Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Identification of Design Patterns in AI Planning Software

Robert Philippsohn

Course of Study:	Software Engineering
Examiner:	Dr. Ilche Georgievski
Supervisor:	Dr. Ilche Georgievski
Commenced:	October 16, 2023
Completed:	April 16, 2024

Abstract

The field of AI Planning has undergone significant growth since the advent of the STRIPS planner in 1971, fueled by the need to tackle an expanding array of complex problem domains ranging from robotics to quantum computing. However, the burgeoning landscape of planners and tools raises concerns regarding software quality assurance amidst increasing complexity. Design patterns offer a promising avenue for addressing this concern, providing structured solutions to recurring design problems and enhancing software development processes. This thesis investigates the systematic identification of design patterns in AI Planning software, guided by a multi-step methodology inspired by Fehling et al. Through reverse engineering and pattern identification processes, this study explores the prevalence and applicability of design patterns across various AI Planning tools and categories. Our findings reveal the widespread utilization of certain patterns, such as Proxy and Factory patterns, reflecting their compatibility with commonly used programming languages. Surprisingly, no novel design patterns specific to AI Planning software were uncovered, highlighting the need for further research in this area. Additionally, the lack of dedicated architectural documentation in research papers emphasizes the importance of identifying effective design patterns to enhance the overall quality of AI Planning software development and maintenance processes.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial intelligence. 13, 15

HTN Hierarchical Task Network. 16

ICAPS International Conference on Automated Planning and Scheduling. 13

IJCAI International Joint Conference on Artificial Intelligence. 13

IoT Internet of Things. 20

IPC International Planning Competition. 21

MOM Message-Oriented mMiddleware. 19

RQs research questions. 13

1 Introduction

Artificial intelligence (AI) Planning, a research field older than even the release of the STRIPS planner in 1971 [RE 71], has witnessed a remarkable evolution marked by the increase of planners and planning tools. This growth is fueled by the imperative to address an expanding array of complex problem domains that require safety, quality, and efficiency. Interesting examples for such diverse domains come from robotics [KM20], space missions [CCF+06][AGA22], logistics [Dra00], video games [KBK08] and, recently, the fields of quantum computing [BDB+18], intelligent buildings [GNN+17] and autonomous driving [AGPA22]. The culmination of these advancements is often showcased at important events such as the International Conference on Automated Planning and Scheduling (ICAPS) or International Joint Conference on Artificial Intelligence (IJCAI), where the latest innovations in AI Planning software are unveiled and evaluated.

However, the burgeoning landscape of planners and tools presents a pressing concern [Geo23d][GB21]: the assurance of software quality amidst increasing complexity. As the intricacy of planners and tools escalates, so does the challenge of maintaining the quality and comprehensibility of the underlying source code. This dilemma underscores the need for effective strategies to ensure the robustness and reliability of AI Planning software.

One promising avenue to understand the current situation is the utilization of design patterns. Design patterns encapsulate proven solutions to recurring design problems, offering a structured and reliable framework for software development. In the context of AI Planning, the application of design patterns holds the potential to improve architecture design of planning tools, streamline the development processes, enhance code maintainability, and foster scalability.

Central to this thesis is one research questions (RQs) split into multiple smaller RQs, with which we try to find out which design patterns are used in the AI Planning domain. Addressing this question requires a proper methodology for the systematic identification and analysis of design patterns within AI Planning tools and planners. Drawing inspiration from methodologies proposed by Fehling et al. [FBUL15] and through the use of reverse engineering source code into UML diagrams, this thesis sets out to explore the landscape of AI Planning software development. Fehling et al.'s methodology involves a multi-step process, including tool identification, sample selection, and pattern identification. Through this methodological framework, we aim to provide a comprehensive overview of design patterns in usage within the AI Planning domain and offer insights into their applicability across different planners and tools. The concrete RQs for this thesis are the following:

1. To what extend do existing AI Planning tools employ design patterns?
    a) How can the design patterns be systematically identified?

b) Which design Patterns are present in most AI Planning tools and which ones are specific to certain AI Planning categories?

c) Are there any observable patterns in existing AI Planning software that can not be covered by existing design patterns?

In addition to elucidating the methodology, this introduction delineates the structure of the thesis. Following this introductory chapter, Chapter 2 provides a comprehensive background on design patterns and AI Planning, laying the foundation for subsequent discussions. Chapter 3 surveys related work, drawing upon guidelines for pattern identification and insights from adjacent fields. Subsequent chapters detail the methodology (chapter 4), present the results of pattern identification (chapter 5), and discuss the implications thereof (chapter 6). Finally, the thesis concludes with a summary of findings and an outlook on future research directions in chapter 7.

2 Background Information

Before we continue with the further thesis, we want to use this chapter to deliver all the necessary information. The groundwork for this report consists of two parts. The first part is the domain of the AI Planning research field. The second section consists of design patterns in software engineering.

2.1 AI Planning

AI is one of the biggest research fields in computer science, and it is gaining even more traction with the new-found hype through ChatGPT. From this singular research field, many more subfields emerged, like machine learning, knowledge engineering, and AI Planning. Before we can continue, we have to define what AI Planning is, what it is used for, and how other subfields might influence AI Planning. In this section, we will define AI Planning, explain the key concepts, talk about planning models, planning functionalities, and more. Planning in general can have different meanings and goals depending on the person and domain in which it is, so let's define this first.

Definition 1 (AI Planning)
Automated planning is an area of AI that studies the deliberation process that chooses and organizes actions by anticipating their expected outcomes computationally. [GNT04].

The outcome of the task, mentioned in the definition, is a plan. The plan is a result of solving a planning problem.

Definition 2 (Planning problem)
A planning problem consists of an initial starting state describing the world, which we wish to transform into a desired goal state through describing the users objective and a set of possible actions which can change the worlds state [GNT04].

This definition gives 3 important things that every planning problem should have: some initial states, goal states and a set of actions. Those parts of a planning written into a formalized way, would look like Definition 3.

Definition 3 (Restricted Model [GNT04])
Given a planning problem $P = (\Sigma, s_i, S_g)$ *where*

- $\Sigma = (S, A, \gamma)$ *is a state transition system,*
- $s_i \epsilon S$ *is the initial state, and*
- $S_g \subset S$ *is a set of goal states,*

find a sequence of actions $<a_1, a_2, \ldots, a_k$

- *corresponding to a sequence of state transitions* $<s_i, s_1, \ldots, s_k>$ *such that*
- $s_1 = \gamma(s_i, a_1), s_2 = \gamma(s_1, a_2), \ldots, s_k = \gamma(s_k - 1, a_k), and s_k \epsilon S$

We also have to mention that there are multiple planning models. Some of the different planning models are the following:

- **Classical planning**: In this kind of problem, we are given an initial situation, a set of action definitions, and a proposition (goal) to be brought about. A solution is a sequence of actions that, when executed, beginning in the initial situation, brings about a situation in which the goal is true. It is assumed that the planner knows everything that is true in the initial situation and knows the effect of every action [McD00].
- **Temporal planning**: In temporal planning, actions do not sequentially follow each other but may temporally overlap and interfere. Whether other actions are being taken may determine the possibility of taking action. The effects of an action may be a complex function of the state and other simultaneous actions, whereas in classical planning they are independent of other actions [Rin07].
- **Hierarchical planning**: Instead of the classical approach, Hierarchical Task Network (HTN) planning involves breaking down tasks into smaller, simpler tasks. It starts with an initial state description and a list of tasks to accomplish, along with rules about how tasks can be broken down into smaller tasks. The planning process continues until all tasks are broken down into their simplest forms, resulting in a plan of action for achieving the desired outcome. This plan consists of a set of straightforward tasks that can be applied to the starting situation [GA15].
- **Non-deterministic planning**: Deciding which actions to execute next in order to achieve a goal is the problem of planning. These actions can have deterministic or non-deterministic effects, with or without a model over their occurrence probability. Furthermore, the world can be either fully, partially, or not observable at all. In a deterministic, fully observable case, the outcome of an action is fully predictable and results in a single state. In a non-deterministic setting, though, the next state depends on which effect of the action actually occurred [MR15].
- **Probabilistic planning**: The probabilistic planning problem is defined as a non-deterministic problem, except that each action has probabilistic outcomes, the initial state is a probability distribution over states, and it has a minimum probability that the plan must satisfy the goal [BCK11].
- **Nummeric Planning**: This extends classical planning by incorporating numerical values into the planning model. In numeric planning, we have an infinite space of numeric states, and the transition function has a possible infinite number of state transitions. It is useful for modeling problems involving the availability of resources, physical quantities like temperature, or spatial information like GPS positions. [GS20].

Aside from these planning problems, we have also learned about planning, which is an extension of the other problem types. Those planners utilize machine learning techniques such as neural networks, decision tree learners, or various forms of regression to enhance efficiency [Kot16]. Considering we want to look at the development of AI Planning tools, we also should take a look at how we develop AI Planning tools. One model shows the software development lifecycle for AI Planning systems, which consists of 10 different phases: *Requirements Analysis, Planning Model Selection, Domain Model Design, Architecture and Design, Planning Technology Selection, Implementation, Testing, Deployment, Monitoring* and finally the *Analysis* phase [Geo23a][Geo23c].

Looking at this general process, design patterns would definitely influence the *Architecture and Design* phase and might be influenced by the *Requirements Analysis* phases, as the decisions of certain demands like scalability or efficiency might influence the choice of design patterns.

2.2 Design Patterns

Design patterns provide blueprints that developers can follow to apply well-known solutions to typical recurring problems. The concept of design patterns originated from architecture and was adapted for use in software engineering. Design patterns serve as abstract templates for software design and cannot simply be implemented directly into the source code. One of the first written collections, that has been shared, was published by the 'Gang of Four' in 1995 [Eri95]. This book by Gamma et al. collects a plethora of different designs, describes how to use design patterns and which pattern could be used for which problem. The underlying idea of design patterns is that they offer a common vocabulary for design, reduce system complexity by naming and defining abstractions, provide a foundation of experience for creating reusable software, and serve as fundamental building blocks for constructing more complex designs [Eri95]. The form in which design patterns are documented is also referenced by Gamma et al. and consists of 11 items. The form of description of a design pattern, mentioned by Gamma et al. is the following: Design pattern name, Intent, Motivation, Applicability, Participants, Collaborations, Diagram, Consequences, Implementation, Examples, See Also. But the most essential are the *pattern name*, the *problem*, the *solution* and the *consequences*. Apart from the form of description of a design pattern, we categorize them according to their scope and purpose. The first criteria, '*Scope*', describes over which domain the patterns is applied. There are two domains we mainly differentiate between: class and object. Class jurisdiction concerns itself with the relationship between classes and subclasses, while object jurisdiction handles relationships between peer objects; which can be more dynamic than classes [Eri95]. The second criteria is called '*Purpose*' and it reflects what a pattern does. For this criteria we also have 3 types: creational, structural and behavioral. Creational patterns concern themselves with the process of creating an object. The structural patterns deal with how classes and objects are composed, either though inheritance or assembly. Lastly we have the behavioral patterns, which describe how classes and objects interact or distribute responsibilities; either through inheritance or cooperation [Eri95]. The original code patterns described in the book of Gamma et al. are depicted in Figure 2.1.

The purpose of design patterns is to deliver know-how and document proven solutions to certain problems for future developers [Eri95]. The effects of design patterns on the software quality has been proven to be mainly in the maintainability attribute [WA20], but the outcome is depending on the prior expertise of the developer applying the pattern [AQ14]. We can also see effects on code smells, where classes that use design patterns have less code smells than classes that do not use them [AAA20].

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method (107)	Adapter (class) (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (object) (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Flyweight (195) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Observer (293) State (305) Strategy (315) Visitor (331)

Figure 2.1: Design pattern space of Gamme et al. [Eri95]

3 Related Work

In this chapter, we want to give an overview of the related works on design pattern identification or designing AI Planning tools. First, we talk about papers in the field of AI Planning, and after that, we broaden the spectrum for all AI-based papers. Lastly, we also include works of research that inspired the process of design pattern identification in other regions of computer science.

3.1 AI Planning

In the AI Planning research field, we could not find any papers talking about design pattern identification in AI Planning. This is why we want to conduct this thesis—to break into this space and inspire others to research this area. In the AI Planning field, there are papers discussing the design process and architecture of AI Planning tools. The first one is talking about the general lack of knowledge and mechanisms for designing AI plans and making deployment and interoperability easier [Geo23d]. It highlights the challenges and proposes answers in the form of architectural patterns and service orientation. The three main challenge areas are development and deployment, process, and architecture. While our thesis does not concern itself with deployment, architecture is a topic of ours, and development too, to a certain degree. While the paper talks more about a service-oriented solution or Message-Oriented mMiddleware (MOM), it also talks about some patterns, like the Strategy design pattern for language selection. Another paper talks about a toolbox (PlanX) for building and integrating AI Planning systems with a full operational cycle [Geo23b]. In that paper, we examine how the toolbox is constructed, the types of components it includes, and the types of architectures utilized. It includes tools like PDDL4J, VAL, or the Planning.Domains web service and utilizes main and dead letter queues in the MOM. It also discusses which user groups are involved in designing and developing an AI Planning system and for whom this toolbox is intended. While that paper demonstrates how to design and create an AI Planning system, it does not address design patterns. A third paper in the AI Planning field tries to explain the usefulness of design patterns in the AI Planning context [VM21]. It argues that reusable abstractions can help in different aspects, but especially with the explainability of domain-independent planning systems. It offers an incentive for why design patterns should be used in AI Planning and delivers through the *Mobile* design pattern. Yet this is as far as it gets; it does not analyze tools to find already existing patterns or give us an example of how to implement other patterns. Also, it limits itself to the usage of design patterns at the domain model level, not in the software itself.

3.2 Works in the broader AI domain

For the AI-based fields, we have one study with the goal of providing an overview of design patterns in AI-based systems, categorizing them, and helping researchers and practitioners alike [HHB23b]. In that study, AI-based systems are described as systems that include AI components. At the end, it resulted in an archive of different patterns that can be accessed on a web archive [HHB23a]. This paper is similar in goal but different in scope. While Heiland et al. focus on all AI-related domains, like machine learning and other fields, we focus solely on AI Planning systems. Another study analyzed the design patterns in the machine learning research domain [WUKG19]. In that study, they collected and analyzed different design patterns for machine learning systems. They conducted the research by surveying the developers of those systems and conducting a systematic literature review. The findings give more insight into the topic of design patterns in the development of machine learning systems. In comparison to our study, it also analyzes the field in question for design patterns and gives us an overview of the current state, but we focus our attention on the AI Planning domain, while they are interested in the machine learning field.

3.3 Works outside the AI domain

In this section, we have a plethora of disciplines in the computer science field. For example, one paper that talks about design patterns outside the AI sphere discusses patterns in the context of Internet of Things (IoT). In the paper of Reinfurt et al., they analyze multiple numbers of production-ready IoT offerings and extract reoccurring solutions into patterns [RBF+16]. In their paper, they talk about five patterns in general and explain how they work. Reinfurt et al., along with the same team, conducted a follow-up study documenting additional design patterns [RBF+19]. The number of additional patterns documented is three. That paper goes even further than the simple identification of design patterns and enters the realm of design pattern authoring. Highlighting new paradigms is not the primary objective of our paper, but if we come across a certain paradigm used in multiple tools, we might also attempt to emphasize those new paradigms. Another interesting paper with design patterns is present in the realm of quantum computing. The paper by Weigold et al. formulates common data encodings into a design pattern format[WBLS14]. Sketches describe 'how' an encoding works, and the consequence section of each design pattern answers the question 'Why should you choose this encoding?' Overall, this paper documents only three design patterns for quantum computing, but more might follow in a future study. Here we have the same thing; our thesis tries to analyze the tools to identify known design patterns, but here we have new paradigms documented and no mention of well-established patterns. In the field of cloud computing, we also have the papers of Fehling [Feh15]. This piece of literature does more than just design pattern identification. It goes two steps beyond and does design pattern authoring and design pattern application. We do not require those steps in our thesis, but they could be pursued if this thesis yields promising results. Overall, we get a deep dive into all the main three phases, including all the steps needed for each phase. We get a textual and graphical format for the design patterns identified and authored, as well as instructions on how to use the patterns. That paper serves as inspiration for how we will conduct our thesis, especially the design pattern identification phase.

4 Methodology

This chapter focuses on describing and depicting how this thesis is conducted. We examine how we gather possible sources and explain how we identify design patterns and the basis for our research. This part might also serve as a blueprint or example for similar studies with the same design or objective.

4.1 Gathering Process

Sadly there is no complete registry for all planning tools available. One known registry is the planning.wiki [GRM+], which has an incomplete collection of AI planners. Some of the planners have a link to their scientific paper and even fewer have a link to their home page, but overall we cannot get the source code and the research paper for all listed planners. Another point is that there are only planners registered, no other tools like validators, parsers or other AI planning tools. The second option to obtain AI planning tools, is tedious work and we have no certainty that we will have a complete and diverse set, an online search. For that we can use well known resources, like scientific libraries (Elsevier, IEEE Xplore, etc.) or google scholar. Lastly, we have the option to look at dedicated conferences, like the ICAPS or IJCAI. This is also the option we took, as we chose the International Planning Competition (IPC). The IPC is a competition hosted by the ICAPS, where many AI planners compete in their respective category (classical, temporal, etc.) and a research paper and the source code are made public for each participating planner. The competition is normally held every two years. For the collected planners, we began with the first planning competition in 1998 and included all the planners mentioned, up to the most recent competition in 2023. For the tools that are not planners, such as validators or parsers, we conducted an online search and also included tools mentioned in the research papers of the collected planners. It's possible that some of the planners included in our final set might be duplicates. However, we consider them unique enough for this initial identification attempt because our final set comprises tools diverse in name, source code link (if present), and document link (competition paper or documentation link). Figure 4.1 depicts how the gathering of our AI Planning tools set was conducted.

Figure 4.1: Gathering progress

4.2 Design Pattern Identification

As the title of this thesis suggests, we try to identify patterns for the software development of AI Planning tools and planners. To identify design patterns in code, we have multiple methods to do so. One possibility is using a two-phase approach, where we have a dataset preparation phase in the first half and, for the second half, we utilize supervised learning algorithms to identify design patterns in source code [DTR19]. Another approach, presented in 1998, tries to use a multi-stage reduction strategy combined with object-oriented software metrics to extract patterns from the design papers or code [AFC98]. A third method is part of a bigger process and also the option we are using for this thesis. Fehling et al. not only display the pattern identification step but also the pattern authoring step and the pattern application step [FBUL15]. The initial phase is the *pattern identification* phase. The structuring and comprehensive collection of domain-specific information characterize it, the very part within which patterns are to be discerned. The principal aim of this phase resides in the systematic organization of the chosen domain, ensuring a conducive environment for the subsequent identification of patterns. Additionally, it requires clearly defining the language and visual elements essential to describing patterns. This ensures a consistent and polished presentation in every documented solution. These actions have an even higher significance in instances where the coordination of larger teams of pattern researchers is required. This also the only phase we are focusing on, but we will give a short explanation for the two other phases. The second phase is called the *pattern authoring* phase. In this phase we would write patterns based on the similarities of existing solutions, which were collected in the previous phase. Lastly, we have the *pattern application* phase. That phase can be performed independently of the previous two phases once we have patterns that can be applied, as said by Fehling et al. [FBUL15]. All three phases have sub-steps that can be iterated through, exactly like the three phases themselves that can be done multiple times.

4.2.1 Domain Definition

The first step of the *pattern identification* phase is the domain definition. Generally, the goal of this step is to clarify and create common knowledge in the domain, where we try to find patterns. This is especially important for a group of researchers working together. To achieve this, we have to "describe with written text and well-accepted definitions."[FBUL15]. If there is not enough information, we would have to create definitions of the domain concept. For a definition of the AI Planning domain, look at chapter 2 in the AI Planning section.

4.2.2 Coverage Consideration

The second step of this phase is coverage consideration. In this step, we try to limit the scope of information we have to consider and look out for. In our case, we limit ourselves to AI planners, domain editors, plan parsers, and validators. Apart from that, we restrain our focus on solutions fulfilling principles such as scalability, efficiency, and explainability.

4.2.3 Information Format Design

With the third step, called *information format design*, we try to create a document where we can note down our findings. In this case the template includes the following information:

- AI Planning tool type: is it a planner, a validator, editor or parser
- Information source: source code, research paper or a combination
- Track: if it is planner, which track did it enter
- Year: When was it presented?
- Design pattern: What patterns where identified?
- Reason: Why those patterns?

4.2.4 Information Collection

For the fourth step of the identification phase, we have the *information collection*. That phase is defined by capturing the solutions and patterns found in the information format we defined in the step before. To do that, we need to review the documents for descriptions of how certain properties or principles are addressed. As a source for the patterns we identify, we mainly utilize the common design patterns from the Gang of Four [Eri95], as well as an archive composed by Heiland et al[HHB23a]. To be more concrete, we read through certain sections in the research papers of the planners or the available documentation of the tools, took notes on how they work, and compared the notes with the descriptions of all the different design patterns. Reverse engineering is another way we utilized to identify design patterns in the AI Planning tools. We used the trial version of the software *Enterprise Architect* [Sys] to reverse engineer the available source code of the included AI Planning tools. Through this process, we got the UML class diagrams, where we looked for certain structures or for class names indicating the usage of known patterns.

4.2.5 Information Review

The last step of this phase is the *information review*. Here we have to provide manageable sets of the current solutions taken into consideration for pattern discovery; the domain organization is improved. Queries may be done on the collection of documented existing solutions to locate comparable ones, depending on the information type.

5 Results

This chapter concerns itself with the presentation of the outcomes of pattern identification. We will talk about the results and describe them. The discussion of the findings and attempt to answer the research questions are given in Chapter 6.

5.1 General

5.1.1 Progression

Figure 5.1 shows the progression of how we went from initially 430 AI Planning tools down to 216 different AI Planning tools. The initial 430 AI Planning tools included many planner duplicates, because some planners competed in multiple competitions over the years. One example is the *Gamer* planner [EK08], which competed in the competitions of 2008, 2011 and 2014. We purged such obvious examples from the initial set to ensure that each planner is represented only once, resulting in a decrease down to 344 AI Planning tools. Afterwards, we removed similar duplicates that had the exact same link for the code base or their research paper, which led to the number of AI Planning tools dropping down to 324. We should clarify that some planners that participated in the IPC 2004, 2011 and 2014 have the same link, because some of those planners were present in a booklet summary for the deterministic tracks. Next, we removed the updated versions of planners, indicating planners that participated in later competitions not under the exact same name but with a suffix indicating their relation to the original, for example, through something like '2.0'. This brought the number down to 261. The last pruning of planners came because some of the papers or code bases for the planners were not accessible anymore; for example, the site for the temporal track of the IPC 2018 was not available anymore. This led to the decrease down to 216 AI Planning tools that were analyzable. Cases where one planner is similar to another planner, were not outright removed. For example there are many planner that use Fast Downward [04] as their base and combine it with another planner or change its purpose from a classical planner to a temporal one, which were not purged from our set of AI Planning tools.

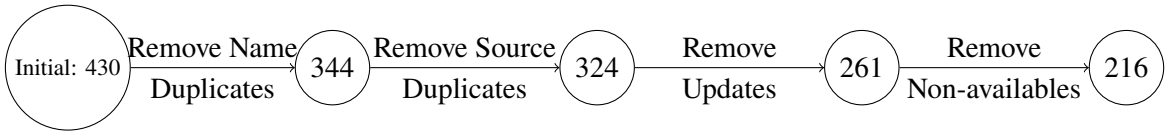

Figure 5.1: Number of papers

5.1.2 Release Year

In Figure 5.2 we see the depiction of the release year of every AI Planning tool we included in our identification process. Overall, we can see that we have 216 unique tools that we analyze for different design patterns. Those tools were either sorted into their respective year, because they competed in that year for the first time at the IPC or because that tool has its first mention or commit in that year. Most of the original planning tools used in this analysis come from 2023, and make up 37 tools (17,13%), while the second highest influx was in 2011 with 36 tools (16,67%). Following these years, in order of most new tools, are the years 2014 with 31 tools (14,35%), 2008 with 28 instruments (12,96%), 2018 introducing 26 implementations (12,04%), 2004 showcasing 24 new planners (11,11%) and the 2000 with ten new tools (4,63%). The competition of 2006 contributed with 8 planners (3,7%) to our thesis, and the years 1998, 2002, and 2020 have only four tools (1,85%) that were included in this thesis. The years 2012 and 2015 had no IPC, so those years constitute a smaller portion of the analyzed set, with one entry (0,46%) for the year 2012 and three entries (1,39%) in the year 2015.

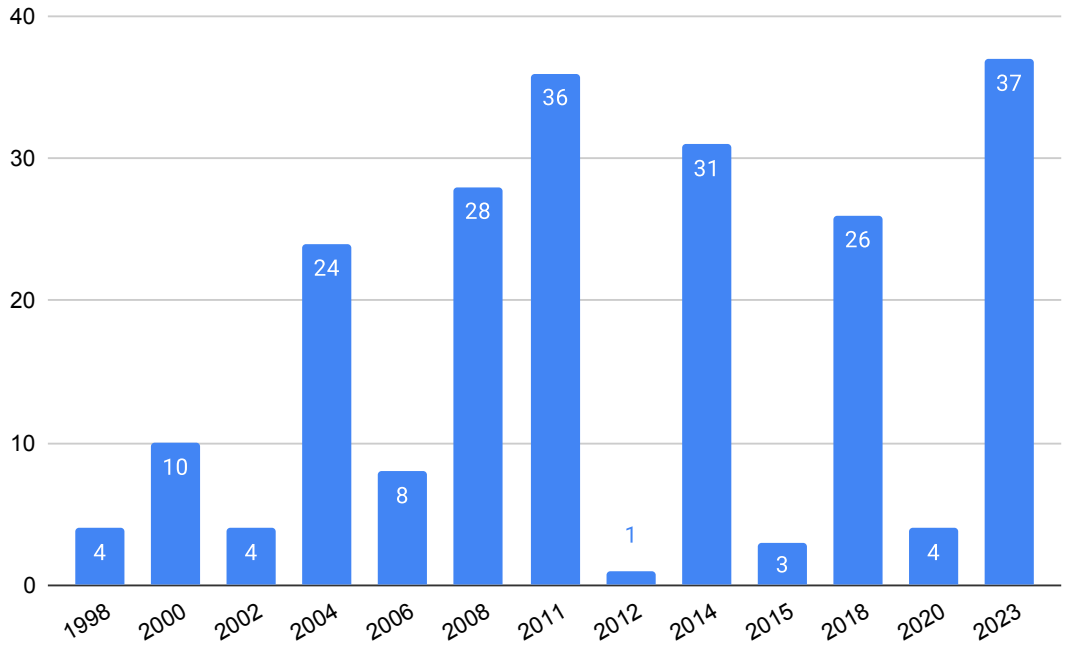

Figure 5.2: Release Year of all included AI Planning tools

5.1.3 Tool types

The next graphic, Figure 5.3, shows the different tool types. The tools are distributed between the following types: planners, validators, editors, and parser. Of those 216 AI Planning tools are 210 classified as planners. All the 210 planners (96,77%) come from the planning competitions, which started in 1998 up to the latest was in 2023. The other seven tools where not presented at the ICAPS planning competitions, but where either extracted from the papers that explained the

planners in the planning competitions or could be found by using a simple internet search. Four (1,83%) of those seven tools are classified as *editors*, two (0,91%) are sorted into the category *parser* and only one tool (0,46%) is labeled as a *validator*. The validator is *VAL* [FLHC14], the parsers are *Universal PDDL parser* [Jon15] and *PDDL4J* [PF18] and lastly the four editors are *itSimple* [VSTC13], *Planning.Domains* [Mui15], *myPDDL* [SK20] and *PDDL Studio* [PCBB12]. For a more detailed look what planners where included take a look at the appendix for Table A.2.

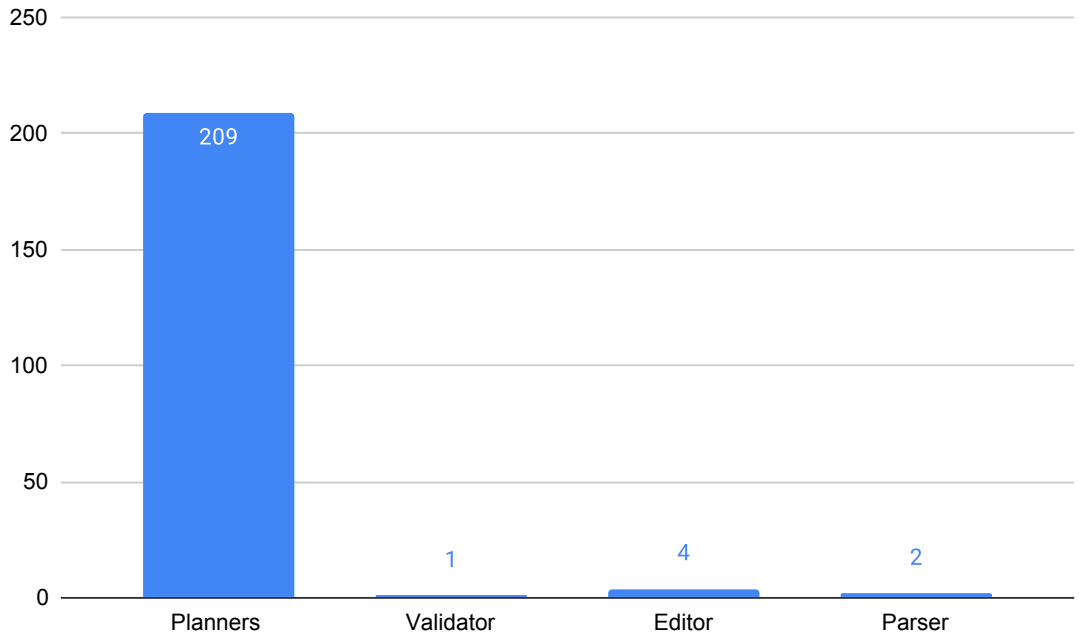

Figure 5.3: What AI Planning tool types were included

5.1.4 Planner Types

Figure 5.4 shows us what kind of planners were included in this study. In figure 5.3, we concluded that there are only 209 planners in this study, but some of the planners participated in multiple competition tracks; for example *DAE-YAHSP*[11] competed in the temporal track and the classical track. There are also cases, where planners competed in the classical track of the IPC, but are no classical planner, for example TALplanner [KD00]. Knowing that, we can now describe the graph. Most of the planners are classified as classical planners, with 125 planners (59,81%) existing in this category. After that we have probabilistic planners in second place, with 27 entries (12,92%), and the learning category in third place, with 26 entries (12,44%) in the set of planner. Afterwards, we have the temporal planners, with 19 different (9,09%) AI planners. Next up we have 11 hierarchical competitors (5,26%) that are included in the set used in this study. After that, there are five different non-deterministic planners (2,39%) that we analyzed in this study. Lastly, we have two numeric planners (0,96%) we had analyzed for this study; for both the correlating paper and code.

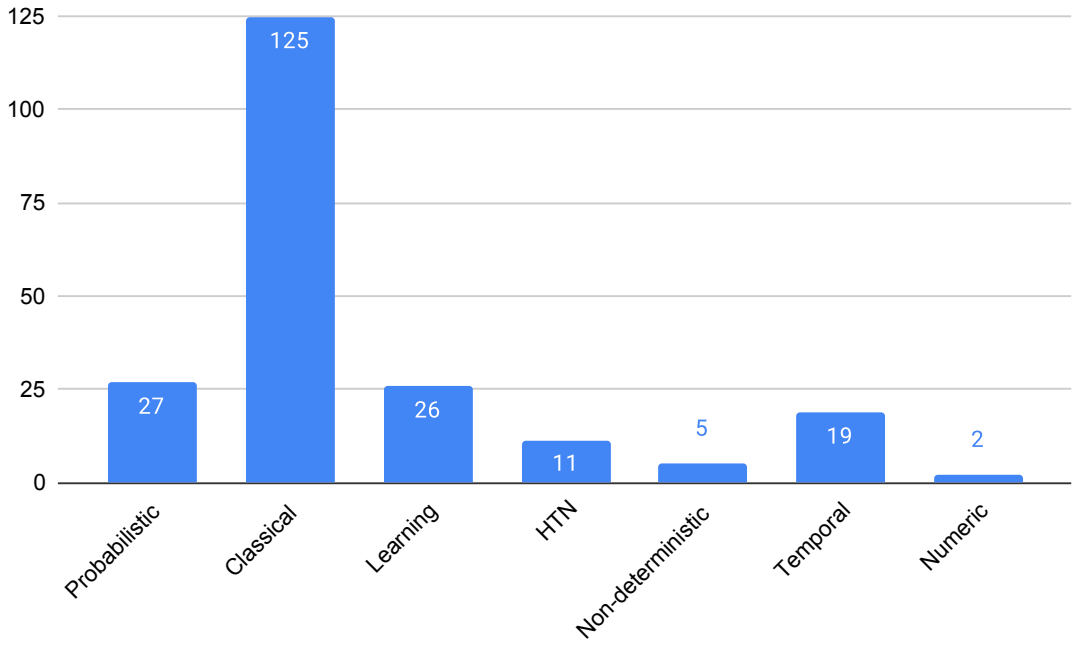

Figure 5.4: Distribution of included AI planners types

5.2 Patterns

5.2.1 Patterns per planner type

In Figure 5.5 we can see the number of patterns found depending on the AI planner type. We have still the same types as in figure 5.4 and the other planning tools (validators, editors and parser) are not included in this graphic. As we can see that 57 classical planners have at least one identified pattern, which means 43,85% of all included classical planners have some sort of identified design pattern. Second biggest groups overall, were probabilistic planners and learning planners but unlike before they do not share the same place in the this comparison. From the 27 learning planners 18 have at least one pattern, which means 66,67% of all learning planners have one identified pattern. On the other hand, from the 27 probabilistic planner only six (22,22%) have any number of identified patterns. Next up, we have the HTN planners where we have two out of 11 planners (18,18%) that have any design pattern. For the temporal planners we have ten out of 14 planners (71,43%) where we could identify any number of design patterns. For the non-deterministic category we have only one planner out of five that has a design pattern identified, which means 20% of those planners contributed meaningfully to our thesis. Lastly, we have full two out of two numeric planners, where we could find at least one design pattern.

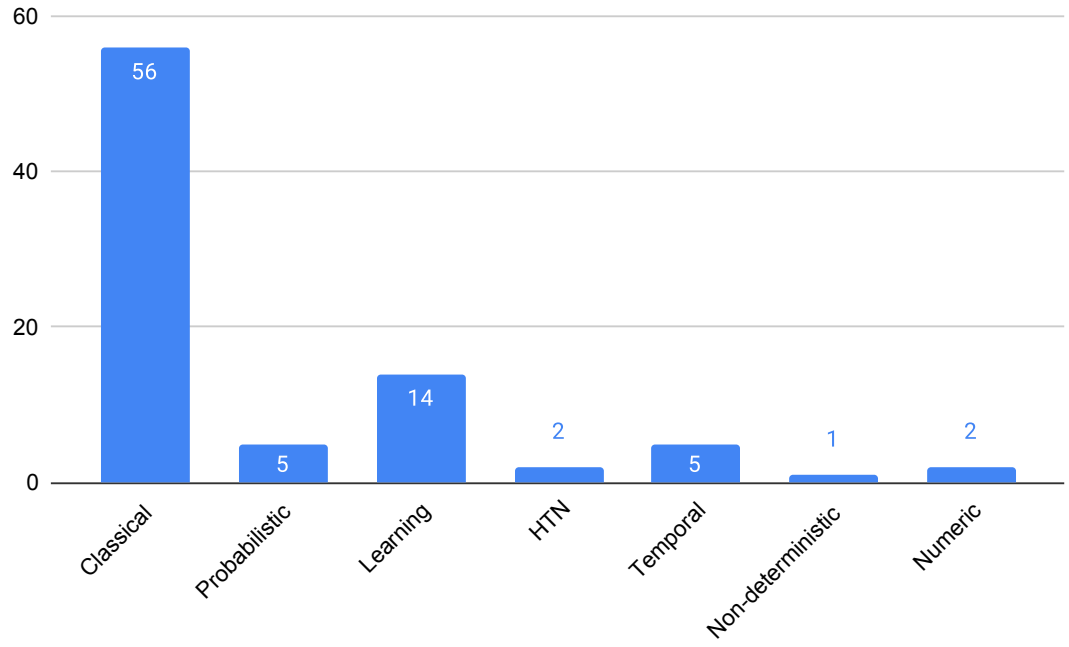

Figure 5.5: The number of patterns found per planner type

5.2.2 Found patterns

Figure 5.6 shows us all the identified patterns for all the AI Planning tools. Before we talk about the patterns we identified, we have to mention that for 123 tools (56,94%) we did not identify a single design pattern; neither in documentation, research paper or in the source code. With that out our way, we can describe the graph further. Two patterns that were sighted mostly together are the Proxy pattern, with 62 appearances (28,70%), followed by the Factory pattern, with 61 occurrences (28,24%), both as described by Gamma et al.[Eri95]. After that we have the Template Method pattern as defined by Gamma et al.[Eri95], with a total of 16 identifications (7,41%). Closely followed comes the Strategy pattern as described by Gamma et al.[Eri95], with 11 occurrences (5,09%). Next up, we have the State pattern as defined by Gamma et al.[Eri95] with ten identified appearances (4,63%) and after that the Registry pattern as described by Fowler et al.[FRF+02] with eight times (3,70%) coming up. Afterwards we identified the Iterator pattern, as explained by Gamma et al. [Eri95], two times (0,93%) in all analyzed AI Planning tools. Lastly, we have the Command pattern and the Singleton pattern as defined by Gamma et al. [Eri95], both identified only once (0,46%). Definitions of the mentioned patterns can be found in the Appendix A.1.

5.2.3 Pattern combinations

Next up we analyzed which pattern combinations were found, in Figure 5.7. This means we look up which design patterns were most commonly seen together. But before we can continue, we have to explain what the abbreviations mean. For simplicity sake, we took only the first letter of the pattern name, for example abbreviating *Factory* to F. In some cases patterns have the same

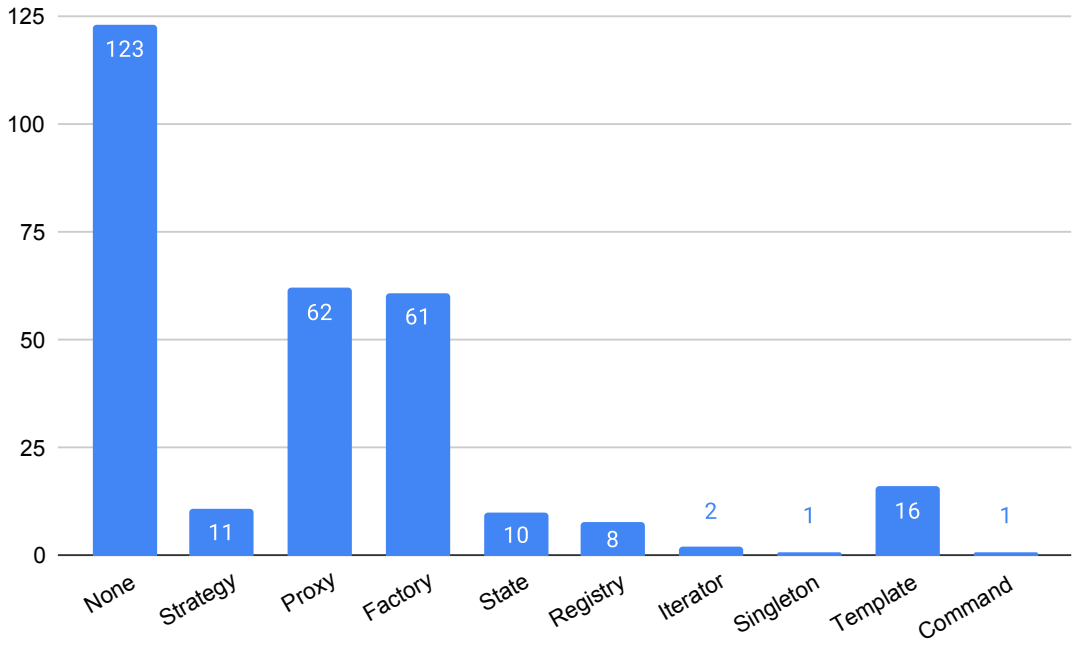

Figure 5.6: Number of identified patterns or the lack thereof

first or even second letter, leading to abbreviations Pr being *Proxy*, Sta being the short version for *State* and Str meaning *Strategy*. With the knowledge of all identified patterns from Figure 5.6 and the abbreviations for the pattern names we can dive into the found data. The pattern combination that occurred the most was the pairing of *Proxy* and *Factory*, appearing in 37 AI Planning tools (17,13%). On second place we 10 appearances (4,63%) of the combination *Proxy*, *Factory* and the *Template Method* pattern. Next on the list, we have the *Proxy*, *Factory*, *State* and *Registry* amalgamation, with 4 occurrences (1,85%) in all 216 AI Planning tools. After that we have the union of *Proxy*, *Factory* and *Registry*, with 3 sightings (1,39%).
Lastly we have a plethora of combinations that were only seen once (0,46%) and those pairings are the following:

- *Proxy*, *Factory* and *Strategy*
- *Proxy*, *Factory*, *Registry* and *Command*
- *Proxy*, *Factory*, *Strategy* and *Template Method*
- *Proxy* and *Template Method*
- *State* and *Template Method*
- *Proxy* and *State*
- *Proxy*, *Factory* and *State*
- *Factory* and *Iterator*

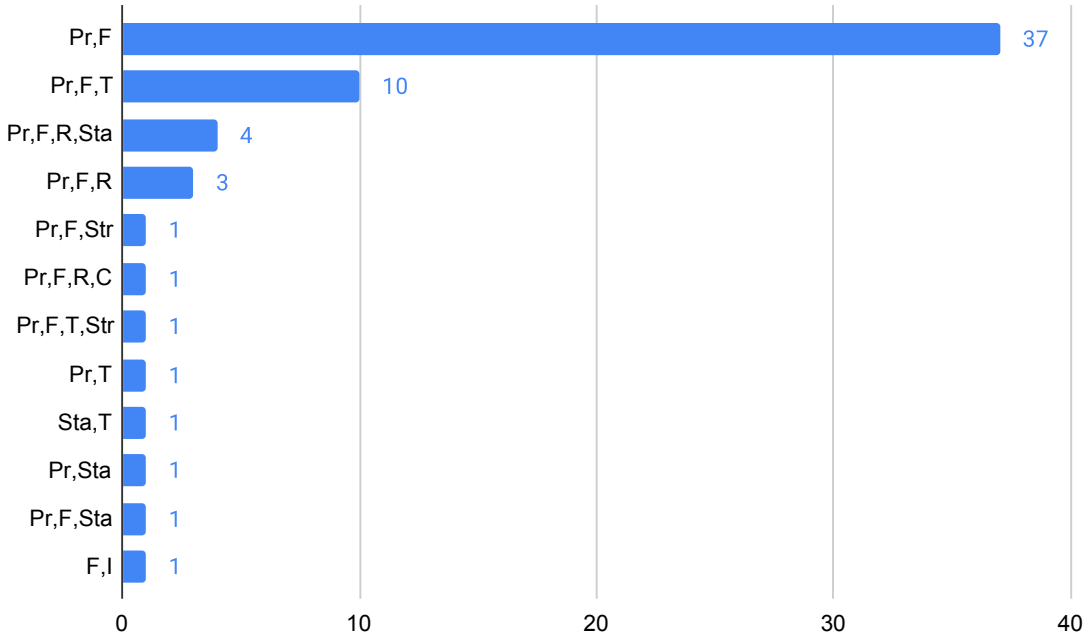

Figure 5.7: Detected pattern combinations

6 Discussion

This chapter discusses the results shown in chapter 5 and tries to answer the research questions stated in chapter 1. We also discuss the validity threats that have to be considered for this study and how we tried to minimize those.

6.1 General observations

First of, we should talk about the results of Figure 5.2. The AI Planning competition first started in 1998 at the AIPS conference and had five planners compete, according to McDermott [McD00]. Afterwards the planning competitions where held almost every two years, which explains the steady influx of included AI Planning tools in this two year pattern. Outliers are the years 2011, which marks the first irregular planning competition, the year 2012, which marks the release of planning editor *PDDL Studio* [PCBB12] and the year 2015, which was the year when planning parser *PDDL4J* [PF18], *Universal PDDL parser* [Jon15] and the online editor *Planning.Domains* [Mui15] were released. Another irregular held planning competition is the one from 2023, which had to be delayed because of the Covid pandemic. The mostly regular influx of tools through planning competitions explains why we have so many AI planners in our set of AI Planning tools, as seen in Figure 5.3. But it does not really explain the low number in other planning tools, for this thesis mainly the validators, editors and parsers. Some of this tools were showcased at ICAPS, for example *itSimple* [VSTC13], which shows we should not have search for those tools only through online search, a Google scholar search and mentions in planner papers; but also through a search of every available ICAPS paper. That would be more time consuming, but might yield a richer variety in AI Planning tools. Through Figure 5.4 we get further insight into the AI Planning planners distribution. As we can see most of the included planners are categorized to be classical planners, fitted for an agile, optimal or satisfying track. That is hardly surprising as most planners from the older generation are classical planners. Planners competing in the learning track, which sometimes utilize machine learning, are a newer breed of planners, were one of the first submitted planners for the learning track was Cabala [ROB08] in 2008. The planners in our set for the HTN category come only from the year 2023 IPC, which might indicate an error in our assessment or an increase in interest for those types of planners.

Something we observed, but did not track, were the planning tools that have a dedicated section for the architecture or design of the tool. We encountered only a few number of tools that had such a section were, for example, AltAlt from the year 2000 [SNK+01] or Crikey from the year 2004 [04]. For the planners, that might be excusable because the planner abstracts for the IPC are fairly short with one to five pages. But for planning tools that have a documentation site or a website and the architecture is not explained, that seems like a shortcoming.

6.2 Patterns

In Figure 5.5 we can see the number of planners per planner type, with at least one design pattern. Although we can see that the classic planner category contains the most planners where design patterns have been found, in percentage it is not on top of the field. In percentage terms, temporal planners and learning planners have a higher rate of hits when it comes to finding design patterns, but it is unclear if it stays the same if we would look at a comparable number of temporal and learning planners as we did with classical planners. Interestingly enough we have a 100% identification rate for the two numeric planners, for which we found the documented patterns solely through the reverse engineered UML diagrams. One of the patterns that we found solely through the reverse engineering process, is the *Proxy* pattern and another is the *Factory* pattern, which are also notably the patterns we found most of the time. If we look at the tools we analyzed and the *Proxy* or *Factory* pattern we documented for them, some might notice that not all tools have source code we might reverse engineer, but we will notice that those planners use Fast Downward [04] as a base or as a part of their new planner, which in turn means that the new tool uses those patterns as well. Other examples for the patterns that where found through reverse engineering are the *State*, *Registry* and *Template Method* pattern. But we have to mention that the application used for reverse engineering was not able to reverse engineer programming languages like Rust, Lisp or Prolog. Supported languages were only the most common object-oriented programming languages, like C++, Java or Python. Patterns we identified through the design pattern identification process, as described by Fehling et al. [FBUL15], are the *Strategy*, *Iterator*, *Command*, and *Singleton* pattern. Considering all the identified patterns, we can note that we mainly analyzed traditional, architectural design patterns, as defined by Gamma et al. [Eri95] or Fowler et al. [FRF+02].

Now that we know which patterns we have identified, we can look at the combinations that we most commonly observed. As Figure 5.7 displays it, the most common pattern pairing is the *Proxy* and *Factory* combination, which is not to surprising as we know that those two patterns are the most common one. We speculate that *Proxy* and *Factory* is that high, because those patterns are common practices in object-oriented languages like C++, Java, or Python; which are also the most common programming languages for AI Planning tools. Both patterns have the drawback of making the code more complex and introducing overhead, but the *Factory* pattern gives us more flexibility, abstraction and organization and the *Proxy* pattern benefits of optimization and transparency. Another pattern that influences the flexibility of some tools is the *Strategy* pattern, but the downside is, again, an increase in development complexity [HHB23a].

Initially, we documented another pattern based on the workflow commonly observed in AI planners. The patterns we initially considered were the *Pipes-and-Filters* pattern or the *Pipeline* pattern because most planners involved some form of preprocessing before the search and planning phase or a post-processing phase after the plan was found. Typical preprocessing steps included translation, pruning, grounding, parsing, or abstraction.
For example, Fast Downward describes its workflow in three phases: translation, preprocessor, and search engine. The translation phase prunes redundant ADL features, grounds the operators and axioms, and simplifies the problem presentation. Preprocessing for Fast Downward includes causal graph computation, domain transition graphs, and handling complicated actions. It determines how things are connected in a planning task, simplifies those connections, and creates graphs showing how actions affect those things. All these steps occur before the actual search process. Other

planners have different steps performed before searching for a plan. For instance, the Scorpion planner computes an abstraction, and then the search heuristic runs. For planners that describe using an improvement process after a plan is found, examples include Arvand Herd [11], LPG-TD [04], and Spock [SFS23]. All three mention some form of pruning, where redundant actions can be removed from the plan without invalidating it. Initially, these workflows led us to believe that we have Pipes-and-Filters or Pipeline patterns present. However, these workflows are not flexible and cannot be changed without major alterations in the source code. Nonetheless, we observe this workflow in over 100 planners out of 209. The inflexibility inherent in these workflows is the reason for this omission. Unlike traditional design patterns, like Pipeline, Pipes-and-Filters, or Chain of Responsibility, which offer modularity and flexibility, the observed workflow in AI planning tools often requires significant alterations in the source code to accommodate any changes. This lack of flexibility makes it challenging to encapsulate the workflow within a single, adaptable pattern. Moreover, while the observed workflow is widespread, its implementation varies significantly across different planning tools. Each tool may prioritize different preprocessing steps, post-processing strategies or may not have either of those two, based on their specific requirements and optimizations. This heterogeneity further complicates the proposition of a unified design pattern. However, it is evident that recognizing and formalizing this observed workflow could constitute an important contribution to the field of AI planning. By demonstrating the mutuality and variations within this workflow, it may be possible to propose a new design pattern tailored specifically for AI planning. Such a pattern could provide guidance for developers in structuring their planning algorithms more effectively, ultimately enhancing the efficiency and robustness of AI planning systems. Therefore, while acknowledging the omission of proposing a new design pattern in the thesis, the potential for this observation to inspire a valuable contribution to the field remains a promising avenue for future research and development in AI planning methodologies.

6.3 Validity threats

Every scientific paper has to ensure the robustness and reliability of its findings. However, it is essential to acknowledge and address potential validity threats that may compromise the integrity of the study. Validity threats are factors that can undermine the validity of research results, leading to erroneous conclusions or interpretations. In this section, we examine potential validity threats inherent in our study methodology and data analysis. By identifying and understanding these threats, we try to reduce their impact on the credibility of our findings. To show that the research is valid, we have to address different categories: *Construct validity*, *Internal validity*, *External validity* and *Reliability*.

To ensure construct validity, we adhere to the identification process described by Fehling et al. [Feh15] [FBUL15], and we use the examples provided by Fehling et al. as references. As this specific research has not been conducted in the field of AI Planning, as outlined in Chapter 3, we had no real expectations for what we would find. To ensure the robustness of this study and its external validity, we analyzed the tools at least three times for design patterns: first while reading the research paper or documentation and taking notes; second, when we reverse engineer the source code and analyze the UML class diagrams; and the third, while reviewing the notes more thoroughly and comparing the description to all pattern sources. This way, we hoped to reduce any observer bias of the researcher and incorporate new knowledge into the final review of all tools. We also tried to mitigate any availability bias that way, which seems to have worked to a degree because this

way we discovered the usage of the *State*, *Registry* and *Template Method* pattern. If we redo the analysis again for all tools, we might get a higher occurrence count for those patterns.
We were also able to eliminate duplicates that slipped through our initial duplicate detection process, by going over our set of AI Planning tools multiple times. This way we were also able to correct errors we overlooked initially.

7 Conclusion

This study investigated the systematic identification of design patterns in AI Planning software. Through methods such as reverse engineering and following established processes, such as those outlined by Fehling et al.[FBUL15][Feh15] in Chapter 4, design patterns can be effectively recognized. The analysis revealed that certain patterns, notably the *Proxy* pattern and the *Factory* pattern, are present in many AI Planning tools, probably because they are commonly used in most object-oriented programming languages. Other patterns that we identified a few times are the following: *State* pattern, *Registry* pattern, *Template Method* pattern, the *Strategy* pattern, and the *Iterator* pattern. Single instances of the *Singleton* pattern and the *Command* pattern were also identified. After thorough investigation, no novel design patterns exclusive to AI Planning software emerged from this study. However, what became apparent was a certain workflow structure present in the majority of AI planners. We can simplify this workflow into three main phases: pre-search, search, and post-search. A significant portion of our tool set had no design pattern identified, which can be attributed to the fact that a lot of research papers for AI Planning tools rarely have a dedicated section for the architecture or design of the tool or have an accessible code base for reverse engineering. Considering all this, the investigation underscores the significance of identifying effective design patterns for enhancing the development, maintenance, and overall quality of AI Planning software.

7.1 Outlook

Through this thesis, we contribute by showing certain shortcomings in the AI Planning domain and displaying some of the most commonly used design patterns. Moving forward, further research into the efficacy and applicability of different design patterns is essential for advancing the field of AI Planning software development. A more thorough analysis with a larger sample size or a more concentrated focus on only planners or other tool types might yield better and more insightful results. The focus of this thesis was the identification of known design patterns, but an approach to finding and authoring novel design patterns specific to the AI Planning domain might also hold interesting findings. An example of a possible novel pattern has been outlined in Chapter 6.2. Such endeavors hold promise for optimizing software architecture, fostering innovation, and ultimately improving the capabilities of AI Planning systems. After conducting more research into this topic and finding more patterns, a guideline for utilizing design patterns in the AI Planning space might be possible, or an analysis of the effectiveness of design patterns might be possible.

Bibliography

[04] "14 th International Conference on Automated Planning and Scheduling". In: 2004. URL: https://ipc04.icaps-conference.org/deterministic/DOCS/IPC-4.pdf (cit. on pp. 25, 33–35, 56, 59–64).

[11] *The 2011 International Planning Competition - Deterministic Track*. June 2011. URL: http://www.plg.inf.uc3m.es/ipc2011-deterministic/attachments/ParticipatingPlanners/ipc2011-booklet.pdf (cit. on pp. 27, 35, 57, 59–64).

[14] "The 2014 International Planning Competition - Deterministic Track". In: (2014). URL: https://helios.hud.ac.uk/scommv/IPC-14/repository/booklet2014.pdf (cit. on pp. 57, 59–64).

[AAA20] M. Alfadel, K. Aljasser, M. Alshayeb. "Empirical study of the relationship between design patterns and code smells". In: *PLoS ONE* 15.4 (Apr. 2020), e0231731. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0231731. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7162509/ (visited on 02/09/2024) (cit. on p. 17).

[AFC98] G. Antoniol, R. Fiutem, L. Cristoforetti. "Using metrics to identify design patterns in object-oriented software". In: *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No.98TB100262)*. 1998. DOI: 10.1109/METRIC.1998.731224 (cit. on p. 22).

[AGA22] E. Alnazer, I. Georgievski, M. Aiello. "On Bringing HTN Domains Closer to Reality - The Case of Satellite and Rover Domains". In: *International Conference on Automated Planning Systems (ICAPS) Workshop on Scheduling and Planning Applications (SPARK)*. 2022. URL: https://icaps22.icaps-conference.org/workshops/SPARK/papers/spark2022_paper_9.pdf (cit. on p. 13).

[AGPA22] E. Alnazer, I. Georgievski, N. Prakash, M. Aiello. "A Role for HTN Planning in Increasing Trust in Autonomous Driving". In: *IEEE International Smart Cities Conference*. 2022, pp. 1–7. URL: https://doi.org/10.1109/ISC255366.2022.9922427 (cit. on p. 13).

[AK01] J. L. Ambite, C. A. Knoblock. "Planning by rewriting". In: *Journal of Artificial Intelligence Research* 15 (2001), pp. 207–261 (cit. on pp. 56, 59, 60).

[AQ14] F. M. Alghamdi, M. R. J. Qureshi. "Impact of Design Patterns on Software Maintainability". In: (2014). URL: https://www.mecs-press.org/ijisa/ijisa-v6-n10/IJISA-V6-N10-6.pdf (cit. on p. 17).

[Asa18] M. Asai. "Alien: Return of Alien Technology to Classical Planning". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team33.pdf (cit. on pp. 57, 58, 60).

[BCEK23] C. Büchner, R. Christen, S. Eriksson, T. Keller. "DALAI – Disjunctive Action Landmarks All In". In: (2023). URL: https://ipc2023-classical.github.io/abstracts/planner4_dalai.pdf (cit. on pp. 58, 60, 62–64).

[BCK11] D. Bryce, W. Cushing, S. Kambhampati. "State agnostic planning graphs: deterministic, non-deterministic, and probabilistic planning". In: *Artificial Intelligence* 175.3-4 (2011), pp. 848–889 (cit. on p. 16).

[BDB+18] K. E. C. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, J. Frank. *Comparing and Integrating Constraint Programming and Temporal Planning for Quantum Circuit Compilation*. 2018. DOI: 10.48550/arXiv.1803.06775 (cit. on p. 13).

[BF08] D. Borrajo, S. Fernandez. "SAYPHI-RULES. On learning control knowledge for forward chaining planners and use them stochastically". en. In: (2008). URL: https://ipc08.icaps-conference.org/learning/documents/abstracts/abstract_sayphirules.pdf (cit. on pp. 57, 59, 61).

[BG05] B. Bonet, H. Geffner. "mGPT: A Probabilistic Planner Based on Heuristic Search". en. In: *Journal of Artificial Intelligence Research* 24 (Dec. 2005), pp. 933–944. ISSN: 1076-9757. DOI: 10.1613/jair.1688. URL: https://jair.org/index.php/jair/article/view/10435 (visited on 02/03/2024) (cit. on pp. 56, 59, 61).

[BG18] T. Balyo, S. Gocht. "The Freelunch Planning System Entering IPC 2018". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/teams_4_34.pdf (cit. on pp. 57, 58, 60).

[BHBM06] J. Baier, J. Hussell, F. Bacchus, S. McIlraith. "Planning with Temporally Extended Preferences by Heuristic Search". en. In: (2006) (cit. on pp. 57, 59, 60).

[Bit23] A. Bit-Monnot. "Experimenting with Lifted Plan-Space Planning as Scheduling: Aries in the 2023 IPC". en. In: (2023) (cit. on pp. 58, 60).

[BKD06] J. Benton, S. Kambhampati, M. B. Do. "YochanPS: PDDL3 Simple Preferences as Partial Satisfaction Planning". en. In: (2006) (cit. on pp. 57, 59, 60).

[BKS06] D. Bryce, S. Kambhampati, D. E. Smith. "Planning Graph Heuristics for Belief Space Search". en. In: *Journal of Artificial Intelligence Research* 26 (May 2006), pp. 35–99. ISSN: 1076-9757. DOI: 10.1613/jair.1869. URL: https://jair.org/index.php/jair/article/view/10453 (visited on 02/03/2024) (cit. on pp. 57, 59, 60).

[BKV06] M. van den Briel, S. Kambhampati, T. Vossen. "IPPLAN: Planning as Integer Programming". In: (2006). URL: https://ipc06.icaps-conference.org/deterministic/booklet/deterministic04.pdf (cit. on pp. 57, 59, 60).

[Bla98] B. Blai. "HSP: Heuristic Search Planner". In: (1998). URL: https://bonetblai.github.io/reports/aips98-competition.pdf (cit. on pp. 56, 59, 60).

[BSSV08] J. Bibaı, P. Saveant, M. Schoenauer, V. Vidal. "DAE: Planning as Artificial Evolution (Deterministic part)". en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/DAE.pdf (cit. on pp. 57, 59–63).

[CC18] A. Coles, A. Coles. *OPTIC*. visited on 21.03.2024. 2018. URL: https://bitbucket.org/ipc2018-temporal/team5/src/master/ (cit. on pp. 57, 61, 63).

[CCF+06] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, N. Policella. “From demo to practice the MEXAR path to space operations”. English. In: *Advances in Applied Articial Intelligence, Proceedings*. Ed. by M. Ali, R. Dapoigny. Vol. 4031. Berlin: Springer-Verlag Berlin, 2006. ISBN: 978-3-540-35453-6 (cit. on p. 13).

[CCK23] P. Chatterjee, A. Chapagain, R. Khardon. “Planning with DiSProD for the IPC 2023”. en. In: (2023) (cit. on pp. 58, 61).

[Cen14] I. Cenamor. “LIBACOP and LIBACOP2 Planner”. en. In: (2014). URL: https://www.cs.colostate.edu/~ipc2014/ipcl2014description-libacop.pdf (cit. on pp. 57, 58, 61, 63).

[CFH+23a] A. B. Correa, G. Frances, M. Hecher, D. M. Longo, J. Seipp. “Levitron: Combining Ground and Lifted Planning”. en. In: (2023) (cit. on pp. 58, 60, 62–64).

[CFH+23b] A. B. Correa, G. Frances, M. Hecher, D. M. Longo, J. Seipp. “Scorpion Maidu: Width Search in the Scorpion Planning System”. en. In: (2023) (cit. on pp. 58, 60, 62–64).

[CFH+23c] A. B. Correa, G. Frances, M. Hecher, D. M. Longo, J. Seipp. “The Powerlifted Planning System in the IPC 2023”. en. In: (2023) (cit. on pp. 58, 60, 62–64).

[CK18] H. Cui, R. Khardon. “The SOGBOFA system in IPC 2018: Lifted BP for Conformant Approximation of Stochastic Planning”. en. In: (2018). URL: https://ipc2018-probabilistic.bitbucket.io/planner-abstracts/conformant-sogbofa-ipc18.pdf (cit. on pp. 57, 58, 61).

[CLH08] Y. Chen, Q. Lv, R. Huang. “Plan-A: A Cost Optimal Planner Based on SAT-Constrained Optimization”. en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/Plan-A.pdf (cit. on pp. 57, 59, 60).

[CS08] A. Coles, A. Smith. “Upwards: The Role of Analysis in Cost-Optimal SAS+ Planning”. en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/upwards.pdf (cit. on pp. 57, 59, 60).

[CV14] L. Chrpa, M. Vallati. “AGAP: As Good As Possible”. en. In: (2014) (cit. on pp. 57, 58, 60, 61).

[CVC18] I. Cenamor, M. Vallati, L. Chrpa. “TemPoRal: Temporal Portfolio Algorithm”. en. In: (2018). URL: https://icenamor.github.io/files/TemPoRal.pdf (cit. on pp. 57, 58, 61–65).

[DGH+23] D. Drexler, D. Gnad, P. Höft, J. Seipp, D. Speck, S. Stahlberg. “Ragnarok”. In: (2023). URL: https://ipc2023-classical.github.io/abstracts/planner17_ragnarok.pdf (cit. on pp. 58, 60, 62, 64).

[DPC23] D. Doebber, A. G. Pereira, A. B. Correa. “OpCount4Sat: Operator Counting Heuristics for Satisficing Planning”. en. In: (2023) (cit. on pp. 58, 60, 62–64).

[Dra00] B. Drabble. “Task decomposition support to reactive scheduling”. English. In: *Recent Advances in Ai Planning*. Ed. by S. Biundo, M. Fox. Vol. 1809. Berlin: Springer-Verlag Berlin, 2000. ISBN: 978-3-540-67866-3 (cit. on p. 13).

[Dre23] D. Drexler. “Vanir: Learning and Executing Width-based Hierarchical Policies”. en. In: (2023) (cit. on pp. 58, 61–64).

[DSS23] D. Drexler, J. Seipp, D. Speck. “Odin: A Planner Based on Saturated Transition Cost Partitioning”. en. In: (2023) (cit. on pp. 58, 60, 62–64).

[DTR19] A. K. Dwivedi, A. Tirkey, S. K. Rath. “Applying learning-based methods for recognizing design patterns”. In: (2019). DOI: https://doi.org/10.1007/s11334-019-00329-3 (cit. on p. 22).

[EH01] S. Edelkamp, M. Helmert. “MIPS: The model-checking integrated planning system”. In: *AI magazine* 22.3 (2001), pp. 67–67 (cit. on pp. 56, 59, 60).

[EK08] S. Edelkamp, P. Kissmann. “GAMER: Bridging Planning and General Game Playing with Symbolic Search”. en. In: (2008) (cit. on p. 25).

[Eri95] J. M. V. Erich Gamma Richard Helm. Ralph E. Johnson. *Design Patterns: Abstraction and Reuse of Object-Oriented Design*. Publication Title: ECOOP’93 - Object-Oriented Programming, 7th European Conference, Kaiserslautern, Germany, July 26-30, 1993, Proceedings. 1995. DOI: 10.1007/3-540-47910-4_21. URL: https://doi.org/10.1007/3-540-47910-4%5C_21 (cit. on pp. 17, 18, 23, 29, 34, 55, 56).

[FBB+18] R. Fuentetaja, M. Barley, D. Borrajo, J. Douglas, S. Franco, P. Riddle. “The Meta-Search Planner (MSP) at IPC 2018”. en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team5.pdf (cit. on pp. 57, 58, 60, 62–64).

[FBUL15] C. Fehling, J. Barzen, Uwe Breitenbücher, F. Leymann. “A Process for Pattern Identification, Authoring, and Application”. In: *Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP)*. ACM, 2015. URL: http://www2.informatik.uni-stuttgart.de/cgi-%20%09bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-50&engl=0 (cit. on pp. 13, 22, 34, 35, 37).

[FC14] R. Fuentetaja, L. Chrpa. “The Rollent planning and learning system at the IPC-8 learning track”. en. In: (2014). URL: https://www.cs.colostate.edu/~ipc2014/ipcl2014description-rollent.pdf (cit. on pp. 57, 58, 61).

[Feh15] C. Fehling. *Cloud computing patterns : identification, design, and application*. 2015. DOI: 10.18419/OPUS-3596. URL: http://elib.uni-stuttgart.de/handle/11682/3613 (cit. on pp. 20, 35, 37).

[FEM23] S. Franco, S. Edelkamp, I. Moraru. “ComplementaryPDB Planner”. In: (2023). URL: https://ipc2023-classical.github.io/abstracts/planner7_ComplementaryPDB.pdf (cit. on pp. 58, 60, 62).

[FFP23] O. Firsov, H. Fiorino, D. Pellier. “OptiPlan-a CSP-based partial order HTN planner”. In: *International Planning Competition in the context of The International Conference on Automated Planning and Scheduling (ICAPS 2023)*. 2023. URL: https://ipc2023-htn.github.io/proceedings/Firsov-2023-IPC-OptiPlan.pdf (cit. on pp. 58, 60).

[FG23] M. Fickert, D. Gnad. “DiSCO - Decoupled Search + COnjunctions”. en. In: (2023) (cit. on pp. 58, 60, 62–64).

[FGLR18] G. Frances, H. Geffner, N. Lipovetzky, M. Ramirez. “Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants”. en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/teams_1_20_30_31_36_47.pdf (cit. on pp. 57, 58, 60, 62, 63).

[FGSH18] M. Fickert, D. Gnad, P. Speicher, J. Hoffmann. "SaarPlan: Combining Saarland's Greatest Planning Techniques". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team7.pdf (cit. on pp. 57, 58, 60, 62–64).

[FH04] Z. Feng, E. A. Hansen. "Symbolic Heuristic Search for Probabilistic Planning". en. In: (2004) (cit. on pp. 56, 59, 61).

[FH18] M. Fickert, J. Hoffmann. "OLCFF: Online-Learning hCFF". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team8.pdf (cit. on pp. 57, 58, 60, 62–64).

[FIT18a] A. Fern, M. Issakkimuthu, P. Tadepalli. "Imitation-Net: A Supervised Learning Planner". en. In: (2018). URL: https://ipc2018-probabilistic.bitbucket.io/planner-abstracts/imitation-net.pdf (cit. on pp. 57, 58, 61, 64).

[FIT18b] A. Fern, M. Issakkimuthu, P. Tadepalli. "Random-Bandit: An Online Planner". en. In: (2018). URL: https://ipc2018-probabilistic.bitbucket.io/planner-abstracts/random-bandit.pdf (cit. on pp. 57, 58, 61, 64).

[FJ18] D. Furelos-Blanco, A. Jonsson. "CP4TP: A Classical Planning for Temporal Planning Portfolio". en. In: (2018). URL: https://planning.wiki/_citedpapers/planners/cp4tp.pdf (cit. on pp. 57, 58, 61–64).

[FK18] D. Fiser, A. Komenda. "MAPlan: Reductions with Fact-Alternating Mutex Groups and h^m Heuristics". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/teams_13_17.pdf (cit. on pp. 57, 58, 60).

[FKIT18] A. Fern, A. Koul, M. Issakkimuthu, P. Tadepalli. "A2C-Plan: A Reinforcement Learning Planner". en. In: (2018). URL: https://ipc2018-probabilistic.bitbucket.io/planner-abstracts/a2c-plan.pdf (cit. on pp. 57, 58, 61).

[FKS+23] P. Ferber, M. Katz, J. Seipp, S. Sievers, D. Borrajo, I. Cenamor. "Hapori Explainable Decision Tree". en. In: (2023). URL: https://ipc2023-classical.github.io/abstracts/planner19_hapori_delfi.pdf (cit. on pp. 58, 60, 62, 63).

[FLB+18] S. Franco, L. H. S. Lelis, M. Barley, S. Edelkamp, M. Martinez, I. Moraru. "The Complementary1 Planner in the IPC 2018". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team9.pdf (cit. on pp. 57, 58, 60, 62–64).

[FLHC14] M. Fox, D. Long, R. Howey, S. Cresswell. *VAL*. 2014. URL: https://github.com/KCL-Planning/VAL/ (cit. on pp. 27, 57, 59, 63, 64).

[Fou00] M. Fourman. "Propositional Planning". In: 2000. URL: https://www.inf.ed.ac.uk/publications/report/0034.html (cit. on pp. 56, 59, 60).

[FRF+02] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002 (cit. on pp. 29, 34, 55).

[FVC20] J. Fernandez-Olivares, I. Vellido, L. Castillo. "Addressing HTN Planning with Blind Depth First Search". en. In: (2020) (cit. on pp. 57, 58, 60).

[GA15] I. Georgievski, M. Aiello. "HTN planning: Overview, comparison, and beyond". en. In: *Artificial Intelligence* 222 (May 2015), pp. 124–156. ISSN: 0004-3702. DOI: 10.1016/j.artint.2015.02.002. URL: http://www.sciencedirect.com/science/article/pii/S0004370215000247 (visited on 11/03/2020) (cit. on p. 16).

[GB21] I. Georgievski, U. Breitenbücher. “A Vision for Composing, Integrating, and Deploying AI Planning Functionalities”. In: *2021 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. 2021, pp. 166–171. DOI: 10.1109/SOSE52839.2021.00025 (cit. on p. 13).

[Geo23a] I. Georgievski. “Conceptualising Software Development Lifecycle for Engineering AI Planning Systems”. In: (2023). DOI: 10.1109/CAIN58948.2023.00019 (cit. on p. 16).

[Geo23b] I. Georgievski. “PlanX: A Toolbox for Building and Integrating AI Planning Systems”. In: *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)* (2023), pp. 130–134. DOI: 10.1109/SOSE58276.2023.00022 (cit. on p. 19).

[Geo23c] I. Georgievski. “Software Development Lifecycle for Engineering AI Planning Systems”. In: *International Conference on Software Technologies*. 2023, pp. 751–760. DOI: 10.5220/0012149100003538 (cit. on p. 16).

[Geo23d] I. Georgievski. “Towards Engineering AI Planning Functionalities as Services”. In: *Service-Oriented Computing – ICSOC 2022 Workshops* (2023). ISBN: 978-3-031-26507-5. DOI: 10.1007/978-3-031-26507-5_18 (cit. on pp. 13, 19).

[GFB08] R. Garcıa-Duran, F. Fernandez, D. Borrajo. “REPLICA: Relational Policies Learning in Planning”. en. In: (2008). URL: https://ipc08.icaps-conference.org/learning/documents/abstracts/abstract_replica.pdf (cit. on pp. 57, 59, 61).

[GGSV08] B. Galvani, A. E. Gerevini, A. Saetti, M. Vallati. “A Planner Based on an Automatically Configurable Portfolio of Domain-independent Planners with Macro-actions: PbP”. In: (2008). URL: https://ipc08.icaps-conference.org/learning/documents/abstracts/abstract_pbp.pdf (cit. on pp. 57, 59, 61, 63).

[GL08] M. Galea, J. Levine. “L2Plan2: Learning Generalised Policies via Evolutionary Computation”. en. In: (2008). URL: https://ipc08.icaps-conference.org/learning/documents/abstracts/abstract_l2plan2.pdf (cit. on pp. 57, 59, 61, 63).

[GL23] P. R. Gzubicki, B. P. Lachowicz. “HUZAR: Predicting Useful Actions with Graph Neural Networks”. en. In: (2023) (cit. on pp. 58, 61–64).

[GNN+17] I. Georgievski, T. A. Nguyen, F. Nizamic, B. Setz, A. Lazovik, M. Aiello. “Planning meets activity recognition: Service coordination for intelligent buildings”. In: *Pervasive and Mobile Computing* 38.1 (2017), pp. 110–139 (cit. on p. 13).

[GNT04] M. Ghallab, D. Nau, P. Traverso. *Automated Planning: Theory and Practice*. en. Elsevier, May 2004. ISBN: 978-0-08-049051-9 (cit. on p. 15).

[GP06] S. Grandcolas, C. Pain-Barre. “FDP: Filtering and Decomposition for Planning”. In: (2006). URL: https://ipc06.icaps-conference.org/deterministic/booklet/deterministic07.pdf (cit. on pp. 57, 59, 60).

[GP08] S. Grandcolas, C. Pain-Barre. “CFDP: an approach to Cost-Optimal Planning based on FDP”. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/C-fdp.pdf (cit. on pp. 57, 59, 60).

[GPT04] C. Gretton, D. Price, S. Thiebaux. “NMRDPP: Decision-Theoretic Planning with Control Knowledge.” en. In: (2004) (cit. on pp. 56, 59, 61).

[GRM+] A. Green, B. J. Reji, C. Muise, E. Scala, F. Meneguzzi, F. M. Rico, H. Stairs, J. Dolejsi, M. Magnaguagno, J. Mounty. URL: https://planning.wiki/ref/planners/atoz (cit. on p. 21).

[GS02] A. Gerevini, I. Serina. "LPG: a planner based on local search for planning graphs with action costs". In: *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*. AIPS'02. Place: Toulouse, France. AAAI Press, 2002, pp. 13–22. ISBN: 1-57735-142-8 (cit. on pp. 56, 59).

[GS18] F. Geißer, D. Speck. "PROST-DD - Utilizing Symbolic Classical Planning in THTS". en. In: (2018). URL: https://ipc2018-probabilistic.bitbucket.io/planner-abstracts/prost-dd.pdf (cit. on pp. 57, 58, 61, 64).

[GS20] A. E. Gerevini, E. Scala. *An Introduction to Numeric Planning*. en. 2020. URL: https://icaps20subpages.icaps-conference.org/wp-content/uploads/2020/11/icaps_2020_lecture_8.pdf (cit. on p. 16).

[GSH18] D. Gnad, A. Shleyfman, J. Hoffmann. "DecStar - STAR-topology DECoupled Search at its best". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team2.pdf (cit. on pp. 57, 58, 60).

[Has08] P. Haslum. "Additive and Reversed Relaxed Reachability Heuristics Revisited". In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/hsps.pdf (cit. on pp. 57, 59, 60).

[HB06] J. Hoffmann, R. I. Brafman. "Conformant planning via heuristic forward search: A new approach". In: 2006. DOI: 10.1016/j.artint.2006.01.003 (cit. on pp. 57, 59, 60).

[HCZ08] R. Huang, Y. Chen, W. Zhang. "DTG-Plan:Fast Planning by Search in Domain Transition Graphs". en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/DTG-Plan.pdf (cit. on pp. 57, 59, 60, 62).

[Hel06] M. Helmert. "Fast (Diagonally) Downward". In: (2006). URL: https://ipc06.icaps-conference.org/deterministic/booklet/deterministic08.pdf (cit. on pp. 57, 59, 60).

[HHB23a] L. Heiland, M. Hauser, J. Bogner. *Design Patterns for AI-Based Systems*. 2023. URL: https://swe4ai.github.io/ai-patterns/ (cit. on pp. 20, 23, 34).

[HHB23b] L. Heiland, M. Hauser, J. Bogner. "Design Patterns for AI-based Systems: A Multivocal Literature Review and Pattern Repository". In: *arXiv preprint arXiv:2303.13173* (2023) (cit. on p. 20).

[Hof01] J. Hoffmann. "FF: The fast-forward planning system". In: *AI magazine* 22.3 (2001), pp. 57–57 (cit. on pp. 56, 59, 60).

[Hol23a] D. Holler. "The PANDA Progression System for HTN Planning in the 2023 IPC". en. In: (2023) (cit. on pp. 58, 60).

[Hol23b] D. Holler. "The TOAD System for Totally Ordered HTN Planning in the 2023 IPC". en. In: (2023) (cit. on pp. 58, 60–64).

[HS] S. Hölldobler, H.-P. Störr. "Solving the Entailment Problem in the Fluent Calculus using Binary Decision Diagrams". In: (). URL: https://www.stoerr.net/pub/hoelldobler.stoerr.00a.pdf (cit. on pp. 56, 59, 60).

[HSHB11] J. Hoey, R. St-Aubin, A. Hu, C. Boutilier. "SPUDD: Stochastic Planning using Decision Diagrams". en. In: (2011) (cit. on pp. 57, 59, 61).

[HSS23] P. Hoeft, D. Speck, J. Seipp. "Dofri: Planner Abstract". en. In: (2023) (cit. on pp. 58, 60, 62–64).

[HTWT23] M. Hao, S. Toyer, R. Wang, F. Trevizan. "Action Schema Networks – IPC Version". en. In: (2023) (cit. on pp. 58, 61).

[Jon15] A. Jonsson. *Universal PDDL Parser*. 2015. URL: https://github.com/aig-upf/universal-pddl-parser (cit. on pp. 27, 33, 57, 59, 63).

[Kat18] M. Katz. "Cerberus: Red-Black Heuristic for Planning Tasks with Conditional Effects Meets Novelty Heuristic and Enchanced Mutex Detection". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/teams_15_16.pdf (cit. on pp. 57, 58, 60, 62–64).

[KBK08] J.-P. Kelly, A. Botea, S. Koenig. "Offline planning with hierarchical task networks in video games". In: 2008 (cit. on p. 13).

[KD00] J. Kvarnström, P. Doherty. "TALplanner: A temporal logic based forward chaining planner". en. In: *Annals of Mathematics and Artificial Intelligence* 30.1 (June 2000), pp. 119–169. ISSN: 1573-7470. DOI: 10.1023/A:1016619613658. URL: https://doi.org/10.1023/A:1016619613658 (visited on 11/03/2020) (cit. on pp. 27, 56, 59, 61).

[KDMW12] A. Kolobov, P. Dai, M. Mausam, D. Weld. "Reverse Iterative Deepening for Finite-Horizon MDPs with Large Branching Factors". en. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 22 (May 2012), pp. 146–154. ISSN: 2334-0843, 2334-0835. DOI: 10.1609/icaps.v22i1.13523. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13523 (visited on 02/03/2024) (cit. on pp. 57, 59, 61).

[KE08] P. Kissmann, S. Edelkamp. "GAMER: Fully-Observable Non-Deterministic Planning via PDDL-Translation into a Game". en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/gamer.pdf (cit. on pp. 57, 59, 60).

[KE12] T. Keller, P. Eyerich. "PROST: Probabilistic Planning Based on UCT". en. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 22 (May 2012), pp. 119–127. ISSN: 2334-0843, 2334-0835. DOI: 10.1609/icaps.v22i1.13518. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13518 (visited on 02/03/2024) (cit. on pp. 57, 59, 61, 64).

[KG08a] R. Kalyanam, R. Givan. "LDFS with Deterministic Plan Based Subgoals". en. In: (2008). URL: https://ipc08.icaps-conference.org/probabilistic/wiki/images/3/37/Team4-LPPFF.pdf (cit. on pp. 57, 59, 61).

[KG08b] E. Keyder, H. Geffner. "The FF(ha) Planner for Planning with Action Costs". en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/ffha.pdf (cit. on pp. 57, 59, 60).

[KG08c] E. Keyder, H. Geffner. "The HMDP Planner for Planning with Probabilities". en. In: (2008). URL: https://ipc08.icaps-conference.org/probabilistic/wiki/images/a/ad/Team9-HMDPP.pdf (cit. on pp. 57, 59, 61).

[KLMT18] M. Katz, N. Lipovetzky, D. Moshkovich, A. Tuisov. "MERWIN Planner: Mercury Enchanced With Novelty Heuristic". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team14.pdf (cit. on pp. 57, 58, 60, 62–64).

[KM20] E. Karpas, D. Magazzeni. "Automated planning for robotics". In: *Annual Review of Control, Robotics, and Autonomous Systems* (2020) (cit. on p. 13).

[Koe99] J. Koehler. *Handling of Conditional Effects and Negative Goals in IPP*. Tech. rep. 1999 (cit. on pp. 56, 59, 60).

[Kot16] L. Kotthoff. *Algorithm Selection for Combinatorial Search Problems: A Survey*. 2016. DOI: 10.1007/978-3-319-50137-6_7. URL: https://doi.org/10.1007/978-3-319-50137-6_7 (cit. on p. 16).

[Kre23] R. Kreft. "Saturated Cost Partitioning for Diverse Sets of Abstractions". en. In: (2023) (cit. on pp. 58, 60, 62–64).

[KS00] H. Kautz, B. Selman. "BLACKBOX: A New ApproachPtroobtlheemASpopllvicinagtion of Theorem Proving to". en. In: (2000) (cit. on pp. 56, 59, 60).

[KS04] E. Karabaev, O. Skvortsova. "FCPlanner: A Planning Strategy for First-Order MDPs". en. In: (2004). URL: https://ipc04.icaps-conference.org/probabilistic/proceedings/skvortsova.pdf (cit. on pp. 56, 59, 61).

[KSB23] R. Kuroiwa, A. Shleyfman, J. C. Beck. "NLM-CutPlan". In: (2023). URL: https://ipc2023-numeric.github.io/abstracts/NLM_CutPlan_Abstract.pdf (cit. on pp. 58, 60–64).

[KSSS18] M. Katz, S. Sohrabi, H. Samulowitz, S. Sievers. "Delfi: Online Planner Selection for Cost-Optimal Planning". en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/teams_23_24.pdf (cit. on pp. 57, 58, 60, 62–64).

[KT23] M. Katz, A. Tuisov. "TFTM-ArgMax Planner: Pruning Preferred Operators with Novelty". en. In: (2023) (cit. on pp. 58, 60, 62, 64).

[LA20] C. Lesire, A. Albore. "PYHIPOP– Hierarchical Partial-Order Planner". en. In: (2020) (cit. on pp. 57, 58, 60, 61, 64).

[LBP23] G. P. Lacroix, R. V. Bettker, A. G. Pereira. "FSM — A Short-Time Learning Planner". en. In: (2023) (cit. on pp. 58, 60, 62–64).

[Leo23] F. Leofante. "OMTPlan: A Tool for Optimal Planning Modulo Theories". en. In: *Journal on Satisfiability, Boolean Modeling and Computation* 14.1 (June 2023), pp. 17–23. ISSN: 15740617. DOI: 10.3233/SAT-220001. URL: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SAT-220001 (visited on 02/03/2024) (cit. on pp. 58, 60–62, 64).

[LF99] D. Long, M. Fox. "Efficient Implementation of the Plan Graph in STAN". en. In: *Journal of Artificial Intelligence Research* 10 (Feb. 1999), pp. 87–115. ISSN: 1076-9757. DOI: 10.1613/jair.570. URL: https://www.jair.org/index.php/jair/article/view/10221 (visited on 02/03/2024) (cit. on pp. 56, 59, 60).

[LLE23] C. Lei, N. Lipovetzky, K. A. Ehinger. "IPC Learning Track: Novelty-Based Generalized Planning". en. In: (2023) (cit. on pp. 58, 61).

[LRG08] N. Lipovetzky, M. Ramırez, H. Geffner. “C3: Planning with Consistent Causal Chains”. en. In: (2008) (cit. on pp. 57, 58, 60).

[Mar08] F. Maris. “TLP-GP: a Planner to Solve Temporally-Expressive Problems”. en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/TLP-GP.pdf (cit. on pp. 57, 59, 61).

[McD00] D. M. McDermott. “The 1998 AI planning systems competition”. In: *AI magazine* 21.2 (2000), pp. 35–35 (cit. on pp. 16, 33).

[MEMF18] I. Moraru, S. Edelkamp, M. Martinez, S. Franco. “Planning-PDBs Planner”. In: 2018. URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team40.pdf (cit. on pp. 57, 58, 60).

[MF01] Y. Meiller, P. Fabiani. “TOKENPLAN: A Planner for Both Satisfaction and Optimization Problems”. In: (2001). URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1578/1477 (cit. on pp. 56, 59, 60).

[MMS22] M. C. Magnaguagno, F. Meneguzzi, L. de Silva. *HyperTensioN and Total-order Forward Decomposition optimizations*. en. July 2022. URL: http://arxiv.org/abs/2207.00345 (visited on 02/03/2024) (cit. on pp. 57, 58, 60).

[MN08] Murugeswari, Narayanaswamy. “Macro-AltAlt: Improving heuristic guided search using frequency based Action Macros”. In: (2008). URL: https://ipc08.icaps-conference.org/learning/documents/abstracts/abstract_macroaltalt.pdf (cit. on pp. 57, 59, 61).

[MR15] G. Markou, I. Refanidis. “Non-deterministic planning methods for automated web service composition”. en. In: *Artificial Intelligence Research* 5.1 (Sept. 2015), p14. ISSN: 1927-6982, 1927-6974. DOI: 10.5430/air.v5n1p14. URL: http://www.sciedupress.com/journal/index.php/air/article/view/7152 (visited on 02/07/2024) (cit. on p. 16).

[Mui15] C. Muise. *Planning.Domains*. 2015. URL: http://planning.domains/ (cit. on pp. 27, 33, 57, 59).

[NCLM01] D. Nau, Y. Cao, A. Lotem, H. Munoz-Avila. “The SHOP planning system”. In: *AI Magazine* 22.3 (2001), pp. 91–91 (cit. on pp. 56, 59, 60).

[NLFL08] M. A. H. Newton, J. Levine, M. Fox, D. Long. “Wizard: Compiled Macro-Actions for Planner-Domain Pairs”. en. In: (2008). URL: https://ipc08.icaps-conference.org/learning/documents/abstracts/abstract_wizard.pdf (cit. on pp. 57, 59, 61).

[OHB23] C. Olz, D. Holler, P. Bercher. “The PANDADealer System for Totally Ordered HTN Planning in the 2023 IPC”. en. In: (2023) (cit. on pp. 58, 60).

[Ols11] A. Olsen. “Pond-Hindsight: Applying Hindsight Optimization to Partially-Observable Markov Decision Processes”. en. In: (2011) (cit. on pp. 57, 59, 61).

[OSSM01] E. Onaindia, O. Sapena, L. Sebastia, E. Marzal. “SimPlanner: An Execution-Monitoring System for Replanning in Dynamic Worlds”. In: *Progress in Artificial Intelligence*. Ed. by P. Brazdil, A. Jorge. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 393–400. ISBN: 978-3-540-45329-1 (cit. on pp. 56, 59).

[OWL04] N. Onder, G. C. Whelan, L. Li. “Probapop: Probabilistic Partial-Order Planning”. en. In: (2004) (cit. on pp. 56, 59, 61).

[PCBB12] T. Plch, M. Chomut, C. Brom, R. Bart. “Inspect, Edit and Debug PDDL Documents: Simply and Efficiently with PDDL Studio”. en. In: (2012). URL: https://icaps12.icaps-conference.org/demo/Plch_et_al.pdf (cit. on pp. 27, 33, 57, 59).

[PF18] D. Pellier, H. Fiorino. “PDDL4J: a planning domain description library for java”. In: *Journal of Experimental & Theoretical Artificial Intelligence* 30.1 (2018). Publisher: Taylor & Francis, pp. 143–176. DOI: 10.1080/0952813X.2017.1409278. URL: https://doi.org/10.1080/0952813X.2017.1409278 (cit. on pp. 27, 33, 57, 59).

[PG08] H. Palacios, H. Geffner. “Conformant Planning through Classical Planning”. en. In: (2008). URL: https://ipc08.icaps-conference.org/probabilistic/wiki/images/7/7a/Team10-T0.pdf (cit. on pp. 57, 59, 60).

[Pou11] P. Poupart. *Symbolic Perseus*. visited on 21.03.2024. 2011. URL: https://cs.uwaterloo.ca/~ppoupart/software.html#symbolic-perseus (cit. on pp. 57, 59, 61, 63).

[QPF23] G. Quenard, D. Pellier, H. Fiorino. “LTP: Lifted Tree Path”. In: (2023). URL: https://ipc2023-htn.github.io/proceedings/Quenard-2023-IPC-LiftedTreePath.pdf (cit. on pp. 58, 60).

[RBF+16] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, A. Riegg. “Internet of Things Patterns”. In: *Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPLoP)*. ACM, 2016. DOI: 10.1145/3011784.3011789 (cit. on p. 20).

[RBF+19] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, A. Riegg. “Internet of things patterns for communication and management”. In: (2019). DOI: 10.1007/978-3-030-14291-9_5 (cit. on p. 20).

[RE 71] N. N. R.E. Fikes. “Strips: A new approach to the application of theorem proving to problem solving”. In: (1971). DOI: https://doi.org/10.1016/0004-3702(71)90010-5. URL: https://www.sciencedirect.com/science/article/pii/0004370271900105 (cit. on p. 13).

[RE08] G. Roger, P. Eyerich. “TFD: A Numeric Temporal Extension to Fast Downward”. en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/TFD.pdf (cit. on pp. 57, 59, 61–64).

[RF14] T. de la Rosa, R. Fuentetaja. “Ensemble-Roller: Planning with Ensemble of Relational Decision Trees”. In: (2014). URL: https://www.cs.colostate.edu/~ipc2014/ipcl2014description-ensemble-roller.pdf (cit. on pp. 57, 58, 61).

[RGP08] N. Robinson, C. Gretton, D.-N. Pham. “CO-PLAN: Combining SAT-Based Planning with Forward-Search”. en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/co-plan.pdf (cit. on pp. 57, 59, 60).

[Rin07] J. Rintanen. “Complexity of Concurrent Temporal Planning”. en. In: *ICAPS* (2007) (cit. on p. 16).

[RJ08] T. de la Rosa, S. Jimenez. “ROLLER: A Lookahead Planner Guided by Relational Decision Trees”. In: (2008). URL: https://ipc08.icaps-conference.org/learning/documents/abstracts/abstract_roller.pdf (cit. on pp. 57, 59, 61).

[RJFT11] A. N. Raghavan, S. Joshi, A. Fern, P. Tadepalli. “Bidirectional Online Probabilistic Planning”. en. In: (2011) (cit. on pp. 57, 59, 61).

[ROB08] T. de la Rosa, A. G. Olaya, D. Borrajo. “CABALA: Case-based State Lookaheads”. In: (2008). URL: https://ipc08.icaps-conference.org/learning/documents/abstracts/abstract_cabala.pdf (cit. on pp. 33, 57, 59, 61).

[RV00] I. Refanidis, I. Vlahavas. “GRT: A Domain Independent Heuristic for STRIPS Worlds Based on Greedy Regression Tables”. In: *Recent Advances in AI Planning*. Ed. by S. Biundo, M. Fox. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 347–359. ISBN: 978-3-540-44657-6 (cit. on pp. 56, 59, 60).

[RW08] S. Richter, M. Westphal. “The LAMA Planner Using Landmark Counting in Heuristic Search”. en. In: (2008). URL: https://ipc08.icaps-conference.org/deterministic/data/planners/LAMA.pdf (cit. on pp. 57, 59, 60, 62–64).

[SBG23] S. Stahlberg, B. Bonet, H. Geffner. “Muninn”. In: (2023). URL: https://ipc2023-learning.github.io/abstracts/muninn.pdf (cit. on pp. 58, 61, 64).

[Sch20] D. Schreiber. “Lifted Logic for Task Networks: TOHTN Planner Lilotane in the IPC 2020”. en. In: (2020) (cit. on pp. 57, 58, 60).

[Sei18a] J. Seipp. “Fast Downward Remix”. en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team43.pdf (cit. on pp. 57, 58, 60, 62).

[Sei18b] J. Seipp. “Scorpion”. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/team44.pdf (cit. on pp. 57, 58, 60, 62–64).

[SFS23] M. Salerno, R. Fuentetaja, J. Seipp. “Spock: Fast Downward Stone Soup with Redundant Action Elimination”. en. In: (2023) (cit. on pp. 35, 58, 60, 62–64).

[SGM18] D. Speck, F. Geisser, R. Mattmueller. “SYMPLE: Symbolic Planning based on EVMDDs{}”. en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/teams_3_10.pdf (cit. on pp. 57, 58, 60, 62–64).

[Sie18] S. Sievers. “Fast Downward Merge-and-Shrink”. en. In: (2018). URL: https://ipc2018-classical.bitbucket.io/planner-abstracts/teams_26_27.pdf (cit. on pp. 57, 58, 60, 62–64).

[SK20] V. Strobel, A. Kirsch. “MyPDDL: Tools for Efficiently Creating PDDL Domains and Problems”. In: *Knowledge Engineering Tools and Techniques for AI Planning*. Springer International Publishing, 2020, pp. 67–90. ISBN: 978-3-030-38561-3. DOI: 10.1007/978-3-030-38561-3_4. URL: http://dx.doi.org/10.1007/978-3-030-38561-3_4 (cit. on pp. 27, 57, 59).

[SKK+08] H. S. Sim, K.-E. Kim, J. H. Kim, D.-S. Chang, M.-W. Koo. “Symbolic Heuristic Search Value Iteration for Factored POMDPs”. In: 2008. URL: https://ailab.kaist.ac.kr/papers/pdfs/SKKCK2008.pdf (cit. on pp. 57, 59, 61).

[SLL+23] A. Singh, C. Lei, N. Lipovetzky, M. Ramirez, J. Segovia-Aguas. “Forward Backward Novelty Search”. en. In: (2023) (cit. on pp. 58, 60).

[SLR+23] A. Singh, N. Lipovetzky, M. Ramirez, J. Segovia-Aguas, G. Frances. “Grounding Schematic Representation with GRINGO for Width-based Search”. en. In: (2023) (cit. on pp. 58, 60).

[SLRS21] A. Singh, N. Lipovetzky, M. Ramirez, J. Segovia-Aguas. "Approximate Novelty Search". en. In: *Proceedings of the International Conference on Automated Planning and Scheduling* 31 (May 2021), pp. 349–357. ISSN: 2334-0843, 2334-0835. DOI: 10.1609/icaps.v31i1.15980. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/15980 (visited on 02/02/2024) (cit. on pp. 58, 60, 62, 63).

[SNK+01] B. Srivastava, X. Nguyen, S. Kambhampati, M. B. Do, U. Nambiar, Z. Nie, R. Nigenda, T. Zimmerman. "AltAlt: Combining Graphplan and Heuristic State Search". In: *AI Magazine* 22.3 (Sept. 2001), p. 88. DOI: 10.1609/aimag.v22i3.1579. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1579 (visited on 02/02/2024) (cit. on pp. 33, 56, 59, 60).

[SOM01] L. Sebastia, E. Onaindia, E. Marzal. "STeLLa: An Optimal Sequential and Parallel Planner". In: *Progress in Artificial Intelligence*. Ed. by P. Brazdil, A. Jorge. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 409–416. ISBN: 978-3-540-45329-1 (cit. on pp. 56, 59, 60).

[Spe23] D. Speck. "SymK - A Versatile Symbolic Search Planner". en. In: (2023) (cit. on pp. 58, 60, 62–64).

[SSH14a] J. Seipp, S. Sievers, F. Hutter. "Fast Downward Cedalion". en. In: (2014) (cit. on pp. 57, 58, 60).

[SSH14b] J. Seipp, S. Sievers, F. Hutter. "Fast Downward SMAC". en. In: (2014) (cit. on pp. 57, 58, 61–64).

[Sys] S. Systems. *Enterprise Architect*. URL: https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/reverseengineersourcecode.html (cit. on p. 23).

[TAB+21] M. Take, S. Alpers, C. Becker, C. Schreiber, A. Oberweis. "Software Design Patterns for AI-Systems". In: (2021). URL: https://ceur-ws.org/Vol-2867/paper5.pdf (cit. on p. 55).

[TF04] F. Teichteil-Konigsbuch, P. Fabiani. "Probabilistic Reachability Analysis for Structured Markov Decision Processes". en. In: (2004) (cit. on pp. 56, 59, 61).

[TG23] A. Torralba, D. Gnad. "Gofai". In: (2023). URL: https://ipc2023-learning.github.io/abstracts/gofai.pdf (cit. on pp. 58, 61–64).

[TIK08a] F. Teichteil-Konigsbuch, G. Infantes, U. Kuter. "FSP: Optimal Forward Stochastic Planning using relaxed PPDDL operators". en. In: (2008). URL: https://ipc08.icaps-conference.org/probabilistic/wiki/images/c/c2/Team1-FSP.pdf (cit. on pp. 57, 59, 61).

[TIK08b] F. Teichteil-Konigsbuch, G. Infantes, U. Kuter. "RFF: A Robust, FF-Based MDP Planning Algorithm for Generating Policies with Low Probability of Failure". en. In: (2008). URL: https://ipc08.icaps-conference.org/probabilistic/wiki/images/f/f6/Team1-RFF.pdf (cit. on pp. 57, 59, 61).

[TNPS08] D.-V. Tran, H.-K. Nguyen, E. Pontelli, T. C. Son. "CPA(C)/(H): Two Approximation-Based Conformant Planners". en. In: (2008). URL: https://ipc08.icaps-conference.org/probabilistic/wiki/images/5/57/Team2-CPA.pdf (cit. on pp. 57, 59–61, 63).

[Tor23a] Á. Torralba. "QDom-LMCut: Enhancing Search with Quantitative Dominance Pruning". In: (2023). URL: https://ipc2023-classical.github.io/abstracts/planner32_dom.pdf (cit. on pp. 58, 60, 62–64).

[Tor23b] Á. Torralba. "SymBD: A Symbolic Bidirectional Search Baseline". In: (2023). URL: https://ipc2023-classical.github.io/abstracts/planner14_SymBD_2023.pdf (cit. on pp. 58, 60, 62–64).

[TSTN23] Á. Torralba, S. Sievers, R. G. Tollund, K. Nielsen. "FTSPlan: Task Reformulation via Merge-and-Shrink". In: (2023). URL: https://ipc2023-classical.github.io/abstracts/planner13_fts.pdf (cit. on pp. 58, 60, 62–64).

[VBA14] J. Virseda, D. Borrajo, V. Alcázar. "LLAMA: Learning LAMA". In: (2014). URL: https://www.cs.colostate.edu/~ipc2014/ipcl2014description-llama.pdf (cit. on pp. 57, 58, 61–64).

[VM21] M. Vallati, T. L. McCluskey. "In Defence of Design Patterns for AI Planning Knowledge Models". In: *AIxIA 2020 – Advances in Artificial Intelligence*. Springer International Publishing, 2021, pp. 191–203. ISBN: 978-3-030-77091-4. DOI: 10.1007/978-3-030-77091-4_12 (cit. on p. 19).

[VR18] O. S. Vercher, E. O. de la Rivaherrera. *TFLAP*. visited on 21.03.2024. 2018. URL: https://bitbucket.org/ipc2018-temporal/team2/src/master/ (cit. on pp. 57, 61).

[VSTC13] T. S. Vaquero, J. R. Silva, F. Tonidandel, J. Christopher Beck. "itSIMPLE: towards an integrated design system for real planning applications". In: *The Knowledge Engineering Review* 28.2 (2013), pp. 215–230. DOI: 10.1017/S0269888912000434 (cit. on pp. 27, 33, 56, 59).

[WA20] F. Wedyan, S. Abufakher. "Impact of design patterns on software quality: a systematic literature review". In: (2020). DOI: https://doi.org/10.1049/iet-sen.2018.5446 (cit. on p. 17).

[WBLS14] M. Weigold, J. Barzen, F. Leymann, M. Salm. "Data Encoding Patterns for Quantum Computing". In: 2014 (cit. on p. 20).

[WKG08] J.-H. Wu, R. Kalyanam, R. Givan. "Planning using Stochastic Enforced Hill-Climbing". en. In: (2008). URL: https://ipc08.icaps-conference.org/probabilistic/wiki/images/9/94/Team6-SEH.pdf (cit. on pp. 57, 59, 61).

[WUKG19] H. Washizaki, H. Uchida, F. Khomh, Y.-G. Guéhéneuc. "Studying software engineering patterns for designing machine learning systems". In: 2019 (cit. on p. 20).

[XCZ06] Z. Xing, Y. Chen, W. Zhang. "MaxPlan: Optimal Planning by Decomposed Satisfiability and Backward Reduction". en. In: (2006) (cit. on pp. 57, 59, 60).

[YFG04] S. Yoon, A. Fern, R. Givan. "Learning Reactive Policies for Probabilistic Planning Domains". en. In: (2004) (cit. on pp. 56, 59, 61).

[YFG07] S. Yoon, A. Fern, R. Givan. "FF-Replan: a baseline for probabilistic planning". In: *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling*. ICAPS'07. Place: Providence, Rhode Island, USA. AAAI Press, 2007, pp. 352–359. ISBN: 978-1-57735-344-7 (cit. on pp. 56, 59, 61).

[YS03] H. L. S. Younes, R. G. Simmons. “VHPOP: Versatile Heuristic Partial Order Planner”. en. In: *Journal of Artificial Intelligence Research* 20 (Dec. 2003), pp. 405–430. ISSN: 1076-9757. DOI: 10.1613/jair.1136. URL: https://jair.org/index.php/jair/article/view/10363 (visited on 02/03/2024) (cit. on pp. 56, 59, 60).

All links were last followed on March 05, 2024.

A Appendix

In this chapter we document all the data we found presented in this thesis. We will also document the design patterns with their description and effects.

A.1 Patterns

In this section we will give descriptions and definitions of the found patterns.

Definition 4 (Strategy [Eri95] [TAB+21])
***Intent**: Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.*
***Consequences**: Switching models/heuristics or achieving flexibility in model behavior is easier, but code complexity is increased and overhead possible.*

Definition 5 (Proxy [Eri95])
***Intent**: Provide a surrogate or placeholder for another object to control access to it.*
***Consequences**: Adds indirection which can hide the fact that an object resides in a different address space or can perform optimizations. It also can reduce the cost of modifying an object and the cost of copying.*

Definition 6 (Factory [Eri95] [TAB+21])
***Intent**: Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.*
***Consequences**: Simplifies the creation of new objects. By ensuring equal representation of classes and features in the training data, it aims to achieve equal prediction accuracy across different categories, which helps with the fairness and equity of AI applications. Drawback is that clients may need to subclass the creator class to create specific objects, adding complexity to the code.*

Definition 7 (State [Eri95])
***Intent**: Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.*
***Consequences**: Overall, the State pattern enhances code structure, clarity, and maintainability by encapsulating state-specific behavior and making state transitions explicit.*

Definition 8 (Registry [FRF+02])
***Intent**: A creational design pattern that provides a centralized point of access to a set of objects. It essentially acts as a global repository or registry for objects of a particular type within an application.*
***Consequences**: It serves as a centralized mechanism for accessing and managing objects, allowing you to locate objects based on certain criteria. But it can lead to tight coupling between components and make the code less modular if overused.*

Definition 9 (Iterator [Eri95])
***Intent**: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.*
***Consequences**: An iterator keeps track of its own traversal state. Therefore you can have more than one traversal in progress at once.*

Definition 10 (Singleton [Eri95])
***Intent**: Ensure a class only has one instance, and provide a global point of access to it.*
***Consequences**: The access is controlled to sole instance, we have a reduced name space and it provides more flexibility than class operations.*

Definition 11 (Template Method [Eri95])
***Intent**: Define the skeleton of an algorithm in an operation, deferring some steps to sub-classes. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.*
***Consequences**: Crucial for reusing code, as they help extract common behavior from classes.*

Definition 12 (Command [Eri95])
***Intent**: Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.*
***Consequences**: The Command pattern simplifies code in several ways: it is reducing dependency, allows easy manipulation and extension and enhances flexibility,*

A.2 Result Tables

The following subsections will depict which AI Planning tool are responsible for the results we can see in chapter 5.

A.2.1 Release Years

Table A.1: The tools according to the release year

Begin of Table	
Year	**Planning tools**
1998	Blackbox [KS00], HSP[Bla98], STAN [LF99]
2000	AltAlt [SNK+01], BDDPlan [HS], FF [Hof01], GRT [RV00], IPP [Koe99], Mips [EH01], PbR [AK01], PropPlan [Fou00], SHOP [NCLM01], TALplanner [KD00], TokenPlan [MF01]
2002	LPG [GS02], Simplanner [OSSM01], Stella [SOM01], VHPOP [YS03]
2004	(BFHSP, Crikey, CPT, Fast Diagonally Downward, Fast Downward, HSP*a, LPG-TD, Macro-FF, Optiplan, SATPLAN, Semsyn, SGPlan, TP-4, YAHSP) [04], UMass [FH04], Purdue-Humans [YFG04], Probapop [OWL04], NMRDPP [GPT04], mGPT [BG05], FF-rePlan [YFG07], FCPlanner [KS04], Classy [YFG04], CERT [TF04], itSimple [VSTC13]

Continuation of Table A.1	
Year	**Planning tools**
2006	Conformant-FF [HB06], Fast Downward-sa [Hel06], FDP [GP06], HPlan-P [BHBM06], IPPLAN-1SC [BKV06], Maxplan [XCZ06], POND [BKS06], YochanPS [BKD06]
2008	Cabala [ROB08], C3 [LRG08], CFDP [GP08], co-plan [RGP08], CPAh [TNPS08], DAE-1 [BSSV08], DTG-Plan [HCZ08], $FF(h_a)$ [KG08b], FSP*-(RBH/RDH)[TIK08a], Gamer [KE08], HMDPP [KG08c], hsp_0^* [Has08], L2Plan2 [GL08], LAMA [RW08], LPPFF [KG08a], Macro-AltAlt [MN08], PbP [GGSV08], Plan-A [CLH08], Replica [GFB08], RFF-(BG/PG) [TIK08b], Roller [RJ08], Sayphi-Rules [BF08], SEH [WKG08], T0 [PG08], Temporal FD [RE08], TLP-GP [Mar08], Upwards [CS08], Wizard [NLFL08]
2011	(ACOPlan, Arvand, Arvand Herd, ayAlsoPlan, BJOLP, Bootstrap-Planner, BRT, CBL, CBP, DAE-YAHSP, Fast Downward Stone Soup, Fast-Downward-Autotune-quality, FD Autotune, Fork Init, Lamar, LM-Cut, LMTD, LPRPG-P, Madagascar, Merge and Shrink, OALDAEYASHP, Par-LPG, PHSFF, POPF-2, Probe, Randward, Roamer, SelMax, Sharaabi) [11], Symbolic Perseus [Pou11], Beaver [RJFT11], Glutton [KDMW12], KAIST AILAB [SKK+08], POND-Hindsight [Ols11], PROST [KE12], SPUDD [HSHB11]
2012	PDDL Studio [PCBB12]
2014	(AIIPACA, BFS(f), BiFD, DPMPlan, Fast Downward Uniform, Freelunch, hflow, IBaCoP, ITSAT, Jasper, Mercury, Metis, MIPlan, NuCeLaR, Planets, Rational Lazy A*, RIDA, RPT, SIW, SPM&S, SymBA*, tBURTON, USE) [14], myPDDL [SK20], VAL [FLHC14], AGAP [CV14], Ensemble-Roller [RF14], FD Cedalion [SSH14a], FD SMAC [SSH14b], LIBaCOP [Cen14], LLAMA [VBA14], RollEnt [FC14]
2015	PDDL4J [PF18], Planning.Domains [Mui15], Universal PDDL parser [Jon15]
2018	A2C-Plan [FKIT18], alien [Asa18], Cerberus [Kat18], Complementary-1 [FLB+18], Conformant-SOGBOFA-B-IPC18 [CK18], CP4TP [FJ18], DecStar [GSH18], Delfi-1 [KSSS18], FD Remix [Sei18a], FDMS [Sie18], freelunch(-double-relaed)/(-madagascar) [BG18], fs-blind [FGLR18], Imitation-Net [FIT18a], maplan-1|2 [FK18], MERWIN [KLMT18], MSP [FBB+18], OLCFF [FH18], OPTIC [CC18], Planning-PDBs [MEMF18], Prost-DD [GS18], Random-Bandit [FIT18b], saarplan [FGSH18], Scorpion [Sei18b], Symple [SGM18], TemPorAl [CVC18], TFLAP [VR18]
2020	HyperTensionN [MMS22], LiloTane [Sch20], pyHiPOP [LA20], SIADEX [FVC20]

2023	Approimate Novelty Anytime [SLRS21], Aries [Bit23], ASNets 2023 [HTWT23], CEGAR++ [Kre23], ComplementaryPDB [FEM23], DALAI 2023 [BCEK23], DiSCO [FG23], DiSProD [CCK23], Dofri [HSS23], Forward Backward Anytime Novelty Search [SLL+23], FSM [LBP23], FTSPlan [TSTN23], GOFAI [TG23], Grounding Schematic Representation with GRINGO for Width-based Search [SLR+23], Hapori [FKS+23], HUZAR [GL23], Levitron [CFH+23a], LNM-Plan [KSB23], LTP [QPF23], Muninn [SBG23], Novelty-based Progressive Generalized Planner [LLE23], Odin [DSS23], OMTPlan [Leo23], OpCount4Sat [DPC23], OptiPlan [FFP23], PandaDealer [OHB23], PANDApro [Hol23a], Powerlifted [CFH+23c], QDom-Lmcut [Tor23a], Ragnarok [DGH+23], Scorpion Maidu [CFH+23b], Spock [SFS23], SymBD [Tor23b], Symk [Spe23], TFTM1 [KT23], TOAD [Hol23b], Vanir [Dre23]
End of Table	

A.2.2 Tool types

Table A.2: The tools according to their type

Begin of Table	
Tool types	**Planning tools**
Planners	Approimate Novelty Anytime [SLRS21], Aries [Bit23], ASNets 2023 [HTWT23], C3 [LRG08], CEGAR++ [Kre23], ComplementaryPDB [FEM23], DALAI 2023 [BCEK23], DiSCO [FG23], DiSProD [CCK23], Dofri [HSS23], Forward Backward Anytime Novelty Search [SLL+23], FSM [LBP23], FTS-Plan [TSTN23], GOFAI [TG23], Grounding Schematic Representation with GRINGO for Width-based Search [SLR+23], Hapori [FKS+23], HUZAR [GL23], Levitron [CFH+23a], LNM-Plan [KSB23], LTP [QPF23], Muninn [SBG23], Novelty-based Progressive Generalized Planner [LLE23], Odin [DSS23], OMTPlan [Leo23], OpCount4Sat [DPC23], OptiPlan [FFP23], Pan-daDealer [OHB23], PANDApro [Hol23a], Powerlifted [CFH+23c], QDom-Lmcut [Tor23a], Ragnarok [DGH+23], Scorpion Maidu [CFH+23b], Spock [SFS23], SymBD [Tor23b], Symk [Spe23], TFTM1 [KT23], TOAD [Hol23b], Vanir [Dre23], HyperTensionN [MMS22], LiloTane [Sch20], pyHiPOP [LA20], SIADEX [FVC20], A2C-Plan [FKIT18], alien [Asa18], Cerberus [Kat18], Complementary-1 [FLB+18], Conformant-SOGBOFA-B-IPC18 [CK18], CP4TP [FJ18], DecStar [GSH18], Delfi-1 [KSSS18], FD Remix [Sei18a], FDMS [Sie18], freelunch(-double-relaed)/(-madagascar) [BG18], fs-blind [FGLR18], Imitation-Net [FIT18a], maplan-1|2 [FK18], MER-WIN [KLMT18], MSP [FBB+18], OLCFF [FH18], OPTIC, Planning-PDBs [MEMF18], Prost-DD [GS18], Random-Bandit [FIT18b], saarplan [FGSH18], Scorpion [Sei18b], Symple [SGM18], TemPorAl [CVC18], TFLAP, AGAP [CV14], Ensemble-Roller [RF14], FD Cedalion [SSH14a], FD SMAC [SSH14b], LIBaCOP [Cen14], LLAMA [VBA14], RollEnt [FC14]

Continuation of Table A.2	
Tool types	**Planning tools**
Planners	(AIIPACA, BFS(f), BiFD, DPMPlan, Fast Downward Uniform, Freelunch, hflow, IBaCoP, ITSAT, Jasper, Mercury, Metis, MIPlan, NuCeLaR, Planets, Rational Lazy A*, RIDA, RPT, SIW, SPM&S, SymBA*, tBURTON, USE) [14], (ACOPlan, Arvand, Arvand Herd, ayAlsoPlan, BJOLP, Bootstrap-Planner, BRT, CBL, CBP, DAE-YAHSP, Fast Downward Stone Soup, Fast-Downward-Autotune-quality, FD Autotune, Fork Init, Lamar, LM-Cut, LMTD, LPRPG-P, Madagascar, Merge and Shrink, OALDAEYASHP, Par-LPG, PHSFF, POPF-2, Probe, Randward, Roamer, SelMax, Sharaabi) [11], Symbolic Perseus [Pou11], Beaver [RJFT11], Glutton [KDMW12], KAIST AILAB [SKK+08], POND-Hindsight [Ols11], PROST [KE12], SPUDD [HSHB11], Cabala [ROB08], CFDP [GP08], co-plan [RGP08], CPAh [TNPS08], DAE-1 [BSSV08], DTG-Plan [HCZ08], $FF(h_a)$ [KG08b], FSP*-(RBH/RDH)[TIK08a], Gamer [KE08], HMDPP [KG08c], hsp_0^* [Has08], L2Plan2 [GL08], LAMA [RW08], LPPFF [KG08a], Macro-AltAlt [MN08], PbP [GGSV08], Plan-A [CLH08], Replica [GFB08], RFF-(BG/PG) [TIK08b], Roller [RJ08], Sayphi-Rules [BF08], SEH [WKG08], T0 [PG08], Temporal FD [RE08], TLP-GP [Mar08], Upwards [CS08], Wizard [NLFL08], Conformant-FF [HB06], Fast Downward-sa [Hel06], FDP [GP06], HPlan-P [BHBM06], IPPLAN-1SC [BKV06], Maxplan [XCZ06], POND [BKS06], YochanPS [BKD06], (BFHSP, Crikey, Fast Diagonally Downward, Fast Downward, HSP*a, LPG-TD, Macro-FF, Optiplan, SATPLAN, Semsyn, SGPlan, TP-4, YAHSP, CPT) [04], UMass [FH04], Purdue-Humans [YFG04], Probapop [OWL04], NMRDPP [GPT04], mGPT [BG05], FF-rePlan [YFG07], FCPlanner [KS04], Classy [YFG04], CERT [TF04], LPG [GS02], Simplanner [OSSM01], Stella [SOM01], VHPOP [YS03], AltAlt [SNK+01], BDDPlan [HS], FF [Hof01], GRT [RV00], IPP [Koe99], Mips [EH01], PbR [AK01], PropPlan [Fou00], SHOP [NCLM01], TALplanner [KD00], TokenPlan [MF01], Blackbox [KS00], HSP[Bla98], STAN [LF99]
Validator	VAL [FLHC14]
Editor	itSimple [VSTC13], Planning.Domains [Mui15], myPDDL [SK20], PDDL Studio [PCBB12]
Parser	PDDL4J [PF18], Universal PDDL parser [Jon15]
End of Table	

A.2.3 Planner types

Table A.3: Planners according to their type

Begin of Table	
Planning types	**Planner**
Classical	(ACOPlan, Arvand, Arvand Herd, ayAlsoPlan, BJOLP, BRT, CBP, DAE-YAHSP, Fast Downward Stone Soup, FD Autotune, Fork Init, Lamar, LM-Cut, LPRPG-P, Madagascar, Merge and Shrink, PHSFF, POPF-2, Randward, Roamer, Sharaabi, SelMax)[11], AGAP [CV14], (AIIPACA, BFS(f), BiFD, DPMPlan, Fast Downward Uniform, Freelunch, hflow, IBaCoP, Jasper, Mercury, Metis, NuCeLaR, Planets, Rational Lazy A*, RIDA, RPT, SIW, SPM&S, SymBA*, USE) [14], alien [Asa18], AltAlt [SNK+01], Approimate Novelty Anytime [SLRS21], BDDPlan [HS], (BFHSP, Fast Diagonally Downward, Fast Downward, LPG-TD, Macro-FF, SATPLAN, Semsyn, SGPlan, YAHSP) [04], Blackbox [KS00], C3 [LRG08], CEGAR++ [Kre23], Cerberus [Kat18], CFDP [GP08], co-plan [RGP08], Complementary-1 [FLB+18], ComplementaryPDB [FEM23], DAE-1 [BSSV08], DALAI 2023 [BCEK23], DecStar [GSH18], Delfi-1 [KSSS18], DiSCO [FG23], Dofri [HSS23], DTG-Plan [HCZ08], Fast Downward-sa [Hel06], FD Cedalion [SSH14a], FD Remix [Sei18a], FDMS [Sie18], FDP [GP06], FF [Hof01], $FF(h_a)$ [KG08b], Forward Backward Anytime Novelty Search [SLL+23], freelunch(-double-relaed)/(-madagascar) [BG18], fs-blind [FGLR18], FSM [LBP23], FTS-Plan [TSTN23], Gamer [KE08], Grounding Schematic Representation with GRINGO for Width-based Search [SLR+23], GRT [RV00], Hapori [FKS+23], HPlan-P [BHBM06], HSP [Bla98], hsp_0^* [Has08], IPP [Koe99], IPPLAN-1SC [BKV06], LAMA [RW08], Levitron [CFH+23a], maplan-1|2 [FK18], Maxplan [XCZ06], MERWIN [KLMT18], MIPlan [14], Mips [EH01], MSP [FBB+18], Odin [DSS23], OLCFF [FH18], OpCount4Sat [DPC23], Optiplan [FFP23], PbR [AK01], Plan-A [CLH08], Planning-PDBs [MEMF18], Powerlifted [CFH+23c], Probe [11], PropPlan [Fou00], QDom-Lmcut [Tor23a], Ragnarok [DGH+23], saarplan [FGSH18], Scorpion [Sei18b], Scorpion Maidu [CFH+23b], SHOP [NCLM01], Spock [SFS23], STAN [LF99], Stella [SOM01], SymBD [Tor23b], Symk [Spe23], Symple [SGM18], TFTM1 [KT23], TokenPlan [MF01], Upwards [CS08], YochanPS [BKD06]
HTN	Aries [Bit23], HyperTensionN [MMS22], LiloTane [Sch20], LTP [QPF23], OptiPlan [FFP23], PandaDealer [OHB23], PANDApro [Hol23a], pyHiPOP [LA20], SIADEX [FVC20], TOAD [Hol23b], VH-POP [YS03]
Numeric	LNM-Plan [KSB23], OMTPlan [Leo23]
Non-deterministic	Conformant-FF [HB06], CPAh [TNPS08], Gamer [KE08], POND [BKS06], T0 [PG08]

Continuation of Table A.3	
Planning types	**Planner**
Temporal	CP4TP [FJ18], DAE-1 [BSSV08], OPTIC [CC18], (CPT, SGPlan, Crikey, HSP*a, TP-4) [04], (DAE-YAHSP, LMTD, POPF-2, Sharaabi) [11], TLP-GP [Mar08], (ITSAT, tBURTON) [14], TemPorAl [CVC18], Temporal FD [RE08], TFLAP [VR18], TALplanner [KD00]
Learning	AGAP [CV14], ASNets 2023 [HTWT23], Cabala [ROB08], DAE-1 [BSSV08], Ensemble-Roller [RF14], FD SMAC [SSH14b], GOFAI [TG23], HUZAR [GL23], L2Plan2 [GL08], LIBaCOP [Cen14], LLAMA [VBA14], Macro-AltAlt [MN08], MIPlan [14], Muninn [SBG23], Novelty-based Progressive Generalized Planner [LLE23], (Bootstrap-Planner, CBL, Fast-Downward-Autotune-quality, OALDAEYASHP, Par-LPG) [11], PbP [GGSV08], Replica [GFB08], RollEnt [FC14], Roller [RJ08], Sayphi-Rules [BF08], Vanir [Dre23], Wizard [NLFL08]
Probabilistic	A2C-Plan [FKIT18], Beaver [RJFT11], CERT [TF04], Classy [YFG04], Conformant-SOGBOFA-B-IPC18 [CK18], DiSProD [CCK23], FCPlanner [KS04], FF-rePlan [YFG07], FSP*-(RBH/RDH) [TIK08a], Glutton [KDMW12], HMDPP [KG08c], Imitation-Net [FIT18a], KAIST AILAB [SKK+08], LPPFF [KG08a], mGPT [BG05], NMRDPP [GPT04], POND-Hindsight [Ols11], Probapop [OWL04], PROST [KE12], Prost-DD [GS18], Purdue-Humans [YFG04], Random-Bandit [FIT18b], RFF-(BG/PG) [TIK08b], SEH [WKG08], SPUDD [HSHB11], Symbolic Perseus [Pou11], UMass [FH04]
End of Table	

A.2.4 Numbers of found patterns per planner type

Table A.4: Patterns found per planner type

Begin of Table	
Planning types	**Planner**
HTN	pyHiPOP [LA20], TOAD [Hol23b]
Numeric	LNM-Plan [KSB23], OMTPlan [Leo23]
Non-deterministic	CPAh [TNPS08]
Temporal	CP4TP [FJ18], DAE-1 [BSSV08], OPTIC [CC18], TemPorAl [CVC18], Temporal FD [RE08]
Learning	AGAP [CV14], (Bootstrap-Planner, Fast-Downward-Autotune-quality, Par-LPG) [11], DAE-1 [BSSV08], FD SMAC [SSH14b], GOFAI [TG23], HUZAR [GL23], L2Plan2 [GL08], LIBaCOP [Cen14], LLAMA [VBA14], Muninn [SBG23], PbP [GGSV08], Vanir [Dre23]
Probabilistic	Imitation-Net [FIT18a], PROST [KE12], Prost-DD [GS18], Random-Bandit [FIT18b], Symbolic Perseus [Pou11]

Continuation of Table A.4	
Planning types	**Planner**
Classical	(AIIPACA, BiFD, Fast Downward Uniform, IBaCoP, Jasper, Mercury, Metis, Rational Lazy A*) [14], Approimate Novelty Anytime [SLRS21], (Arvand, Arvand Herd, BJOLP, BRT, Fast Downward Stone Soup, FD Autotune, Lamar, Merge and Shrink, Randward, Roamer, SelMax) [11], CEGAR++ [Kre23], Cerberus [Kat18], Complementary-1 [FLB+18], ComplementaryPDB [FEM23], DAE-1 [BSSV08], DALAI 2023 [BCEK23], Delfi-1 [KSSS18], DiSCO [FG23], Dofri [HSS23], DTG-Plan [HCZ08], (Fast Diagonally Downward, Fast Downward) [04], FD Remix [Sei18a], FDMS [Sie18], fs-blind [FGLR18], FSM [LBP23], FTSPlan [TSTN23], Hapori [FKS+23], LAMA [RW08], Levitron [CFH+23a], MERWIN [KLMT18], MSP [FBB+18], Odin [DSS23], OLCFF [FH18], OpCount4Sat [DPC23], Powerlifted [CFH+23c], QDom-Lmcut [Tor23a], Ragnarok [DGH+23], saarplan [FGSH18], Scorpion [Sei18b], Scorpion Maidu [CFH+23b], Spock [SFS23], SymBD [Tor23b], Symk [Spe23], Symple [SGM18], TFTM1 [KT23]
End of Table	

A.2.5 Patterns

Table A.5: Patterns found per planner type

Begin of Table	
Patterns	**Planner**
Proxy	(Arvand, Arvand Herd, BJOLP, Bootstrap-Planner, BRT, Lamar, Roamer, Fast Downward Stone Soup, Fast-Downward-Autotune-quality, Merge and Shrink, Randward, SelMax, FD Autotune) [11], (BiFD, Fast Downward Uniform, Mercury, Metis, Rational Lazy A*, Jasper) [14], CEGAR++ [Kre23], Cerberus [Kat18], Complementary-1 [FLB+18], ComplementaryPDB [FEM23], CP4TP [FJ18], DALAI 2023 [BCEK23], Delfi-1 [KSSS18], DiSCO [FG23], Dofri [HSS23], (Fast Diagonally Downward, Fast Downward) [04], FD Remix [Sei18a], FD SMAC [SSH14b], FDMS [Sie18], FSM [LBP23], FTSPlan [TSTN23], GOFAI [TG23], HUZAR [GL23], LAMA [RW08], Levitron [CFH+23a], LLAMA [VBA14], LNM-Plan [KSB23], MERWIN [KLMT18], MSP [FBB+18], Odin [DSS23], OLCFF [FH18], OMTPlan [Leo23], OpCount4Sat [DPC23], Powerlifted [CFH+23c], QDom-Lmcut [Tor23a], Ragnarok [DGH+23], saarplan [FGSH18], Scorpion [Sei18b], Scorpion Maidu [CFH+23b], Spock [SFS23], SymBD [Tor23b], Symk [Spe23], Symple [SGM18], TemPorAl [CVC18], Temporal FD [RE08], TFTM1 [KT23], TOAD [Hol23b], Vanir [Dre23]
Registry	DiSCO [FG23], Dofri [HSS23], FSM [LBP23], LNM-Plan [KSB23], SymBD [Tor23b], Symk [Spe23], Symple [SGM18], TFTM1 [KT23]

Continuation of Table A.5	
Patterns	**Planner**
Iterator	Symbolic Perseus [Pou11], VAL [FLHC14]
Singleton	CPAh [TNPS08]
Command	DiSCO [FG23]
Factory	Universal PDDL parser [Jon15], VAL [FLHC14], Vanir [Dre23], TOAD [Hol23b], Temporal FD [RE08], TemPorAl [CVC18], Symple [SGM18], Symk [Spe23], SymBD [Tor23b], Spock [SFS23], Scorpion Maidu [CFH+23b], Scorpion [Sei18b], saarplan [FGSH18], QDom-Lmcut [Tor23a], Powerlifted [CFH+23c], OPTIC [CC18], OpCount4Sat [DPC23], OLCFF [FH18], Odin [DSS23], MSP [FBB+18], MER-WIN [KLMT18], LNM-Plan [KSB23], LLAMA [VBA14], Levitron [CFH+23a], LAMA [RW08], HUZAR [GL23], GOFAI [TG23], FTSPlan [TSTN23], FSM [LBP23], fs-blind [FGLR18], FDMS [Sie18], FD SMAC [SSH14b], (Fast Downward, Fast Diagonally Downward) [04], Dofri [HSS23], DiSCO [FG23], Delfi-1 [KSSS18], DALAI 2023 [BCEK23], DAE-1 [BSSV08], CP4TP [FJ18], Complementary-1 [FLB+18], Cerberus [Kat18], CEGAR++ [Kre23], (BiFD, Fast Downward Uniform, Mercury, Metis, Rational Lazy A*, Jasper) [14], (Arvand Herd, Arvand, BJOLP, BRT, Bootstrap-Planner, Lamar, Roamer, Fast Downward Stone Soup, Fast-Downward-Autotune-quality, Merge and Shrink, Randward, SelMax, FD Autotune) [11]
None	A2C-Plan, ACOPlan, alien, AltAlt, Aries, ASNets 2023, ayAlsoPlan, BDDPlan, Beaver, BFHSP, BFS(f), Blackbox, C3, Cabala, CBL, CBP, CERT, CFDP, Classy, co-plan, Conformant-FF, Conformant-SOGBOFA-B-IPC18, CPT, Crikey, DAE-YAHSP, DecStar, DiSProD, DPMPlan, Fast Downward-sa, FCPlanner, FD Cedalion, FDP, FF, FF-rePlan, $FF(h_a)$, Fork Init, Forward Backward Anytime Novelty Search, Freelunch, freelunch-madagascar, FSP*-(RBH/RDH), Gamer, Glutton, Grounding Schematic Representation with GRINGO for Width-based Search, GRT, hflow, HMDPP, HPlan-P, HSP, HSP*a, hsp_0^*, HyperTensionN, IPP, IPPLAN-1SC, ITSAT, KAIST AILAB, LiloTane, LM-Cut, LMTD, LPG, LPG-TD, LPPFF, LPRPG-P, LTP, Macro-AltAlt, Macro-FF, Madagascar, maplan-1|2, Maxplan, mGPT, MIPlan, Mips, NMRDPP, Novelty-based Progressive Generalized Planner, NuCeLaR, OALDAEYASHP, Optiplan, OptiPlan, PandaDealer, PANDApro, PbR, PHSFF, Plan-A, Planets, Planning-PDBs, POND, POND-Hindsight, POPF-2, Probapop, Probe, PropPlan, Purdue-Humans, Replica, RFF-(BG/PG), RIDA, RollEnt, Roller, RPT, SATPLAN, Sayphi-Rules, SEH, Semsyn, SGPlan, Sharaabi, SHOP, SIADEX, Simplanner, SIW, SPM&S, SPUDD, STAN, Stella, SymBA*, T0, TALplanner, tBURTON, TFLAP, TLP-GP, TokenPlan, TP-4, UMass, Upwards, USE, VHPOP, Wizard, YAHSP, YochanPS, Planning.Domains, itSimple, myPDDL, PDDL Studio, PDDL4J
Strategy	(AIIPACA, IBaCoP) [14], Approimate Novelty Anytime [SLRS21], Delfi-1 [KSSS18], Hapori [FKS+23], L2Plan2 [GL08], LIBaCOP [Cen14], MSP [FBB+18], (AGAP, Par-LPG) [11], PbP [GGSV08]

Continuation of Table A.5	
Patterns	**Planner**
Template Method	DALAI 2023 [BCEK23], HUZAR [GL23], Imitation-Net [FIT18a], Jasper [14], LAMA [RW08], (Arvand Herd, Lamar, Roamer) [11], Levitron [CFH+23a], LLAMA [VBA14], MERWIN [KLMT18], MSP [FBB+18], Muninn [SBG23], OMTPlan [Leo23], PROST [KE12], pyHiPOP [LA20]
State	DTG-Plan, PROST [KE12], Prost-DD [GS18], Ragnarok [DGH+23], Random-Bandit [FIT18b], SymBD [Tor23b], Symk [Spe23], Symple [SGM18], TemPorAl [CVC18], TFTM1 [KT23]
End of Table	

A.2.6 Pattern combinations

Table A.6: Patterns found per planner type

Begin of Table	
Pattern Combinations	**Planner**
Proxy, Factory	(Arvand, BJOLP, Bootstrap-Planner, BRT, Fast Downward Stone Soup, Fast-Downward-Autotune-quality, Merge and Shrink, Randward, SelMax, FD Autotune) [11], (BiFD, Fast Downward Uniform, Mercury, Metis, Rational Lazy A*) [14], CEGAR++ [Kre23], Cerberus [Kat18], Complementary-1 [FLB+18], CP4TP [FJ18], (Fast Diagonally Downward, Fast Downward) [04], FD SMAC [SSH14b], FDMS [Sie18], FTSPlan [TSTN23], GOFAI [TG23], Odin [DSS23], OLCFF [FH18], OpCount4Sat [DPC23], Powerlifted [CFH+23c], QDom-Lmcut [Tor23a], saarplan [FGSH18], Scorpion [Sei18b], Scorpion Maidu [CFH+23b], Spock [SFS23], Temporal FD [RE08], TOAD [Hol23b], Vanir [Dre23]
Proxy, Factory, Template	(Arvand Herd, Lamar, Roamer) [11], DALAI 2023 [BCEK23], HUZAR [GL23], Jasper [14], LAMA [RW08], Levitron [CFH+23a], LLAMA [VBA14], MERWIN [KLMT18]
Proxy, Factory, State, Registry	SymBD [Tor23b], Symk [Spe23], Symple [SGM18], TFTM1 [KT23]
Proxy, Factory, Registry	Dofri [HSS23], FSM [LBP23], LNM-Plan [KSB23]
Proxy, Factory, Strategy	Delfi-1 [KSSS18]
Proxy, Factory, Registry, Command	DiSCO [FG23]
Proxy, Strategy, Factory, Template	MSP [FBB+18]
Proxy, Template	OMTPlan [Leo23]
State, Template	PROST [KE12]
Proxy, State	Ragnarok [DGH+23]
Factory, Iterator	VAL [FLHC14]

Continuation of Table A.6	
Pattern Combinations	**Planner**
Proxy, Factory, State	TemPorAl [CVC18]
End of Table	

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature