Universidad Carlos III de Madrid

PhD Thesis

Automated Planning Through Abstractions in Dynamic and Stochastic Environments

Author:

Moisés Martínez Muñoz

Thesis advisors:

Dr. D. Daniel Borrajo Millán and

Dr. D. Fernando Fernández Rebollo

Computer Science Department

Leganés, September, 2016

PhD Thesis

Automated Planning Through Abstractions in Dynamic and Stochastic Environments

Author: Moisés Martínez Muñoz

Advisors: Dr. D. Daniel Borrajo Millán
and Dr. D. Fernando Fernández Rebollo

Selection Board		Signature
Chairman:	.....	.....
Member:	.....	.....
Secretary:	.....	.....

Score: .....

The thesis defense was on Leganés, ..., 2016 at the Universidad Carlos III de Madrid.

In the computer field, the moment of truth is a running program; all else is prophecy

---

Herbert Simon

ACKNOWLEDGEMENTS

These pages could not exist without the presence and help of many people with whom I shared these years at Carlos III de Madrid University.

First of all I want to thank my PhD advisors Daniel Borrajo and Fernando Fernández, because they gave me the opportunity of doing this thesis in the first place. They followed me in all these years, always encouraging, with patience and understanding. Daniel is one of those persons that leaves his mark, for his knowledge, kindness and insight. I learned a lot about Artificial Intelligence from him, he made me improve in many aspects of life, eventually making me a better researcher. Fernando has tried to focus my crazy mind in this work during these amazing four years. I learned a lot about Machine Learning from him, he made me more self-confident and a better researcher too.

Also, I want to thank the other people that I have collaborated with during this time. I am grateful to Yago Sáez, he introduced my first steps in Artificial Intelligence with kindness and passion. My visit to Örebro, where I met Franziska Klügl. She introduced me to multi-agent simulated environments that helped me to improve some aspects of this thesis. Javier García Polo because it has been amazing to work with him at any crazy idea related to Automated Planning, Machine Learning and/or Robotics. I am deeply grateful to Nerea Luis because we shared so much together, from the days in the office spent running for dead-lines, supervising bachelor thesis, coding for StarCraft, bothering about Artificial intelligence or Robotics and/or looking for sponsors to T3F.

I also want to thank Carlos Linares, Frederic Py and Francisco Melo for their valuable comments that helped me out to improve this work. I am truly grateful of all the effort they did into their reviews.

Thanks to the people of the Planning and Learning Group (PLG) at Carlos III de Madrid University. Especially with my 'labo' mates: Eloy Flórez, Javier García, Daniel Pérez, Javier Ortiz, Emilio Martín, Ezequiel Quintero, Vidal Alcazar, Álvaro Torralba, Jesús Virseda, Sergio Nuñez and Jose Carlos González. Because, it is amazing to work with people like them. Thanks to Neli for solving each administrative stuff making my life a bit easy during these years. Also I want

to include in these acknowledgements all the institutions that provided me with funding for my activities, without their generous contributions I could not finish this work.

To my friends, the old and the new: Nacho, Jaime, Ignacio, Victor, Carlos, Marga, Ana, Edu, Eloy, Javi, Alvaro, Ricardo, Jesús, Sergio, Nerea ... to be there in the good and bad times during this path. I would like to thank all the people that feed my curiosity about maths, physics and computer sciences. My father, my high school teachers and my teachers, mates and students in the university. They all made possible this work.

A final thanks goes to my family, that has been always supportive, despite myself. I am deeply grateful to my Father, Susana and David who always are there for everything. I would not be the person that I am without them.

Thank you all so much.

ABSTRACT

Generating sequences of actions – plans – for an automatic system, like a robot, using Automated Planning is particularly difficult in stochastic and/or dynamic environments. These plans are composed of actions whose execution, in certain scenarios, might fail, which in turn prevents the execution of the rest of the actions in the plan. Also, in some environments, plans must be generated fast, both at the start of the execution and after every execution failure. These problems have contributed to generate new Automated Planning models (Planning under Uncertainty) to tackle these situations. These models include changes in the representation of the information to manage the dynamics of the environment (action outcomes, observability of the environment, etc). In spite of the initial advantages of these models, there are some important disadvantages that increase the cost of generating a plan. These models require an accurate definition of the environment's dynamics. Frequently, it is extremely difficult to define such accurate models, and in some environments the amount of information needed is huge. The most common solution to avoid these problems consists on repairing or re-planning when a failure in execution is detected due to the lack of information. Therefore, at each planning (re-planning) step, a new plan of actions is generated including the possible changes in the environment.

The main objective of this thesis consists on developing a new planning-execution approach that reduces the computational effort of the planning task in dynamic and stochastic scenarios. These scenarios present some challenges that increase the complexity of the planning-execution process: (i) new information about the environment can be discovered during action execution, modifying the structure of the planning task; (ii) actions' execution can fail which in turn prevents the execution of the rest of the plan; (iii) the execution of the actions in the plan can generate states from which there is no solution (dead-ends); (iv) plans may need to be generated quickly to offer a real time interaction between the automated planning system and the environment; and (v) planning tasks present scalability problems. For these reasons, the process of generating and executing a plan of actions can be prohibitively expensive in real world environments.

In the first part of this thesis, we detail novel methods for generating predicate abstractions from planning tasks. We then propose a way of using these predicate

abstractions during search to generate partial abstract plans, while considering the far future information with a different level of detail, by selectively removing predicates of the planning task. This approach generates partial abstract solution plans where the k-first actions in the plan are guaranteed to be applicable as long as the information about the environment does not change or the actions' execution is correct. Meanwhile, the rest of the plan might not necessarily be applicable due to the level of the details of the predicate abstraction. In the second part of this thesis, we focus on improving the technique developed on the first part to implement a method to generate predicate abstraction automatically using different extraction methods (Landmark based and Relaxed Plan based). Finally in the third part of this thesis, we compare the performance of both extraction methods conducting a detailed study about the generation time and the type of predicate chosen in order to analyze their effect in the planning-execution process.

Finally, we provide an outlook on possible extensions of our work in different directions: (1) investigating more complex ways to deploy abstractions during search using different level of abstractions; (2) deploying predicate abstractions to generate partial plans which can be used to guide the search; and (3) modifying the value of k dynamically in order to improve the quality of the partial plans generated for our approach.

RESUMEN

Generar secuencias de acciones – planes – para un sistema automático, como un robot, mediante la utilización de Planificación Automática es particularmente difícil en entornos estocásticos y/o dinámicos. Estos planes están compuestos por acciones cuya ejecución puede fallar en algunas ocasiones, evitando que se puedan ejecutar el resto de acciones que componen el plan. Además, en algunos entornos, los planes deben ser generados rápidamente, tanto al comienzo de la ejecución, como cuando un fallo es detectado durante ésta. Estos problemas han contribuido a que aparezcan nuevos modelos de Planificación Automática (Planificación con Incertidumbre) capaces de manejar estos problemas. Estos modelos incluyen cambios en la representación de la información para gestionar las dinámicas del entorno (resultados de las acciones, observabilidad del entorno, etc). A pesar de la ventajas iniciales de estos modelos, existen algunas importantes desventajas que producen un incremento en el coste de generación de los planes de acciones. Esto es debido a que es necesario tener una representación precisa de la dinámica del entorno y frecuentemente es extremadamente complicado obtenerla o es demasiado grande para manejarla. La solución más común para evitar estos problemas consiste en reparar o re-planificar cuando se detecta un fallo durante la ejecución debido a la falta de información. Por lo tanto, en cada proceso de planificación (re-planificación), se genera un nuevo plan de acciones incluyendo los posibles cambios detectados en el entorno.

El objetivo principal de esta tesis consiste en definir un nuevo enfoque de planificación-ejecución basado en abstracciones que permita reducir el coste computacional de la tarea de planificación en escenarios dinámicos y estocásticos. Estos escenarios presentan algunos desafíos que aumentan la complejidad del proceso de planificación-ejecución: (i) nueva información sobre el entorno puede ser descubierta durante la ejecución de las acciones, modificando la estructura de la tarea de planificación; (ii) la ejecución de las acciones puede fallar impidiendo la ejecución del resto del plan; (iii) la ejecución de las acciones puede generar estados a partir de los cuales no hay solución (caminos sin salida); (iv) los planes de acciones pueden necesitar ser generados rápidamente para ofrecer una interacción más realista entre el sistema de planificación automática y el entorno; y (v) este tipo de entornos presentan problemas de escalabilidad debido a la explosión combinatoria implícita por la cantidad de información y la complejidad del modelo de acciones.

Por estas razones, el proceso de generar y ejecutar un plan de acciones puede ser prohibitivamente costoso en entornos del mundo real.

En la primera parte de esta tesis, vamos a describir de forma detallada el método desarrollados para generar abstracciones basadas en predicados para tareas de planificación. A continuación, proponemos una forma de utilizar abstracciones basadas en predicados durante la búsqueda para generar planes parciales, considerando la información futura con diferentes niveles de detalle, mediante la eliminación de forma selectiva de predicados de la tarea de planificación. Este enfoque generará planes parciales abstractos, donde las k-primeras acciones del plan (horizonte) podrán ser ejecutadas siempre y cuando la información sobre el entorno sea completa y no varíe, mientras que el resto del plan podría no ser necesariamente aplicable. En la segunda parte de esta tesis, nos centramos en la mejora de la técnica descrita en la primera parte mediante la implementación de diferentes técnicas para generar abstractiones basadas en predicados de forma automática utilizando distintos métodos de extracción (Landmarks y Plan relajado). Finalmente en la tercera parte de la tesis, comparamos el comportamiento de ambos métodos mediante la realización de un estudio detallado sobre el tiempo de generación y los predicate extraidos con el fin de analizar su efecto sobre el proceso de planificación y ejecución.

Por último, ofrecemos una perspectiva sobre las posibles extensiones de nuestro trabajo en diferentes direcciones: (1) investigando formas más complejas de desplegar las abstracciones durante la búsqueda utilizando diferentes niveles de abstracciones; (2) desplegando abstraciones basadas en predicados para general planes parciales que puedan ser utilizados con el fin de guiar el proceso de búsqueda; y (3) modificando el valor de k de forma dinámica con el fin de mejorar la calidad de los planes parciales generados por nuestra técnica.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1

INTRODUCTION

In the last decade, there was a growing need to build control systems with the ability to interact in complex stochastic and dynamic environments. This kind of environments generates significant challenges related to the tasks of sensing, control and deliberation. In particular, it is critical to design control algorithms that determine an appropriate action to take based on the current state of the world. But the process to choose these actions could be a difficult task depending on the information known about the environment or the time used to make decisions.

A solution could be to get some inspiration from human reasoning processes when solving problems from the real world. Within the field of **Artificial Intelligence** (AI), **Automated Planning** (AP) is the branch of AI that studies the generation of an ordered set of actions – plan – that allows a system to transit from a given initial state to a state where a set of goals have been achieved. AP has been successfully used to solve real world problems such as planning Mars exploration missions (Ai-Chang et al., 2004), managing fire extinctions (Asunción et al., 2005) or controlling underwater vehicles (Rajan et al., 2007). Despite of these examples, the application of AP systems to stochastic and dynamic environments still presents some challenges, mainly because these scenarios increase the complexity of the planning-execution process: (i) new information about the environment can be discovered during action execution, modifying the structure of the planning task; (ii) actions' execution can fail which in turn prevents the execution of the rest of the plan; (iii) the execution of the actions in the plan can generate states from which the rest of the plan cannot be successfully executed (dead-ends); (iv) plans may need to be generated quickly to offer a real time interaction between the AP system and the environment; and (v) planning tasks present scalability problems according to the combinatorial explosion. For these reasons, the process of generating and executing a plan of actions can be prohibitively expensive for this kind of scenarios.

There are two main (extreme) approaches to solve problems in stochastic and dynamic environment: deliberative and reactive. At one extreme, we find deliberative systems which are based on interleaving planning and execution with full or partial observability. Depending on **what** information about the environment

is known to build the plan of actions, **when** this information is collected and **how** this information is used to build the plan, we can group deliberative systems into three perspectives: (1) Off-line planning; (2) Situated planning; and (3) On-line planning.

Off-line planning systems are based on generating a unique plan of actions which is computed using an accurate domain model in order to handle the contingencies (failures, new information, sensing, etc) which can appear during execution. In order to generate such plan, the planning system needs to have full knowledge about the dynamics of the environment (failures in the actuators of a robot, the structure of the terrain, accuracy of sensors) to define an accurate action model. There are different alternatives based on off-line planning which consist on: (1) building conditional plans (Peot and Smith, 1992) where plans take into account all possible outcomes; (2) generating a set of policies by solving the problem as a Markov Decision Process (MDP) (Bonet and Geffner, 2005; Yoon et al., 2008; Kolobov et al., 2010); or (3) translating the planning task into another representation to apply algorithms which can solve the task in the new representation (Palacios and Geffner, 2005). But, unfortunately, the dynamics of the environment are not usually known or cannot be easily modeled. As alternative, we can try to learn the dynamics of the environment and then apply one of the previous approaches. However, the learning effort is huge except for small tasks (Zettlemoyer et al., 2005).

The most common solution consists on using **Situated planning**. Situated planning systems are based on generating a plan using a deterministic action model and replans and/or repairs the previous plan when a failure is detected during execution (the robot is not in the expected location, the robot hands do not hold the object, ...). When re-planning (Yoon et al., 2007), the planner generates an initial applicable plan and executes it, one action at the time. If an unexpected state is detected, the system generates a new plan from scratch. This process is repeated until the system reaches the problem goals. Therefore, at each planning (re-planning) step, including the initial one, the system is devoting a huge computational effort on generating a valid plan (an applicable plan that achieves the goals), when most of it will not be used. When repairing a running plan (Krogt and Weerdt, 2005; Fox et al., 2006; Borrajo and Veloso, 2012), the planner generates an initial applicable plan from scratch and executes it. If an unexpected state is detected, the system generates a new plan by reusing the plan generated previously adding and/or removing some actions. In general, deliberative systems require a huge computational effort to generate a complete and sound plan. Depending upon the dynamics of the environment, most probably the plan will not be fully executed.

On-line planning systems are based on executing actions continuously until goals are reached. These planning systems generate *partial solutions* of the planning task which do not guarantee that the plan will actually reach the goals. Partial plans are commonly generated using two alternatives: (1) Sub-goaling planning which generates a solution plan using a subset of goals (Champandard et al., 2009) chosen manually or automatically. These subsets can be generated using the different predicates which are part of the goals of the problem or computing an ordered set of landmarks choosing them as subgoals (Sebastia et al., 2006). (2) Limited-horizon planning searches for a solution plan until the planner reaches the goals or reaches a user-supplied bound (time, number of actions, etc), and then it returns the best solution found (Korf, 1990).

On the other extreme, there are several approaches that solve problems in stochastic and dynamic scenarios using reactive systems. These systems are based on greedily selecting the next action to be applied according to some knowledge which has been programmed or learned previously. If the knowledge about the environment is only used to select the next action, we can consider a pure reactive system without any deliberation, where the system perceives and generates the next action in a continuous cycle. Systems based on the Subsumption architecture (Brooks, 1986; Amir and Maynard-reid, 1999; Butler et al., 2001) are built using a control layer set, where different layers are interconnected with signals.During each execution step, one layer is chosen depending on the information perceived. Then, this layer execute a set of action in the environment. Other reactive approaches are based on building reactive behavioural navigation controllers using neural networks (Zalama et al., 2002; Yang and Meng, 2003) or fuzzy logic (Aguirre and González, 2003; Zhu and Yang, 2010). In general, reactive systems require much less computational effort and are "mostly" blind with respect to the future; they usually ignore the impact of the selected action on the next actions and states. Thus, they often get trapped in local minima or dead-ends.

The main idea of this thesis consists on extending some works on abstractions to the field of planning under uncertainty, by generating plans that provide detailed actions in the first steps of the plan. Meanwhile in later steps of the plan, our approach will only provide limited details, since the actions that are planned to be executed in the future are very unlikely to be used, given the uncertainty during execution. Our research has been inspired by the work of Zickler and Veloso (Zickler and Veloso, 2010), where a motion planning technique is presented. This technique generates a collision-free trajectory from an initial state to a goal state in dynamic environments. They introduced the concept of Variable Level-Of-Detail (VLOD), which focuses search on obtaining accurate short-term motion planning, while considering the far future with a different level of detail, by selectively ignoring the physical

interactions with dynamic objects. This technique decreases the computational effort of the motion planning process, so that information about different elements of the environment is not used to search a path to reach all goals. This approach only works for motion planning, so we have generalized it for task planning.

1.1 Objectives of the thesis

Current Automated Planning models (classical planning and planning under uncertainty) based on heuristic search, decision making and/or dynamic programming, only obtain robust plans when they have complete and accurate action models. But unfortunately obtaining an accurate action model from the real world is not always possible or the cost to get it is huge. Besides, if the information used to represent the environment is huge, the time required to generate a plan could be prohibitively large. Planning in real-world environments present some important drawbacks: (1) the complexity of solving a real-world planning task that encodes full information about the contingencies of the environment is EXPSPACE-complete (Littman et al., 1998); (2) it is extremely difficult to build an accurate action model of the real-world which encodes full knowledge about the contingencies of the environment and the learning effort to generate it is huge (Zettlemoyer et al., 2005); and (3) real-world scenarios imply quick interactions with the environment which cannot be performed if the time required to generate a plan is prohibitively large according to the complexity of the real-world scenario. According to this, the main objective of this thesis consists in proving our initial hypothesis:

> It is possible to solve planning tasks in dynamic and stochastic (real-world) environments using deterministic planning by generating k-bounded plans by means of abstractions which are built removing some predicates that represent future information about the environment.

This hypothesis is based on two main ideas: it is not always possible to collect all the information about the dynamics of a real world environment, or if it is possible to get it, the amount of information that describes the environment can not be managed by current planners; and in dynamic and stochastic environments it is not useful to spend a long time generating a detailed long plan, when most of it will not be applied due to the dynamic behavior of the environment (changes in the information known, unexpected states, new information, etc). The main objective has been divided into several sub-objectives, which are described in detail next:

- **Analyzing the existing literature related with this thesis** (Objective 1): the first objective of this thesis is to perform an exhaustive review of the existing literature in classical planning, planning under uncertainty, abstractions on AP and planning and execution. It requires a deep study about the advantages and limitations of the existing paradigms and how they can be improved to solve similar planning tasks in the dynamic and stochastic environments.

- **Decreasing the computational effort of the planning task** (Objective 2): solving problems in dynamic and stochastic environments is challenging in Automated Planning. A second objective of this thesis is to propose a new planning approach to decrease the time of the planning task that applies abstractions over the information known about the environment. The abstractions will be created over information about the future, which could change during the planning and execution processes.

- **Designing an automatic way to generate abstractions** (Objective 3): it is very important to identify **what** information can be used to generate the abstractions. This objective consists on designing a technique that generates abstractions automatically. This technique must analyze the information used to represent the environment and generate a set of abstractions, which can be used by the technique designed to solve the objective 2. The rules used to select the information to be abstracted must be based on the importance of the information for the planning process.

- **Building a set of test benchmarks** (Objective 4): The current test benchmarks for Automated Planning have been designed to analyze the capability of planners to find a solution, but these solutions are not usually executed in an environment. For this reason, we are going to generate a set of test benchmarks which will be composed of both domains from previous works and domains designed for this thesis. Besides, we are going to define a way to evaluate the features of the different techniques developed in this Thesis.

1.2 Thesis Outline

This introductory chapter has briefly summarized the research background, motivation, and the main ideas of this thesis. The remainder of this thesis is organized as follows:

Part I: State of the Art

- Chapter 2 provides background on the state of the art in Automated Planning. Specifically, it reviews the literature about deterministic, probabilistic, contingent and conformant planning approaches.
- Chapter 3 introduces the concept of **abstraction** and how abstractions have been deployed over deterministic planning.
- Chapter 4 introduces the concept of **Real Time Search**, the different techniques developed and the concept of lookahead in Heuristic Search.

Part II: Predicate Abstractions over Automated Planning

- Chapter 5 introduces the concept of Planning and Execution describing in detail the planning and execution architecture developed to conduct the empirical evaluation. Besides, this chapter also describes the method of empirical evaluation used to test the thesis.
- Chapter 6 introduces Variable Resolution Planning (VRP). Next, it describes the concept of predicate abstraction introducing a theoretical definition, a set of properties based on previous works (Knoblock et al., 1990; Yang et al., 1991) and a method to deploy predicate abstractions in Automated Planning choosing the predicate used to build the predicate abstraction and the horizon value that defines when the predicate abstraction is applied manually.
- Chapter 7 introduces the different methods developed to generate the predicate set which can be used to generate the predicate abstraction.

Part III: Conclusions and Future Work

- Chapter 8 summarizes the contributions of this thesis.
- Chapter 9 points out some directions for future research.

Part IV: Appendixes

- Appendix A describes the execution sequence of the modules that compose the different versions of the planner implemented in this thesis.

- Appendix B describes how to deploy and configure the LPELEA architecture implemented to conduct the empirical evaluation of this thesis.
- Appendix C includes the detailed results for the experiments conducted in Chapter 6.
- Appendix D includes the detailed results for the experiments conducted in Chapter 7.

Part I

STATE OF THE ART

2

AUTOMATED PLANNING

One of the aims of Artificial Intelligence consists on generating autonomous systems able to produce intelligent behaviours related to the abilities of the human brain. One of the distinct abilities of the human brain relates to its capability of long-term reasoning which consists on generating solutions to complex tasks. The general problem of finding a solution to a complex task has been tackled following three main trends: (1) programming-based approaches where a program encodes a specific method that solve a specific task; (2) model-based approaches where a general solver infers automatically a solution using an action model, the information about the problem and the objectives (goals); and (3) learning-based approaches where an automatic system improves itself by learning the adequate solution according to the information provided by an instructor or the environment.

Automated Planning is a model-based approach to autonomous behaviour which generates a plan of actions (behaviours) that solve a specific task according to the available information about the environment. In other words, a planning task can be defined such as a long-term reasoning process in which a set of agents must achieve a set of goals by executing a sequence of actions (Nau et al., 2004). The ability of long-term reasoning about the actions to perform, before acting, is certainly an important point to generate behaviours in real-world environments. Despite the complexity of long-term reasoning, Automated planning has proved to be effective to solve diverse real-world problems such as managing fire extinctions (Asunción et al., 2005), controlling underwater vehicles (Rajan et al., 2007), natural-language generation (Koller and Hoffmann, 2010), intermodal transportation problems (García et al., 2013) and controlling quadcopters (Bernardini et al., 2014).

This chapter is a review of Automated Planning which is the main topic addressed in this thesis. We introduce the relevant concepts in Automated Planning based on deterministic, probabilistic and non-deterministic approaches.

2.1 Introduction

Automated Planning is a field of Artificial Intelligence based on the **Problem Space** framework created by Herbert Simon and Allen Newell (Newell and Simon, 1972). A problem space is defined by a set of states and a set of operators that configure the domain model. The domain model can be represented as a directed graph whose nodes represent the states, and whose edges represent the operators. Formally, a planning task consists of an action model, an initial state and a goal state where a solution plan is a path from the node in the graph that represents the initial state to a goal node that represent a state identified as a goal state. This framework is extremely general to represent tasks in a large variety of domains, ranging from simple tasks to move a robot from one point to another to more complex planning tasks. A set of planning tasks can be defined within a Problem Space where each task is composed of an initial state and a set of goals. An Automated Planning system is based on a set of three common features:

- A Conceptual Model. A formal definition of the task to be solved and the structure of the solution
- A Modeling Language. A formal language that describes the problem solving task and the environment
- An Algorithm. The technique used to find a solution

At start, some assumptions were made to simplify problem solving. They are related to the representation of the environment, the actions or the states of the environment:

- Finite world: the environment is represented as a finite set of states
- Static world: the environment only changes when an action is executed
- Determinism: the execution of the same action in the same state always yields the same new state
- Total observability: there is complete knowledge about the state of the environment
- Implicit time: the execution of an action has no duration. Then, the state transitions are instantaneous
- Reachability goals: the objective of the planning task is to find a set of actions that transform a given initial state into another state satisfying the goals

Depending on which of these assumptions are relaxed we can define different types of planning paradigms. In this thesis, we are interested in the use of AP in dynamic and stochastic environments. This implies that it is not possible to have complete knowledge about both the environment and the action outcomes. Besides the actions have different durations. In more detail:

- Non-determinism: determinism is an unrealistic assumption because, when an action is executed in a real environment, predicting the effects is often difficult. We can differentiate three kinds of outcomes: (1) deterministic; (2) disjunctive, when different actions outcomes are possible (actions do not follow a probabilistic model); and (3) stochastic, when the effects of the actions follow a probabilistic model.
- Environment observability: in several systems, the state of the environment cannot be observed completely. Depending on the amount of information that is known, three levels of observability can be defined: (1) total observability, when full information about the environment can be captured or sensed; (2) partial observability, when the state cannot be fully observed and (3) no observability, when no information about the environment can be captured or sensed, except for the initial state.

According to these concepts (Non-deterministic and Environment Observability), different planning models (and thus, techniques) can be used to generate plans of actions. Table 1 shows some AP models that can be defined depending on these dimensions.

Planning Paradigms		
Observability	**Action Model**	
	Deterministic	Stochastic
Total	Deterministic	Probabilistic
Partial	Contingent	Probabilistic-Contingent
None	Conformant	Probabilistic-Conformant

Table 1: Automated planning paradigms.

According to Table 1, four different planning paradigms can be defined. Classical or Deterministic Planning is the sub-field of AP that studies the full relaxation of both dimensions. This sub-field works in a fully observable environment where action

outcomes are deterministic. Meanwhile, Planning under uncertainty is the sub-field of AP that studies the relaxation of the total observability and/or deterministic world assumptions. This sub-field can be divided into three fields depending on how the different dimensions are defined. Probabilistic planning works in a fully observable environment where action outcomes are stochastic. Conformant planning works in environments where no information about the environment can be observed except for the initial state. And finally, Contingent planning works in deterministic environments where the information known about the world is partially observable. Next, each paradigm is explained in more detail.

2.2 Classical Planning Task

Classical Planning can be defined as the process of generating a set of actions, which sequentially executed into the environment transform the initial state into another state where the goals are reached. This generation process can be considered as a path-finding process in a directed graph whose nodes represent the states of the environment, and whose edges represent the state transitions resulting from an action execution.

An example of a planning task of the Rovers domain (Mcdermott, 2000) is presented in Figure 1[1]. The environment is represented as a grid of 16 cells, called waypoints. Each one is denoted with a bi-dimensional coordinate (x, y), starting on the left bottom cell of the grid. White cells represent waypoints in which the rover can stay (free); and black cells represent obstacles. Two types of samples can be collected: rocks and soil. Rocks are denoted with a small black circle. Soil is denoted with a small black square. Rovers are located at waypoints and can move between any two free waypoints which are adjacent. Besides, rovers can take samples of rocks and soil and analyze them. Finally, there is a lander base which is used by the rovers to send information to the Earth about the analysis made over the samples. The typical goal of a Rovers task is to take some soil and rock samples, analyze them and send the result to the lander base. In the example, the dashed red line represents the deterministic plan to solve the planning task. We are going to use this example to explain some concepts about deterministic planning.

2.2.1 The Conceptual Model

The conceptual model for a classical planning task defines the formalization of the environment, the actions that can be used to modify the environment, and how

[1] This domain is a simplified version of the one used to provide plans for the actual Mars rovers.

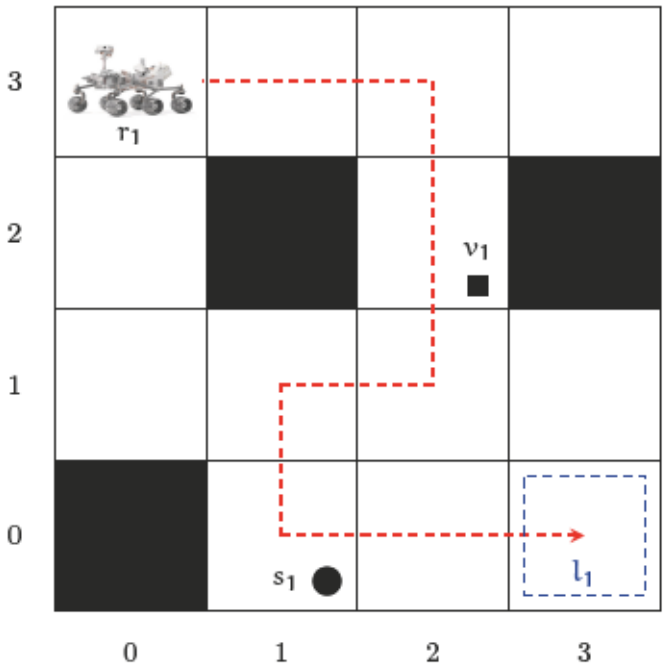


Figure 1: Classical planning task of the Rovers domain. Rover r_1 must take soil sample v_1 from location $(2, 2)$ and rock sample s_1 from location $(1, 0)$.

actions are applied in the environment to achieve the goals. Classical planning tasks are frequently encoded as transition graphs whose nodes represent the states of the environment, and whose edges represent the state transitions resulting from action execution (DUDA). This graph representation is the search space of the planning task which is commonly used for graph search algorithms to find a path (plan of actions) between the initial state and a goal state that solve the task.

Definition 1. (Deterministic Planning Model) A Deterministic Planning task can be defined as a 5-tuple $\Pi = (S, A, T, S_0, G)$, where:

- S is a finite set of states, where each state is a non-empty set of grounded literals (also known as facts or atoms).
- A is a finite set of grounded actions derived from the action schemes of the domain. Each action $a_i \in A$ can be defined as a tuple $a_i = (\text{Pre}, \text{Add}, \text{Del})$, where $\text{Pre}(a_i)$ are the preconditions of the action, $\text{Add}(a_i)$ are its add effects, and $\text{Del}(a_i)$ are the delete effects. $\text{Eff}(a_i) = \text{Add}(a_i) \cup \text{Del}(a_i)$ are the effects of the action. Besides, each action a_i has an associated non-negative integer cost, $\text{cost}(a)$ (the default cost is one).

- T is a state transition function $T : S \times A \rightarrow S : (s, a) \mapsto T(s, a) = s'$, where s' is the state resulting from applying the action a in a given state s.
- $s_0 \in S$ is a finite set of literals which encode the initial state.
- $G \subseteq S$ is a finite set of goal states.

An action $a \in A$ is applicable in a state $s_i \in S$ when a target state s_{i+1} exists such that $T(s_i, a) = s_{i+a}$. A solution plan π for a planning task Π is an ordered set of actions (commonly, a sequence) $\pi = (a_1, \ldots, a_n)$, $\forall a_i \in A$, that transforms the initial state s_0 into a state s_n where $G \subseteq s_n$. This plan π can be executed if the preconditions of each action are satisfied in the state in which it is applied; i.e. $\forall a_i \in \pi$, $Pre(a_i) \subseteq s_{i-1}$ such that state s_i results from executing the action a_i in the state s_{i-1}. s_0 is the initial state.

The plan length corresponds with the cost of the plan when all actions have unitary costs. In a conceptual model with non–unitary costs, plans with lower cost are preferred to plans with higher cost. A satisficing planning task consists in finding a plan of actions or showing that no such plan exists. An optimal planning task consists in finding a plan of actions that minimizes the value computed by Definition 2 or showing that no such plan exists.

Definition 2. (Plan Cost) The cost of a plan $\pi = (a_1, \ldots, a_n)$, $\forall a_i \in A$ is given by a function

$$cost(\pi) = \sum_{i=1}^{n} C(a_i)$$

There are two main ways of formalizing the conceptual model of a classical planning task in the literature: Propositional and Multi-Valued.

2.2.1.1 Propositional Formalization

Classical planning tasks are formalized in Propositional Logic (Mendelson, 1987) using propositions that describe the state of the world in terms of objects (robots, locations, rocks, etc), predicates which describe static or dynamic features of these objects or relations among them (e.g. locations are connected by roads) and actions that manipulate those relations (a robot can move from one location to another,

a package can be grasped by a robot) and describe what propositions are added or deleted when the action is applied. Each proposition is Boolean, so each such variable indicates whether a proposition about the world is true or false in a given state.

Definition 3. (Propositional Formalization of Classical Planning) A deterministic planning task using a propositional formalization is defined as a 4-tuple $\Pi = (F, A, I, G)$, where:

- F is a finite set of grounded literals (also known as facts or atoms).
- A is a finite set of grounded actions derived from the action schemes of the domain. Each action $a_i \in A$ can be defined as a tuple $a_i = (Pre, Add, Del)$, where $Pre(a_i) \subseteq F$ are the preconditions of the action, $Add(a_i) \subseteq F$ are its add effects, and $Del(a_i) \subseteq F$ are the delete effects. $Eff(a_i) = Add(a_i) \cup Del(a_i)$ are the effects of the action. Besides, each action a_i has an associated non-negative integer cost, $cost(a)$ (the default cost is one). Any state s is a subset of facts that are true at a given time step. An action a is applicable in s_i, if $Pre(a) \subseteq s_i$. Then, the result of applying an action a in state s_i generates a new state that can be defined as: $s_{i+1} = s_i \setminus Del(a) \cup Add(a)$.
- $I \subseteq F$ is a finite set of grounded literals that are true in the initial state.
- $G \subseteq F$ is a finite set of grounded literals which must be true in the goal state.

The example shown in Figure 1 depicts a simple task of the Rovers domain which consists on taking some images and analyzing some soil and rock samples. This information must be communicated to the lander base using the available rovers. The initial state is shown on Figure 1 and the goal consists on taking and analyzing one soil sample from waypoint *w22* and one rock sample from waypoint *w01*. States are composed of propositions of the type `(at r w)`, `(can_traverse r x y)`, ..., `(at_soil_sample w)`. Each proposition is composed of a sequence of parameters where the first parameter corresponds to the identifier of the action and the rest correspond to particular objects. For instance, the action `(at r w)` is identified The initial state s_0 is shown in Figure 2. The actions are navigate, sample-soil, sample-rock, drop, calibrate, take-image, communicate-soil-data, communicate-rock-data and communicate-image-data.

Figure 3 shows the structure of the partial transition graph that represents the conceptual model (action/state) for the planning task shown on Figure 1. This

(visible w10 w20) ... (visible w33 w23)
(at_soil_sample w22)
(at_rock_sample w10)
(at_lander general w30)
(channel_free general)
(at r1 w03) (available r1)
(store_of rs1 r1)
(empty rs1)
(equipped_for_soil_analysis r1)
(equipped_for_rock_analysis r1)
(can_traverse r1 w10 w20) ... (can_traverse r1 w33 w23)

Figure 2: Initial state of the planning task shown in Figure 1.

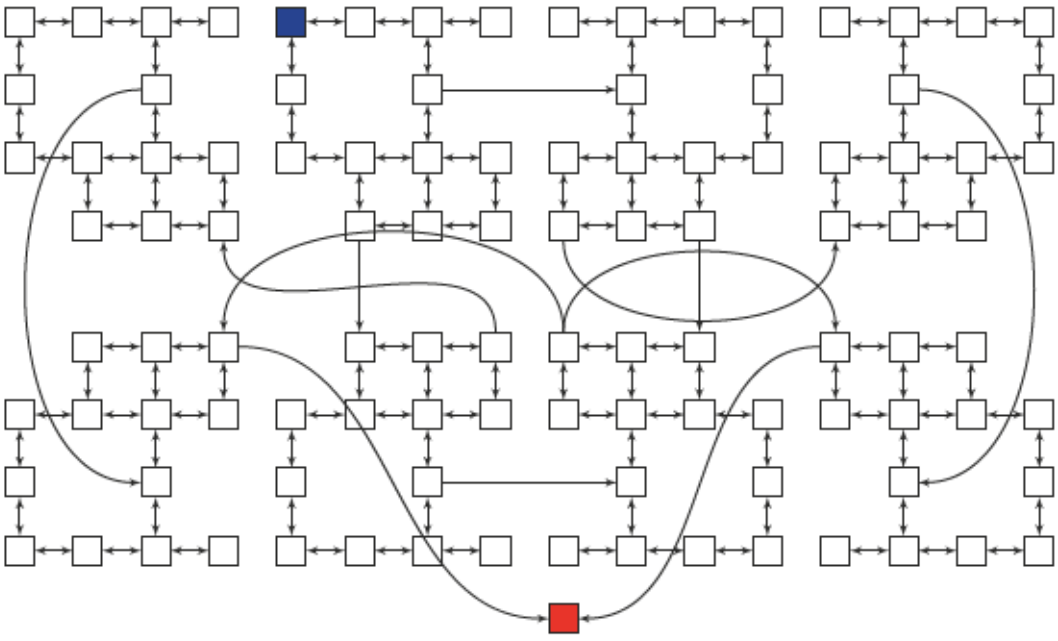


Figure 3: Transition graph for the propositional encoding of the planning task shown in Figure 1. The initial state is the blue one and the goal state is the red one.

graph represents the state space and the different actions which can be executed to transit between the different states that describe the planning task.

2.2.1.2 Multi-Valued Formalization

Planning tasks are formalized using multi-valued state variables that describe the state of the world and actions that describe how the values of the state variables change when the action is applied. There are two multi-valued formalizations: SAS^+ (Bäckström and Nebel, 1993) and Finite-Domain Representation (Helmert, 2006) which is based on SAS^+.

Definition 4. (Multi-valued Formalization of Classical Planning) A planning task using a multi-valued formalization is defined as a 4-tuple $\Pi = (V, A, s_0, s_*)$, where:

- V is a finite set of state variables, where each state variable $v \in V$ has an associated extended domain $D_v^+ = D_v \cup \{u\}$. The domain of a state variable is composed of a set of values D_v and the undefined value u which is used to denote when the value is unknown. A value of a state variables $v \in V$ in a given state s, also known as fluent, is defined as $s[v] \in D_v^+$. These state variables define the planning space $S^+ = D_{v_0}^+ \times \ldots \times D_{v_n}^+$. A partial variable assignment or partial state is a state in which at least a fluent $s[v] = u$.
- A is a finite set of actions over V. Each action $a \in A$ is a 3-tuple $a = (pre(a), post(a), prev(a))$ where $pre(a)$ represents the preconditions, $post(a)$ the post-conditions and $prev(a)$ the prevail-conditions. The preconditions of $a \in A$ are fluents which must be true prior to the application of the action a and become not true after its application. The post-conditions of $a \in A$ are fluents which are not true prior to the application of the action a and become not true after its applications. The prevail-conditions of $a \in A$ are fluents which must be true before and after the application of the action a.
- s_0 is the initial state which is defined over V such that $s_0[v_i] \neq u \; \forall \, v_i \in V$.
- s_* is a partial state over V such that $s_*[v_i] \in D_v^+ \; \forall \, v_i \in V$.

Any state s is a complete assignment to all the variables in V. This means that each variable is represented by a fluent. Therefore an action $a \in A$ is applicable in the state s if $\forall \, v_i \in V : (prev(a)[v_i] = u \vee prev(a)[v_i] = s[v_i]) \wedge (pre(a)[v_i] = u \vee pre(a)[v_i] = s[v_i])$. Then, the result of applying an action a in state s_i generates a new state that is equal to s_i except that $\forall \, v_i \in V post(a)[v_i] \neq u : s_{i+1}[v_i] = post(a)[v_i]$. An action sequence $\pi = (a_1, \ldots, a_n), \forall a_i \in A$ is a solution plan if $s_* \subseteq s_n$ where s_n is the final state generated after the sequential execution of the plan π.

$V = \{v_{r_1}, v_{v_1}, v_{s_1}, v_{rs_1}, v_{a_1}, v_{a_2}\}$

$D_{v_{r_1}} = \{\text{at-r1-w01, at-r1-w02, } \ldots, \text{ at-r1-w33}\}$

$D_{v_{v_1}} = \{\text{communicated-rock-data-w10, } \neg\text{communicated-rock-data-w10}\}$

$D_{v_{s_1}} = \{\text{communicated-soil-data-w22, } \neg\text{communicated-soil-data-w22}\}$

$D_{v_{rs_1}} = \{\text{empty-rs1, full-rs1}\}$

$D_{v_{a_1}} = \{\text{have-rock-analysis-r1-w10, } \neg\text{have-rock-analysis-r1-w10}\}$

$D_{v_{a_2}} = \{\text{have-soil-analysis-r1-w22, } \neg\text{have-soil-analysis-r1-w22}\}$

$A = \{a_1, a_2, \ldots, a_{37}\}$

$S_0 = \{v_{r_1} \rightarrow \text{at-r1-w03, } v_{v_1} \rightarrow \neg \text{ communicated-rock-data-w10,}$
$v_{s_1} \rightarrow \neg \text{ communicated-soil-data-w22, } v_{rs_1} \rightarrow \text{empty-rs1,}$
$v_{a_1} \rightarrow \neg \text{ have-rock-analysis-r1-w10, } v_{a_2} \rightarrow \neg \text{ have-soil-analysis-r1-w22}\}$

$s_* = \{v_{r_1} \rightarrow \text{communicated-rock-data-w10, } v_{s_1} \rightarrow \text{communicated-soil-data-w22}\}$

Figure 4: Partial *SAS+* encoding of the planning task of Figure 1.

Figure 4 shows part of a SAS^+ encoding of the planning task depicted in Figure 1. The planning task is composed of six state variables: (1) a variable that describes where the rover is, v_{r_1}; (2) a variable that describes the state of the rover store, v_{rs_1}; (3) a variable that describes if the rock sample from waypoint (1,0) has been communicated, v_{v_1}; (4) a variable that describes if the soil sample from waypoint (2,2) has been communicated, v_{s_1}; (5) a variable that describes if the rock sample from waypoint (1,0) has been analyzed, v_{a_1}; and (6) a variable that describes if the soil sample from waypoint (2,2) has been analyzed, v_{a_2}. For instance, the variable $v_{r_1} \in V$ is defined to represent the location of rover r_1 such that $D_{v_{r_1}} = \{$at-r1-w01, at-r1-w02, $\ldots$, at-r1-w33$\}$. $D_{v_{r_1}}$ describes the different values which can be assigned to variable v_{r_1}. Figure 5 shows the structure of the domain transition graph for the variables v_{r_1} and v_{rs_1}.

2.2.2 The Modeling Language

The modeling language is a notation that describes the semantic representation of the planning task. The most representative languages used to describe planning tasks are based on a variation of First Order Logic (FOL) where each atom of information is defined using predicates. Different languages have been defined in Automated Planning to represent the conceptual model described previously, and some of them are summarized next.

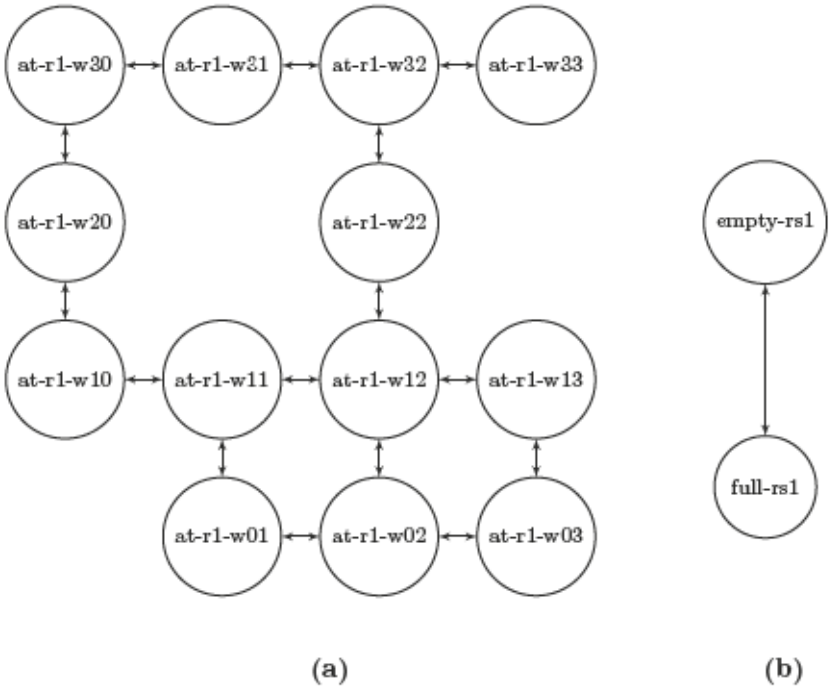


Figure 5: Multi-valued conceptual model corresponding to the planning task defined in Figure 1. (a) Domain transition graph for rover r_1. (b) Domain transition graph for rover store rs_1.

2.2.2.1 STRIPS

The STanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson, 1971) language is the base of most modern planning representation languages. This language is based on two important logic assumptions: (1) the **Closed World Assumption** (Reiter, 1987) which assumes that all facts that are not known to be true, are false; and (2) the **STRIPS Assumption** to handle the **Frame Problem** (McCarthy and Hayes, 1969) which consists on representing only the propositions that change after applying an action and assuming that the rest of propositions do not change their values. In other words, STRIPS assumes that only actions can change a specific part of the world state according to their own effects.

Commonly, the planning task is provided in two input files: a problem and a domain. The problem file defines a set of objects (instantiations of types in the domain), an initial state (I), and a set of goals (G). Figure 6 shows the problem description encoded using STRIPS of the planning task shown in Figure 1.

```
(define (problem rover1) (:domain rover)
(:objects
   general - Lander
   colour high_res low_res - Mode
   r1 - Rover
   rs1 - Store
   w00 w01 w02 w03 w10 w11 w12 w13 w20 w21 w22 w23 w30 w31 w32
   w33 - Waypoint
   camera0 - Camera
)
(:init
   (can_traverse r1 w01 w02)
   (can_traverse r1 w01 w11)
   (can_traverse r1 w02 w03)
   (can_traverse r1 w02 w12)
   ....
   (at_soil_sample w10)
   (at_rock_sample w22))
   (:goal (and
   (communicated_soil_data w10)
   (communicated_rock_data w22)))
```

Figure 6: Partial problem description encoded in STRIPS of the planning task shown in Figure 1.

The domain file contains the definition of a set of generalized actions (whose instantiations with problem objects will lead to actions in A) and a set of ungrounded predicates (whose instantiations will generate literals in F). Table 2 shows actions of the Rovers domain in STRIPS language.

2.2.2.2 ADL

The Action Description Language (ADL) (Pednault, 1994) is one of the first extensions of STRIPS. This language increases the expressivity of the STRIPS language. This language is based on the **Open World Assumption** which means that any proposition which is not asserted by the effects of the actions in the state of the world or defined in the initial state is taken to be unknown. The main contributions of this language are:

Action	Preconditions	Added	Deleted
navigate	can-traverse(*r*, x, y) available(*r*) at(*r*, x) visible(x, y)	at(*r*, y)	at(*r*, x)
sample-soil	at(*r*, x) at-soil-sample(x) equipped-for-soil-analysis(*r*) store-of(*r*, s) empty(s)	full(s) have-soil-analysis(*r*, x)	empty(s) at-soil-sample(x)
sample-rock	at(*r*, x) at-rock-sample(x) equipped-for-rock-analysis(*r*) store-of(*r*, s) empty(s)	full(s) have-rock-analysis(*r*, x)	empty(s) at-soil-sample(x)
drop	store-of(*r*, s) full(s)	empty(s)	full(s)
calibrate	at(*r*, x) equipped-for-imaging(*r*) calibration-target(c, o) visible-from(o, x) on-board(c, *r*)	calibrated(c, *r*)	
take-image	on-board(c, *r*) calibrated(c, *r*) equipped-for-imaging(*r*) at(a, x) supports(c m) visible-from(o, x)	have-image(*r*, o, m)	calibrated(c, *r*)
communicate-soil-data	at(*r*, x) at(l, y) have-soil-analysis(*r*, p) visible(x, y) available(*r*) channel-free(l)	available(*r*) channel-free(l) communicated-soil-data(p)	available(*r*) channel-free(l)
communicate-rock-data	at(*r*, x) at(l, y) have-rock-analysis(*r*, p) visible(x, y) available(*r*) channel-free(l)	available(*r*) channel-free(l) communicated-rock-data(p)	available(*r*) channel-free(l)
communicate-image-data	at(*r*, x) at(l, y) have-image(*r*, o, m) visible(x, y) available(*r*) channel-free(l)	available(*r*) channel-free(l) communicated-image-data(p)	available(*r*) channel-free(l)

Table 2: STRIPS definition of the Rovers domain.

- The preconditions of the actions and the goals can be expressed using negations, disjunctions and quantified formulas.
- The effects of the actions can be expressed using conditional effects. This allows actions to have different outcomes according to the current state.
- The problem goals can be expressed as conjunctions and disjunctions.

2.2.2.3 PDDL

The Planning Domain Definition Language (PDDL) (Mcdermott, 2000) was created in order to define a standard planning language and to allow comparative analysis of the different planning systems. PDDL was also developed as the planning input language of the First International Planning Competition (IPC), whose objective was to compare the state-of-the-art planning systems. During the following years different versions of PDDL have been developed, each adding new features.

- PDDL 1.2 (IPC1 and IPC2) is the first version of the language and contains STRIPS and ADL features. Besides, it includes the use of typed variables.
- PDDL 2.1 (IPC3) increases the features of PDDL 1.2 adding numeric values which can be modified in the effects of the actions. Besides, it includes a new type of actions: **durative actions**. Durative actions introduce the concept of time in PDDL and allow discrete and continuous effects.
- PDDL 2.2 (IPC4) increases the features of PDDL 2.1 adding derived predicates (also known as axioms). Additionally, it added timed initial literals, which are propositions that become true after a given amount of time independent of the planning actions which have been selected previously.
- PDDL 3.0 (IPC5) allows the use of soft goals. A soft goal does not need to be achieved by the plan, but a cost is paid if it is not achieved at the end of the plan (Smith, 2004).
- PDDL 3.1 (IPC6) introduces functional STRIPS (Geffner, 2000). Functional STRIPS offers a different way to encode planning tasks mapping the objects of the planning task to their properties. This encoding provides a more natural modeling for some domains and makes the extraction of information from some heuristic functions easier (Edelkamp, 2002; Francès and Geffner, 2015).

Although PDDL 3.1 offers an extensive set of features and functionalities most the current planners do not support it. Actually, most planning systems only support

the propositional fragment of PDDL 2.2 that approximately corresponds to what was supported by STRIPS, typing (definition of types), the use of the equality predicate and numeric values. Figure 7 shows an example of a PDDL action. In this case, the action corresponds to the action navigate from the Rovers domain.

```
(:action navigate
 :parameters (?x - rover ?y - waypoint ?z - waypoint)
 :precondition (and (can_traverse ?x ?y ?z)
                    (available ?x)
                    (at ?x ?y)
                    (visible ?y ?z)
 :effect (and (not (at ?x ?y)) (at ?x ?z)))
```

Figure 7: PPDL definition for action *navigate* from Rovers domain.

2.2.3 The Algorithms

As commented above, solving a classical planning task can be considered as a path-finding process in a directed graph where graph search algorithms can be conducted in different ways: (1) forward search (also called progression), if the graph search algorithm starts searching at the initial state and goes towards the goal state; (2) backward search (also called regression), if the graph search algorithm starts searching at the goal state and goes towards the initial state; and (3) bidirectional search, when two graph search algorithms are performed simultaneously, one starting from the initial state and another one starting from the goals.

In general, graph search in Automated Planning is a extremely difficult task because the size of the search space may be exponential. STRIPS planning system was developed as a deliberative module of the software that controls the autonomous robot Shakey. This planning system implements a Depth-First Search (DFS) algorithm which expands the states of the problem space until a goal state is found without any type of guidance. This planning system tries to satisfy iteratively each goal independently of the others. In the next years, similar planning systems were developed adding domain-independent heuristics and Machine Learning techniques.

Other approaches tried to explore the problem space by other graph representations like Planning Graph. A Planning Graph is composed of the levels obtained by alternating propositions and actions layers. The first layer includes all the propositions defined in the initial situation, then the second layer is composed

of all the actions which can be applied in the previous layer. The third layer is composed of the propositions generated from applying the actions of the second layer. GRAPHPLAN (Blum and Furst, 1997) was the first planning system that performed search in a Planning Graph. This planner starts with an initial graph that only contains the proposition layer which is composed of literals of the initial state. Then, the graph is built expanding sequentially action and proposition layers until a proposition layer that satisfies the goals is reached. Then, the planner tries to extract a plan or determines that the goals are not achievable by a plan of length N. If no plan is found, it extends the graph one time step (the next action layer and the next propositional layer), and then it searches again for a solution plan. This graph is composed of all potential plans up to a certain length N, where N is the number of action layers. Different planning systems (Kambhampati et al., 1997; Smith and Weld, 1998) have applied this kind of representation.

Domain-independent heuristics have shown to be effective improving graph search algorithms to guide or prune the search towards the goal states. Planning as Heuristic Search (Bonet and Geffner, 2001; Hoffmann and Nebel, 2001) is based on the definition of an evaluation function, $f(n)$, that scores all states in the search graph according to how close they are to a goal state. Heuristic functions are usually obtained by solving a relaxed version of the original task, which is simpler than the original one (DUDA) relaxing some elements of the task description (Pearl, 1984). In the last years, researchers are analyzing other ways to define heuristic functions, like abstractions.

The Best First Search (BFS) algorithms used for heuristic search expand the state with lowest heuristic value according to the function $f(s) = g(s) + h(s)$, where $g(s)$ is the cost of the current path from the initial state to s and $h(s)$ is the heuristic value of s. The most common algorithms used for heuristic search are Greedy Best-First Search (GBFS) (Pearl, 1984) and A* (Hart et al., 1968). A* can be modified to speed up the search by weighting the heuristic function with a factor $w > 1$ (Pohl, 1970), thus using the selection function $f(s) = g(s) + w * h(s)$. This new algorithm, called weighted A*, searches more greedily the larger w is.

One of the first planning systems which used domain-independent heuristics was the Heuristic Search Planner (HSP) (Bonet and Geffner, 2001). This planning system uses a weighted A* guided by the heuristic function h^{add} which approximates the distance between two states by summing the distance between the propositions in the states. This function is based on a delete-relaxation of the original problem (a version of the original planning task in which all the delete effects of all the actions $a \in A$ are ignored). Fast Forward (FF) (Hoffmann and Nebel, 2001) is a forward-chaining planning system which implements a domain-independent

heuristic similar to HSP but in this case an explicit solution of the relaxed task is used to compute the distance between two states by means of a Planning Graph. Besides, FF introduces some new techniques: (1) the Enforced Hill Climbing algorithm (EHC), a local search algorithm that performs a variation of a Breadth-First Search algorithm (BFS) which searches exhaustively until a node with better heuristic value than the last best found node is discovered; (2) helpful actions, which are actions extracted from the relaxed plan used to compute the heuristic function that increase the greediness of the search algorithm. FF starts searching for a plan using EHC and helpful actions. If EHC fails, FF automatically switches to a weighted best-first search algorithm.

Finally, Fast Downward (FD) (Helmert, 2006) is a forward-chaining heuristic planning system similar to HSP and FF, but this planner uses a multi-valued representation of the planning task (Helmert, 2009). FD implements different domain-independent heuristic functions which can be combined during search such as: (1) Landmark counting heuristic (Richter and Westphal, 2010) which consists in counting unachieved propositions that must be true in every solution of a planning task (Porteous et al., 2001); or (2) Merge and Shrink (M&S) (Helmert et al., 2007) which generates abstractions about the structure of the planning task.

2.2.4 Other Algorithms

As described in the previous section, classical planning tasks are commonly formalized as a directed graph. This graph describes the search space which can be huge depending on the information used to encoding the planning task increasing the complexity of finding a solution. Simultaneously to the evolution of search graph algorithms, other types of algorithms were developed using different ways of formalization.

2.2.4.1 Planning as Satisfiability

Planning as satisfiability is one of the most powerful approaches to Automated Planning (Kautz and Selman, 1992). The different approaches based on Boolean Satisfiability Task (SAT) translate a planning task into SAT. A SAT task is an *NP*-Complete problem which consists on determining whether a set of truth values can be assigned to the variables of a boolean formula so that is it satisfied.

The first SAT solvers translate a STRIPS task and a horizon in a propositional logical formula whose satisfiability is checked sequentially one at a time for horizon lengths $(1, 2, 3, \ldots, n)$ until a plan is found (Kautz and Selman, 1992). These

approaches try to solve the SAT task using a solver that checks the boolean formula with horizon length n starting with $n = 1$. If the formula is found satisfiable, the solver returns a solution and terminates. If the formula is found unsatisfiable, a new solving process is starting with a higher horizon value. In the next years, SAT algorithms were extended by using different encoding schemes (Rintanen et al., 2006; Tompkins et al., 2011; Cai et al., 2015) in order to improve the performance of SAT solvers.

2.3 Probabilistic Planning Task

In the previous section we described classical planning where full information about the environment is available and actions are deterministic. These assumptions are extremely useful to decrease the complexity of the planning tasks but they prevent to model more realistic domains. Probabilistic planning is an extension of non-deterministic planning with information on the probabilities of non-deterministic events. This planning paradigm tries to find a plan that transforms the initial state into a goal state with full or partial information about the state of the environment and actions with probabilistic effects. In stochastic environments, actions can generate different outcomes, so it is not possible to find a plan of actions that assure reaching the goals. For this reason, probabilistic planning systems must reason about the likelihood of the actions' outcomes. This fact increases the complexity of the planning process, so planners must generate plans which maximize the probability to reach the goals and minimize the probability to generate unexpected states. The most common way of solving probabilistic planning tasks (Bonet and Geffner, 2005) consists in representing the planning task as an optimization problem using a Markov Decision Process (MDP) (Puterman, 1994), Partially Observable MDPs (POMDPs) (Cassandra et al., 1994) and factored MDPs (Boutilier et al., 2001). Probabilistic planning is based on the following three ideas:

- A planning domain is modeled as a stochastic system, where action outcomes are modeled as a probability distribution function.
- Goals are represented as an utility function, numeric function or a set of goals.
- Solutions are represented as policies that specify the action to execute in each state or as conditional sequences of actions.

2.3.1 The Conceptual Model

The conceptual model for a planning task defines the formalization of the environment, the actions that can be used to change the environment, and how actions are applied in the environment to achieve the goals. Probabilistic Planning tasks are frequently encoded as probabilistic transition graphs.

The conceptual model for a Probabilistic Planning task is a stochastic, finite and fully observable state-transition model, where each transition has associated a probability.

Definition 5. (Probabilistic Planning Model) A Probabilistic Planning task can be defined as a 6-tuple $\Pi = (S, A, T, s_0, G)$, where:

- S is a finite set of states, which is composed of all states that can be reached.
- A is a set of actions whose effects follow a probabilistic model.
- $T(s'|s, a)$, is the probability that the action $a \in A$ executed in state $s \in S$ results in a state $s' \in S$. This means that for each state $s \in S$, if there exists an action $a \in A$ and a state $s' \in S$ such that $T(s'|s, a) \neq 0$, then $\sum_{i'} T(s_i|s, a) = 1$.
- $s_0 \in S$ is a finite set of literals which encode the initial state.
- $G \subseteq S$ is a finite set of goal states.

The solution plan of a probabilistic planning task π is a sequence of actions $\pi = \{a_0, a_1, \ldots, a_{n-1}\} \forall\, a_i \in A$ or a policy $p : S \rightarrow A$ such as $p(s') = a$ where the best action is chosen according to the current state.

2.3.2 The Modeling Language

The modeling language is a notation that allows a syntactic representation of the planning task. Different languages have been defined in Automated Planning to represent the conceptual model of a probabilistic planning task, and some of then are summarized next.

2.3.2.1 PPDDL

The Probabilistic Planning Domain Definition Language (PPDDL) (Younes and Littman, 2004) is a modeling language that extends PDDL 2.1 allowing to describe planning tasks in stochastic environments. This language was created for the first IPC with a non-deterministic track (Younes et al., 2005) and supports actions with probabilistic effects, probabilistic literals in the initial state and markovian rewards.

Probabilistic effects allow us to define a set of possible outcomes of an action. Equation 1 shows the structure of probabilistic effects, where $\forall e_i \; \exists \; p_i$, this means that effect e_1 occurs with a probability p_i.

$$(\text{probabilistic } p_1 \; e_1 \; p_2 \; e_2 \; \dots \; p_{n-1} \; e_{n-1} \; p_n \; e_n)$$

This means that for each action, it is mandatory that $0 \leqslant p_i \leqslant 1$ and $\sum_{i=0}^{n} p_i \leqslant 1$. If the sum of the probabilities is lower than 1, it is assumed that there is a probability $P = 1 - \sum_{i=0}^{n} p_i$ that the action (DUDA). This representation of the actions increases the capabilities of PDDL allowing probabilistic effects on the actions, but it also increases the complexity of the actions. According to the results of Littman (Littman et al., 1998), PPDDL, after grounding, is equivalent to Dynamic Bayesian Networks (DBN) (Dean and Kanazawa, 1989), which is also another common representation of the probabilistic planning tasks. Figure 8 shows an example of a probabilistic action in the Rovers domain. In this action, the rover *?x* will navigate to waypoint *?z* from waypoint *?y* with 0.8 probability and it will not navigate with a 0.2 probability.

```
(:action navigate
 :parameters (?x - rover ?y - waypoint ?z - waypoint)
 :precondition (and (can_traverse ?x ?y ?z)
                    (available ?x)
                    (at ?x ?y)
                    (visible ?y ?z))
 :effect (probabilistic 0.8 (and (not (at ?x ?y)) (at ?x ?z))))
```

Figure 8: PPDDL definition for action *navigate* from Rovers domain.

Probabilistic literals allow us to generate different initial states for a similar planning task. A set of literals can be defined using a probabilistic relation in the initial state.

Markovian rewards are encoded using fluents (numerical variables in PDDL) and are associated with actions. PPDDL reserves a special fluent called **reward** to represent

```
(:init
       (probabilistic 0.5 (at r1 w11)
                      0.4 (at r1 w12)
                      0.1 (at r1 w10)))
```

Figure 9: PPDDL definition for probabilistic literals in the initial state for the Rovers task shown in Figure 1.

the total accumulated reward of the planning task. This special fluent is initialized to zero and cannot be used as part of the preconditions or the effects of actions. These restrictions on the use of the reward fluent allow a probabilistic planner to handle domains with rewards, without implementing full support for fluents.

2.3.2.2 RDDL

RDDL (Relational Dynamic influence Diagram Language) (Sanner, 2011) is the representation language of the uncertainty tracks of the IPC 7 (Coles et al., 2012). This language was created in order to introduce some features which are difficult to formalize using PPDDL. RDDL can be defined as a Dynamic Bayes Network (DBN) (with potentially many intermediate layers) extended with an influence diagram (Howard and Matheson, 1984). This allows for the efficient description of Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) by representing everything (state-fluents, observations, actions) with variables. A probabilistic planning task using the RDDL language can be defined as a 7-tuple $\Pi = (C, S, A, P, R, I, O)$ where:

- C is a finite set of constant variables.
- S is a finite set of state variables.
- A is a finite set of action variables.
- P is a finite set of functions. Each one defines the conditional probability function for each next state variable in terms of the previous state variable and action.
- R is a reward function.
- $I \subseteq S$ is a finite set of state variables which are initialized with a value.
- O is a finite set of objectives (goals).

```
domain rovers {

types {rock-sample: object;}

pvariables {
        //Constants fluents
        ROCK_XPOS(rock-sample): {non-fluent, real, default = 0.0 };
        ROCK_YPOS(rock-sample): {non-fluent, real, default = 0.0 };
        ROCK_VALUE(rock-sample): {non-fluent, real, default = 1.0 };
        //State fluents
        robot-at(xpos, ypos) : {state-fluent, bool, default = false};
        //Action fluents
        moveX: {action-fluent, real, default = 0.0 };
        moveY: {action-fluent, real, default = 0.0 };
        takeRock: {action-fluent,  bool, default = false}
};

cpfs {
        xPos' = xPos + xMove + Normal(0.0, MOVE\_VARIANCE\_MULT*xMove);
        yPos' = yPos + yMove + Normal(0.0, MOVE\_VARIANCE\_MULT*yMove);
        .
        .
        .
}

reward = if (takeRock) then 1.0 else 0.0;

state-action-constraints {
        takeRock => ((xMove == 0.0) ^ (yMove == 0.0));
};
};
```

Figure 10: RDDL codification of the Rovers domain.

Figure 10 shows a simplified example of the Rovers domain on RDDL. In this example variables and actions are encoded in the parameterized variable section (pvariables). In this case three types of fluents have been defined: (1) non-fluents which are static variables of the environment; (2) state-fluents which are dynamic variables of the environment and (3) action-fluents that describe the actions which can be executed. More types of fluents can be defined in this section. Next, the conditional probability function for each state variable is defined in the corresponding section (`cpfs`). In this example, conditional probability functions are encoded to variables `xpos` and `ypos` which are used to define the location of the rover. Finally, the reward function is defined, as well as the constraints that are applied to actions. In this case, a constraint over the action `takeRock` is defined, such that the rover cannot execute action `xMove` and `yMove` while action `takeRock`

is executing. We refer the reader to the Language Description document (Sanner, 2011) for further details on the RDDL syntax.

2.3.3 The Algorithms

There are several techniques used to generate a plan in Probabilistic Planning: (1) extending classic planners to handle probabilistic effects; (2) finding policies (mappings between world states and the preferred action to be executed to achieve the goals) that optimize a utility function, which gives preference to the different states and transitions of the MDP; and (3) compiling the planning task into another representation for which there are effective algorithms.

2.3.3.1 Approaches based on extending classical planning

The first approaches tried to solve probabilistic planning tasks extending some techniques developed to solve deterministic planning tasks by including probabilistic effects. The BURIDAN planning system (Kushmerick et al., 1995) is considered the first probabilistic planning system. This planner uses a probability distribution over some possible world states to model imperfect information about the initial state and actions. This planning system generates a sequential plan where the probability of the plan achieving the goals is greater than a user-supplied probability threshold. Other approaches modified the Planning Graph structure introducing probabilistic outcomes in the actions. Paragraph (Little and Thiébaux, 2006) is a probabilistic planning system based on Graphplan. This planning system modifies the structure of the Planning Graph introducing a new layer composed to the action's outcomes.

Handling probabilistic effects increases the complexity of the directed graph which is used to represent the search space. Besides, the planning system must have full knowledge about the dynamics of the environment which is not commonly possible in real world scenarios. Some approaches try to solve probabilistic planning tasks generating sequential plans which can be modified during execution according to the information which is available about the environment.

On one hand, plan repair consists on adapting the previous plan to the new context. LPG-ADAPT (Fox et al., 2006) is a stochastic planning system which conducts a local search algorithm into a search space which is represented as an action graph of partial plans. This planning system can be used to generate a new plan from scratch or repair a previous one. The repairing process uses the previous plan to build the initial graph structure and introduces some modifications according to the information obtained from the environment. ERRT-PLAN (Borrajo and Veloso,

2012) is a stochastic planning system which guides the search by using a previous plan. On the other hand, replanning approaches generate a new plan from scratch according to the new information obtained from the environment. FF-Replan (Yoon et al., 2007) translates the probabilistic domain into a deterministic domain. There are two ways of performing the translation: (1) generating a new deterministic domain where action effects are composed of the outcome with the higher probability of the probabilistic version of the action or (2) generating a new deterministic domain splitting each probabilistic action on a set of them, one per outcome. Next, the planning system generates a plan using FF and executes the actions in the environment. During execution, if an unexpected state is reached the planner FF generates a new plan from scratch using the deterministic domain generated previously and the current state of the environment.

2.3.3.2 Approaches based on finding policies

The most common technique consists on generating a set of policies by solving a task which is modeled as a MDP which describes the environment with full information and stochastic actions. A MDP can be characterized by a state space S, an action space A, a state transition function $T(s_{i+1}|s_i, a)$ which defines the probability to move from state $s_i \in S$ to $s_{i+1} \in S$ executing the action $a \in A$, and a reward function r(s) which specify the immediate utility of being in state s. The goal consists on finding an optimal policy that solves the task maximizing the expected reward.

This representation model presents some important disadvantages: (1) they need accurate action models for non-deterministic environments which is extremely complex and sometimes impossible to generate (Bresina et al., 2005); and (2) the complexity of these models grow polynomially according to the size of the state space. This state space explosion problem limits the use of the approaches based on fully observable MDP models. Commonly, MDPs are solved by means of dynamic programming algorithm such as policy iteration or value iteration, but in the last years heuristic search algorithms have shown to be effective to solve MDPs.

The first approaches used value iteration algorithms (Bellman, 1957), which are also called backward induction algorithms. These algorithms define a randomly selected cost for each state $c_n(s_n)$ on the MDP and refine the value of each state finding an action that minimizes the expected cost. Value iteration algorithms are composed of two phases: (1) a value determination phase, in which the expected cost of each state is calculated and (2) a value refinement phase, in which the algorithm finds an action that minimizes the cost of a state and stores it in the policy. These algorithms

require enumeration of the state space which increases the computational overhead according to the size of the task to solve.

Boutilier (Boutilier et al., 2001) developed an algorithm that transforms a probabilistic planning task into a First-Order MDP (FOMDP) performing a value iteration algorithm without explicit enumeration of either the state or action spaces of the MDP. First-order approximate linear programming (FOALP) (Sanner and Boutilier, 2005) translates a probabilistic planning task into a FOMDP and approximates its value function using First-Order Approximate Linear Programming. This algorithm approximates value functions by representing them as a linear combination of first-order basic functions using a first-order generalization of approximate linear programming techniques for propositional MDPs.

Wang (Wang et al., 2007) transforms the probabilistic planning task into First Order BDDs and applies a variation of the algorithm developed by Boutilier. Stochastic Enforced Hill-climbing (SEH) (Wu et al., 2011) generalizes the enforced hill-climbing algorithm to stochastic domains. This planner builds a breadth-first local MDP around the current state and searches a policy that expects to solve this MDP with a better valued state. When a policy is found, the method executes the policy until the local MDP exits. The policy is computed using the value iteration algorithm over the local MDP, where the rewards are defined as the heuristic value assigned when execution finishes.

Other approaches use policy iteration algorithms (Howard, 1960) which generate a randomly selected initial policy and refines it repeatedly. Commonly, the algorithm alternates between two phases: (1) an evaluation phase, in which the cost of the actual policy is computed and (2) a policy refinement phase, in which the actual policy is refined to a new policy with a smaller expected cost. This kind of algorithms has been rarely extended to solve planning tasks.

Finally, domain-independent heuristics have shown to be effective solving deterministic planning tasks. Other works have extended search algorithms from heuristic search to solve MDPs. The first probabilistic planning system which uses a heuristic search algorithm to solve MDPs is LAO* (Hansen and Zilberstein, 2001) which is an extension of AO* algorithm. LAO* generates an optimal policy performing a graph search over the MDPs without evaluating the entire state space. mGPT (Bonet and Geffner, 2005) is a probabilistic planning system based on heuristic search algorithms for solving MDP models. This planning system combines heuristic search algorithms with methods for obtaining lower bounds from deterministic relaxations. Learning Depth-First Search (LDFS) (Bonet and Geffner, 2006) combines dynamic programming and heuristic search. LDFS searches

for solutions by combining iterative, bounded depth-first searches, with learning. RFF (Teichteil-königsbuch et al., 2008) generates an off-line policy combining classical planning and simulation. This planner compiles the probabilistic problem into a deterministic one solving it by means of the deterministic planner FF. Next, RFF uses Monte-Carlo simulation to estimate the probability of failure of the action step. If this probability exceeds a threshold at a given step, RFF computes a new plan for overcoming the failures of this step and starts a new Monte-Carlo simulation for the new plan.

2.3.3.3 Approaches based on translating to another representation

Some approaches compile the probabilistic planning task into another representation to apply algorithms which can solve the task in the new representation. The most common compilations of the probabilistic planning task are:

- Deterministic Planning Task compilation: The simplest solution consists on compiling the probabilistic planning task into a deterministic planning task. These approaches transform actions from the probabilistic representation to a deterministic representation and solve the planning task using a deterministic algorithm. This compilation generates a deterministic action for each outcome of each probabilistic action, where the cost of the deterministic action is defined by the probability associated with the outcome used to generate them (Jimenez et al., 2006; Kalyanam and Givan, 2008).
- Planning as boolean satisfiability (SAT): MAXSAT (Majercik and Littman, 1998) is the first planner that transforms a planning task into an E-MAJSAT problem which is a probabilistic version of a SAT problem. E-MAJSAT compilation is similar to a traditional SAT compilation except for the actions' encoding. Each action effect is encoded with a clause consisting of random propositions which are true with a given probability value. After compilation, the solver determines all possible satisfying assignments of the variables. For each satisfying assignment, it is computed the product of probabilities associated to the satisfied clause. The planner finds the assignment of truth values that produces the highest product of the probabilities of satisfied clauses.
- Constraint Satisfaction Problem (CSP) compilation: CSPs are mathematical problems which are defined by a set of variables and a set of constraints. Each variable is defined by a set of possible values. Each constraint involves a subset of variables specifying the allowable combinations of values for that

subset. A solution of a CSP consists of a complete assignment of all variables that satisfies all the constraints. The Probabilistic Planning Task is compiled into a state-variable representation. This representation is solved using a CSP algorithm and the result generated by the algorithm is encoded to PDDL (Hyafil and Bacchus, 2004).

2.3.4 Other Algorithms

As described in the previous section, probabilistic planning tasks are commonly formalized as a MDP or a POMDP in order to generate a policy that solves the planning task. These techniques represent the state space as a directed graph where the transitions between the states follow a probability distribution which describes the dynamic of the environment. This means that the size of the state space can be huge depending of the information used to encode the planning task and the dynamics of the environment must be known a priori. In the last years, other types of algorithms have been developed to solve problems in stochastic domains without information about the dynamics of the environment.

2.3.4.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a domain independent search approach (Brügmann, 1993) which consists on finding decisions in a full or partial observable environment by taking random samples in the search space building a search tree according to the most promising results. The main idea of this algorithm is to use the first iterations in order to create statistics that guiding the next iterations to the most promising parts of the search space. There are two main algorithms for node selection used to implement MCTS: (1) Upper Confidence Bound (UCB) algorithm (Auer et al., 2002); and (2) Upper Confidence Bound for Trees (UCT) algorithm (Kocsis and Szepesvári, 2006). MTCS has been used in probabilistic planning (Keller and Eyerich, 2012).

2.4 Conformant Planning Task

Conformant Planning tries to find a plan that transforms the initial state into a goal state in a non-deterministic environment and without any sensing capabilities during plan execution. This means that actions' effects may be non-deterministic (Goldman and Boddy, 1996), exogenous events are possible and the initial state can be partly known. Conformant Planning can be modeled as belief state-transition system where a belief state is the set of all states that are possible.

For this reason, a conformant planning task can be formulated as a path-finding problem in belief space BS where a sequence of actions that map a given initial belief state into a goal belief state (Bonet and Geffner, 2000a). The use of this representation increases the complexity of conformant planning generating two main problems: the representation of belief states in a compact way; and the generation of effective heuristic functions over the belief space state.

2.4.1 The Conceptual Model

The conceptual model for a Conformant Planning task is a stochastic, finite and partial observable belief state-transition model.

Definition 6. (Conformant Planning Model) A Conformant Planning task can be defined as a 5-tuple $\Pi = (S, A, F, b_0, G)$, where:

- S is a finite set of states. S is used to build the belief state set BS.
- A is a finite set of grounded deterministic actions derived from the action schemes of the domain. Each action $a_i \in A$ can be defined as a tuple $a_i = (Pre, Eff)$. $Pre(a_i)$ are the preconditions of the actions, and $Eff(a_i)$ is a list of non-deterministic effects.
- T is the state transition function for non deterministic actions $T : S \times A \rightarrow 2^S : (s, a) \mapsto (s_1, \ldots, s_n)$, where n is the number of non-deterministic effects.
- $b_0 \subseteq BS$ is the initial non empty belief state.
- $G \subseteq BS$ is a finite set of belief states that contains the goal states.

According to this conceptual model, an action a is applicable in the belief state $b = (s_0, \ldots, s_m) \forall s_i \in S$ if $T(s_i, a)$ gives at least one target state for any $s_i \in b$. Therefore, applying an action a in a given belief state b_i results in the successor belief state b_{i+1} defined when a is applicable in every state $s \in b$.

A solution plan π for a conformant planning task Π is an ordered set of actions (commonly, a sequence) $\pi = (a_0, \ldots, a_n)$, $\forall a_i \in A$ corresponding to a sequence of belief-states $(b_0, b_1, \ldots, b_n)$, $\forall b_i \in BS$ such that b_{i+1} is the result of executing the action a_i in the belief state b_i. Finally, the complexity of Conformant Planning is increased when action outcomes are probabilistic. In this case, it is called Conformant Probabilistic Planning.

2.4.2 The Representation Language

Conformant planning tasks can be described using an extension of PDDL 2.1 which has been introduced in the first IPC with a non-deterministic track (Younes et al., 2005). This extended language supports non-deterministic effects, probabilistic literals in the initial state, literal disjunctions in the initial state and markovian rewards:

- Disjunctions in the initial state by using the **oneof** statement for expressing different initial states **in the same** planning task.

  $$(\text{oneof } l_1, l_2, \ldots, l_n)$$

  The **oneof** statement describes a set of literals which cannot be true in the same state. This statement is used to define different initial states for Contingent planning tasks.

- Mutual exclusion for literals by means of the **or** statement for expressing the mutual exclusion between two literals.

  $$(\text{or } (\text{not } l_1) \ (\text{not } l_2))$$

  The **or** statement describes the mutual exclusion of literals in the initial state. This means that **only one** of the two literals defined in the expression must be true in each possible initial state.

2.4.3 The Algorithms

There are several techniques used to generate a plan in Conformant Planning: (1) extending deterministic planning to handle representations based in belief states; (2) generating search algorithms for belief-states space; and (3) compiling the conformant planning task into other representations.

2.4.3.1 Approaches based on extending deterministic planning

The first approach attempts to solve Conformant Planning tasks extending the ideas applied over classical planners allowing multiple initial states. Conformant-Graphplan (CGP) (Smith and Weld, 1998) is a planning system based on Graphplan-based planner that generates sound (non-contingent) plans. This

planning system builds one different plan graph for each possible initial state and searches a solution in all graphs simultaneously. This process is composed of two phases: a graph expansion phase in which separate plan graphs (worlds) are generated for each possible state of the environment and one for each possible non-deterministic outcome; and a solution extraction phase in which a valid plan in all possible worlds is generated considering the mutual exclusion relations between the different graphs.

2.4.3.2 Approaches based on searching in the belief-states space

These approaches search a solution plan explicitly searching in the belief-state space. The first approach was introduced by Bonet and Geffer (Bonet and Geffner, 2000b) which solves a Contingent Planning task as a problem of heuristic search in a belief-states space using standard search algorithms such as A^*. However, the size of the space of belief-states could be extremely large depending of the complexity of the task and this approach usually fails to scale up. (CPA) (Son et al., 2005) implements a depth-first search algorithm in the spaces of partial states instead of the space of belief states. Partial states are a set of fluent literals. Conformant-FF (Hoffmann and Brafman, 2006) extends the FF planner into Conformant Planning performing a forward search into the belief-states space. Belief states are defined by the sets of known and negatively known propositions by using a Conjunctive Normal Form (CNF). The heuristic function is a variation of the relaxed planning method of FF with an approximate linear-time reasoning about known propositions using a 2-CNF projection of the formula that captures the true belief state semantics.

2.4.3.3 Approaches based on compiling to another representation

The most common approaches consist on compiling the Conformant Planning task into another representation in which there are efficient algorithms to solve the task.

- Conformant Planning task can be translated into a Deterministic Planning task using the compilation $\mathsf{K}(\mathsf{P})$ (Palacios and Geffner, 2006). The translation $\mathsf{k}(\mathsf{p})$ generates a non-deterministic, fully observable planning task by including new literals. After, the $\mathsf{K}_i(\mathsf{P})$ (Palacios and Geffner, 2007) extends the theoretical and practical limitations of $\mathsf{K}(\mathsf{P})$.
- Conformant Planning tasks can be compiled into SAT following two different approaches:

  - Generate and Test Strategy: This technique consists on checking the existence of plans of length n with incomplete information about the initial state. SAT tasks can be described using different languages: (1) Quantified Boolean Formulas (QBFs) is a propositional representation that extends propositional logic where variables can be quantified over either existentially, or universally. This representation has been used by the QBFPLAN planner (Rintanen, 1999); and (2) Action language ζ is based on the causal theories (McCain and Turner, 1997). This language is an extension of propositional logic for expressing causal knowledge which is represented by inference rules. This representation has been used by the ζ-PLAN solver (Castellini et al., 2003).
  - Transformed Task Strategy: The technique consists on generate and test the existence of conformant plans using a single SAT call over a transformed task. This transformed task is generated by projecting the original theory over the action variables. This strategy has been used by the COMPILE-PROJECT-SAT (Palacios and Geffner, 2005)

- Symbolic Model checking (SMC) (Burch et al., 1990) is a formal verification technique which permits the automatic verification of systems modeled in a specific language for describing finite state systems by means of Binary Decision Diagrams (BDDs) (Bryant, 1992). A BDD is a directed acyclic graph representing a boolean function. Terminal nodes are either True or False and non-terminal nodes are associated with a boolean variable, and two BDDs, called left and right branches. Conformant Model Based Planner (CMBP) (Cimatti and Roveri, 2000) is a conformant planner implementing SMC.

2.5 Contingent Planning Task

Contingent Planning is an extension to Conformant Planning including sensing actions. When a planning system is solving a real-world task, it is not possible to have complete knowledge about the state of the environment, but it is possible to sense some information relevant to solve the task. Contingent Planning tries to find a conditional plan of actions that transforms the initial state on a goal state in a deterministic or stochastic and partial-observable environment. This means that the state of the environment is not fully known, but it is possible to collect information about the state of the environment during the execution. These features imply an important change in the structure of the solution plan regarding the other planning paradigms. In the other paradigms, plans are given by a sequence of actions, while in Contingent Planning plans are trees of actions branching on observations.

2.5.1 The Conceptual Model

The conceptual model for a Contingent Planning task is a stochastic, finite and full or partial observable belief state-transition model. This model is similar to Classical or Probabilistic models including two new features: (1) sensing actions to capture information about the environment after action execution; and (2) disjunctive effects to model sensing actions.

Definition 7. (Contingent Planning Model) A contingent planning task can be defined as a 7-tuple $\Pi = (S, A, \delta, O, \sigma, b_0, G)$, where:

- S is a finite set of states which are used to build the belief state set BS.
- A is a finite set of grounded deterministic or probabilistic actions derived from the action schemes of the domain. Each action $a_i \in A$ can be defined as tuple $a_i = (Pre, Eff)$. $Pre(a_i)$ are the preconditions of the action, and $Eff(a_i)$ is a set of conditional effects. A conditional effect $e(a_i)$ is a triple $e(a_i) = (con(e), add(e), del(e))$ corresponding to the effect's condition, add, and delete lists respectively.
- T is the state transition function for a non-deterministic action, and it is a map $T : S \times A \to S : (s, a) \mapsto (e_1, \ldots, e_n)$, where n is the number of non-deterministic effects.
- O is a finite set of grounded sensing actions (observations) which generate the possible observed states after applying the sensing action o in the state b. Sensing actions can generate different outcomes. But in this case, each effect is not defined by a probability. They are defined by conditions that depend of the observations of the state. This kind of actions introduces a fork into the plan when they are applied assuming binary observations, one branch marked with $obs(o_i)$ and another branch marked with $\neg obs(o_i)$.
- σ is an observation function $\sigma : S \to O$ which associates to each state a possible observation.
- $b_0 \subseteq SB$ is a finite set of states which define the different initial states.
- $G \subseteq SB$ is a finite set of goal states.

In Contingent Planning the information known about the initial state might not be complete and the action effects might not be predictable. But, the system has the

ability to observe some aspects of the current state using observation actions. This means that a solution plan is composed to two different types of actions which are interleaved: (1) actions; and (2) observations.

- An action $a \in A$ is applicable in the belief state $b = (s_0, \ldots, s_m) \forall s_i \in S$ if $T(s_i, a)$ gives at least one target state for any $s_i \in b$. Therefore, applying an action a in a given belief state b_i results in the successor belief-state b_{i+1} when a is applicable in at least one state $s \in b_i$.
- An observation $o \in A$ can be applied in every belief state. Observations are performed to remove or add states to the belief-state. The application of an observation o in a belief-state b_i generates a belief-state $b_{i+1} = s' \in S | s' = o(s), s \in b$.

A solution plan π for a contingent planning task Π is an action-observation tree where the nodes of the tree correspond to observations and actions of the current state of the environment and the leafs of the tree correspond to the possible belief goal states. Finally, the complexity of Contingent Planning is increased when action outcomes are probabilistic. In this case is called Contingent Probabilistic Planning.

2.5.2 The Representation Language

There is not a standard representation language for Contingent Planning. Commonly, each contingent planner defines its own representation language, so different languages have been defined to represent the conceptual model of Contingent Planning.

2.5.2.1 PDDL

An extension of PDDL 2.1 was proposed (Bonet and Geffner, 2000b) which includes sensing actions. Sensing actions are planning actions with effects which generate some information observed from the current state. Besides, this language supports non-deterministic effects, probabilistic literals in the initial state, literal disjunctions in the initial state and markovian rewards.

2.5.2.2 NPDDL

The Non-deterministic Planning Domain Definition Language (NPDDL) (Bertoli et al., 2003) is a representation language that extends PDDL 2.1 allowing users to

describe planning tasks in non-determinism environments. This language supports some features:

- Incomplete information in the initial states is characterized by describing the set of possible initial states using the *unknown* and the *oneof* statements (Younes et al., 2005).
- Non-deterministic actions. These actions are characterized by introducing several possible outcomes using the *unknown* and the *oneof* statements.
- Partial Observability is expressed by introducing observable variables and observations. An observable variable is a variable whose value is observed continuously during planning. An observation is a sensing action over a variable ν which is characterized by a boolean formula over ν and the domain of values of ν.

2.5.3 The Algorithms

There are different techniques used to generate a plan in Contingent Planning: (1) extending deterministic planning for dealing with contingencies; and (2) compiling the contingent planning task into another representation.

2.5.3.1 Approaches based on extending deterministic planning

The first approach attempts to solve contingent planning tasks extending the ideas applied over classical planning allowing sensing actions. The Conditional Non-Linear Planner (CNLP) (Peot and Smith, 1992) is a conditional version of the Systematic Nonlinear Planner (SNLP) (McAllester and Rosenblitt, 1991) that includes sensing actions. This planning system represents uncertain information about some literals using the special predicate **unknown**. Sensing actions are used to know if unknown literals are true or false during execution when the lack of information can prevent it to achieve the goals.

Other approaches like Sensory GRAPHPLAN (SGP) (**W**eld et al., 1998) extends the planning graph structure to handle sensing actions. This planning system builds an individual planning graph for each possible state of the environment maintaining the original structure based on propositions and actions layers . Observation actions are represented as actions without preconditions and sensing effects. This kind of actions corresponds to primitive observations that return information about one literal defining if the literal is true or false after the execution of the observation action.

2.5.3.2 Approaches based on translating to another representation

Another way to solve a contingent planning task consists on compiling the task into another form of problem solving. The most common method translates a contingent planning task into a fully-observable non-deterministic planning task transforming the sensing actions into a set of non-deterministic actions (Albore et al., 2007).

2.6 Discussion

A planning task can be modeled as a search problem in a directed graph, where nodes represent the different states of the search space and edges represent the actions which define the transitions between the different states. States are represented as a set of variables (logic or numerical), and actions are represented as operations that change the variables' value, in terms of pre-conditions and post-conditions. Different planning paradigms (classical, probabilistic and non-deterministic) can be defined depending on how this information is modeled and extracted from the environment.

On one hand, classical planning systems are deterministic and full information on the initial state is assumed. This means that the state of the environment is always perfectly known (full observability) and the action execution always yields to the expected state (determinism). These unrealistic assumptions were defined in order to decrease the complexity of finding a solution of a planning task. The first planning systems such as STRIPS (Fikes and Nilsson, 1971) performed an exhaustive depth-first search algorithm with any guidance which just allow them to solve simple linear tasks. In the next years, some important contributions such as the planning graph framework (Blum and Furst, 1995), heuristic search (Bonet and Geffner, 2001; Hoffmann and Nebel, 2001) and the different heuristic functions (Helmert et al., 2007; Richter and Westphal, 2008) increased the powerful of classical planning systems. However, these assumption do not hold anymore in an dynamic and stochastic environment where the system has to deal with incomplete information, because the world is partially observable and non-deterministic.

On the other hand, probabilistic, conformant and contigent planning systems are non-deterministic and partial information about the environment, including the initial state, is assumed. This means that the state of the environment is partially known and the effects of the actions are not deterministic generating different outcomes. Several approaches have been developed including sensing actions (Albore et al., 2007) and non-deterministic effects with incomplete information about the initial state (Palacios and Geffner, 2007; Wu et al., 2011).

However, these approaches require full information about the dynamics of the environment which is commonly unknown or cannot be easily modeled.

In general, if we are trying to solve a planning task in a stochastic and dynamic environment (real-world environment) it is not possible to capture full information about the environment to generate a perfect plan of actions. There are a huge number of contingences which are unknown a priori and might be detected during the execution of the plan. This implies changes in the actions model and/or the information about the environment which in turn prevents the execution of the rest of the plan.

3

ABSTRACTIONS IN AUTOMATED PLANNING

The ability of **abstraction** is one of the most important features of the human brain related to the capabilities of perception, conceptualization and reasoning. Human reasoning processes performs simplifications about the environment in order to solve complex tasks which arise from the interaction with the environment. The different approaches to abstraction, developed in AI, commonly consider an abstraction as a relation between a complex task, which is represented in a specific formalism, and a simple task and its own representation. This simpler task is generated using a bidirectional mapping function that maps the complex task from the original space of representation to an abstract space which is smaller than the original. In this thesis, we focus in deploying abstractions over AP in order to simplify the structure of the planning task decreasing the complexity of the search process.

In this chapter, we review the state of the art in **abstractions** for domain-independent planning from two perspectives. The first one is based on building abstractions which change the structure of the planning task; and the second one is based on building abstraction heuristics to guide search algorithms. In both cases, we explain the different theoretical definitions of abstraction and the different methods developed to build them.

3.1 Introduction

The first planning systems (Newell and Simon, 1972) performed goal regression in the state-space without any understanding about what parts of the state-space were more promising to find a solution plan that reaches the goals. But, the process of finding a path in a directed graph can be huge in terms of time and computation resources depending on both the structure and the size of the graph.

A technique which has shown to be effective diminishing the complexity of planning tasks is to use abstractions in order to help focus the search in the most promising parts of the search-space. In this context, an abstraction can be defined as a surjective function that transforms a planning task into another **simpler** one, where some details about the structure of the state-space of the task are ignored

or removed. As described below, the state-space of a planning task is represented as a directed graph whose nodes represent the states, and whose edges represent the transitions made possible by each action. This representation offers some opportunities to deploy abstractions in order to simplify the structure of the state-space: (1) joining some states; and/or (2) simplifying the structure of the actions. Then, if an abstraction is applied over the state-space of a planning task, a new search-space is generated. This new search-space is commonly called abstract space and is a simplification of the original state-space.

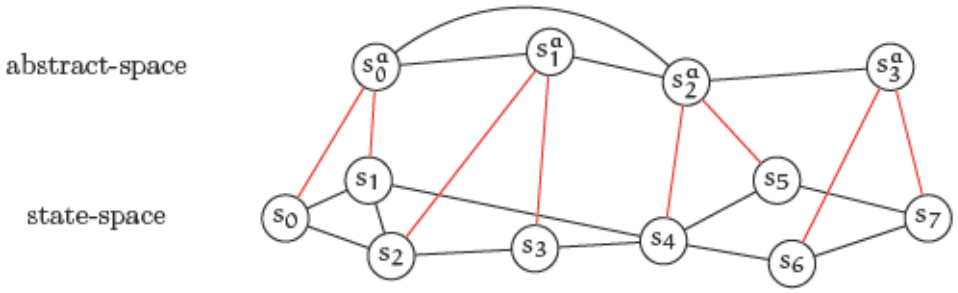


Figure 11: Example of an abstraction applied over a simple planning task.

Figure 11 shows the state-space of a simple planning task and the abstract space generated after an abstraction over the structure of the states has been applied. In this example the original state-space is composed to 8 states and 10 actions, which are reduced to 4 states and 3 actions. In this example, the states s_0 and s_1 are joined to create a new abstract state called s_0^a. In this case, the abstraction not only reduces the number of states of the state-space, it also decreases the number of original actions which can be applied over the new abstract states. Then, it is possible to build two types of abstractions over Automated Planning:

- Abstractions over the states: This kind of abstraction builds a new state-space composed of abstract states. The new states are built by compositions of the states of the original state-space.
- Abstractions over the actions: This kind of abstraction includes some changes in the original state-space. These changes consists on new abstract actions which modify the structure of the graph by changing the connections between the states.

Abstractions on Automated Planning are usually done by first solving a planning task defined in an abstract space and then using the abstract solution to guide the

search process solving the original task. In order for an abstraction to be useful, the abstract task should be easier to solve and the total time spent should be less than without using the abstraction. This could be considered the most important requirement to build good abstractions, nevertheless it has turned out very difficult to achieve in practice.

Multiple abstraction-based algorithms have been defined depending on two important aspects: (1) what kind of transformations must be applied to the original planning task and (2) how we can use the abstract planning task or the abstract solution to solve the original task. These algorithms have been used successfully in two different ways: (1) decreasing the complexity of the search process modifying the structure of the state-space of the planning task to solve it incrementally; or (2) generating domain-independent heuristics which are used to guide the search process in the original state-space.

3.2 Abstractions over the state-space

As described below, the state-space of a planning task is represented as a directed graph. In this context, an abstraction over the state-space applies a transformation function that generates a simpler version of the original state-space (abstract space). Commonly, the abstract space is built using abstractions over the actions where some aspects about actions' structure is relaxed. Then, the set of actions used to built the abstract space is composed of both original actions and abstract actions. This kind of abstraction changes the structure and the size of the state space: (1) including new transitions between states which are generated by new abstract actions; and (2) pruning some parts of the state-space which are not achieved using the new set of actions and/or combining some states generating new ones.

The most common techniques based on applying abstractions over the state-space generates a hierarchical representation of different state-spaces where the ground level corresponds to the original state-space and the rest of the levels of the hierarchy correspond to abstract spaces which commonly are sorted in descending order of size or complexity. This representation is used to solve the planning task starting with the most abstract space in the hierarchy and solving it, and then the abstract solution is refined through successively more detailed abstract levels until the original task is solved. This technique has been successfully used in different planning systems, some of them are described next.

3.2.1 General Problem Solving

General Problem Solver (GPS) (Newell and Simon, 1972) is considered one of the first planning systems. This planner implements a hierarchical algorithm to solve planning tasks using an abstractions of the state-space. This planner receives as input a planning task and an abstraction of the state-space. Then, GPS maps the original task into the abstract space generating an abstract planning task which is solved. The solution of the abstract task is used to guide the search process in the original state-space to generate a solution. This planner provided the first automated use of abstraction over the state-space, but the abstraction was built manually by an expert.

3.2.2 Abstractions for STRIPS

ABstraction for STRIPS (ABSTRIPS) (Sacerdoti, 1972) was a planning system that uses abstractions modifying the structure of the search space. This approach extended the work of Newell and Simon on GPS (Simon and Newell, 1969) combining abstractions with STRIPS (Fikes and Nilsson, 1971) to generate a hierarchy of abstractions decreasing the complexity of solving the planning task.

ABSTRIPS introduced the concept of abstraction spaces, which are generated by removing predicates of the preconditions from the actions of the original planning task. An abstraction space is a reduced version space of the original task space in which a single abstract state corresponds to one or more states in the original task space. Several levels of abstractions (abstraction spaces) are generated forming an abstraction hierarchy. Abstractions levels are defined by assigning criticalities to predicates. Criticalities are natural numbers that indicate how difficult is to achieve a specific literal and are used to define the order in which literals are removed to build the different abstraction spaces.

This approach is composed of two phases: the first phase defines the structure of the hierarchy of abstractions and how they are built. A predetermined (partial) ordering of all the predicates that describe the domain is defined manually. This set of predicates determine the order in which the literals of the preconditions are analyzed by the algorithm that determines the criticalities of each predicate. This algorithm is composed of two steps: (1) the first step assigns a criticality of two plus the maximum value in the partial order to each literal which cannot be changed by any operator. This means that the operator is static. (2) the second step analyzes each literal according to the partial ordering defined initially. If a short plan could be computed to achieve the literal from a state in which all previously analyzed literals were assumed to be true, then the criticality of the literal is assigned equal to its rank

in the partial ordering. But, if no short plan is found, the criticality of the literal is assigned greater than the highest rank in the partial order. The second phase uses the abstractions hierarchy to generate a solution. First, an abstract plan is found that satisfies only the preconditions of the operators with the highest criticality values. The abstract plan is then refined by considering the preconditions at the next level of criticality and inserting steps into the hierarchical problem solving using abstract spaces until the original problem is solved in the ground level.

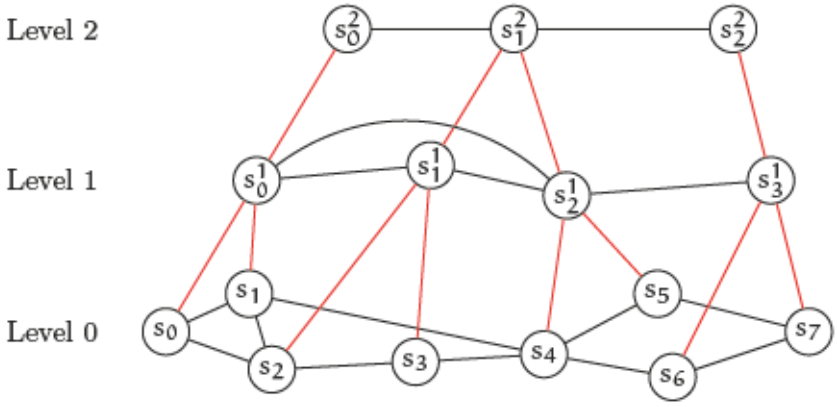


Figure 12: Hierarchical representation of the state space.

Figure 12 shows an example of a planning task where the abstraction spaces are arranged in a hierarchy. The ground level corresponds to the structure of the original task and the rest levels correspond with different level of abstraction where the last level represents the most abstract state space. In this example, the state-space is reduced in each level of the hierarchy. The states of the state-space are indirectly combined between them generating abstract states according to the predicates removed from the preconditions of the actions of the previous level. For instance, states s_2 and s_3 are combined generating the abstract state s_1^1 in the Level 1. This state is combined again with the state s_2^1 generating a the abstract state s_1^2 in the Level 2.

3.2.3 ALPINE

Alpine (Knoblock et al., 1991) is a system that builds automatically abstractions based on the interactions between literals, which are part of the preconditions of the operators. The generation process uses the operators and, optionally, the goals of the planning task and it produces an ordered abstraction hierarchy. The generation process partitions the literals of the domain in different classes and orders them according to the interactions among them. Literals which are defined in one level

cannot interact with the literals in higher abstract levels. This order set is used to generate an ordered set of abstraction spaces, where the highest level in the hierarchy is the most abstract one and the lowest level is the most detailed one. This hierarchy is used in the hierarchical version of Prodigy (Carbonell et al., 1991).

There are some important differences between the abstractions generated by Alpine and ABSTRIPS: (1) Alpine generates abstractions hierarchies using only the initial task definition, while ABSTRIPS requires an initial order of all predicates to form the abstractions; (2) Alpine generates abstractions for a specific problem, and ABSTRIPS builds a single abstractions hierarchy for an entire domain; and (3) Alpine generates an abstraction of the original task space for each level, while ABSTRIPS generates a global relaxed model.

3.3 Abstractions over the heuristic function

Heuristics functions are a way of ranking a set of nodes in order of define their quality and choose what the best successor is to explore the state-space of a planning task. They are modeled as a function h that returns a number for each node of the state-space, which is used to estimate the distance from a state s to a goal state g. Heuristic functions are usually obtained by solving a relaxed or abstract version of the original task, which is a simpler than the original one relaxing some elements of the task description (Pearl, 1984).

3.3.1 Delete relaxation

The FF heuristic (Hoffmann and Nebel, 2001), h_{FF}, is a domain independent heuristic function derived as the cost of the plan of a relaxed problem. The planning problem relaxation consists on ignoring the delete list of all actions and extracting a solution using a Graphplan-style algorithm (Blum and Furst, 1995). The number of actions in the relaxed solution is used as a goal distance estimate. The relaxation can be considered as an abstraction. The process of creating a graph of the search space where delete lists are ignored for each action is a simplification of the original problem decreasing the complexity of the plan generation process.

3.3.2 Pattern Databases

A Pattern Database (PDB) is a set of patterns, where each pattern is a pair. The first component is a partial (abstract) specification of a state of the problem and the second component is the cost of solving the abstract problem derived from the

original one using as initial state the partial state. These were originally defined to improve the search efficiency of the A* algorithm decreasing the number of expanded nodes in the sliding-tiles puzzle (Culberson and Schaeffer, 1998). After, PDBs were used in AP (Edelkamp, 2001) to store the cost of solving abstract tasks derived from the original planning task. In this case, patterns are composed of abstract state and the cost of solving an abstract task starting from the abstract state. This cost is a lower bound on the corresponding cost in the state space of the original planning task.

PDBs are formally defined in AP using a Multi-Valued Formalization (Definition 4) where abstractions are built by means of a projection of the original planning task over a subset of variables.

Definition 8. (Projection (Helmert et al., 2007)) A projection of the planning task Π over a set of variables $\upsilon \in V$ is defined by restricting the initial state, goals and preconditions/effects of the operators to υ. In other words, a projection is an abstraction α, so that two states s_1 and s_2 are equivalent if and only if they are agree on the value of variables in υ, i. e. $s_1 \sim^{\alpha} s_2$ if and only if $s_1[v] = s_2[v] \forall v \in \upsilon$.

3.3.3 Merge-and-Shrink

The Merge-and-Shrink (M&S) is a technique that generates abstraction spaces that are directly associated with the variables of the planning task. M&S was originally proposed in the context of Model Checking (Dräger et al., 2006; Dräger et al., 2009) and it was adapted to AP (Helmert et al., 2007). The definition of the variables is based on the Multi-Valued Formalization (Bäckström and Nebel, 1993), where each variable is defined as a multi-valued state variable. The abstract state space is built incrementally, starting with a set of atomic abstractions associated with each individual variable and merging two abstractions (replacing them with their synchronized product) and shrinking them (aggregating pairs of states in one). M&S is a generalization of PDBs, since for any PDB is possible to build an equivalent M&S abstraction by merging the projections that correspond to variables in the pattern. Variables which are not in the pattern are shrunk to a single abstract state, so that the abstraction does not distinguish their value, just as PDBs. This heuristic function has been successfully used in different planning systems (Reyna et al., 2013; Helmert et al., 2014).

3.4 Discussion

In this chapter, we have reviewed state-of-the-art abstraction heuristics for domain-independent planning from two different perspectives. The first one is related with approaches that manipulate the structure of the search space in order to speed up search. Some of these approaches, like ABstraction for STRIPS (ABSTRIPS) and Alpine, show that is possible to solve planning tasks decomposing them in different abstract levels of details where each level is more detailed than the previous one. The complexity of solving an abstract planning task is commonly less than the original one if the abstraction is consistent with a set of properties discovered by Knoblock (Knoblock et al., 1991). Then, it is possible to build detailed plans incrementally by including new partial plan between the actions of the previous abstract plan. But, it is important to analyze what information can be abstracted. Choosing an incorrect abstraction selection can increase the complexity of the planning process.

The second perspective is related with approaches that use abstractions to build more effective heuristic functions which are used to guide search in order to decrease the computational overhead. Merge-and-shrink (M&S), which is a generalization of PDBs, shown that is possible to generate flexible heuristic functions by means of abstractions. These heuristic functions have shown their capacity to guide the search in order to solve planning task optimally, but the time needed to build the abstraction is prohibitively large according to the complexity of the task.

In general, abstractions offer a real opportunity to decrease the complexity of planning which is an important issue in real-world environments when the planning time to generate a solution is small and the environment can change due to external agents. In the following chapters we will make use of abstractions in combination with classical planning to decrease the computational effort of solving planning task for dynamic and stochastic environments.

4

REAL-TIME SEARCH

The process of building effective autonomous systems that interact with the real-world is a cornerstone of AI. Autonomous systems must consider a range of issues including: (1) what information can be obtained about the environment; (2) how the system deliberates about what action to perform; (3) how the system goes about executing its strategy; and (4) how much time is allowed to deliberate. All of these key issues must be considered to build interactive autonomous systems which must adapt to unexpected changes in the environment, and to be flexible enough to react nimbly and modify the previous actions when it must.

Commonly, autonomous systems do not have time enough to compute a complete solution before performing an action in real-world scenarios due to the complexity of the problem. Real-world environments are characterized for needing quick responses to offer a real time interaction between the autonomous system and the environment. These time restrictions are an important drawback for Automated Planning where the complexity of solving a real-world problem using planning that encodes full information about the contingencies of the environment is EXPSPACE-complete (Littman et al., 1998). In the last years, several approaches based on heuristic search have been developed to generate plans of actions using upper-bounds (time, expanded nodes, depth, etc) which limit search and act in real-world scenarios.

In this chapter, we review the state of the art in Real-Time Search from two perspectives. The first one is based on Incremental Search Heuristic techniques which conduct incremental local search increasing the complexity of the search space explored; and the second one is based on Real-Time Heuristic Search which conducts local search where the heuristic function is updated according to the information obtained in previous searches.

4.1 Introduction

The ability of generating a solution plan in a reasonable time in real-world scenarios is one of the most important facets that an autonomous system must display.

Autonomous systems have to adapt their actions (i.e. plans) continuously according to changes in the world or changes of their action models of the world. In these cases, the original set of actions might no longer apply or might no longer be good for the new state. Then, a new set of actions must be generated. But, in these cases, the autonomous system has to choose its actions in a limited amount of time using partial information which is related to the current state of the environment. For instance, an autonomous system (i.e. a robot) cannot spend much time generating a plan when is interacting with a human that is looking for information about his flight which is close to taking off. Different approaches have been developed to speed up the cost of generating a plan using Incremental Heuristic Search (Pemberton and Korf, 1994) or Real-Time Heuristic Search (Korf, 1990; Ishida and Korf, 1991). These approaches are commonly called Agent-Centered Search algorithms (Koenig, 1996), because an agent conducts a local search over a small part of the search space which is close to the current state of the agent.

As we described in the Chapter 1 of this dissertation, two deliberative perspectives based on search can be defined to solve real-world problems. On one hand, **off-line planning** systems focus on defining methods that solve a one-time search problem. These systems generate a full plan of actions from scratch using algorithms like A* (Hart et al., 1968) and IDA* (Korf, 1985) and then actions in the plan are executed into the environment. But, if the environment changes, the plan cannot be executed. On the other hand, **on line planning** systems interleaves search and action execution in order to adapt actions to the contingencies of the environment which cannot be expected previously. Furthermore, generating effective plans according to the restrictions of real-time environments implies that search must be restricted to the areas of the search space around the current state by limiting the exploration time.

These algorithms have been used to solve real-time problems from the real-world because they can compute a partial set of actions before an entire solution is found adapting it to new information from the environment. Many of these algorithms have been used in Real-Time Strategy games (Buro, 2003; Bulitko and Lee, 2006), navigation systems (Stentz and Hebert, 1995), control in robotics (Koenig, 1996; Gutmann et al., 2005) and tactical mobile robot prototypes for urban reconnaissance (Matthies et al., 2002).

4.2 Incremental Heuristic Search

Incremental Heuristic Search methods are based on reusing information from previous searches to solve more complex search tasks potentially faster than solving each search task from scratch. Commonly, this kind of algorithms is used to

solve robot navigation problems in unknown environments where the search time is bounded to offer quick responses. Dynamic A* (D*) (Stentz, 1995a) was an evolution of the A* algorithm for Real-Time Search. This algorithm computes an initial solution from the goal state to the initial state. Then, the solution is efficiently repaired according to the changes in the cost of the arcs during execution. The repairing phase is only executed when new information is found that changes the structure of the problem. The algorithm computes an optimal solution assuming that all information captured at each execution step is correct. Besides, these approaches offer the option of distributing the computational effort between the on-line and off-line phases.

Dynamic A* introduced the concept of incremental heuristic search solving navigation problems whose complexity is increased by changes in the structure of the graph. However, Dynamic SWSF-FP (Ramalingam and Reps, 1996) is considered the first real algorithm for Incremental Heuristic Search. This algorithm computes shortest paths from a set of nodes to the goal node recomputing the paths which have only changed during execution. Lifelong Planning A* (LPA*) (Koenig et al., 2004) combines Dynamic A* and Dynamic SWSF-FP to repeatedly find shortest paths in a graph, while the edge cost or the structure of the graph change (vertices are added or deleted). This algorithm starts searching in the same way as A*. But, the subsequent searches are potentially faster, because they reuse some parts of the previous search tree which are similar to the new search tree. This algorithm interleaves search and action execution updating the previous solution according to the new information discovered during execution.

Other approaches introduced the concept of anytime search to improve the features of Incremental Heuristic Search. Anytime A* (Likhachev et al., 2004) (ARA*) computes a first solution using A* with inflated heuristics and then continues to improve the solution reusing search efforts from previous executions in such a way that the sub-optimality bounds are still satisfied. The bounds are changed progressively as time allows to compute an optimal solution for the problem. Anytime Dynamic A* (Likhachev et al., 2005) is a planning and replanning algorithm which computes bounded suboptimal solutions in a anytime fashion reusing previous search efforts. When new information about the environment is received, the algorithm incrementally repairs its previous solution. This algorithm tunes the quality of the solution based on the available search time (bound).

4.3 Real-time Heuristic Search

Real-time Search (RTS) methods are a special type of agent-centered search where search is limited by a horizon. These methods are characterized for two properties:

(i) the search time per action can be upper-bounded by a user-supplied constant (hence real-time); (ii) they associate a heuristic function with each state (hence heuristic) which is updated (or learned) in order to avoid local minima and/or dead-ends. Most of the RTS algorithms are improvements over the Real-time A* algorithm.

Real-time A* (Korf, 1990) (RTA*) is considered the first RTS algorithm. This algorithm combines A* and MiniMax algorithms where a solution is computed searching from the current state to a fixed depth determined by computational resources and/or the information available about the environment. Besides, this algorithm adapts the heuristic function to evaluate the nodes inside to perimeter defined to the search frontier. This algorithm interleaves two search phases: (1) a simulation phase which consists on a partial tree expansion (lookahead) search in a simulated environment where actions are not actually executed; and (2) an execution phase in which the best action found is executed in the real-world. After each execution phase a new simulation phase is performed using the new current state.

Learning Real-time A* (Korf, 1990) (LRTA*) was an evolution of the previous algorithm. This algorithm performs an iterative execution of the RTA* search with an updating rule to enable the algorithm to learn from previous runs. The update rule is based on changing the heuristic value of the current state only with respect to its immediate neighborhoods. A variation of the LRTA* algorithm (Ishida and Korf, 1991) was developed to avoid the local minima of the heuristic function used (heuristic depressions). This variation ran a limited A* search when a heuristic depression was detected and then used the results of the A* search to correct the depression at once.

In the following years, Shimbo and Ishida (Shimbo and Ishida, 2003) introduced variations on LRTA* for bounding the amount of state space exploration. LRTA*(k) (Hernández and Meseguer, 2005) is an algorithm based on LRTA* with a different updating strategy. This algorithm updates the heuristic value of k states per iteration following a bounded propagation strategy. Bulitko (Bulitko and Lee, 2006) developed a three-parameter framework (named LRTS) which deploys different Learning RTS algorithms tunning three parameters.

4.4 Discussion

In this chapter, we have reviewed the state-of-the-art in real-time search from two different perspectives. The first one is related with approaches that conduct an incremental heuristic local search reusing information from previous searches.

These algorithms are commonly used to solve navigation problems in real-time environments where the structure of the search space changes during execution. These algorithms have shown to be effective in real-time environments decreasing the search time, but they cannot guarantee a constant time bound on search time per action execution. Depending on the changes in the environment, this time can be huge. This is an important drawback in real-world environments with partial information where some changes can modify the complexity of the problems avoiding to generate a solution or increasing the time needed to compute one.

The second one is related with approaches that conduct a heuristic search with partial information where the search time is limited by a lookahead commonly called horizon. The horizon guarantees a constant time bound on search time per action execution. However, the quality of the solution depends of the time bound which is chosen manually. Besides some RTS approaches discover pathological cases of deeper lookahead which increase both execution and planning costs (Bulitko et al., 2003). This means that choosing a good lookahead in RTS is really important in order to obtain a balance between search time and execution cost.

In general, time or computational bounds allow RTS algorithms to handle real-world problems. This is an important feature in real-world environments where the available time to generate a solution is often short. In the following chapters, we will use lookahead in classical planning in order to generate partial plans to act in dynamic and stochastic environments.

Part II

PREDICATE ABSTRACTIONS OVER AUTOMATED PLANNING

5

PLANNING AND EXECUTION

In previous chapters, we focused mainly on the problem of plan generation describing the different planning paradigms. However, the main objective of this thesis consists of using planning as reasoning system in real-world applications like robotics, industrial applications, aerospace or video-games. These applications require control systems with situated planning capabilities, systems that interleaves planning and execution, monitoring control, updating state strategies, failure recovery, plan supervision and replanning or repairing mechanisms.

When a plan that theoretically solves a planning task is executed in the environment, some discrepancies between predicted and observed states of the environment may occur. These discrepancies can be caused by: (1) new information about the environment can be discovered during action execution modifying the structure of the planning task; (2) unanticipated exogenous actions can change the environment; (3) actions' execution which can fail generating unexpected states which in turn prevents the execution of the rest of the plan; and (4) the execution of the actions in the plan can generate states from which no plan can be successfully executed (dead-ends). Regardless of the cause, when a discrepancy is detected during execution, it brings into question whether the plan being executed remains valid or must be changed.

This chapter describes the concept of Planning and Execution, the different techniques for plan generation and validation and some of the most important architectures based on Automated Planning. Next, it presents a detailed description of the Light Planning, Execution, LEarning Architecture (LPELEA) which has been developed to perform the empirical evaluation of this thesis. Finally, it shows the different methods used to conduct the empirical evaluation of this thesis describing the domains, the metrics and the different planners.

5.1 Introduction

Planning and execution consists of generating a plan and executing it in a real-world environment. This means that the planning and execution system must handle the

complexity of the real-world to solve the planning task. First planning systems were not able to handle the complexity that implies executing a plan of actions in a real-world environment. In order to handle this kind of problems some assumptions were made to simplify the complexity of the planning process omitting the execution process. These assumptions are related to the representation of the environment, the actions and the goals.

- **Assumptions about the environment.** There are two assumptions related to the representation of the environment and how the information about it is observed. The most common and unrealistic simplification consists of assuming that the planning system has full knowledge about the state of the world. This means that the planning system has full information about the environment. However, real-world environments are complex and difficult to model. In these cases, the planning system must generate a plan using partial information which is unknown a priori or must be obtained using the available sensors. The second one consists of considering the world as static. A world is considered static when only the action execution can change the environment (Veloso et al., 1998a). But, there is a wide number of exogenous events (uncontrolled agents, people, environmental events, etc) which can change the environment when a plan is executed in a real-world environment (Nareyek and Sandholm, 2003). These unpredictable changes must be detected using monitoring mechanisms and included in the planning model to generate a new plan which considers the new information about the environment. These monitoring mechanisms are described in section 5.1.2.

- **Assumptions about the action execution.** These assumptions are related to the structure of the actions and how they are executed in the environment. The first one consists of considering that actions are deterministic. This means that the effects of the actions are always known and predictable and the execution of the action only depends on the current state. However, this is an unrealistic simplification in real-world scenarios when action execution may generate different outcomes which cannot be predicted for the planning system. This signifies that action's execution must be controlled using monitoring mechanism which can analyze the result of the action execution (Blythe, 1994). The second one considers that the execution of the action is immediate. But, this simplification can be a problem in situations where some resources are only available in certain time intervals (Coddington et al., 2001; Mausam and Weld, 2008) or the resource-usage impacts on the feasibility of the plan (Coles and Coles, 2012). In real-world scenarios the action execution time depends on different factors related to the environment which must be controlled. Actions execution must be monitored to detect failures during execution or excessive

execution time generating a new solution according to the new information sensed from the environment.

- **Assumptions about the goals.** The objective of a planning task is defined as a set of goals which must be achieved. However, planning systems do not have considerations about what happens if any of these goals cannot be achieved or if more information is needed about the environment to reach the planning task. Planning systems assume that the objectives are static and can change during the execution process. On many occasions goals can change during execution or new goals can be discovered according to the unpredictable changes of the environment (Haigh and Veloso, 1996).

Most of these assumptions have been relaxed by new planning paradigms (probabilistic, contingent, conformant, temporal, etc) which try to handle some aspects of the real-world environments. But, these paradigms increase the complexity of the planning process and also include new assumptions related to the knowledge about the environment which must be used to model realistic planning tasks. For instance, Probabilistic planning introduces stochastic effects which imply an accurate knowledge about the dynamics of the environment which in real-world domains is impossible to acquire except for small tasks (Zettlemoyer et al., 2005).

In general, many practical approaches of planning and execution are based on using classical planning combined with monitoring and execution mechanisms which remove partially some of these assumptions without increasing the complexity of the planning model. In fact, the first work that combined planning and execution in a real-world environment was based in classical planning. This approach (Fikes et al., 1972) was developed to control the robot Shakey (see Figure 13). This robot is considered the first general-purpose mobile robot to be able to reason about its own actions. It was composed of a television camera, a group of contact sensors and sonar range finders to obtain information about the environment and a drive motor to move around different rooms. The reasoning system was a basic planning and execution architecture which used STRIPS (Fikes and Nilsson, 1971) to generate a plan and PLANEX (Fikes, 1971) to validate the plan according to the information of the environment which is obtained by the sensors.

This first approach was the starting point for new control systems based on planning. These control systems can be divided in two trends (off-line and situated or on-line planning) depending of what information is known about the environment and how this information is used to generate a solution plan.

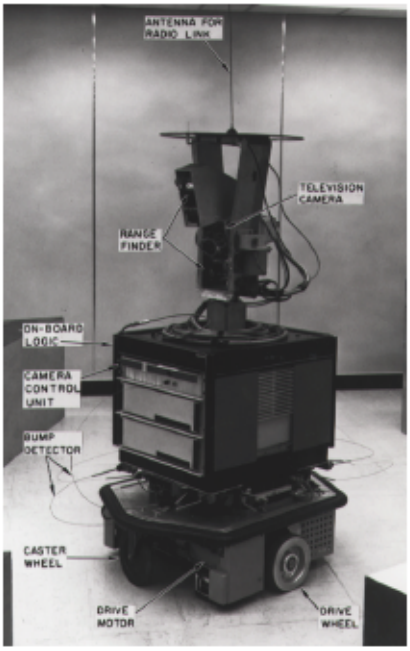


Figure 13: Shakey: the first general-purpose mobile robot (This image is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported (wikipedia)).

On one hand, if we have full information about the dynamics of the environment (accurate model of the actuators of a robot, failure probabilities of actions, accuracy models of sensors, contingencies of the environment, ...), we can define a domain model with probabilistic information (such as in PPDDL or RDDL) and/or with non-deterministic information (such as in PDDL+contingencies or NPDDL). In this case, we can proceed in different ways: (1) generating a conditional plan (Peot and Smith, 1992; Draper et al., 1994) that takes into account all possible states; (2) building a policy by solving an MDP (Hansen and Zilberstein, 2001; Bonet and Geffner, 2003); or computing a conformant plan (Bryce and Kambhampati, 2004). These planning system have been used to build control systems which are often composed of two modules: (1) a planning module which generates only one solution plan from scratch before the execution (Washington, 1995) by using a non-deterministic planning system (probabilistic, conformant, contingent, ...); and (2) an execution module which executes the solution plan in the environment. These control systems are called **off-line** planning systems, because they generate only one plan before the execution without taking into account the changes in the environment. This is an important drawback, because they need accurate information about the dynamics of the environment which is not possible in real-world scenarios.

On the other hand, if we do not have full information about the dynamics of the environment, we can define a basic domain model with deterministic actions which can include action cost and/or temporal constraints. In this case, we can generate a deterministic solution plan which is executed in the environment until an unexpected state is detected. Then, a new plan is computed including the information obtained from the environment. The control systems which follow this scheme are called **on line** planning systems, because they adapt to the contingencies of the environment generating a new solution. Most of the control systems based on this trend deploy a control architecture that include planning, monitoring, and execution. The simplest control system that can be defined to interleave planning and execution (see Figure 14) is composed of three main modules:

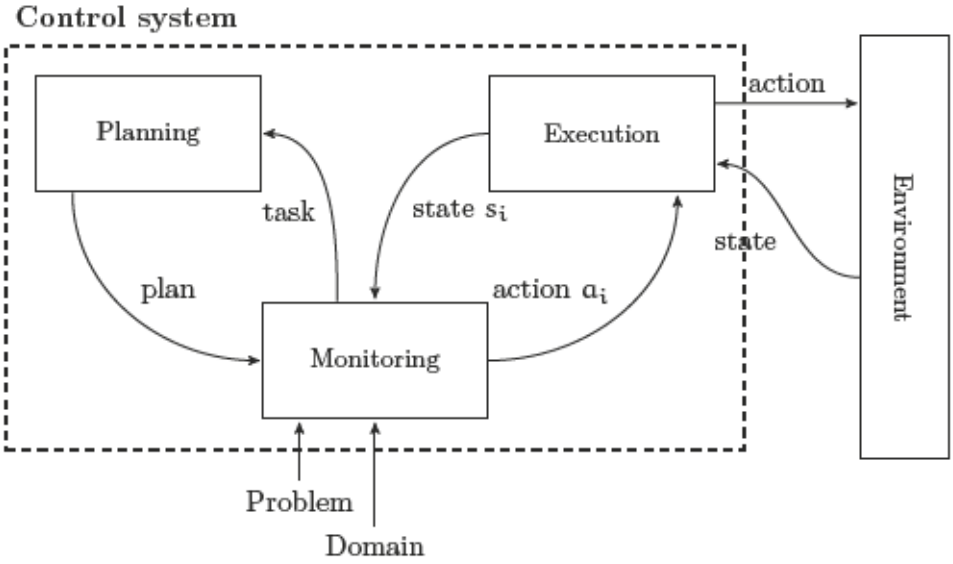


Figure 14: Example of a basic planning and execution control system

- The planning module generates a plan that solves a specific planning task. The simplest solution consists of using a planning system with a deterministic action model that is simple and incomplete. Different monitoring mechanisms are described in section 5.1.1.
- The execution module which is a sensor-actuator system. This module receives information from the environment using the sensors and executes the actions of the plan using the actuators.
- The monitoring module controls the execution process sending actions to the execution module and observing the result of each one, analyzing if an

unexpected state has been observed and deciding whether execution should proceed, or it should generate a new plan of actions. The most common monitoring mechanisms are described in section 5.1.2.

5.1.1 Plan generation strategies

In this thesis, we focus on on-line planning systems which interleave planning a execution in order to use the information obtained from the environment to adapt to the contingencies that appear during execution. This process can be performed using a plan generation mechanism (replanning, repairing, computation or plan reuse and case-base reasoning). All of these mechanisms proceed by trying to adapt the previous plan to the contingencies detected with the hope that such adaptation will decrease search time holding the plan quality. However, given an unexpected state which generates a similar problem to the previous one, the adaptation process does not guarantee neither to save planning effort or to hold plan quality (Nebel and Koehler, 1995). In spite of this, several plan generation strategies have shown to be effective decreasing planning effort in many situations:

- Replanning: This strategy generates a new plan from scratch using the new information obtained from the environment during execution (Yoon et al., 2007).
- Repairing: This strategy consists of modifying the previous plan by removing obstructing actions from the plan and/or adding actions to achieve the goals. One of the first approaches was developed in the System for Interactive Planning and Execution (SIPE-2) (**W**ilkins, 1990) where some sub-plans are removed from the original plan to replace them by better ones. More recently, the repairing system LS-ADJUST-PLAN (Gerevini and Serina, 2000) uses two local search algorithms: (1) the first one to analyze the inconsistencies of the plan; and (2) the second one to search on a sub-graph of the planning task to repair the plan according to the new state of the world. Another approach for plan repair was included in SimPlanner (Onaindia et al., 2001) which uses heuristic search to find what part of the previous plan is valid and can be used to build a new plan. In general, repairing techniques are faster that replanning techniques, but they depend on the planning system that is used to generate the first plan from scratch. If this first plan cannot be generated, the repairing system cannot be used to repair the plan during execution.
- Computation reuse: This strategy consists of using data structures computed during the first planning process. The SHERPA system (Koenig et al., 2002) can be combined with different planning systems to generate plans keeping

the quality and decreasing the planning time. This approach retains the search tree built during the previous planning episodes to determine how changes in the environment may affect the current plan.

- Plan reuse: This strategy consists of using past solutions to guide the search or to build a new plan (Borrajo and Veloso, 2012).
- Case-base Reasoning (CBR): This strategy consists of storing information about particular solutions (cases) in which problems are successfully solved. These solutions are stored in a case-base where each solution is composed of a plan, the initial state and the goals which are reached using that plan. Commonly, approaches based on CBR proceed by generating a plan using a traditional planning system and executes each action into the environment. During execution, if an unexpected state is reached a past solution is retrieved from the case-base and it is used to solve the problem or to build a more complex solution according to the information about the environment. There are many approaches that used mechanism based on CBR (Veloso, 1993; Ihrig and Kambhampati, 1997; Britanik and Marefat, 2004; Borrajo et al., 2014).

5.1.2 Monitoring strategies

A plan monitoring mechanism consists of finding differences between the real state of the environment and the expected state of the control system (Giacomo et al., 1998). There are different types of contingencies which can be detected during the execution.

- Execution failures: These contingencies are produced when action execution fails due to hardware or software failures. For instance, a failure in the actuators of a robot that make it move to a wrong location.
- Exogeneous events: These contingencies are produced when some information about the environment changes due to actions produced by external agents or new information is discovered.
- Opportunities: Changes in the environment might be produced during execution such that they do not prevent the action execution, but they can be used to improve the solution.

Different monitoring mechanisms have been developed to control the execution in dynamic and stochastic environments. On one hand, Plan Monitoring (PM) mechanisms consist on checking whether a remaining sub-plan is still executable. A

sub-plan is considered executable when the preconditions of the sub-plan actions that were not established by actions of the sub-plan, are holding in the environment. If the sub-plan cannot be executed, the monitoring module has to request a new plan from the planning module. The PLANEX system (Fikes, 1971) is considered the first monitoring mechanism. This system was developed to monitor the execution of the robot Shakey. Rational-Based monitoring (Veloso et al., 1998b) captures information both about the plan currently under execution and the alternative choices that were found but they did not purposed. This information is used to introduce changes into the execution plan according to the information which is captured. Other PM mechanisms monitor plan validity using algorithms that exploit knowledge about the environment and the different discrepancies which are detected during execution. These algorithms introduce annotations on the plan which can be exploited by the plan validation algorithm to quickly discern conditions that are relevant to a situation trying to avoid replanning episodes (Fritz and McIlraith, 2007).

On the other hand, Action Monitoring (AM) mechanisms consist on analyzing if the next action can be executed. An action is considered executable if the preconditions of the action are true when it is going to be executed. If the preconditions of the actions are not true, the plan cannot be executed and a new plan has to be generated from the current state. The Livingston system (Williams and Nayak, 1996) was developed to monitor the execution of a reactive control system. This system uses AM mechanism where the expected state is compared with the observed state. If some discrepancies are detected the system analyzes the state by searching the information of the expected state which are consistent with the observed state. Then, a new set of actions is generated using the HSTS planning and scheduling system (Muscettola, 1994). Another way to implement AM consists of building a probabilistic representation of the action model using a Partial-Observable MDP (POMDP) where some preconditions are monitored (Boutilier, 2000). But, this AM mechanism needs full information about the dynamics of the preconditions which are monitored.

Finally, there are some approaches which modify

5.2 Architectures

This section describes some previous control architectures that have influenced the implementation of the control architecture used in this thesis for the empirical evaluation.

5.2.1 Task Control Architecture

The Task Control Architecture (TCA) is one of the first deliberative architectures developed to control autonomous robots that must work in dynamic and uncertain environments (Simmons, 1992). TCA was developed to control both the Ambler six-legged walker robot designed for planetary exploration by NASA and the Xavier robot (Simmons et al., 1997). This architecture supports: (i) distributed processing of the sensors; (ii) hierarchical task decomposition; (iii) temporal synchronization of subtasks; (iv) execution and monitoring; and (v) resources management. TCA is based on a hierarchical representation of task execution called task trees. Task trees describe the hierarchical relationships between the different tasks and the temporal constraints among them.

Figure 15 shows the structure of the task control architecture for the Ambler Walking System. This architecture consists of a set of task modules and a general purpose Control Central Module (CCM). Task modules communicate among them by message passing. CCM controls the communication between modules by scheduling and executing the messages. The CCM coordinates the execution of the messages taking into account the available resources (sensors and actuators) and the temporal constraints of the tasks. Besides, this module monitors the messages and the task execution to prevent failures.

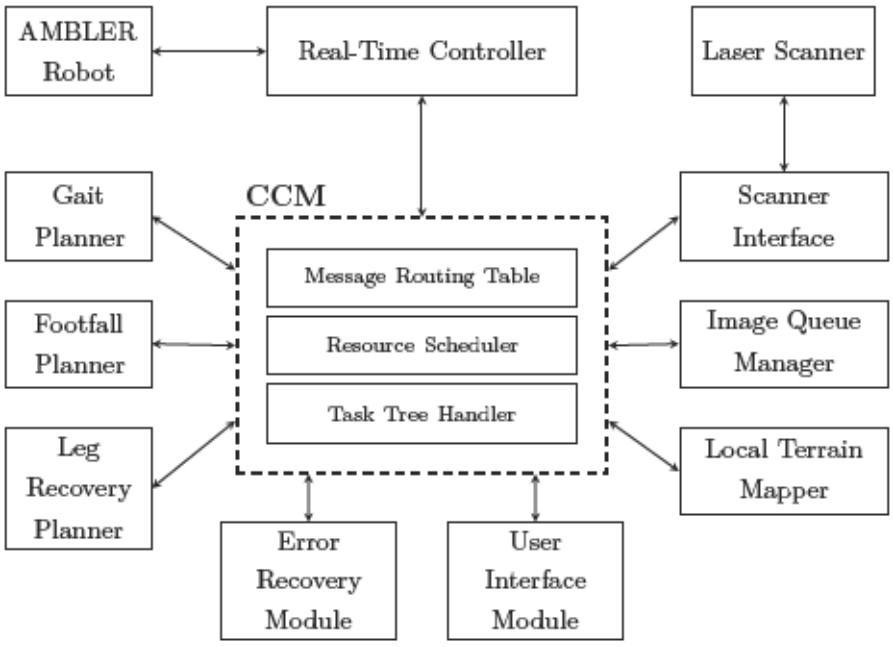


Figure 15: Task Control Architecture for Ambler Walking System.

5.2.2 LAAS Architecture for Autonomous System

LAAS Architecture for Autonomous System is a three-layered (3T) architecture (Gat, 1998) that generates tasks plans (each task is composed of a set of actions) taking into account temporal constraints (Alami et al., 1998) which are common in real-world environments. Besides, this architecture includes software tools to programming each layer. The architecture is composed of three layers:

1. The lowest layer, called function layer consists of a hierarchy of interconnected modules which implement act and perception capabilities. Modules are software entities which implement a control loop related to a given resource. Resources may be physical or logical sensors or actuators. Modules are written in GenoM language, which generates standardized templates that simplify the development of modules. Besides, this design makes each module hardware independent which means that the architecture is portable from one robot to another.

2. The intermediate layer, called executive layer, controls and coordinates the execution of the functions distributed in the modules according to the task requirements. This module is considered as a communication interface between the decision and the functional layers which selects, parameterizes and synchronizes dynamically the modules of the functional layer related to the tasks received from the decision layer. This layer is written in the Kheops language which automatically generate an automata which can be formally verified in order to check the logical and temporal properties of the tasks.

3. The highest layer, called decision layer, consists of two modules: (1) a temporal planner; (2) a supervisor based on Procedural Reasoning System (PRS) (Ingrand et al., 1996). The first version of the architecture started using the IxTeT temporal planner (Ghallab and Laruelle, 1994), but this planner was replaced by the Fape temporal planner (Dvorak et al., 2014). The PRS decomposes tasks, chooses alternative methods for achieving tasks, and monitors execution. Besides, LAAS architecture allows for multiple decision layers, such as a high-level "global mission" layer and a lower-level "task" layer.

Figure 16 shows the structure of the LAAS Architecture. This is composed of three layers and an interface to connect with sensors and actuators of the robot controlled by the architecture.

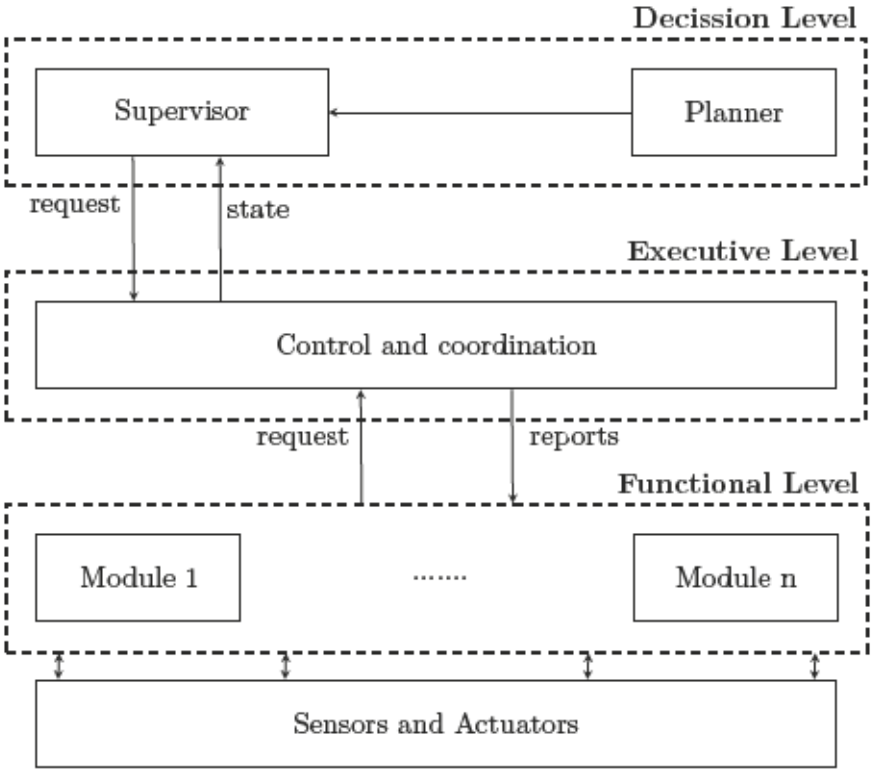


Figure 16: LAAS Architecture for Autonomous Systems.

5.2.3 Teleo-Reactive Executive

T-REX (Teleo-Reactive Executive) (McGann et al., 2008a) is an architecture based on the IDEA architecture (Finzi et al., 2004) developed to control Autonomous Underwater Vehicles (AUV) in real oceanographic scientist missions. This architecture is composed of different modules or Tele-Reactors (McGann et al., 2008b) organized in a hierarchical structure, where each reactor solves a specific task. Tele-reactors receive goals to generate their behaviour, send goals to lower level reactors, and receive observations from lower level reactors. Besides, tele-reactors can use different planning systems: (1) Europa which is a temporal constraint satisfaction based planner that encodes the planning task in NDDL; or (2) APSI-TRF which is a temporal planner that encodes the planning task in DDL (Borgo et al., 2014).

Figure 17 shows an example of the structure of the T-REX architecture. As it can be seen, in this case the architecture is composed of three modules: two reactors (mission manager and navigator) and a functional layer (vehicle control subsystem

(VCS)). In this case, VCS works as a functional layer that connects with the autonomous underwater vehicle. This architecture divides the planning task among the different reactors.

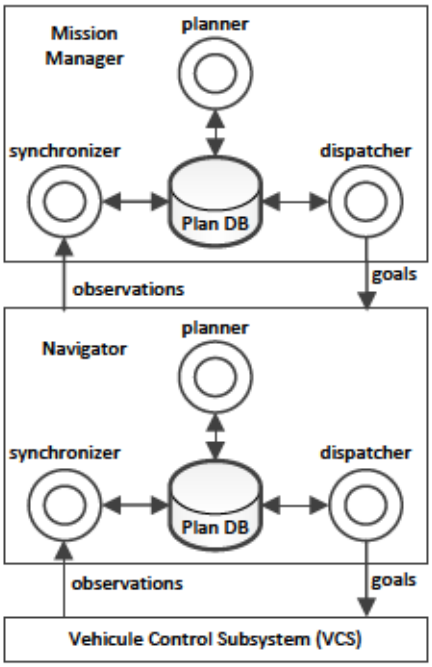


Figure 17: T-REX architecture..

5.2.4 Planning, Execution and Learning Architecture

The Planning, Execution and Learning Architecture (PELA) (Jiménez et al., 2008) is an architecture that integrates planning, execution and learning techniques. This architecture generates a plan using a planner, which can be deterministic or probabilistic; executes the plan storing the result as success, failure or dead-end; and learns the patterns for predicting these outcomes. Figure 18 shows the structure of the architecture along with the integration of the modules. As it can be seen, PELA has three components: (1) a planning module which generates a solution plan; (2) an execution module which executes the actions in a simulated environment; and (3) a learning module which stores the results of the executed actions. The planning and learning components can be exchanged for others that provide the same functionality with the same inputs and outputs.

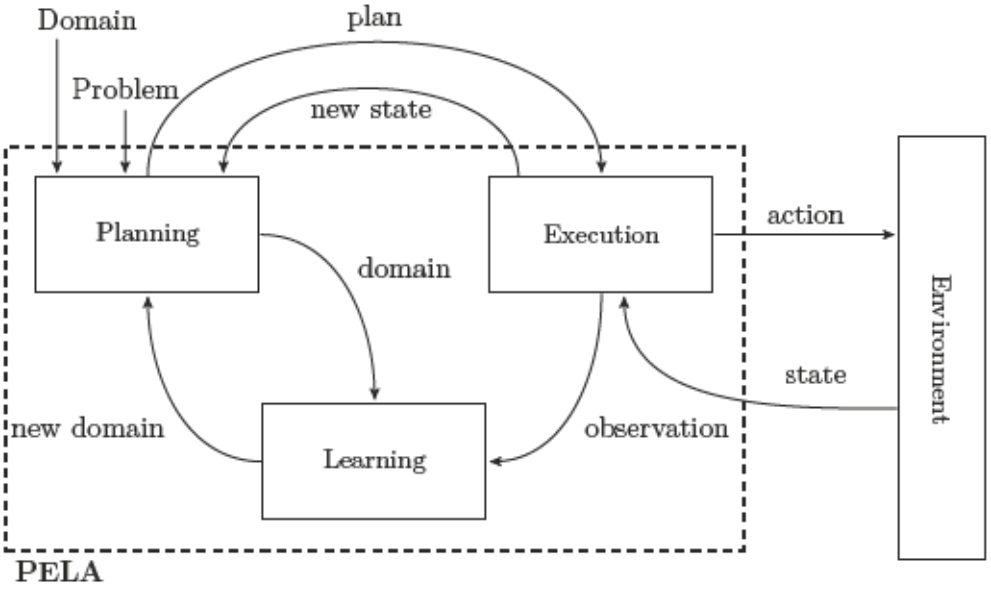


Figure 18: PELA architecture

5.2.5 Summary

In the previous sections of this chapter, we have partially reviewed the state-of-the-art of planning and execution. First, we have described the most important components of a planning and execution system which includes reactive and deliberative components. Besides, we have emphasized in the different plan generation and monitoring strategies, which in our opinion are extremely important in planning and execution. Next, we have described some of the most relevant architectures for planning and execution.

Most of the architectures described in this section are focus on the execution process offering different functionalities related to the execution of concurrent actions or tasks, task synchronization between different robots and temporal execution. Besides many of these architectures have not been developed to used planners based on PDDL. However in this dissertation, we are focus on using planning systems which use PDDL to model the environment and the action model. More precisely, we use **deterministic planning systems** due to three important factors: (1) the complexity of describing a dynamic and stochastic environment using other modeling languages like PPDDL and RDDL; (2) the complexity of generating a solution using a more complex representation which includes probabilistic and/or non-deterministic effects, sensing action, etc; and (3) PDDL has become a standard for current planning system.

According to these reasons, we have chosen the theoretical PELEA architecture (Quintero et al., 2011) which uses PDDL as the main language to model the inputs of the planning systems, the global and partial states of the environment and the execution plans. In this thesis, we have implemented a new version of the PELEA architecture, called Ligth PELEA (LPELEA) which deploys some of the modules of the architecture introducing new algorithms and data structures which are described in the next section.

5.3 Planning, Execution and LEarning Architecture

Planning, Execution and LEarning Architecture (PELEA) (Quintero et al., 2011) is a theoretical architecture that includes different modules which integrate planning, execution, monitoring, re-planning and learning techniques to solve planning tasks in dynamic and stochastic environments. This architecture has been designed in order to create a hybrid architecture which combines two of the most extended control paradigms: Deliberative and Reactive.

- The Deliberative paradigm, commonly called **high-level**, is the oldest control paradigms used in Artificial Intelligence (Fikes et al., 1972; Keiji Nagatani and Thrun, 1998). This paradigm is characterized by using a global world model which is provided by user information or sensory information. This information is used by long-term reasoning algorithms which generate a sequence of actions to reach a set of goals.
- The Reactive paradigm, commonly called **low-level**, it was first developed by Brooks (Brooks, 1986; Amir and Maynard-reid, 1999). This paradigm is characterized by using a local world model which is provided by sensory information. This information is used by simple algorithms which solve one specific task. Commonly, those algorithms are distributed in a hierarchy of interconnected modules.

The PELEA architecture has been designed to use both levels in order to combine the advantages of each one. The division in two levels allows control systems to recover from execution failures at either level. For instance, if a reactive failure is detected, it can be solved in the low-level and generating a new high level plan is not needed. Besides, the structure of the architecture can be easily adapted to the requirements of the control system. Although, the architecture must be composed of at least three modules: a monitoring module, a execution module and decision support module. Figure 19 shows the structure of the PELEA architecture along

with the integration of the modules. As it can be seen, the theoretical definition of PELEA is composed of eight modules that exchange a set of Knowledge Items (in XML) during the reasoning and execution steps. The knowledge used by the architecture is composed of the information about the environment (Action model, world model) in different levels of detail (High and Low). In this thesis, we have implemented a new version of PELEA called Light PELEA (LPELEA) which is composed by only six of the modules presented in Figure 19 (Execution, Monitoring, Decision Support, LowToHigh, Low Level Planner and Goals and Metrics), but we have only used four of them for the empirical evaluation (Execution, Monitoring, Decision Support and Goals and Metrics) [1].

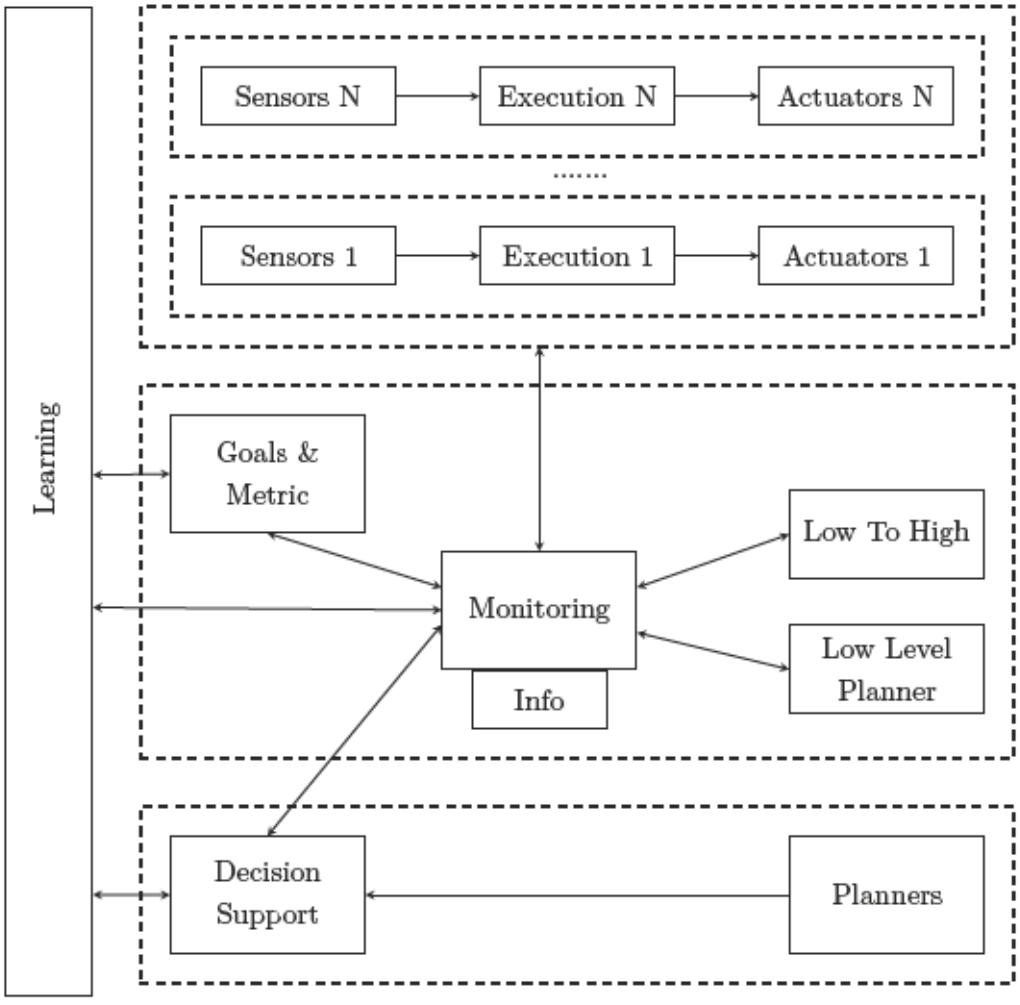


Figure 19: The PELEA architecture.

[1] The source code of the LPELEA architecture is available at https://bitbucket.org/momartin/pelea

5.3.1 Monitoring

The Monitoring module is the main component of the PELEA architecture. This module deploys the control algorithm that synchronizes communications among the other modules, supervises the execution of the actions and dispatch the actions to the corresponding execution modules. The Light PELEA architecture developed in this thesis offers two different control algorithms for monitoring:

- A control algorithm for monitoring an architecture composed of three types of modules (Monitoring, Execution and Decision support). This configuration has been defined to control systems where there is an one-to-one correspondence between the actions of high and low level.
- A control algorithm for monitoring an architecture composed of four types of modules (Monitoring, Execution, Decision support and Goals and Metrics). This configuration has been defined to control systems where there is a one-to-one correspondence between the actions of high and low level and the information obtained from the environment is used to change the planning strategy.

This module stores the global state of the world which is built using both the initial information of the environment and the sensory information obtained from the different execution modules. Besides, it analyzes the current state of the world in order to detect some discrepancies between the current state and the expected state. The expected state is generated using the previous state and the action executed.

Algorithm 5.1 shows the monitoring algorithm for control systems composed of three types of modules. This algorithm observes and analyzes action execution, splitting the parallel action set EA among the different execution modules (line 11), sending the next action to the corresponding Execution Module (line 14), and asking for a new plan to the Decision support module (line 19) if an execution failure is detected (line 6). Besides, it is responsible for checking differences between the expected state and the observed state of the environment sent by the Execution module (line 15). If an observed state is not valid (detailed on 5.3.1.1), this module starts another planning episode to generate a new plan according to the observed state (line 19). This algorithm has been defined to control the execution of different devices.

Algorithm 5.1: Pseudo-code of the main loop of the Monitoring Module for High Level

```
   input: Planning task: Π = (S, A, I, G)
   input: Horizon value: k
   Data: Execution modules : E
 1 begin
 2     Set s ← I;
 3     Set plan ← ∅;
 4     plan ← send getPlan(s, A, k) to DecisionSupport;
 5     repeat
 6         if validState(s) then
 7             if goalReached(s, G) then
 8                 return Successful Execution;
 9             else
10                 if plan ≠ ∅ then
11                     EA ← getNextAction(plan);
12                     for each action a in EA do
13                         e ← getExecutionModule(a, E);
14                         send ExecuteAction(a) To e;
15                         s ← send getState() To Execution;
16                 else
17                     Return No solution;
18         else
19             plan ← send getPlan(s, A, k) to DecisionSupport;
20     until problemSolved();
```

5.3.1.1 State comparison techniques

As we described in the previous section, after every execution of an action, the monitoring module compares the current state of the environment and the expected state to determine if the current state is valid according to the deterministic version of the domain. The expected state is generated after each action execution using the available information about the environment and the action which has been executed in the environment. The comparison process can be performed in three different ways depending on the level of similarity among the states: (1) analyzing if the expected state and the current state are equal according to definition 9; (2) analyzing if the expected state and the current state are similar according to definition 10; or (3) analyzing if the next action of the plan can be executed in the current state according to definition 11.

Definition 9. (Valid full state). Let $\Pi = (S, A, I, G)$ be a planning task, $s \in S$ is the expected state of the system and $s_c \in S$ is the current state of the environment. s_c is a full valid state if $s = s_c$.

Definition 10. (Valid partial state). Let $\Pi = (S, A, I, G)$ be a planning task, $s \in S$ is the expected state of the system and $s_c \in S$ is the current state of the environment. s_c is a valid partial state if $\forall\, p_i \in s_c\ p_i \in s$.

Definition 11. (Valid state based on next action). Let $\Pi = (S, A, I, G)$ be a planning task, $a \in A$ is the next action to be executed in the current plan and $s_c \in S$ the current state of the environment. $s_c \in S$ is a valid state for the execution of action a if $\forall p_i \in Pre(a), p_i \in s_c$.

5.3.2 Execution

The execution modules are the communication interfaces between the architecture and the environment. These modules execute actions in the environment (real or simulated) and obtain information about the environment after execution which is sent to Monitoring in order to update the global state of the environment. Depending of how the LPELEA architecture is configured, execution modules can be implemented in two different ways:

- Simulator interface: The execution module is implemented as an interface between the architecture and a simulator (MDPSim). In this case, there is only one execution module which executes all actions in the simulator.
- Robotic interface: The execution module is implemented as an interface between the architecture and the robot which must be controlled. In this case, the architecture can deploy different execution modules (1 ... N), one for each robot unit. Each execution module can be developed as a native module of the LPELEA architecture or a rosjava module (Luis et al., 2016).

5.3.3 Decision Support

The Decision Support module handles a set of planning systems which are used to generate a plan of actions. This node can be configured to use different types of planning systems, but the most common configuration consists of two planners: one for planning from scratch and another one to perform replanning or plan repair. The first planner receives as input the initial state and the domain and the second

planner receives as input the current state, the domain and the previous plan. However, it is possible to send more parameters to the planner (metrics, achieved goals, identifier of the planner etc).

5.3.4 The simulated environment

In this section, we describe the structure of the simulator used to simulate dynamic and stochastic environments. The simulator environment is composed of two components: (1) A MDP simulator which simulates the execution of the action in the environment generating different outcomes for the actions based on a probability distribution; and (2) an Error Simulator which generates additional information which is introduced to the probabilistic action as parameters in order to generate events which cannot be produced by the simulator.

5.3.4.1 MDP Simulator

The MDP Simulator (MDPSim) was developed during the first IPC probabilistic track (Younes et al., 2005). This simulator, based on an MDP, was developed to simulate the execution of PDDL actions in stochastic environments. MDPSim executes each action according to a given probabilistic action model described in PPDDL and sends back the resulting state. The PPDDL model generates execution failures, such as actions that generate effects that were not present in the original deterministic PDDL model used by the planner.

5.3.4.2 Error Simulator

MDPSim can simulate a probabilistic model to generate different outcomes for each action using PPDDL, but it is extremely difficult to generate outcomes that change some details of the environment. For instance, in the Rovers domain, in the real world a rover could move to another waypoint different to the expected one depending on the ground properties (roughness, slippery) or the lack of precision of the actuators. But, this is difficult to encode using PPDDL. Therefore, we have developed an Error Simulator that introduces additional information in the environment. This additional information is encoded as parameters into the action and used in the effects to generate different outcomes, which cannot be defined by the grammar accepted by the MDPSIM parser.

Figure 20 shows the structure of the Error Simulator module. This node is composed of two processes: (1) an action handler which chooses the error function according

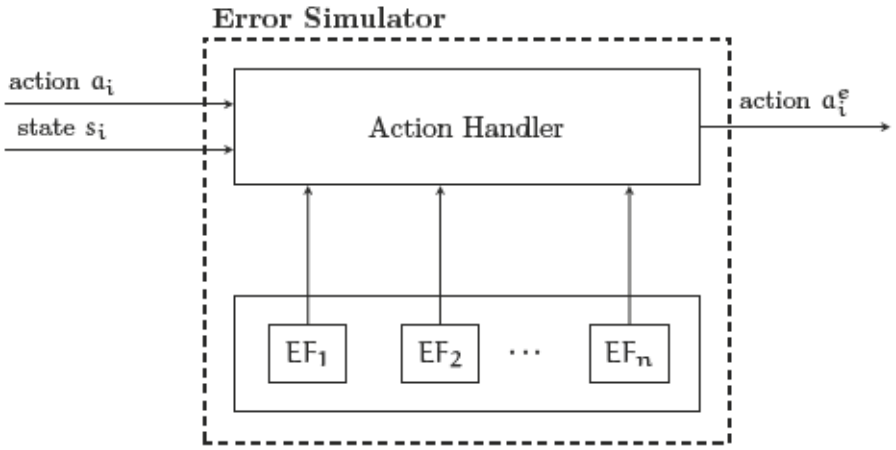


Figure 20: Error Simulator structure

to the name of the action received as input; and (2) an error function which generates some additional parameters for a specific action. If an action is received, the action handler chooses the error function associated with the action and executes it generating a new action with other parameters as output. If there is not an error function associated with the action, the Error Simulator generates the same action as output.

Algorithm 5.2: Pseudo-code of the handler function for action navigate

```
   input: Current state of the environment: s
   input: Action: a
   Output: error parameters: p = [p1, p2]
 1 begin
 2     rover ← getParameter(a, 0);
 3     currentLoc ← getParameter(a, 1);
 4     futureLoc ← getParameter(a, 2);
 5     preMovements ← getPredicates(s, "can_traverse", [rover, currentLoc]);
 6     postMovements ← getPredicates(s, "can_traverse", [rover, futureLoc]);
 7     preMovements.remove([rover, futureLoc]);
 8     postMovements.remove([rover, currentLoc]);
 9     while p ≠ ∅ do
10         le1 ← preMovements.getRandom();
11         le2 ← postMovements.getRandom();
12         if le1 ≠ le2 then
13             p ← [le1, le2];
14     return p;
```

Algorithm 5.2 shows the pseudo-code of the error function for action navigate of the Rovers domain. This error function receives two parameters as input: (1) the current state s defined as a set of predicates; and (2) the action a defined as a tuple composed of the name of the action and an ordered set of parameters according to the PDDL definition of the action. The error function generates two error locations for a rover trying to simulate actuators failures. In order to obtain the values of the parameters of the action, the function `getParameter` is applied (lines 2, 3 and 4). Then, two predicates set composed of available waypoint locations are generated: (1) the first one is composed of all waypoint locations which are accessible using the current location of the rover as start point (line 5); and (2) the second one is composed of all waypoint locations using the expected position of the rover as start point (line 6). Both sets are modified removing the expected location from postMovements and the current location from preMovements (line, 7 and 8). Finally, the loop (line 9) chooses two different locations which are included as new parameters of the actions.

```
(:action navigate
 :parameters (?x - rover ?y - waypoint ?z - waypoint
                          ?we1 - waypoint ?we2 - waypoint)
 :precondition (and (can_traverse ?x ?y ?z)
                    (available ?x)
                    (at ?x ?y)
                    (visible ?y ?z)
                    (can_traverse ?x ?y ?we1)
                    (can_traverse ?x ?z ?we2)
                    (not (equal ?z ?we1))
                    (not (equal ?y ?we2))
                    (not (equal ?we1 ?we2)))
 :effect (probabilistic 0.80 (and (not (at ?x ?y)) (at ?x ?z))
                        0.05 (and (not (at ?x ?y)) (at ?x ?we1))
                        0.05 (and (not (at ?x ?y)) (at ?x ?we2))))
```

Figure 21: PPDDL definition for action *navigate* from Rovers domain. This action include new information which is computed for the Error Simulator module

Figure 21 shows a navigate action from Rovers domain encoded in PPDDL. It represents the possible outcome of an action with stochastic effects, where the Rover will move to the desired waypoint (denoted by variable ?z) with a 0.8 probability, and two different accessible waypoints (denoted by variables ?we1 and ?we2) with a 0.05 probability. Since probabilities do not sum one, with the remaining probability (0.1), there will be no changes in the state (the rover will remain in the current waypoint, ?y). In this case, the action `navigate` includes two new parameters which

represent two waypoint locations which have been computed using the algorithm 5.2. These parameters are incorporated to the action model by the Error simulator.

5.4 Discussion

In this Chapter, we have presented the Light PELEA architecture, a new implementation of the theoretical PELEA architecture which has been implemented in order to conduct the empirical evaluation of this thesis. Most of the functionalities of the Light PELEA architecture have not been described in this dissertation, because they are not relevant to the objectives. However, this new version includes some advantages over the theoretical architecture in order to make it easier and more portable to other scenarios.

On one hand, the LPELEA architecture offers some advantages about the architecture, the communications between the different modules and the configuration of each one. Modules have been implemented as independent entities which can been replaced by other modules. New modules can be implemented extending the generic classes and coding the methods that process the generic messages which receive the modules. LPELEA allows two communications modes: (1) one based in Java Remote Method Invocation (RMI) to allow communication between the LPELEA modules; and (2) another one based in the message passing interface implemented by the Robot Operating System (ROS) in order to control different types of robots. Finally, LPELEA offers a configuration system based in XML files which allows LPELEA to introduce new properties to the modules. The basic structure of the XML files is described in Appendix B.

On the other hand, the LPELEA architecture implements a system to simulate dynamic and stochastic environments composed of two system: (1) the MDPSim simulator; and (2) the Error simulator which allows LPELEA to introduce more complex errors in the simulated environment by generating action handlers which introduce new parameters which are difficult or impossible to compute using PDDL. Besides, LPELEA introduces different monitoring algorithms, a set of different state comparison techniques and different wrappers to use different planning system.

6

VARIABLE RESOLUTION PLANNING

Humans have the ability of solving hard problems from the real world by forming abstract mental constructions (Trope and Liberman, 2010). Humans cannot experience what is not present, but our cognitive models can make predictions or speculate about the future according to the information which is available. A prediction is an abstract mental construction composed of a set of behaviours (actions) which can be conducted to solve the problem. However, behaviours of the abstract mental construction become more abstract according to a psychological distance (horizon) is increased. Psychological distance is a subjective experience that something is close or far away from the self, here, and now. Human reasoning processes construct abstract mental constructions with different levels of abstraction based of the psychological distance which is based on different metrics. For instance, an arbitrary number of basic behaviours, a maximum time, the level of veracity of the information, etc.

For an automatic system to operate effectively in a dynamic and stochastic environment, its behaviour must be flexible in the face of unexpected changes. In the context of AP, some important drawbacks can appear when a planning task must be solved in dynamic and stochastic environments: (i) new information about the environment can be discovered during action execution; (ii) actions' execution can fail; (iii) the execution of the actions can generate unexpected states from which no plan can be successfully executed (dead-ends); and (iv) plans may need to be generated quickly to offer a real time interaction between the planning systems and the environment. Planning under these conditions is an extremely hard task, and often plans that guarantee reaching the goals in spite of incomplete run-time information cannot be generated. One option in such cases is to find contingent plans or policies, but both require accurate information about the dynamics of the environment. But, getting accurate information about the dynamics of this kind of scenarios is hugely difficult. Then, the most viable option consists of interleaving planning and execution in order to handle these contingencies.

In this chapter, we present a novel approach that generates a Sequential Abstract Plan (SAP) which can be used to solve planning tasks in dynamic and stochastic

environments which are close to real-world. Unlike a sequential plan, which specifies a set of detailed actions, a SAP specifies a set of actions that provide detailed actions in the first steps of the plan. But, in later steps of the plan, it will only provide abstract actions which are computed using limited details, since the actions that are planned to be executed in the future are very unlikely to be used due to the uncertainty in plan execution. Our first approach for computing a SAP uses abstractions based on removing some predicates which are chosen manually by an expert. We also present a new planner, called AKFD, which implements VRP. We empirically compare our approach to other approaches and find that our technique decreases the complexity of solving planning tasks in dynamic and stochastic environments.

6.1 Introduction

Dynamic and stochastic environments are characterized by the presence of uncertainty in the actions and partial information about the real state of the environment. This means that the actions' execution can fail which in turn prevents the execution of the rest of the solution due to the uncertainty of the environment. This lack of information increases the complexity of building accurate action models in order to generate detailed plans. Besides, the time needed to generate a detailed plan in these complex scenarios can be prohibitively huge freezing the planning system during the execution process. In order to face these important drawbacks, we propose Variable Resolution Planning (VRP) which is a novel technique for interleaving planning and execution in stochastic and dynamic environments.

The key idea in VRP is to generate a Sequential Abstract Plan (SAP) quicker than traditional planning approaches relaxing information about the environment which is changing continuously. This idea is based on three important assumptions according to the characteristics of dynamic and stochastic environments: (1) it is not possible to obtain enough information about the uncertainty of the environment to build an accurate action model; (2) it is not useful to spend a lot of time generating a detailed plan, because the plan might not be executed fully; and (3) the process of interacting in these scenarios must be quick in order to produce a real interaction with the environment. With the purpose of facing these assumptions, the VRP technique combines predicate abstractions and lookaheads to decrease the search time in classical planning. Abstractions are used to build a relaxed version of the action model and lookaheads are used to define when this relaxed version is deployed during search.

To offer the reader an overview of the VRP's architecture, Figure 22 shows the VRP architecture main phases and how these are connected. The VRP planning system is

composed of two different phases. The first phase, called **Abstraction Generation**, builds a set of abstract actions and variables (abstract states). The second phase, called **Search Algorithm**, executes a heuristic search algorithm to compute a SAP using two sets of actions: regular actions and abstract actions. Regular actions are the set of standard actions defined in the original planning task. Abstract actions are the set of actions generated in the previous phase. The search algorithm uses the regular actions set until it reaches horizon k; then, the algorithm uses the abstract actions set until it reaches the goals. Upon termination, the search algorithm either outputs a SAP or no solution. This means that the most search effort of the planning task is devoted to compute a valid plan head of length k; and the rest of the plan is only generated by checking for potential reachability by relaxing the actions' model.

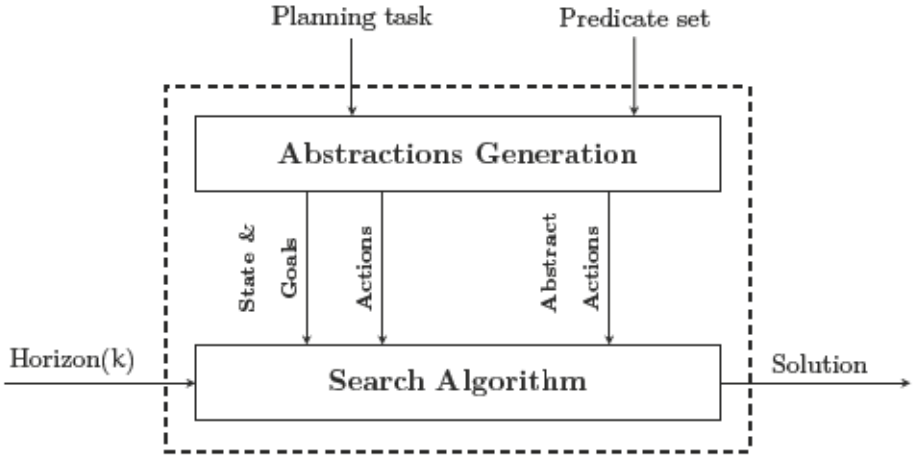


Figure 22: VRP's architecture.

6.2 Formalization of Predicate Abstractions

As we describe below, VRP is based on removing some future details about the planning task to speed up search on hard problems, which are executed in a dynamic and/or stochastic environment. From a planning perspective, the information about the environment is represented using **predicates** in FOL that describe the features or the state of the different objects or agents. For instance, in order to describe the current location of a robot, we define the predicate $(\texttt{at robot}_1\ \texttt{location}_1)$. Predicate **at** describes that the $\texttt{robot}_1$ is located in the position $\texttt{location}_1$. This information can be selectively ignored in order to simplify the state space generating an abstract representation of the planning task. This abstract representation is built by removing some predicates from the structure of propositional representation described in definition 3. This process is called **Predicate Abstraction** and

consists of removing literals from F, I and G and from the preconditions and effects of the grounded actions in A.

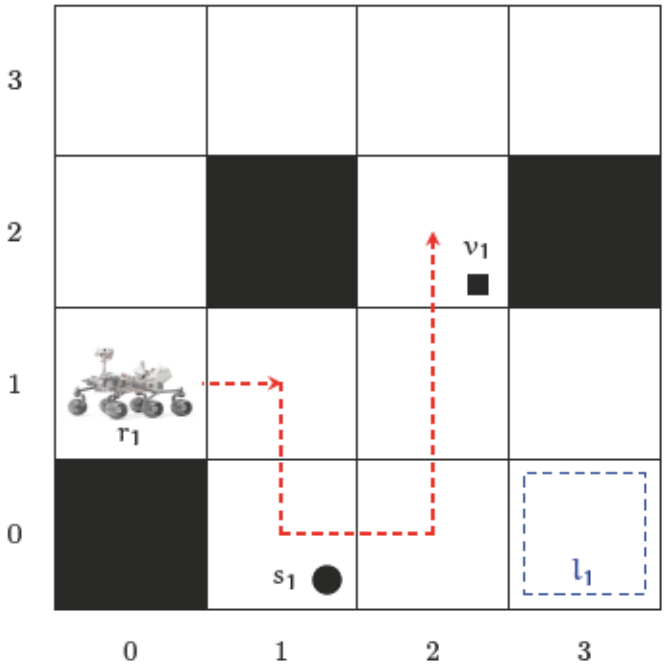


Figure 23: Simple planning task of the Rovers domain

To illustrate the concept of predicate abstraction used in our approach, a task of the Rovers domain is shown in Figure 23. The environment is represented as a grid of 16 cells, called waypoints. Each one is denoted with a bi-dimensional coordinate (x, y), starting on the left bottom cell of the grid. White cells represent waypoints in which the rover can stay (free); and black cells represent obstacles. Two types of Mars samples can be collected: rocks and soil. Rocks are denoted with a small black circle. Soils are denoted with a small black square. Rovers are located at waypoints and can move between any two free waypoints which are adjacent. Besides, rovers can take samples of rocks and soil or take images and analyze them. Finally, there is a lander base which is used by the rovers to send information about the analysis made over the samples. The typical goals of a Rovers task are to take images and soil and rock samples, analyze them and send the result to the lander base. We are going to use this example to explain different concepts related to predicate abstractions.

First, we are going to describe the concept of Grounding mapping which generates a mapping between the ungrounded (PDDL) definition of a predicate p and a set of grounded facts F.

Definition 12. (**Grounding mapping**) Let L be a set of facts, and p an ungrounded predicate. The **grounding mapping** $g(p, L)$ is the set of grounded propositions (facts) of predicate p in L.

Figure 23 describes a planning task where a rover, denoted as `r1`, can move between different waypoints ($w00, \ldots, w33$). For instance, the ungrounded (lifted) predicate $p = (\texttt{at ?x ?y})$[1] is defined to represent the current location in which the rover `r1` is located. Then, the grounding mapping of predicate p in F is composed of only the groundings of predicate `at`:

$$g(p, F) = \{(\texttt{at r1 w00}), (\texttt{at r1 w01}), \ldots, (\texttt{at r1 w33})\}$$

As we describe below, VRP performs an abstraction over some predicates. Then, we first need to define a projection over some predicates that will be the basis for the abstraction.

Definition 13. (**Projection of a set of facts over a predicate or a predicate set**). Let L be a set of grounded propositions (facts) and p a predicate. A **projection** of L over p is a function defined as:

$$\text{Proj}_p(L) = \{x \mid x \in L, x \in g(p, L)\}$$

If P is a set of ungrounded predicates, we also define the projection over L as:

$$\text{Proj}_P(L) = \{x \mid x \in L, p \in P, x \in g(p, L)\}$$

We can generalize definition 13 to any first-order logic formula by recursively applying the previous definition.

Definition 14. (**Projection of a formula over a set of predicates**) Let ϕ_1 be a formula, let ϕ_2 be a formula and P be a set of ungrounded predicates. A **projection** of ϕ_1 and ϕ_2 over P can be recursively defined as:

[1] In PDDL any symbol preceeded by a question mark denotes a variable that can be grounded with particular objects. Also, in PDDL, variables in predicates are further denoted with the appropriate object types that can substitute the variable. In this case, we would have `(at ?x - robot ?y - waypoint)`.

$$\begin{aligned}
&Proj_P(\phi_1 \wedge \phi_2) = Proj_P(\phi_1) \wedge Proj_P(\phi_2)\\
&Proj_P(\phi_1 \vee \phi_2) = Proj_P(\phi_1) \vee Proj_P(\phi_2)\\
&Proj_P(\neg\phi_1) = \neg Proj_P(\phi_1)
\end{aligned}$$

where ϕ_1 and ϕ_2 are first-order logic formulas.

A projection of some planning task Π over some predicate p removes the information of the rest of predicates in all the components of Π.

Definition 15. (Projection of a planning task over a predicate set) Let $\Pi = (F, A, I, G)$ be a planning task and let P be a set of predicates. A **projection** of the planning task Π over P is defined as:

$$Proj_P(\Pi) = (Proj_P(F), Proj_P(A), Proj_P(I), Proj_P(G))$$

where $Proj_P(A) = \{a' \mid a \in A, a = (Pre, Add, Del), a' = (Proj_P(Pre), Proj_P(Add), Proj_P(Del))\}$ and $Proj_P(F), Proj_P(I), Proj_P(G), Proj_P(Pre), Proj_P(Add)$ and $Proj_P(Del)$ are computed according to Definitions 13 and 14.

In the case of VRP, instead of projecting over a set of predicates, we will generate abstractions by projecting over the complement of a set of predicates. That is, we will remove from the planning task everything related to that set of predicates.

Definition 16. (Abstraction of a planning task over a predicate set) Let $\Pi = (F, A, I, G)$ be a planning task, P the original set of predicates of Π and P_1 a subset of predicates $P_1 \subseteq P$. An abstraction of the planning task Π over P_1 can be defined as: $\Pi^{abs}_{P_1} = Proj_{P \setminus P_1}(\Pi)$. Also, $\Pi^{abs}_{P_1} = (F^{abs}_{P_1}, A^{abs}_{P_1}, I^{abs}_{P_1}, G^{abs}_{P_1})$ where $F^{abs}_{P_1} = Proj_{P \setminus P_1}(F)$, $A^{abs}_{P_1} = Proj_{P \setminus P_1}(A)$, $I^{abs}_{P_1} = Proj_{P \setminus P_1}(I)$ and $G^{abs}_{P_1} = Proj_{P \setminus P_1}(G)$.

Let us see an example of abstracting an action from a planning task. Given the planning task Π shown in Figure 23, if $P = \{(\texttt{at ?x ?y})\}$ and a is the grounded action `navigate(r1, w03, w13)`[2], where:

[2] Moving the rover from one waypoint to another.

$$Pre(a) = \{(\texttt{can\_traverse r1 } w03\ w13),\ (\texttt{visible } w03\ w13),\ (\texttt{available r1}),\ (\texttt{at r1 } w03)\}$$

$$Add(a) = \{(\texttt{at r1 } w13)\}$$

$$Del(a) = \{(\texttt{at r1 } w03)\}$$

The abstraction of action a over P is $a^{abs} = (Pre(a^{abs}), Add(a^{abs}), Del(a^{abs}))$ where:

$$Pre(a^{abs}) = \{(\texttt{can\_traverse r1 } w03\ w13),\ (\texttt{visible } w03\ w13),\ (\texttt{available r1})\}$$

$$Add(a^{abs}) = \emptyset$$

$$Del(a^{abs}) = \emptyset$$

If we build the abstraction using the predicate **at** (**at** is removed from Π), rovers are not required to be at any location to perform any task, and therefore to move between locations. Thus, a predicate abstraction over the predicate **at** modifies the structure of the planning task changing the connections between the states of the state space. According to the previous example, if the abstraction process is applied over the navigate action, the effects (adds and deletes) of the action become empty sets and then navigate actions are pruned from the abstract action set decreasing its size. This fact also changes the structure of the state space decreasing the number of states used to built the abstract state space, because removing the predicate that defines the location of the robot merges some states of the state space.

Corollary 1. If a predicate abstraction removes all effects of an action. The action is pruned from the abstract action set.

Finally, we have to define how the abstraction affects plans.

Definition 17. (Sequential Abstract Plan (SAP)) Let $\Pi = (F, A, I, G)$ be a planning task, let P_1 be a set of predicates, let $\Pi^{abs}_{P_1} = (F^{abs}_{P_1}, A^{abs}_{P_1}, I^{abs}_{P_1}, G^{abs}_{P_1})$ be a sequential abstract planning task and let k be a number $k > 0$. A **Sequential Abstract Plan** $\pi^{abs}_{P_1}$ is an action sequence $\pi^{abs}_{P_1} = \{a_0, \ldots, a_{k-1}, a_k, \ldots, a_{n-1}\}$

divided into two sub-plans: the first one is a sound plan composed of the k first actions $\forall i = 0 \dots k-1, a_i \in A$; and the second one is composed of $n-k$ abstract actions $\forall i = k \dots n-1, a_i \in A^{abs}$.

Sequential Abstract Plans are computed in order to decrease the complexity of problem solving in Automated Planning. Actions before the horizon k are actions in the original state space, so the head of the plan is a valid one and hence, sound. After the horizon, VRP only uses abstract actions, which allows for a much faster pace of problem solving. In many cases, these abstract actions will not be executed in the environment, because some required preconditions have been removed from the definition of the action. Assuming the world is stochastic, in some cases the plan will eventually fail before even attempting to execute abstract actions. So, it is not that important from the point of view of planning and execution that the actions are not completely valid, because they will most probably not be used in the current run. When (if) the plan fails, a new plan will be computed with horizon k again, so the first abstract actions of the previous plan will be replaced by original valid actions.

Figure 24 compares the abstract plan generated from the example abstract planning task with $k = 4$ to the plan generated from the example planning task (Figure 23). Since predicate *at* is removed from horizon $k = 4$, no more navigation actions appear in the abstract plan, because they are not included in the abstract action set. According to this, the abstract plan is easier to compute than the standard plan, but it is composed of abstract actions which cannot be executed in the environment.

6.3 Properties of Predicate Abstractions

In this section, we study the different properties which can be drawn from predicate abstractions and if these properties can be used to build good abstractions detecting what predicates are good candidates to build them. A good abstraction is one that separates out some parts of the problem which can be solved first and then held invariant while other parts of the problem are solved. Most of these properties have been discovered by previous works based on generating abstractions by removing predicates.

Property 1. (**Monotonicity Property (Knoblock et al., 1990)**) The existence of a ground-level solution implies the existence of an abstract solution that can be refined into a ground solution while leaving the literals established in the abstract plan unchanged.

1	navigate(r1 w10 w11)	navigate(r1 w10 w11)
2	navigate(r1 w11 w01)	navigate(r1 w11 w01)
3	sample_soil(r1 rs1 w01)	sample_soil(r1 rs1 w01)
3	navigate(r1 w01 w02)	navigate(r1 w01 w02)
		k = 4
5	communicate_soil_data(r1 g w01 w02 w02)	communicate_soil_data(r1 g w01 w02 w02)
6	drop(r1 rs1)	drop(r1 rs1)
7	navigate(r1 w02 w12)	sample_rock(r1 rs1 w22)
8	navigate(r1 w12 w22)	communicate_rock_data(r1 g w22 w13 w03)
9	sample_rock(r1 rs1 w22)	
10	navigate(r1 w22 w12)	
11	navigate(r1 w12 w02)	
12	communicate_rock_data(r1 g w22 w13 w03)	

Figure 24: Plans generated for the Rovers task described in Figure 23. Left: sequential plan generated by a traditional classical planner. Right: sequential abstract plan generated by removing predicate at with horizon k = 4.

The property 1 defines that an abstract plan can be refined in a non-abstract plan if the grounded predicates added and deleted from the actions in the abstract plan are unchanged in the non-abstract plan. In other words, the predicates added to or removed from the abstract state space must be added to and removed from the original state space too. However, this property does not ensure the order in which the predicates are added or removed or if they are added or removed by the same actions in both the abstract plan and the non-abstract plan. Then, sometimes the refinement process can be more complex than generating a plan without using abstractions. In order to simplify the refinement process, Knoblock introduces a stronger version of the previous property.

Property 2. (Ordered Monotonicity Property (Knoblock, 1994)) The refinement of an abstract plan leaves all the literals that comprise the abstract space unchanged.

This property was defined in order to build abstraction hierarchies. But, it can be very useful for our approach in order to define which predicates can be combined in order to build good abstractions. In the previous section, we defined how to build a predicate abstraction using a set of ungrounded predicates. This means that if the predicate abstraction is built using too many predicates, the abstract solution may not be able to be refined because the actions of the abstract plan may not be used in the non-abstract plan.

6.4 VRP Algorithm

As we have discussed previously, predicates can be removed from the original planning task to decrease the size of the search space and generate a new smaller abstract search space. But first, we need to show **how** the predicates manually chosen are used to build predicate abstractions (discussed in Section 6.4.1) and **when** these abstractions are deployed during search (discussed in Section 6.4.3).

6.4.1 Predicate Selection

Previously to the first phase of VRP, it is mandatory to choose a set of predicates to abstract over. The relevance of each predicate depends of different factors: (i) the type of predicate — whether the predicate is added or deleted during search —; (ii) whether the information that describes the predicate is relevant to achieve the goals; and (iii) whether the information about a predicate is relevant to select other predicates. During this phase, some predicates are selected according to some of these factors, and will form the **predicate set** of a planning task $ps(\Pi)$. We have defined four categories of predicates:

- Static: they are predicates which do not appear in the effects of any action of the planning task, so no action can remove them.
- Dynamic: they are predicates which are part of the effects of at least one action of the planning task.
- Goal: they are predicates that represent some of the goals of the planning task.
- Function: they are predicates that describe a property of the world that is represented using numeric values.

According to this categorization, it is easy to identify the relevance of the different types of predicates in relation to selecting them for abstraction. Static predicates are invariant, so they do not change during the planning process. For this reason, eliminating them does not offer any real opportunity to decrease the complexity of the search task. Goal predicates cannot be removed, since the planning task would be transformed into unsolvable planning task. Our objective consists in relaxing some parts of the planning task while keeping the same task. Functions can be good candidates but, initially, we are going to focus in predicates. Finally, dynamic predicates change during search. Clearly, these are the best option to simplify the

planning task by abstracting them without losing goals reachability. As an example, Table 3 shows the predicate categorization for the Rovers domain.

Static	Dynamic	Goals
at_lander	at	communicated_soil_data
can_traverse	empty	communicated_rock_data
equipped_for_soil_analysis	have_rock_analysis	communicated_image_data
equipped_for_rock_analysis	have_soil_analysis	
equipped_for_imaging	full	
available	calibrated	
supports	have_image	
visible	channel_free	
visible_from	at_soil_sample	
store_of	at_rock_sample	
calibration_target	communicated_soil_data	
on_board	communicated_rock_data	
	communicated_rock_data	

Table 3: Predicate categorization for the Rovers domain.

There can be potentially many different ways to choose what predicates can be used to generate abstractions. The first approach to select the predicates to remove is to rely on human expert knowledge. We have explored this alternative in previous studies over the FF planner (Martínez et al., 2012; Martínez et al., 2013). In the Rovers domain, the best candidate from the standpoint of an expert is the predicate `at`. This predicate describes the current location of a robot which is continuously moving between locations that are connected by different paths. The robots need to move to complete other tasks. Since there are different paths to move from one location to any another location, removing predicate `at` is a good choice to decrease the complexity of the search process, while not introducing dead-ends. Thus, it decreases the number of actions which can be applied during search, so the planner does not have to reason about the position of the rover. The planner will deal with those computations when the next action fails, ignoring again movement actions in the far future. For each domain, we will provide in the experimental section a rationale for choosing some predicates over others. The advantage of this approach is that it works quite well if the domains at hand are well-known in advance. The disadvantage is that it might require some manual trial and error.

6.4.2 Abstractions Generation

The next phase generates abstractions using the information of the **predicate** set computed in the previous phase. According to the definitions in section 6.2, VRP builds an abstraction over the propositional representation (Martínez et al., 2013). But, currently, most planners use a multi-valued state variable representation, which is an extension of the SAS+ formalism (Bäckström and Nebel, 1993). The complete abstraction generation process is composed of two steps (encoding abstractions on SAS+ and generating abstract actions).

6.4.2.1 Encoding Abstractions on SAS+

Planners that use SAS+ transform a planning task expressed in PDDL into SAS+ using multi-valued state variables. Each variable value (a fluent in SAS+) is equivalent to a grounded predicate. According to the example depicted in Figure 23, a SAS+ planner would generate a state variable r1 to define the position of the rover r_1. It will be composed of 16 fluents, one for each position in which the rover can be:

$$\texttt{r1} = \{\texttt{at-w00}, \ \ldots, \ \texttt{at-w33}\} \cup \{u\}.$$

In this case, if in a state s $s[\texttt{r1}] = \texttt{at-w00}$, it would represent the same information as the grounded predicate `at-r1-w00`. However, in many domains, variables' values can come from groundings of different predicates. For instance, if rovers could hold rocks and move rocks to other locations, the variable that represents the location of the rock `rc1` can have as values all waypoints in the planning task (`at-w00` to `at-w33`), as well as all robots that can hold it (`holds-r1` to `holds-rn` in the case of a planning task with n robots). Thus, in order to apply the abstraction mechanism described before, we cannot just remove variables from the planning task, but values of variables that correspond to predicates in the predicate set.

Therefore, the first stage of the process of the generation of abstractions consists of marking fluents from SAS+ variables. A new property must be defined for each fluent. Property $a(v_i)$ indicates whether a fluent must be removed when abstractions are deployed. The marking process consists of identifying which fluent must be removed. In order to determine if a fluent is part of the predicate set, a function $get\_fact\_name(v_i)$ is defined. This function extracts the name of the predicate that represents the fluent v_i. The name of the predicate of each fluent is compared with each element from the predicate set. If the fact name of the fluent is part of

the predicates set, the fluent is marked as abstract. The result of this stage is a variation of the state variable set, where some fluents are marked as abstract.

Algorithm 6.1 details the marking process for a state variable set and a predicate set. The algorithm iterates over all state variables of the state space (line 2). Then, all available fluents of each state variable v are analyzed (line 3) in order to check if they can be abstracted or not. The corresponding fact name of each fluent f is extracted using the function $get\_fact\_name(f)$ (line 4) which is compared to each predicate in the predicate set P. If the fact name is a member of the predicate set (line 6), the fluent is marked as abstract fluent changing the value of the property $a(f)$ to true (line 7).

Algorithm 6.1: Pseudo-code of the marking process

```
   input: ordered state variable set V
   input: predicate set P
1  begin
2      foreach v ∈ V do
3          foreach f ∈ v do
4              name ← get_fact_name(f)
5              foreach p ∈ P do
6                  if p = name then
7                      a(f) ← true
8                      break
```

6.4.2.2 Generating Abstract Actions

The second stage of the abstraction generation process consists of generating abstract actions using SAS+ fluents marked as abstract. A new set of actions is defined as the abstract action set A^{abs}. It is composed of the new actions computed after removing fluents marked as abstract from standard actions. For each action $a \in A$ a new action a^{abs} is generated, where $pre^{abs}(a) = pre(a) \setminus AP$, $prev^{abs}(a) = prev(a) \setminus AP$ and $post^{abs}(a) = post(a) \setminus AP$. If all fluents from the post and pre sets are removed, the action is not added to A^{abs}.

Algorithm 6.2 describes the generation process of the abstract actions. This generation process is composed of three functions. The first one is the main function of the generation process which iterates over all actions of the action space (line 3).

For each action a from the original space a new action is generated in the abstract space. This process is composed of two phases: (1) the first one generates the pre-conditions and the post-conditions using the `generate_pre_and_post_fluents` function (line 4); and (2) the second one generates the prevail-conditions using the `generate_prevail_fluents` function (line 6), if the fluent set generated by the function `generate_pre_and_post_fluents` is not empty (line 5). Finally, if the abstract action keeps at least one effect, the action is added to the abstract action set (line 8).

Algorithm 6.2: Pseudo-code of the generation process of the abstract action space

```
   input : ordered state variable set V
   input : actions set A
   output : abstract action set A^abs
1  begin
2      Set A^abs ← empty
3      foreach a ∈ A do
4          prepost ← generate_pre_and_post_fluents(get_effects_fluents(a))
5          if prepost is not empty then
6              prevail ← generate_prevail_fluents(get_prevail_fluents(a))
7              name ← get_action_name(a)
8              A^abs ∪ action(name, prevail, prepost)
9      return A^abs
```

The `generate_pre_and_post_fluents` function (line 1) generates both the pre-conditions and the post-conditions set (predicates) for the new action. The function iterates over all fluents of the original effect set adding to the predicate set each pre-condition and post-condition which has not been marked as abstract (lines 5 and 8). Finally, the function returns the predicate set. The `generate_prevail_fluents` function (line 11) generates the set of prevail-conditions for the new abstraction. As the previous function, this function iterates over all fluents of the original prevail set adding to the prevail set each prevail-condition which has not been marked as abstract (line 14). Finally, the function returns the prevail set.

Algorithm 6.3: Pseudo-code of the generation functions which generate the prevail-conditions, the pre-conditions and the post-conditions of the abstract actions.

```
1  Function generate_pre_and_post_fluents(F)
       input  : original fluent set: F
       output : new pre-condition and post-condition fluent set: p
2      Set eff ← empty
3      foreach f ∈ F do
4          if f.pre is not empty then
5              if a(f.pre) then
6                  predicates ∪ f.pre
7          else
8              if a(f.post) then
9                  predicates ∪ f.post
10     return predicates
11 Function generate_prevail_fluents(F)
       input  : original prevail fluent set: F
       output : new prevail fluent set: prevail
12     Set prevail ← empty
13     foreach f ∈ F do
14         if a(f) then
15             prevail ∪ f
16     return prevail
```

Figure 25 illustrates the description file generated for the example described in Figure 23. This file represents the planning task in a high level language used by the preprocessor module of FD[3]. This file is used by the planning module to solve the abstract planning task. As we can see in the example, action `navigate-r1-w01-w02` does not generate any abstract action when the abstraction is applied. However, action `communicate_rock_data-r1-general-w10-w20-w30` generates an abstract action with one precondition less. The result of this step is a high level representation including both abstract actions and abstract fluents generated using the set AP.

Corollary 2. Let $\Pi = (F, A, I, G)$ be a planning task, let P be a predicate set, and let $\Pi_P^{abs} = (F_P^{abs}, A_P^{abs}, I_P^{abs}, G_P^{abs})$ be the abstract planning task. The size of the abstract action set will always be $\left|A_P^{abs}\right| \leqslant |A|$.

[3] The syntax of the high level language used by the preprocessor of FD is described in http://www.fast-downward.org/PreprocessorOutputFormat

Proof. (Corollary 2) According to definition 13, the projection $\text{Proj}_p(a)$ of an action $a \in A$ transforms the action a in an abstract action a_{abs} which is added to A_p^{abs} if $\text{eff}(a_{abs}) \neq \emptyset$. Otherwise, the action a_{abs} is not added to A_p^{abs}. Therefore, given that actions in A are transformed in abstract actions and they can either be added to A^{abs} or not, and the projection operation does not create any new action, then it must be true that $\left|A_p^{abs}\right| \leqslant |A|$.

6.4.3 Search using Predicate Abstractions

Finally, the third phase of VRP generates a sequential abstract plan using both the action set A from the original space and the abstract action set A^{abs} from the abstract space. The search algorithm implemented in this phase is based on relaxing some far future details about the planning task in order to decrease the time needed to generate a solution plan. Thus, we have to define **how** the execution time must be measured during the search process and from **what** point predicate abstraction must be deployed. The most common way to measure the execution time during search consists of monitoring the **depth** of the nodes according to the actions which have been chosen to build the solution. Then, we introduce the concept of horizon (Stentz, 1995b) into the search which is bounded using the depth of the nodes of the search space.

Definition 18. (Horizon). A valid **Horizon** is any k natural number such that $k > 0$.

The horizon described in Definition 18 is employed to define which action set must be used to generate the successors of a state during search. The function `getApplicableActions(A, A^abs, s, k)` generates the applicable actions for the state s according to the horizon value k. This function compares the depth of the state d(s) with k. If the horizon value k is higher than d(s), the applicable actions of the state s are generated using the original action set A. Otherwise, the function getApplicableActions(s, k) uses the abstract action set A^{abs} to choose them.

$$\text{getApplicableActions}(A, A^{abs}, s, k) = \begin{cases} \text{getActions}(A, s) & d(s) < k \\ \text{getActions}(A^{abs}, s) & d(s) \geqslant k \end{cases} \quad (1)$$

During the search process, the effect of the horizon is combined with the predicate abstraction, effectively pruning the search space. The combination of both

Variables:

r1 ∈ {at-w01, at-w02, at-w03, at-w10, at-w11, at-w13, at-w20, at-w21, at-w22, at-w23, at-w30, at-w31, at-w32, at-w33}

rs1 ∈ {empty-rs1, full-rs1}

c1 ∈ {communicated_rock_data-w10, ¬communicated_rock_data-w10}

c2 ∈ {communicated_soil_data-w22, ¬communicated_soil_data-w22}

v1 ∈ {have_rock_analysis-r1-w10, at_rock_sample-w10}

s1 ∈ {have_soil_analysis-r1-w22, at_soil_sample-w22}

Initial State:

r1 = at-w33

rs1 = empty-rs1

c1 = ¬communicated_rock_data-w10

c2 = ¬communicated_soil_data-w22

v1 = at_rock_sample-w10

s1 = at_soil_sample-w22

Goal:

c1 = communicated_rock_data-w10

c2 = communicated_soil_data-w22

Action communicate_rock_data-r1-general-w10-w20-w30

PREV: r1 = at-w10 PRE: v1 = have_rock_analysis-r1-w10

POST: c1 = communicated_rock_data-w10

...

Action navigate-r1-w01-w02

PRE: r1 = at-w01 POST: r1 = at-w02

Abstract action communicate_rock_data-r1-general-w10-w20-w30

PRE: v1 = have_rock_analysis-r1-w10 POST: c1 = communicated_rock_data-w10

...

Figure 25: Multi-valued state variable representation of the Rovers planning task including abstract actions when predicate **at** is removed. This representation is described using a high level language used by the preprocessor of FD.

parameters decreases the number of applicable actions for each state s_i when depth is greater than k. Thus, it is expected that solving time decreases, because the number of generated states will be much less. However, the quality of the generated abstract plan depends on the value of k. We believe that small values of k decrease the planning time and decrease the number of expanded nodes during search.

Potentially, this diminishes the accuracy of the plan and increases the number of replanning steps when this technique is used on a planning and execution cycle (more abstract actions means more execution failures). On the other hand, high values of k increase the accuracy of the plan, but increase the planning time. In the worst case, if the value of k is close to the size of the standard plan, the planning time using the horizon and the predicate abstraction will be equal than the standard planning time. In order to understand how the value of k influences the search process, we will choose a representative set of values to use on the experiments reported in the next section.

Algorithm 6.4 shows the pseudo-code of the k-bounded Best-First Search (BFS). The open list, open, organizes the generated states in a list according to their f-value. The open list stores search states which are composed of one state, the action which generates the state, the h value and the g value. The algorithm starts inserting the initial state using the `state` function (line 2). Then, the algorithm iterates the open list until it is empty or a solution is found (line 5). At each iteration, the next state to be expanded is extracted from the top of the open list (line 6) and inserted in the closed list (line 7). First, the algorithm checks whether the current state is a goal state (line 8). If the current state is a valid goal state, the algorithm stops the search (line 9) and builds the solution path going over the closed list starting from the current state (line 23). Otherwise, the algorithm generates the applicable actions using the `getApplicableActions` function (line 12). If d(current) is lower than k, the successors are generated using the original action set (line 12). But, if d(current) is greater than k, the successors are generated using the abstract action set (line 14). Finally, the successors of the current state are generated. Each successor is evaluated (line 15) and added to the corresponding list (open or closed). Then, the algorithm checks if the successor has been generated previously and can be reopened or updated (line 17). Otherwise, the algorithm inserts the successor in the open list or in the closed list if the successor is a dead-end. Finally, the algorithm builds the solution using the `buildSolution` function (line 23). This function finds the forward path from I to the current state iterating the closed list if the current state is a goal state. Otherwise, the function outputs `'no plan'`.

6.5 Empirical Evaluation

In this section, we describe the method that we have followed to conduct the empirical evaluation of the different approaches which have been developed in this thesis. During the last years, the planning community has done many efforts to develop common methods and tools to measure the capabilities of the different planning systems and compare their performance. These efforts led to the creation in 1998 of the International Planning Competition (IPC). This competition has

Algorithm 6.4: Pseudo-code of the k-bounded Best First Search algorithm

```
input : Planning task: Π = (F, A, I, G)
input : Abstract planning task: Π_P^abs = (F_P^abs, A_P^abs, I_P^abs, G_P^abs)
input : Horizon value: k
output : Sequential abstract plan or "no plan"
1  begin
2      Set open ← open ∪ state(I, empty, evaluate(I), 0)
3      Set closed ← empty
4      Set solved ← false
5      while open is not empty and not solved do
6          current ← getBestState(open)
7          closed ← closed ∪ {current}
8          h ← getH(current)
9          if G ⊆ current then
10             solved ← true
11         else
12             acts ← getApplicableActions(A, A_P^abs, current, k)
13             successors ← generateSuccessors(current, acts)
14             foreach suc ∈ successors do
15                 h_suc ← evaluate(suc)
16                 g_suc ← getG(current) + getCost(suc)
17                 if reopen(suc, closed) or update(suc, h_suc, open) then
18                     open ←
                       open ∪ state(getState(suc), getAction(suc), h_suc, g_suc)
19                 else if not isDeadEnd(suc) then
20                     open ←
                       open ∪ state(getState(suc), getAction(suc), h_suc, g_suc)
21                 else
22                     closed ←
                       closed ∪ state(getState(suc), getAction(suc), h_suc, g_suc)
23     return buildSolution(s_0, current, A, A_P^abs, solved, closed)
```

as objective to offer a common environment and rules in order to compare the performance of current state-of-the-art planners.

In the last years, IPC rules have become a standard to compare planner performance and evaluate research in AP. However, most of these rules have been created to compare the performance of the plan generation process without any consideration about the execution process. In order to evaluate this thesis, we have implemented

a new version of the PELEA architecture to solve planning tasks in a simulated or real-world environment. The PELEA architecture automatically tracks the planning and execution process measuring the planning and replanning time, the number of planning episodes, the number of executed actions, the evolution of the goals and memory usage. Besides, it validates all the solution plans using the plan evaluation system provided for MDPSim (Younes et al., 2005).

As described in Chapter 5, LPELEA architecture deploys a planning, monitoring and execution loop to solve a planning task in simulated or real-world environment. In order to perform a set of experiments to analyze the capabilities of the different approaches developed in this thesis, we have defined a basic setting. The maximum planning time for a problem has been set to 1000 seconds and the maximum execution time for a complete planning-execution-replanning loop for each planning task has been set to 86400 seconds (1 day). Each problem has been executed 10 runs where a run is a complete execution of a planning task in the simulated environment. A run is considered complete when the goals are reached, the maximum planning time is reached, the maximum execution time is reached or a dead-end is detected.

6.5.1 Evaluation metrics

We measure the performance of planning and execution with two types of metrics: coverage and time score. The time score metric has been extended to measure the planning time, the execution time and the replanning time.

- Coverage: This metric measures the total number of planning tasks solved from the benchmark within the time and memory bounds. It is the official metric for the optimal track of the IPC since 2008.
- Time Score: This metric measures the reward that planners get for solving the planning tasks as early as possible. Planning systems get a time score in the interval (0, 1] for each task solved. The fastest planner is awarded one whole point, while other planners solving the same problem in more time receive a fraction of a point. If a planner does not solve the problem before the time limit it gets 0 points for that problem. The total score of a planner in a domain is the sum of its score in all the problems of that domain. This metric is used to score three different values: (1) first planning time; (2) full planning time; and (3) full planning and execution time.

  Different equations may be proposed to determine the score of the planning systems. In the context of this thesis, we use the time score metric from the Sequential Satisficing Track of the IPC-2011. Let t_* be the minimum time

in seconds required by the fastest planner to solve the task, rounding all the times to the upper second, let t_p be the time in seconds obtained by the planner p and let t_{max} be the maximum planning time. Then, the time score of a planner p that solves the problem in t_p, is computed following Equation 2. Any planner solving the problem in less than one second receives the maximum score.

$$t_{score}(t_*, t_p, t_{max}) = \begin{cases} 0 & \text{if } t_p > t_{max} \\ 1 & \text{if } t_p < 1s \\ t_*/t_p & \text{if } t_p > 1s \end{cases} \quad (2)$$

- Score: This metric is used to measure different variables which are collected during planning and execution (planning steps, quality of the solution, plan deviation, etc). As the previous metric, planning systems get a score in the interval (0, 1] for each task solved. The best planner is awarded one whole point, while other planners solving the same problem but obtaining a higher value for the variable measured receive a fraction of the point. Let v_* be the minimum value required by the best planner to solve the task, let v_p be the value obtained by the planner p and let v_{max} be the maximum value for the variable measured. Then, the score of a planner p that solves the problem with the value v_p, is computed following Equation 3.

$$v_{score}(v_*, v_p, v_{max}) = \begin{cases} 0 & \text{if } v_p > v_{max} \\ v_*/v_p & \text{if } v_p <= v_{max} \end{cases} \quad (3)$$

6.5.2 Planners

In order to evaluate the results of our research, in this thesis we have compared the performance of the different versions of our approach against the closest competitors models in terms of planning and execution systems based on PDDL and its variants:

- Classical techniques, which use planning and replanning when an execution failure is detected. We use LAMA11, an anytime planner developed within the Fast-Downward framework (Richter and Westphal, 2010). Once LAMA11 has found a first solution, it continues to search for better solutions until it exhausts the search space or the available resources (memory and/or time). LAMA11 was the winner of the sequential satisficing track of IPC2011.

LAMA11 assumes it will be given 30 mins to generate a plan. This is unreasonable for most robotics tasks. Therefore, in our case, we only use the first solution and we call this configuration LAMAF.

- Probabilistic techniques, which use a PPDDL model, so it has information on probabilistic outcomes of actions. We have used mGPT (Bonet and Geffner, 2005), the winner of the last Probabilistic track based on PPDDL. mGPT is a planning system based on heuristic search for solving Markov Decision Processes (MDPs) by extracting and using different classes of lower bounds along with various heuristic-search algorithms.
- Reactive techniques, which choose the best action according to the current state of the environment. In order to provide a better behaviour than a pure reactive system, we tune our approach to work like a reactive system.

6.5.3 Benchmark Domains

A set of different benchmark of planning tasks has been defined in each competition to test the planners' capabilities of solving tasks of different kinds. These planning tasks are classified in domains. Problems of the same domain are of the same type, having a common structure but varying in difficulty. However, most of the planning domains have been designed with the aim of checking the limits of both the heuristic functions and/or the search algorithms. Then, our benchmark is composed of both domains from previous IPCs and domains designed to solve real-world problems.

In this thesis, we consider a benchmark set of 6 domains. For each domain we have chosen 5 different problems per domain as a baseline to generate our benchmark. Problems were chosen by running state-of-the-art planners and choosing problems that could be considered as easy, medium and hard according to the time it takes the planners to solve them. Each problem is executed 15 times in the simulated environment, so we already have 150 different problem execution traces. During execution, due to the use of the error simulator (see section 5.3.4.2) and MDPSim (see section 5.3.4.1), the problem's structure changes according to the different failures and the exogenous events. We present a detailed description of the different planning domains which have been considered:

- The **Rover domain** was designed for the sequential track of IPC-3 (2002) and it was inspired on the Mars exploration rovers missions where an area of the planet is represented as a grid of cells, called waypoints. They contain samples of rock or soil that can be collected by the robots. Each robot can traverse across different waypoints and can perform a set of different actions

(analyze rock or soil samples or take pictures of a specific waypoint). All data collected by robot units has to be sent to the lander, that is placed in a specific waypoint.

- The **Depots-robots domain** is inspired on the current efforts of some companies to automate warehouse management processes by using homogeneous robot units like Amazon company which is using Kiva robots (Wurman et al., 2007) in its warehouses. In this domain, there is a set of robot units, a set of humans and a set of pods which are located in a warehouse. The warehouse is defined as a grid composed of cells. Robot units can move among adjacent free cells and carry pods to humans. Humans are located in specific locations where they can use products which are contained in the pods.

- The **TidyBot domain** was designed for the sequential track of the IPC-6 (2012) and it was inspired on a household cleaning task. There is one or more robot units which must pick up a set of objects and put them into goal locations. The environment is represented as a bi-dimensional grid, divided into navigable locations and surfaces (tables and cupboards) on which objects may lie. Robots have a gripper, which moves relative to the robot, up to some maximum radius. Existing objects block the gripper, so that it may be necessary to move one object out of the way to put another one down. Robots can carry one object at a time in the gripper, but may also make use of a cart, that can hold multiple objects.

- The **Port domain** is inspired on the current container handling systems used on ports to move the containers carried on by the ships. In this domain, there is a set of ships, a set of hoists and a set of crates which are located in ships. There are some docks which are shared by all hoists. Hoists are assigned to one ship and can move and stake crates between ships and the dock. Besides, crates can be stacked on the dock. The goals of this domain consist on loading crates on ships.

- The **Satellite domain** was designed for the sequential track of IPC-3 (2002) and it was inspired on the Satellite observation missions. In this domain, there is a set of satellites equipped with different instruments, which can operate in different modes. The goal is to acquire images, dividing the observation tasks among the satellites, based on the capabilities of their instruments. Satellites must acquire images from different objectives from the space.

- The **Warehouse domain** is a variant of the Sokoban Domain inspired on the warehouse management processes by using heterogeneous robots. In this domain, there are two different robot units which can move around the place.

The first robot unit (cargo robot) offers a basic mechanism to push and pull pods in specific storage locations. The second robot unit (coordinator robot) analyzes the label of the pods to obtain information about the pod's final location using Computer Vision. Pods are identified using a Quick Response code (QR) label, which has information about the pod (storage location, priority, weight, etc). This information can be critical during the storage process. In order to solve a problem in this environment, coordinator robots must obtain information about the different pods and command to the cargo robots to move the pods to the correct storage location. Thus, it implies a collaboration process between different robot units. Cargo robots can only move and transport pods, meanwhile coordinator robots can move and get information about the pods using the QR labels. This domain was used in one of the papers references in Chapter 8.

6.5.4 The evaluation environment

The evaluation environment has been configured using the LPELEA architecture described in Chapter 5.3 which uses both the MDPSim to emulate the execution of plans and the Error Simulator which increases the variability of the execution dynamics introducing exogenous events. The VRP technique described in this chapter has been implemented over the Fast-Downward (FD) planning system. The source code, written in C++, has been built as a variation of the preprocessor and the solver system of FD. The new planner has been called Abstract k Fast Downward (AKFD)[4].

The LPELEA architecture has been configured as shown in Figure 36[5]. This loop is used to evaluate VRP using the AKFD planning systems. The planning-execution process starts with the Execution node. Given a planning task (consisting of a deterministic PDDL domain and problem) and a horizon k, the Execution node requests a plan π to the Monitoring node, which, in turn, requests a plan to the Decision Support node. This node generates a plan using VRP, taking as input the deterministic PDDL domain and problem as well as k. Next, the Monitoring node iteratively sends every action a_i in the plan to the Execution node based on Algorithm 5.1. Finally, the Execution node sends each action to the real world to be executed and receives the new observed state. In our case, the real world has been realistically simulated by using two nodes. The Error simulator transforms the action a_i into another action a_i^e in PPDDL such that it incorporates new stochastic effects into the action model. Next, the Error simulator sends the new

[4] The source code is available at https://bitbucket.org/momartin/akfd

[5] The source code of the VRP modules is available at https://bitbucket.org/momartin/peleahorizon. There are some detailed instructions in the repository on how to use it.

action to the actual simulator, MDPSim in this case. MDPSim executes each action using a stochastic action model in PPDDL, manually generated from the original action model (PDDL domain).

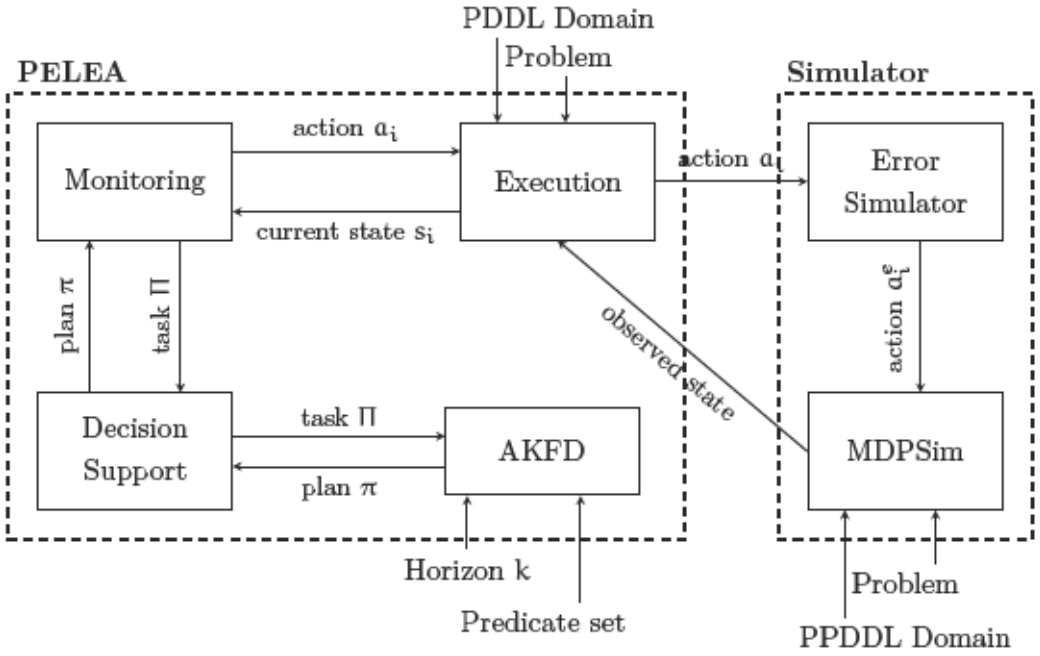


Figure 26: Light PELEA Architecture used to evaluate VRP.

After every execution of an action, if the observed state is not valid according to the deterministic version of the domain (as in Definition 11), PELEA generates a new plan using the observed state as the new initial state. This process is repeated using a re-planning-execution-monitoring loop until it senses a state where the goals have been reached. Due to the characteristics of the simulator, we assume that the different agents (robots) can wait in a secure state during the computation of a new plan. This means that the state of the different agents will not change during the planning steps.

The performance of AKFD is compared to FF-Replan (Hoffmann and Nebel, 2001), mGPT (Bonet and Geffner, 2005) and Lama (Richter and Westphal, 2010) over five problems of the Rovers domain. Our approach has been configured manually where a horizon value of 10 actions and a predicate set composed of the predicate `at`. In the Rovers domain, which we have described before, we have designed an error model based on four failures which have been encoded into PPDDL. (1) There is a general error which prevents the execution of any action. Each action has a probability equals to 0.05 that the action is not executed properly; (2) A calibration error happens when a rover tries to take a picture, and the camera accidentally removes

its calibration, so the rover needs to calibrate the camera again. This error has a probability equals to 0.1; (3) A communication error occurs when a rover tries to send samples or images to the lander and the information sent by the rover is never received. The sample is lost and the rover must take the sample again. This error has a probability equals to 0.1. The domain definition has been modified allowing this kind of failures. Rock and soil samples do not disappear from a waypoint when a rover uses them. And (4) a navigation error happens when a rover moves to a different waypoint than the expected one when it is navigating. In the case that the rover finished its movement in a different waypoint, this waypoint must be connected with the origin or destination waypoint. This error has a probability equals to 0.15. All these experiments were conducted in a Intel Xeon 2.93 GHZ Quad Core processor (64 bits) running under Linux. The maximum available memory for the planners was set to 8 GB.

Table 4 shows the results for five different problems of the Rovers domain where an abstraction has been computed using the predicate **at**. This predicate has been selected manually. The results report the average over 15 executions and the standard error over five different metrics (M). F corresponds to planning time for the first planning episode in seconds. T corresponds to the total planning time (including first planning time) of all runs in seconds. R corresponds to the number of replanning episodes. A corresponds to the number of actions executed in the simulated environment and C corresponds to the number of problems solved. Coverage is described in terms of four values: (1) number of solved problems; (2) number of unsolved problems that have exceeded the maximum planning time (1000 seconds); (3) number of unsolved problems that have exceeded the maximum planning and execution time (86400 seconds); and (4) number of unsolved problems by dead-ends. The last value is the sum of the other four values. LAMAF solves all problems, but using much longer planning time than our approach. Besides, our planner decreases the first planning time on one order of magnitude in all problems chosen from the Rovers domain. We can also see that an approach based on MDPs, mGPT, that receives as input more information than our approach (the probabilistic effects of actions), is not able to scale up and does not solve any problem. In general, the time performance of AKFD is better than the other three planning systems solving the problems quicker than them. Besides, AKFD needs less time to compute a sequential abstract plan which is an important advantage over the other two approaches in dynamic and stochastic environments where the reasoning time is important to offer real interaction with the environment. FF-Replan can only solve two planning tasks of the Rovers benchmarks. Finally, mGPT cannot solve any planning task, because this planner needs more than 1000 seconds to build a solution according to the probability distribution of the action model.

Planner	M	Problem				
		rovers 36 (14,80,40)	rovers 37 (14,80,40)	rovers 38 (14,85,57)	rovers 39 (14,95,62)	rovers 40 (14,100,68)
FF-Replan	F(s)	93.9 ± 1.1	-	632.4 ± 1.9	-	-
	T(s)	1178.9 ± 329.1	-	4946.1 ± 1131.8	-	-
	R	40.1 ± 6.7	-	39.5 ± 10.2	-	-
	A	311.7 ± 37.4	-	350.5 ± 24.1	-	-
	C	10, 5, 0, 0/15	0, 15, 0, 0/15	11, 4, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15
LAMAF	F(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	45.8 ± 5.4	75.60 ± 7.3	59.2 ± 7.3	68.2 ± 5.1	70.1 ± 10.2
	A	323.6 ± 24.2	512.20 ± 28.9	432.8 ± 42.3	486.8 ± 21.2	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
mGPT	F(s)	-	-	-	-	-
	T(s)	-	-	-	-	-
	R	-	-	-	-	-
	A	-	-	-	-	-
	C	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15
AKFD (k = 10)	F(s)	9.6 ± 0.1	13.8 ± 0.1	20.2 ± 0.1	14.6 ± 0.1	23.9 ± 0.1
	T(s)	591.5 ± 68.4	1271.3 ± 95.3	1347.7 ± 144.9	1380.1 ± 52.9	1996.5 ± 69.3
	R	64.3 ± 6.6	104.1 ± 8.8	73.1 ± 8.2	110.7 ± 5.8	95.1 ± 2.8
	A	354.1 ± 21.3	592.3 ± 25.7	426.1 ± 25.7	593.7 ± 23.5	568.3 ± 18.9
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 4: Comparing AKFD to LAMAF, FF-Replan and mGPT over five problems from the Rovers domain. AKFD has been tuned removing the predicate **at** and using a horizon value of 10 actions. Both parameters have been chosen manually. The performance of the planners is measured over 5 metrics (M). F corresponds to planning time for the first planning episode in seconds. T corresponds to the total planning time (including first planning time) of all runs in seconds. R corresponds to the number of replanning episodes. A corresponds to the number of actions executed in the simulated environment and C corresponds to the number of problems solved. The best results are highlight in bold. Besides, for each problem we have defined three values which describe the complexity of the problem: (i) number of rovers; (ii) number of waypoints and (iii) number of goals.

These results show that VRP can be used to solve planning task in dynamic and stochastic environment decreasing both the first planning time and the full planning time keeping the quality of the solution. However, VRP depends of three parameters (the predicate set, the horizon value and the environment) which must be analyzed in order to discover the influence of these parameters in the process of planning and execution. In the next sections, we analyze the effects of each parameter.

6.5.5 The predicate set

The predicate set defines which predicates are going to be removed in order to generate the abstract state space. In this section, we analyze the effect of using different predicate sets composed of one or more predicates in order to analyze the effects over the planning time and the quality of the executed plan. Table 5 shows the results for five different problems of the Rovers domain where an abstraction has been computed using one predicate in order to analyze the influence of different predicates when abstractions are built. We have chosen five dynamic predicates from the categorization shown of Table 3 (Page 95). The predicate sets used to build abstractions are: $AP_1 = \{at\}$, $AP_2 = \{have\text{-}image\}$, $AP_3 = \{have\text{-}rock\text{-}analysis\}$, $AP_4 = \{have\text{-}soil\text{-}analysis\}$ and $AP_5 = \{calibrated\}$. We only compare AKFD with LAMAF, because mGPT does not solve any problem and FF-Replan does not solve difficult problems. The results show small differences in the first planning time (F) depending on which predicate is used to build the abstraction. Neither, there are significant differences in the full planning time (T). However, the predicate `have_rock_analysis` provides the best results for all problems. In our opinion, this happens because there are more goals of the type `communicated_rock_data` or rock goals are the most difficult to reach. This is an important fact, because the predicate used to build good abstractions can be chosen depending on the structure on the problem or the type of goals.

Definition 19. (**Domination between predicates**) Let $\Pi = (F, A, I, G)$ be a planning task, let $P_1 \in F$ a predicate, let $P_2 \in F$ a predicate, let $\pi^{abs}_{P_1}$ the sequential abstract that solve the abstract planning task $\Pi^{abs}_{P_1}$ and let let $\pi^{abs}_{P_2}$ the sequential abstract that solve the abstract planning task $\Pi^{abs}_{P_2}$. A predicate P_1 dominate over P_2, if $\pi^{abs}_{P_1} \subset \pi^{abs}_{P_2}$.

Regarding to the number of replaning steps, we observe that some predicates like `calibrated` or `have_rock_analysis` reduce the number of replanning steps (R) and the full planning time (T). This fact can be related to the similarity of the plan generated by AKFD and LAMAF. Table 6 shows the size of the plans generated in the first planning step. Predicates `have-image` and `calibrated` generate the most accurate plans, but none of these predicates generates the predicate abstraction which produces the best results. Then, the predicate used to built the predicate abstraction depends on the structure of the planning task.

The previous results show that there are some differences in the performance of AKFD depending on the predicate used to build the predicate abstraction. According to this, we analyze if some predicate can be used to build a more complex predicate

Planner	M	Problem				
		rovers 36 (14,80,40)	rovers 37 (14,80,40)	rovers 38 (14,85,57)	rovers 39 (14,95,62)	rovers 40 (14,100,68)
LAMAF	F(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	**45.8** ±5.4	**75.60** ± 7.3	**59.2** ± 7.3	**68.2** ± 5.1	**70.1** ± 10.2
	A	**323.6** ± 24.2	**512.20** ± 28.9	432.8 ± 42.3	**486.8** ± 21.2	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD AP_1	F(s)	**9.6** ± 0.1	**13.8** ± 0.1	**20.2** ± 0.1	14.6 ± 0.1	**23.9** ± 0.1
	T(s)	591.5 ± 68.4	1271.3 ± 95.3	**1947.7** ± 144.9	1380.1 ± 52.9	1996.5 ± 69.3
	R	64.3 ± 6.6	104.1 ± 8.8	73.1 ± 8.2	110.7 ± 5.8	95.1 ± 2.8
	A	354.1 ± 21.3	592.3 ± 25.7	**426.1** ± 25.7	593.7 ± 23.5	568.3 ± 18.9
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD AP_2	F(s)	11.3 ± 0.2	18.6 ± 0.1	24.0 ± 0.1	20.6 ± 0.0	29.1 ± 0.2
	T(s)	594.6 ± 89.5	1313.0 ± 204.4	1517.2 ± 257.8	1289.1 ± 136.6	2050.4 ± 464.5
	R	58.3 ± 8.9	90.8 ± 14.6	73.2 ± 12.6	86.2 ± 10.1	84.4 ± 19.3
	A	377.7 ± 46.9	593.6 ± 44.7	475.8 ± 36.6	594.6 ± 63.3	524.2 ± 39.0
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD AP_3	F(s)	11.6 ± 0.1	20.3 ± 0.1	25.3 ± 0.1	20.1 ± 0.1	31.8 ± 0.1
	T(s)	**535.4** ± 63.5	**1205.5** ± 230.5	1437.1 ± 252.4	**1112.2** ± 126.9	**1693.9** ± 190.9
	R	52.3 ± 6.7	85.2 ± 16.0	68.6 ± 12.9	74.6 ± 10.5	69.6 ± 7.9
	A	353.0 ± 27.7	588.6 ± 79.9	486.0 ± 55.5	564.8 ± 35.8	549.2 ± 42.2
	C	15.0.0.0/15	15.0.0.0/15	15.0.0.0/15	15.0.0.0/15	15.0.0.0/15
AKFD AP_4	F(s)	10.9 ± 0.3	16.3 ± 0.1	23.7 ± 0.2	15.7 ± 0.2	28.4 ± 0.1
	T(s)	704.9 ± 62.1	1403.2 ± 171.3	1623.9 ± 115.3	1560.8 ± 63.9	2275.6 ± 174.7
	R	71.2 ± 7.4	103.6 ± 12.0	79.8 ± 5.9	116.2 ± 4.8	95.8 ± 6.8
	A	385.0 ± 23.3	596.2 ± 51.2	466.4 ± 30.2	664.0 ± 32.3	569.6 ± 12.5
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD AP_5	F(s)	11.2 ± 0.2	16.9 ± 0.1	23.4 ± 0.0	17.8 ± 0.1	27.9 ± 0.2
	T(s)	566.1 ± 79.0	1320.0 ± 170.8	1635.7 ± 191.9	1251.5 ± 211.7	1932.9 ± 188.3
	R	56.5 ± 8.2	94.8 ± 13.7	80.2 ± 9.5	90.0 ± 15.1	80.4 ± 8.2
	A	349.7 ± 34.6	600.8 ± 48.7	481.2 ± 34.3	558.0 ± 48.4	**519.3** ± 36.6
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 5: Comparing AKFD to LAMAF over five problems from the Rovers domain. AKFD has been tuned removing one predicate and horizon of 10 actions. Each configuration is denoted to AP which means Abstract Predicate and a number. Both parameters have been chosen manually. The best results are highlight in bold.

abstraction in order to improve the performance of AKFD. We have defined 5 different predicate sets:

$AP_1 = \{\text{at}\}$
$AP_2 = \{\text{have-image}\}$
$AP_3 = \{\text{have-rock-analysis}\}$
$AP_4 = \{\text{have-soil-analysis}\}$
$AP_5 = \{\text{calibrated}\}$

Planner	Problem				
	rovers 36 (14,80,40)	rovers 37 (14,80,40)	rovers 38 (14,85,57)	rovers 39 (14,95,62)	rovers 40 (14,100,68)
LAMAF	250	393	306	358	385
AKFD ($P = AP_1$)	119	174	156	183	203
AKFD ($P = AP_2$)	206	334	315	392	338
AKFD ($P = AP_3$)	192	239	218	320	167
AKFD ($P = AP_4$)	141	251	226	175	334
AKFD ($P = AP_5$)	228	355	330	382	360

Table 6: Comparing the size of the first plan generated by AKFD and LAMAF for five problems from the Rovers domain.

AP_6 = {at, have-rock-analysis}
AP_7 = {at, have-rock-analysis, have-soil-analysis}
AP_8 = {at, have-rock-analysis, have-soil-analysis, have-image}
AP_9 = {at, have-rock-analysis, have-soil-analysis, have-image, calibrated}

Table 7 shows the results for five different problems of the Rovers domain where the abstractions have been built using different predicate sets. We have used the predicate **at** as a base to build the other predicate sets. Each set has been built by including a new predicate over the previous until all dynamic predicates used in the previous section are part of the predicate set. For instance, the set $AP_7 = AP_6 \cup \{have-soil-analysis\}$. According to the results shown in Table 7, there is no significant difference between the results of the different configurations of AKFD. This effect is produced due to the way in which the abstract action set is built. The predicate **at** decreases the size of the action space when the predicate abstraction is built using it, because the navigate actions are pruned from the original action space. This means that rovers can take pictures, rock and soil samples immediately without moving across the environment. Then, a predicate abstraction built using more predicates related to these tasks will not improve the abstraction produced by the predicate **at**. Predicate **at** dominates over other dynamic predicates like **have-rock-analysis**, **have-soil-analysis** and **have-image**, because it simplifies most of the tasks performed by the rovers in the environment (see Definition 19). Besides, a predicate abstraction composed of many predicates can produce an excessive simplification of the abstract space. This fact may increase the number of

replanning steps due to the bad quality of the sequential abstract plans generated in each planning step.

Planner	M	Problem				
		rovers 36 (14,80,40)	rovers 37 (14,80,40)	rovers 38 (14,85,57)	rovers 39 (14,95,62)	rovers 40 (14,100,68)
LAMAF	F(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	**45.8** ±5.4	**75.60** ± 7.3	**59.2** ± 7.3	**68.2** ± 5.1	**70.1** ± 10.2
	A	**323.6** ± 24.2	**512.20** ± 28.9	432.8 ± 42.3	**486.8** ± 21.2	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD ($P = AP_1$)	F(s)	9.6 ± 0.1	13.8 ± 0.1	20.2 ± 0.1	14.6 ± 0.1	23.9 ± 0.1
	T(s)	591.5 ± 68.4	1271.3 ± 95.3	**1347.7** ± 144.9	1380.1 ± 52.9	**1996.5** ± 69.3
	R	64.3 ± 6.6	104.1 ± 8.8	73.1 ± 8.2	110.7 ± 5.8	95.1 ± 2.8
	A	354.1 ± 21.3	592.3 ± 25.7	**426.1** ± 25.7	593.7 ± 23.5	568.3 ± 18.9
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD ($P = AP_6$)	F(s)	9.6 ± 0.1	13.7 ± 0.1	19.9 ± 0.3	14.4 ± 0.1	22.9 ± 0.2
	T(s)	570.8 ± 75.2	1358.1 ± 175.1	1711.6 ± 125.3	1322.3 ± 154.4	2084.5 ± 95.8
	R	62.2 ± 8.8	111.1 ± 15.3	93.6 ± 7.5	106.4 ± 12.6	98.4 ± 5.2
	A	347.1 ± 24.5	635.2 ± 48.2	477.8 ± 30.5	615.6 ± 49.1	557.8 ± 23.4
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD ($P = AP_7$)	F(s)	9.6 ± 0.1	13.0 ± 0.1	19.3 ± 0.1	13.6 ± 0.1	22.3 ± 0.3
	T(s)	605.7 ± 80.4	**1151.1** ± 22.9	1518.1 ± 118.1	1325.8 ± 119.5	2226.6 ± 142.1
	R	66.6 ± 8.5	95.1 ± 0.9	83.4 ± 6.9	108.2 ± 9.4	105.6 ± 6.3
	A	355.8 ± 50.4	549.4 ± 26.5	466.6 ± 23.2	589.6 ± 17.7	589.8 ± 21.4
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD ($P = AP_8$)	F(s)	9.4 ± 0.1	13.1 ± 0.1	19.1 ± 0.1	13.6 ± 0.1	22.1 ± 0.2
	T(s)	**554.8** ± 61.6	1321.7 ± 175.2	1601.5 ± 202.7	1259.8 ± 159.9	2140.7 ± 122.8
	R	61.4 ± 7.2	109.1 ± 14.7	87.8 ± 10.7	102.6 ± 13.2	102.4 ± 6.6
	A	371.1 ± 25.2	621.4 ± 59.1	472.2 ± 39.9	587.2 ± 69.5	583.2 ± 44.1
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD ($P = AP_9$)	F(s)	**9.1** ± 0.1	**12.8** ± 0.2	**19.1** ± 0.1	**13.2** ± 0.1	**21.9** ± 0.2
	T(s)	576.1 ± 81.3	1238.5 ± 108.8	1496.4 ± 144.4	**1235.2** ± 91.9	2059.1 ± 319.4
	R	63.4 ± 9.4	103.1 ± 9.3	82.4 ± 8.3	100.2 ± 8.1	98.2 ± 15.9
	A	375.2 ± 43.7	606.7 ± 43.8	447.6 ± 29.6	565.6 ± 33.3	570.1 ± 73.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 7: Comparing AKFD to LAMAF over five problems from the Rovers domain. AKFD has been tuned removing different predicate sets. The best results are highlight in bold.

Table 8 shows the size of the original action space and the abstract action sets generated using different predicate sets to build the predicate abstraction. The results show that the combination of predicates `calibrated` and `at` generates the smallest abstract action set. However, this combination does not produce the best results for AKFD.

In conclusion, we believe that some predicates dominate over others depending on the structure of the planning task. In order to generate good abstractions analyzing

Planner	Problem				
	rovers 36 (14,80,40)	rovers 37 (14,80,40)	rovers 38 (14,85,57)	rovers 39 (14,95,62)	rovers 40 (14,100,68)
LAMAF	12096	22854	25577	26353	26371
AKFD ($P = AP_1$)	9914	20572	23157	23789	23635
AKFD ($P = AP_2$)	12096	22854	25577	26353	26371
AKFD ($P = AP_3$)	12096	22854	25577	26353	26371
AKFD ($P = AP_4$)	12096	22854	25577	26353	26371
AKFD ($P = AP_5$)	9763	12710	18296	13397	26371
AKFD ($P = AP_6$)	9914	20572	23157	23789	23635
AKFD ($P = AP_7$)	9914	20572	23157	23789	23635
AKFD ($P = AP_8$)	9914	20572	23157	23789	23635
AKFD ($P = AP_9$)	3457	4050	6419	5689	7044

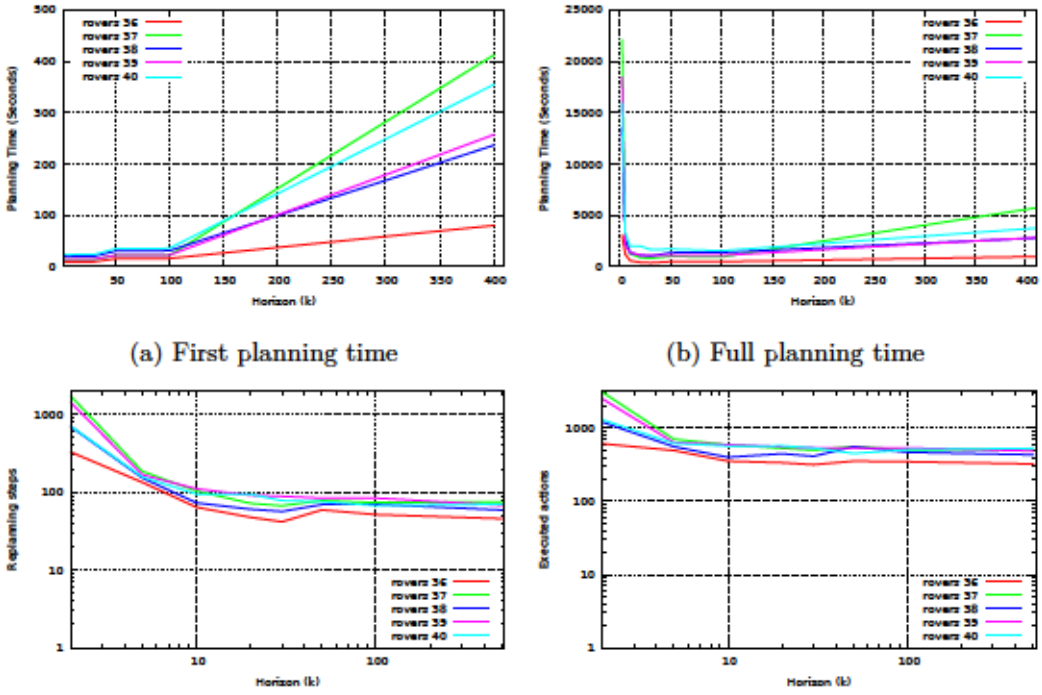
Table 8: Comparing the size of the action set generated by AKFD and LAMAF for five problems of the Rovers domain.

the planning task is important to identify which predicates can be used to simplify the complexity of the search process keeping the maximum information in the abstract planning task. Besides, we think that a good abstraction must be built using predicates with no domination among them in order to avoid unnecessary simplifications in the abstract space. In Chapter 7, we are going to analyze if it is possible to build good predicate abstractions automatically taking advantage of the information of the problem in order to choose the best predicate set.

6.5.6 THE HORIZON VALUE

The horizon value defines when the predicate abstraction is deployed during search. In this section, we analyze the effect of using different horizon values in order to analyze the effects over the planning time and the quality of the executed plan. We have run AKFD with different values of k = (2, 5, 10, 20, 30, 50, 100) and a predicate abstraction built using predicate at. Besides, we have chosen the value of k = 400 to represent the results of LAMAF in the figures. This value has been chosen given that the longest plan generated by LAMAF is 393 actions for rovers 37 problem and that means that AKFD will not switch to the abstract action set, and thus all search will be standard LAMAF search.

Figure 27 shows the evolution of four metrics depending on the horizon value for five different problems of the Rovers domain. We have not shown the coverage metric (C) because all problems are solved by all AKFD configurations and LAMAF. Figure 27 (a) shows the average first planning time for the whole cycle of planning and execution. AKFD reduces the first planning time by one order of magnitude in all problems in comparison with LAMAF. But, there are no significant changes on the first planning time among the different configurations of AKFD. Meanwhile, Figure 27 (b) shows the average full planning time for the whole cycle of planning and execution. In general, AKFD decreases the full planning time for most configurations. If the horizon value is smaller than 5, AKFD needs much more time than LAMAF to solve all problems. But, when the horizon value is bigger than 5, AKFD can solve all problems decreasing the full planning time. Besides, our approach decreases on a order of magnitude the planning and execution time on problem rovers 37 when the horizon value takes values between 20 and 30.



(a) First planning time

(b) Full planning time

(c) Replanning steps

(d) Executed actions

Figure 27: Evolution of four metrics during the whole cycle of planning and execution depending on the value of k for five problems from the Rovers domain. The x axis shows the value of k and the y axis shows the value of the metric. The horizon value of 400 corresponds to LAMAF due to the longest plan generated by LAMAF is 393 action for rovers 37 problem. Figures (a) and (b) are shown in decimal-scale and Figure (c) and (d) are shown in log-scale.

On the other hand, Figure 27 (c) shows that the number of replanning steps is similar regardless of the horizon value except for small values like 2 and 5. As we explained before, the accuracy of a plan is the difference between the original plan generated by LAMAF and the abstract plan generated by AKFD. If the horizon value is small, the number of replanning steps is huge due to the accuracy of the abstract plans. Nevertheless, the accuracy of the abstract plans increases as the horizon value is increased. In general, AKFD obtains better results than LAMAF decreasing the number of actions executed to solve the planning tasks when k=30. In conclusion, if AKFD is tuned with medium horizon values (20, 30 and 50) it solves planning tasks executing a similar number of actions than LAMAF, but decreasing in one order of magnitude the planning time. The data used to generate the different figures shown in Figure 27 are presented in Table 9.

The complexity of the planning task decreases during the cycle of planning and execution. Then, there is a point in which the planning task is simple enough to be solved by LAMAF in seconds. At this point, it is not useful to simplify the planning task using predicate abstractions. Figure 28 shows the average planning time for the first 60 iterations of the cycle of planning and execution for problem 40 using different values of k = (2, 5, 10, 20, 30, 50). These results show that LAMAF needs less replanning steps to solve the problem, because the plans generated by LAMAF do not contain abstract actions and the plans generated for each planning step are fully informed. Plans generated by the different configurations of AKFD need more replanning steps according to the accuracy of the sequential abstract plan, but in this case the planning time is always very short. Interestingly, there is an iteration during the planning and execution cycle at which the planning time for LAMAF is similar to the planning time for AKFD and after this iteration LAMAF takes less time to compute plans keeping the accuracy of the plan. These results suggest that AKFD does not improve over standard planning from scratch from that point on. For problem rovers 40, this point is around 40 planning steps. Besides, the computational cost of the planning task of AKFD (time) is constant regardless of the horizon value as we can see in Figure 29. This Figure shows the average planning time for the whole cycle of planning and execution for problem 40.

In conclusion, we think that the horizon value is an important parameter for VRP. These results shows that VRP can be easily parameterized by appropriately setting a value for k so that its behavior gradually transits from a more deliberative approach (using large values of k) to a more reactive approach (using small values of k). In the extremes, if k=1, VRP becomes an almost pure reactive system, while if k=∞, VRP behaves as the standard deliberative planner in which is based.

Planner	M	Problem				
		rovers 36 (14,80,40)	rovers 37 (14,80,40)	rovers 38 (14,85,57)	rovers 39 (14,95,62)	rovers 40 (14,100,68)
LAMAF	F(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	45.8 ± 5.4	75.60 ± 7.3	59.2 ± 7.3	68.2 ± 5.1	70.1 ± 10.2
	A	323.6 ± 24.2	512.20 ± 28.9	432.8 ± 42.3	**486.8 ± 21.2**	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=2)	F(s)	9.7 ± 0.1	14.7 ± 0.1	21.5 ± 0.1	15.3 ± 0.1	25.3 ± 0.1
	T(s)	3059.7 ± 325.8	22069.1 ± 9344.2	13503.3 ± 3226.8	18442.6 ± 8622.77	15922.1 ± 2466.2
	R	328.2 ± 33.67	1717.4 ± 715.7	695.4 ± 169.1	1423.3 ± 688.9	704.4 ± 113.7
	A	611.3 ± 64.81	3082.6 ± 1293.1	1203.60 ± 290.37	2499.5 ± 1233.3	1297.6 ± 192.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=5)	F(s)	9.7 ± 0.06	**13.8 ± 0.1**	**20.1 ± 0.1**	**14.5 ± 0.1**	**23.7 ± 0.3**
	T(s)	1225.3 ± 92.1	2311.6 ± 218.2	2789.9 ± 94.5	2083.1 ± 149.8	3350.5 ± 194.8
	R	134.2 ± 11.4	185.7 ± 16.5	152.7 ± 4.8	167.7 ± 13.5	157.3 ± 10.5
	A	494.6 ± 36.7	705.1 ± 62.2	557.1 ± 25.6	633.7 ± 43.9	617.7 ± 61.5
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=10)	F(s)	**9.6 ± 0.1**	**13.8 ± 0.1**	20.2 ± 0.1	14.6 ± 0.1	23.9 ± 0.1
	T(s)	591.5 ± 68.4	1271.3 ± 95.3	1347.7 ± 144.9	1380.1 ± 52.9	1996.5 ± 69.3
	R	64.3 ± 6.6	104.1 ± 8.8	73.1 ± 8.2	110.7 ± 5.8	95.1 ± 2.8
	A	354.1 ± 21.3	592.3 ± 25.7	400.1 ± 25.7	593.7 ± 23.5	568.3 ± 18.9
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=20)	F(s)	9.7 ± 0.1	13.9 ± 0.1	20.4 ± 0.1	14.6 ± 0.1	23.9 ± 0.1
	T(s)	436.4 ± 31.8	885.4 ± 34.1	1135.9 ± 220.5	1135.4 ± 120.7	2020.4 ± 267.1
	R	47.7 ± 3.4	72.1 ± 2.5	60.7 ± 11.9	91.3 ± 9.2	94.1 ± 12.6
	A	336.7 ± 18.8	528.7 ± 13.1	447.7 ± 54.6	553.7 ± 15.9	571.3 ± 56.8
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=30)	F(s)	9.7 ± 0.1	14.1 ± 0.1	20.4 ± 0.2	14.8 ± 0.1	23.9 ± 0.1
	T(s)	389.5 ± 31.9	**828.5 ± 61.9**	**1052.6 ± 319.1**	1115.1 ± 142.1	1668.8 ± 198.4
	R	41.7 ± 3.9	67.1 ± 4.2	56.7 ± 17.4	88.7 ± 12.5	78.1 ± 9.4
	A	316.5 ± 18.1	**500.1 ± 5.7**	**412.7 ± 48.8**	538.1 ± 42.6	534.3 ± 45.9
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=50)	F(s)	15.7 ± 0.9	22.2 ± 2.1	30.7 ± 0.2	22.4 ± 0.2	35.2 ± 0.1
	T(s)	501.1 ± 96.9	1025.3 ± 158.3	1377.7 ± 200.9	**1107.1 ± 149.1**	1727.8 ± 315.2
	R	59.3 ± 13.1	78.4 ± 12.4	69.6 ± 10.4	84.1 ± 11.6	75.8 ± 14.3
	A	354.2 ± 23.8	562.1 ± 35.9	558.1 ± 22.2	536.8 ± 63.6	**520.8 ± 21.6**
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=100)	F(s)	16.3 ± 0.8	22.3 ± 0.1	31.4 ± 0.3	22.9 ± 0.2	36.1 ± 0.2
	T(s)	507.2 ± 74.8	969.1 ± 98.8	1395.9 ± 299.1	1116.3 ± 115.5	**1561.6 ± 215.2**
	R	51.7 ± 8.9	73.6 ± 7.7	70.1 ± 15.3	84.2 ± 9.2	67.6 ± 8.8
	A	347.7 ± 29.8	503.8 ± 26.68	468.4 ± 43.1	535.8 ± 20.6	506.6 ± 41.9
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 9: Comparing AKFD using different horizons to LAMAF over five problems from the Rovers domain where predicate **at** has been removed manually. F corresponds to planning time for the first planning episode in seconds. T corresponds to the total execution time (including planning time) of all runs in seconds. R corresponds to the number of replanning episodes. A corresponds to the number of actions executed in the simulated environment and C corresponds to the number of problems solved. The best results are highlight in bold. Besides, for each problem we have defined three values which describe the complexity of the problem: (i) number of rovers; (ii) number of waypoints and (iii) number of goals.

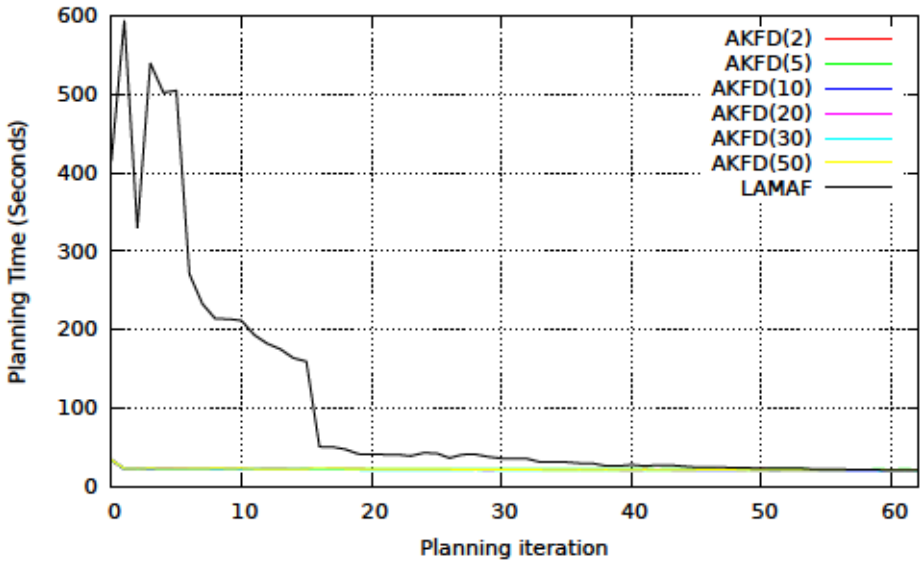


Figure 28: Average planning time for the first 60 iterations of planning and execution for Rovers problem 40. The x axis shows the number of planning steps and the y axis shows the planning time for each step. Both axis of the figure are shown in decimal-scale.

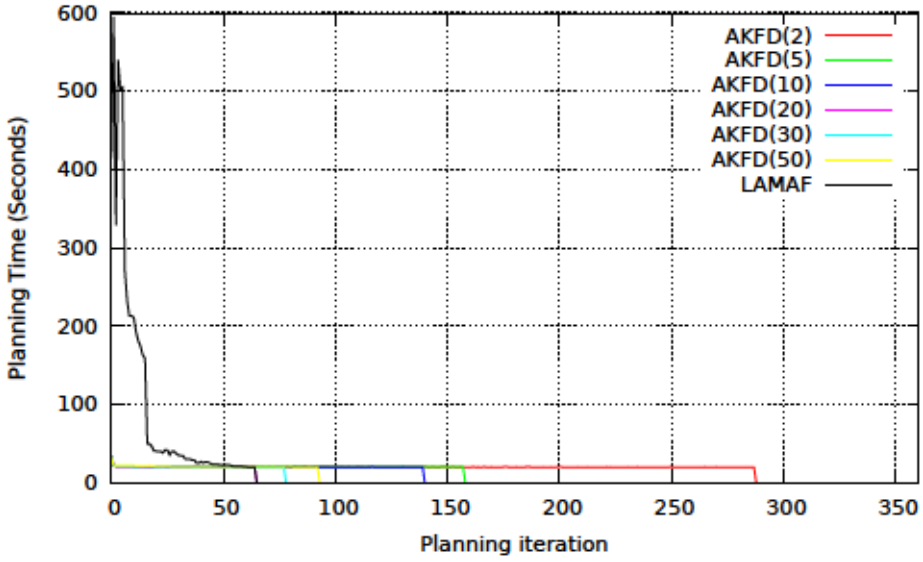


Figure 29: Average planning time for the whole cycle of planning and execution for Rovers problem 40. The x axis shows the number of planning steps and the y axis shows the planning time for each step. Both axis of the Figure are shown in decimal-scale.

6.5.7 THE ENVIRONMENT

The environment is not a real parameter of VRP, but its complexity can influence the other two parameters (the predicate set and the horizon value). In this section, we analyze how VRP works in more complex environments. Then, we evaluate the performance of VRP in two different ways: (1) increasing the percentage of errors and exogenous events of the environment; and (2) increasing the size of the environment.

First, we analyze the effect of increasing the number of failures and the number of exogenous events. Figure 30 shows the full planning time of problem rovers 40 using different values of k in four different environments. The environments have been configured using four different stochasticity levels (10%, 20%, 30% and 40%) which are used to increment both the number of errors and the number of exogenous events. The results show that VRP has a similar behavior regardless to the stochasticity level of the environment. In conclusion, a higher percentage of errors and exogenous events implies more replanning steps. And, then the total time obviously increases with the stochasticity level.

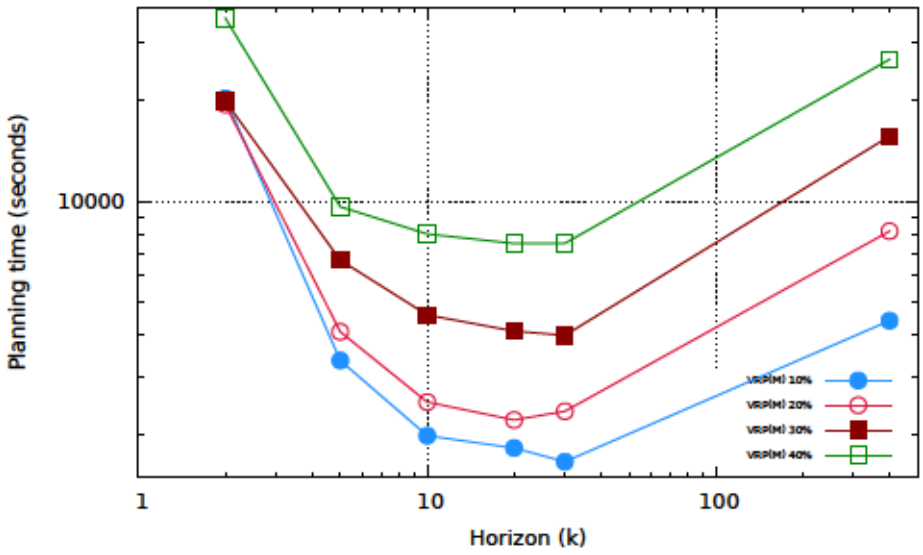


Figure 30: Evolution of the full planning time depending on the value of k for problem 40 with different stochasticity levels. The x axis shows the value of k and the y axis shows the full planning time. The value of 500 of the x axis corresponds to LAMAF, because the solution plan of problem 40 has less than 500 actions and no abstraction is applied to solve the problem. Both axis of the Figure are shown in log-scale.

Second, we analyze the effect of increasing the complexity of the environment in order to analyze the performance of VRP in more complex environments. Table 10

shows the results of five complex problems of the Rovers domain. These problems have been designed increasing the number of waypoints (175, 200, 250 and 300), the number of goals (98, 110, 145, 150 and 180) and the number of rovers (60). The results show that our approach can solve these difficult problems while LAMAF cannot solve any, because the first planning step consumes the maximum planning time of 1000 seconds. This means that VRP can handle difficult problems solving them partially at each planning step decreasing the planning time in a cycle of planning and execution.

Planner	M	Problem				
		rover 175 (175,15,145)	rovers 200 (200,15,110)	rovers 250 (250,15,180)	rovers 300 (300,15,150)	rovers 100 (100,60,98)
LAMAF	F(s)	1072.8 ± 0.4	1092.9 ± 0.8	1102.5 ± 0.2	1177 ± 0.6	1043.4 ± 0.3
	T(s)	-	-	-	-	-
	R	-	-	-	-	-
	A	-	-	-	-	-
	C	0, 0, 5, 0/5	0, 0, 5, 0/5	0, 0, 5, 0/5	0, 0, 5, 0/5	0, 0, 5, 0/5
AKFD (k = 10)	F(s)	**115.9 ± 0.4**	**137.85 ± 0.15**	**212.1 ± 0.3**	452.1 ± 2.4	**86.3 ± 0.1**
	T(s)	**16052.2 ± 7146.3**	19962.5 ± 1156.7	**21989.9 ± 953.6**	**19941.3 ± 1234.1**	6447.9 ± 384.9
	R	**155.4 ± 95.7**	196.7 ± 10.6	**186.5 ± 45.3**	**53.3 ± 35.6**	92.5 ± 5.5
	A	**972.4 ± 583.1**	1149.3 ± 7.6	**1075.33 ± 40.09**	**335.3 ± 225.6**	517.7 ± 14.2
	C	5, 0, 0, 0/5	5, 0, 0, 0/5	5, 0, 0, 0/5	5, 0, 0, 0/5	5, 0, 0, 0/5
AKFD (k = 20)	F(s)	123.8 ± 0.5	292.9 ± 0.48	-	335.7 ± 24	86.7 ± 0.1
	T(s)	16855.5 ± 545.6	**18838.1 ± 521.2**	-	22037.9 ± 1111.8	**4289.5 ± 207.5**
	R	178.3 ± 14.9	**172.4 ± 87.9**	-	72.6 ± 4.2	**59.5 ± 14.5**
	A	1287.3 ± 65.9	**1138.3 ± 48.5**	-	567.6 ± 14.9	**476.1 ± 71.4**
	C	5, 0, 0, 0/5	5, 0, 0, 0/5	0, 0, 0, 0/0	5, 0, 0, 0/5	5, 0, 0, 0/5

Table 10: Comparing AKFD to LAMAF on five difficult problems from the Rovers domain. The meaning of the metrics is the same as in Table 4. The best results are highlight in bold.

6.5.8 VRP IN DIFFERENT DOMAINS

This section reports the experimental results obtained in five different domains. For each domain, we have developed an error simulator which prevents the execution of the actions and generates exogenous events simulating a real world environment. Table 11 shows a summary for six different planning domains. In order to measure the performance of VRP and LAMAF in different domains, we have computed the score using the equations described in section 6.5.1. We compute the first planning time (F) and the planning time (T) using equation 2 and the replanning steps (R), the number of executed actions (A) and the coverage (C) using equation 3. The maximum number of points for each metric is 75, because for each domain we have executed 75 planning tasks. In general, VRP obtains much higher scores than those obtained by LAMAF. On one hand, if the horizon value is less than 5, VRP cannot

solve many of the problems of the benchmarks. As we described in section 6.5.6, VRP can behave like a reactive system when is tuned with small horizon values. However in these cases, the number of replanning steps is increased due to an excessive number of abstract actions in the solution plan. This fact can be an important problem in some domains in which the system is executing the same actions until the maximum execution time is reached.

On the other hand, if the horizon value is equal to or greater than 5, VRP can solve most of the problems of the benchmark decreasing the first planning time and the full planning time. However, our approach produces a small increment in both the number of planning steps and the number of actions executed in the environment except for some problem in which VRP needs less actions. Interestingly, no horizon value dominates over the others in any domain. This means that the features of the domain and/or the problem are important in order to choose **which** predicates can be used to build the abstractions and **when** abstractions should be deployed during search.

6.6 Discussion

In this Chapter, we have presented Variable Resolution Planning (VRP), a novel technique that uses an abstraction mechanism that dynamically removes some predicates during the planning process. Our approach is able to significantly decrease the computational effort of the search process. The predicate abstraction is only used in nodes of the search tree that are far away from the initial state of the search. The exact computation of a plan in those nodes is not crucial, given that most probably the actions will not be executable, since the execution system (robot) will find an execution failure earlier on. Abstractions are started to be used from a given planning horizon k. According to the results presented in this chapter, VRP can gradually control the relation between reasoning and execution using two parameters: (1) the predicate set, our approach can change the accuracy of the sequential abstract plan depending on what predicates are removed; and (2) the value of k, our approach can work as a reactive system, generating short plans of actions, or as a deliberative system, generating full sound plans.

There are several lines to research further in the context of this work in order to improve the results on domains with different features. Predicate abstractions are built using predicates that are chosen manually according to the knowledge about both the problem and the domain. Besides, we have observed that some predicates generate smaller abstract space than others as we showed in Table 8. This fact can be important in order to decrease the number of replanning steps or generate more informed sequential abstract plans. Interestingly, we can observe that some

Planner	Metrics	Domain					Total
		Rovers	Depots Robots	TidyBot	Port	Satellite	Total
LAMAF	F(s)	5.4	22.3	20.7	25.9	0	74.3
	T(s)	19.6	27.4	27.5	12.2	0	86.7
	R	**68.3**	42.2	28.1	34.5	0	173.1
	A	**65.9**	39.2	27.8	18.3	0	151.2
	C	**75**	45	30	45	0	195
AKFD (k=2)	F(s)	70.8	0	10	0	0	80.8
	T(s)	5	0	0.1	0	0	5.1
	R	18.3	0	0.3	0	0	18.7
	A	4.8	0	0.1	0	0	4.9
	C	**75**	0	10	0	0	85
AKFD (k=5)	F(s)	**74.3**	52.8	64.3	**73.3**	**67.3**	332.1
	T(s)	32	18	13.1	41.2	49.4	153.7
	R	50.4	20.3	20.3	67.6	68.2	226.8
	A	28.4	17.4	11	37.4	**51.2**	145.4
	C	**75**	55	65	**75**	70	340
AKFD (k=10)	F(s)	74.2	**64.9**	**72.6**	73.1	52.1	**336.9**
	T(s)	56.7	32.7	23.8	**59.6**	43.3	216.1
	R	64.7	34.4	32.6	66	54.7	252.5
	A	50.9	31.3	23.6	**56.4**	49.2	211.5
	C	**75**	71	**75**	**75**	59	**355**
AKFD (k=20)	F(s)	73.6	63.9	70.7	72.4	**49.4**	330.1
	T(s)	65	**49**	52.7	51.2	44.2	**262.1**
	R	63.9	**48.3**	54.5	71.5	**50.6**	288.8
	A	59	**47.4**	46.4	46.3	49.8	**248.8**
	C	**75**	68	**75**	**75**	52	345
AKFD (k=30)	F(s)	73.5	53.8	67.7	69.7	47.1	311.7
	T(s)	**67.7**	44.8	**54.3**	46.5	37.7	251
	R	67.4	47.2	**57.2**	**71.8**	48.5	**292.2**
	A	61.8	44.9	**50.6**	44.1	44.8	246.3
	C	**75**	59	**75**	**75**	50	334

Table 11: Results of planning and execution on six different planning domains. The first column corresponds to the planners used to solve the benchmark. The meaning of the metrics is the same as in Table 4. However, the results of each metric have been computed using the equations described in 6.5.1. The next five columns correspond to the score obtained for each domain. The last column corresponds to the total scores. The best ordering scores are highlighted in bold.

predicates improve the time performance of VRP more than others; this means that there is a relation between the structure of the problem and the predicate set used to build the predicate abstraction. In Chapter 7, we are going to explore how to generate predicate sets automatically using different techniques based on the information of the problem and the domain.

While the selected predicates have some influence over the quality of the plan after the horizon, the value of k influences the accuracy of the plan before the horizon and even the number of replanning steps. Small k values decrease the quality of the plan before the horizon, increasing the number of replanning steps. Meanwhile, large k values increase the quality of the plan before the horizon, but planning time increases. Besides, the value of the horizon has some influence over the planning time, which is extremely important in robotic environments where robots cannot spend much time on reasoning. In general, smaller values of k decrease planning time. If the value of k is changed dynamically during execution, VRP could decrease the number of replanning steps and increase the quality of the plan before the horizon. Some of the experiments show that k could be tuned during planning and execution depending on the size of the plan or the number of goals that must be reached. Interestingly, we can observe that different values of k solve better different groups of problems. This fact leads us to think that changing the value of the horizon during planning and execution dynamically could improve the performance of the process increasing the coverage and decreasing the full planning time.

7

GENERATING PREDICATE SETS AUTOMATICALLY

As described in the previous chapter, Variable Resolution Planning (VRP) is a novel technique that generates Sequential Abstract Plans (SAP) removing far future information. These plans are used to solve planning tasks in dynamic and stochastic environments in a planning and execution cycle in order to avoid failures and/or capture information about the environment which commonly cannot be known initially. However, VRP must be tuned with two parameters: (1) the horizon value which defines when the future information is removed; and (2) the predicate set which defines which information about the planning task is removed. In the previous chapter, these parameters have been chosen manually which implies an advanced knowledge about both the domain and the problem in order to choose the best configuration for VRP.

In this chapter we introduce a variation of Variable Resolution Planning where the predicate set is computed automatically using some information which is not encoded directly in the planning task. This means that this information must be extracted using other mechanisms or other sources. We try to extract this information from two different data structures extensively used in Automated Planning: (1) the landmark graph; and (2) a relaxed plan. We present also a new version of the planner AKFD which generates a sequential abstract plan using a predicate set computed automatically. We empirically compare this new version of AKFD to both the previous version and other approaches finding that our technique decreases the complexity of solving planning tasks in dynamic and stochastic environments generating the predicate set automatically using the information of the planning task.

7.1 Introduction

Planning tasks are described by means of two input files: a domain and a problem. The domain file contains a definition of the action model, a set of ungrounded predicates F and a set of types. Meanwhile, the problem file defines a set of objects, an initial state (I), and a set of goals (G). The information encoded in both files can be used to identify some features related to the type of predicates, but

it is not possible to identify features related to the complexity of the planning task or the relevance of each predicate according to both the initial state and the goals. Therefore, in order to generate good abstractions extracting other type of information about the planning task is necessary. In order to collect this information, we use different techniques that have been previously used to build heuristic functions. The information generated for these techniques is used to obtain information related to the relevance of predicates.

Figure 31 shows the new structure of the VRP architecture. The new version of the VRP planning system is composed of three different phases. In the first phase, called **Knowledge Gathering**, information about the planning task is extracted. This phase receives three parameters: the domain, the problem and the generation technique. This phase can collect data from two different data sources: (1) the landmark graph which produces a hierarchy of some grounded predicates that must be true to reach the goals; and (2) the relaxed plan which generates a relaxed sequence of actions where the delete effects of each action have been ignored. These data structures and their generation process are described in the section 7.2. The information extracted on the previous phase is used by the **Abstraction Generation** phase where a set of abstract actions and variables (abstract states) are built. Finally, the **Search Algorithm** phase is executed to generate a sequential abstract plan. Upon termination, the search algorithm either outputs a SAP or no solution.

7.2 Data extraction techniques

In this section, we describe the different mechanisms used to build the data structures from which we collect information to built the predicate set automatically. We have chosen two different data structures: (1) the landmark graph and (2) the relaxed plan.

7.2.1 The Landmark Graph

Landmarks can be defined as logical formulas (possible consisting of a single fact) that must be achieved in every valid plan. Landmarks can be defined as facts that must be achieved at some point of the solution plan before the goals are reached. It is possible to use landmarks to simplify the planning task. For example, if a robot must go from room A to room B and the rooms are connected by a door which is locked, the door must be open at some point in every solution plan. Hence, if a human tries to solve this task, it would most likely first find the way to open the door and then go to room B. This task can be decomposed into two small tasks, one

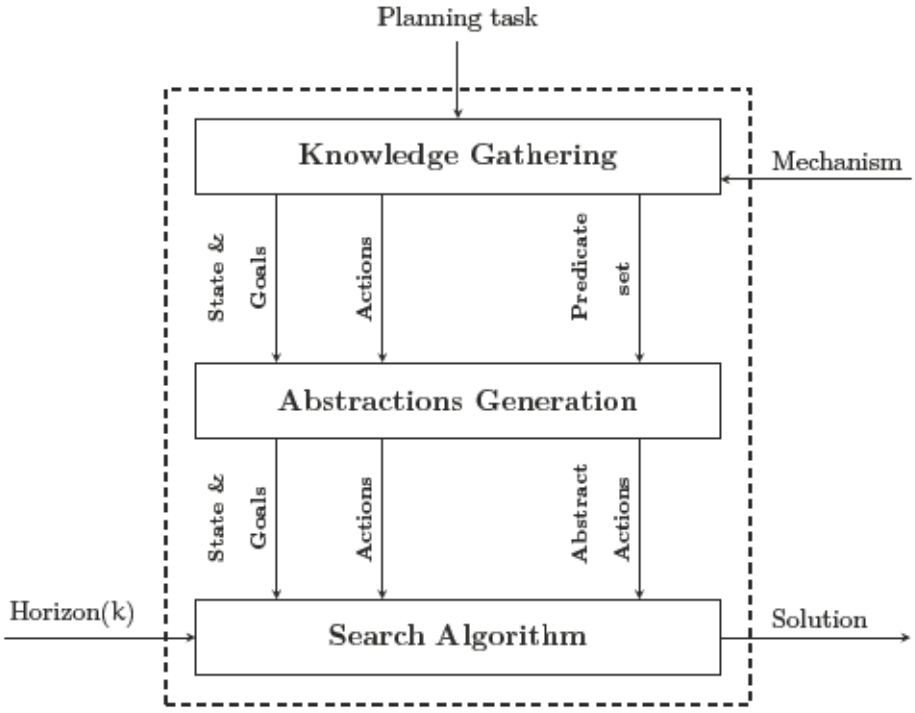


Figure 31: VRP's architecture.

of finding a way to open the door, and other to go from room A to room B when the door has been opened. Both tasks are usually easier to solve than the original task.

Landmarks were first described by Porteous et al. (2001) and were later studied in more detail by Hoffman (2004). Landmarks can be considered as subgoals that must be achieved in every plan. There are two kinds of landmarks: (1) fact landmarks; and (2) action landmarks. Both concepts are formally defined in the next two definitions:

Definition 20. (Fact Landmark) Let $\Pi = (F, A, I, G)$ be a planning task and let $f \in F$ be a fact. f is a fact landmark of the planning task Π, if for each valid plan π that solves Π, f is true at some point [1].

Definition 21. (Action Landmark) Let $\Pi = (F, A, I, G)$ be a planning task and let $a \in A$ be a grounded action. a is an action landmark of the planning task Π, if for every valid plan π that solves Π, $a \in \pi$.

[1] Facts in the initial state and facts in the goal state are always considered landmarks by definition.

Finding the complete set of landmarks for a planning task is PSPACE-complete (Hoffmann et al., 2004), but there are some methods that can efficiently compute a subset of the landmarks using a graph generated by the delete-relaxation heuristic. This subset of landmarks is generated as a graph in which some partial orders between fact landmarks are defined. The **landmark graph** is an important part of many of the techniques that exploit landmarks and describes the interactions and orders between landmarks. Besides, the landmarks graph is a directed graph composed of the fact landmarks of the planning task and the orders between them. Orders between landmarks are relations between two facts landmarks that represent the partial order in which they must be achieved. Different types of orders can be defined:

- Natural order: Let a and b be fact landmarks of a planning task Π, a is naturally ordered before b, denoted a $<_{nat}$ b, if in each successful plan in which a is true at some time i and b is true at some time j, j>i.
- Necessary order: Let a and b be fact landmarks of a planning task Π, a is necessarily ordered before b, denoted a $<_{n}$ b, if in each successful plan in which a is true at some time i then b is added at time i+1.
- Greedy-necessary order: Let a and b be fact landmarks of a planning task Π, a is greedy-necessarily before b, denoted a $<_{gn}$ b, if in each successful plan in which a must be true at some time i then b is true at time i+1, when b is first achieved.

Figure 32 shows a Logistics task composed of a set of locations which are grouped into two cities; City 1 contains three locations A, B and C, while city 2 contains locations D, E. Locations C and E are airports. Two types of vehicles are available: two trucks (t_1, t_2) and one plane (a_1). The goal consists on transporting the package p_1 from location B to location D.

A partial landmark graph for our running example is depicted in Figure 33. This graph includes facts landmarks where natural orderings are represented by bold arcs and necessary ordering are represented by dashed arcs.

Commonly, the landmark graph has been used to build domain-independent heuristics (Richter et al., 2008; Karpas and Domshlak, 2009). In this thesis, we are interested in the different mechanisms that these heuristic functions use to generate the landmarks graph. Several landmark generation techniques have been developed in order to collect the maximum number of landmarks:

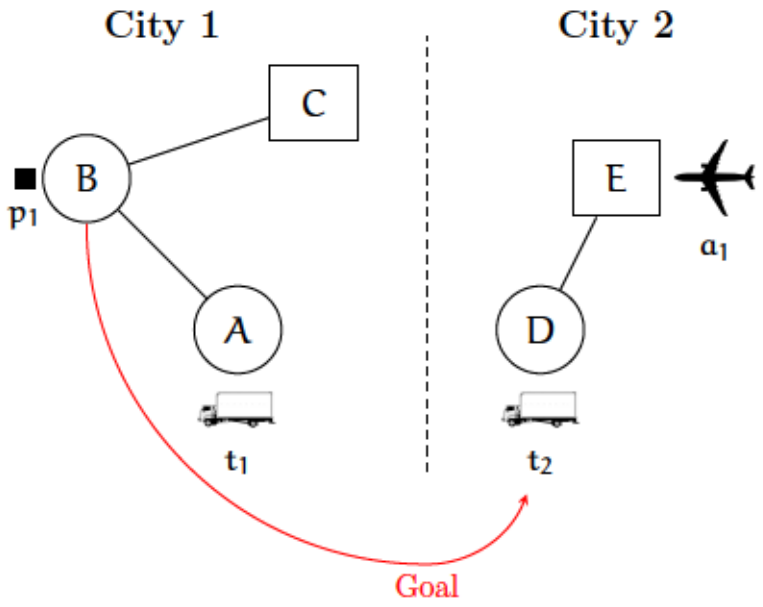


Figure 32: Deterministic planning task of the Logistics domain: the goal consists on transporting package p_1 from location B to D

.

- Zhu/Givan technique (Zhu and Givan, 2003) is an incomplete method for finding causal landmarks based on planning graph propagation. First, this technique identifies the action landmarks of the planning task. Action landmarks are represented as a conjunction of propositions (preconditions of the actions). Then, these propositions are analyzed in order to identify candidate fact landmarks. If a candidate is identified, a planning graph propagation process is performed to verify if the candidate is a fact landmark.
- h^m technique (Richter et al., 2008) is an incomplete method for finding fact disjunctive landmarks. This technique generates a set of landmarks using a queue which stores the predicates which can be landmarks. This queue is started with the facts in the goal. The algorithm analyzes each element of the queue checking if the element is a landmark until the queue is empty. During the checking process new landmarks can be added to the queue. The algorithm finishes when the queue is empty.
- Exhaustive technique is a complete method for finding all fact landmarks implemented in the FD planning system. This technique checks for each fact of the planning task if it is a fact landmark. The checking process is done using the relaxed plan.

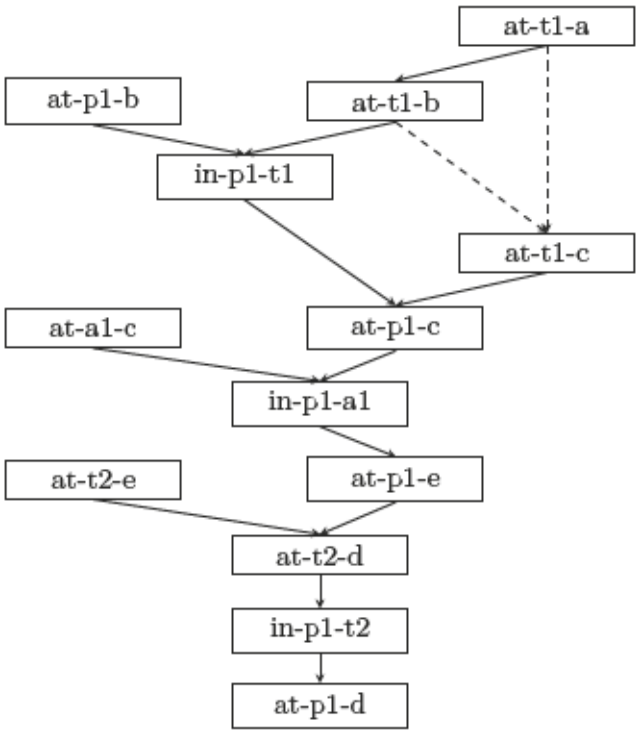


Figure 33: Partial landmark graph for the example task shown in Figure 1. Bold arcs represent natural orderings, dashed arcs represent necessary orderings.

7.2.2 The Relaxed Plan

A relaxed plan can be defined as a sequential plan of actions which has been computed by removing the delete effects of the actions of the Planning Task (Hoffmann and Nebel, 2001). These plans are built using the Planning Graph (Blum and Furst, 1997) which is a layered representation of the planning task. The Planning Graph is composed of different levels generated by alternating propositions and actions layers. The first layer includes all the propositions defined in the initial situation, then the second layer is composed of all the relaxed actions which can be applied in the previous layer. The third layer is composed of the propositions generated from applying the actions of the second layer. The generation process of the relaxed plan extends the different layers until a goal state is generated. Then, the algorithm goes over the planning graph from the goal state to the initial state generating the relaxed plan. Formally the relaxed plan can be defined as:

Definition 22. (Relaxed plan (Hoffmann and Nebel, 2001)) Let $\Pi = (F, A, I, G)$ be a planning task and let π_r be a plan. Then, π_r is a relaxed plan of Π if $\forall a \in \pi_r$ $del(a) = \emptyset$.

Figure 34 shows both the solution plan and the relaxed plan that solve the planning task depicted in Figure 32. This plan is composed of relaxed actions. There are some similarities between relaxed plans and sequential abstract plans. A relaxed plan can be considered as a sequential abstract plan which has been computed removing all dynamic predicates from the delete effects of the actions with a horizon value of 0. As we see in the Figure 34, the relaxed plan is shorter or has equal length to the solution plan.

1	drive(t_1, A, B)	drive(t_1, A, B)
2	load(p_1, t_1, B)	load(p_1, t_1, B)
3	drive(t_1, B, C)	drive(t_1, B, C)
4	unload(p_1, t_1, C)	fly(a_1, E, C)
5	fly(a_1, E, C)	unload(p_1, t_1, C)
6	load(p_1, a_1, C)	load(p_1, a_1, C)
7	fly(a_1, C, E)	unload(p_1, a_1, E)
8	unload(p_1, a_1, E)	drive(t_2, D, E)
9	drive(t_2, D, E)	load(p_1, t_2, E)
10	load(p_1, t_2, E)	unload(p_1, t_2, D)
11	drive(t_2, E, D)	
12	unload(p_1, t_2, D)	

Figure 34: Relaxed plan of the planning task depicted in Figure 32. Left: sequential plan. Right: relaxed plan.

7.3 VRP Algorithm

As shown in the previous chapter, predicates can be removed from the original planning task to decrease the size of the search space and generate a new smaller abstract search space. In order to build the predicate abstractions is mandatory to choose a predicate which can be built manually. The knowledge gathering phase has been included in the VRP architecture to provide some mechanism to generate a predicate set automatically. In the following, we only provide a detailed description of the knowledge gathering phase because the other phases are equal to the original VRP.

7.3.1 Knowledge Gathering

The first phase of the new version of VRP consists of gathering knowledge from the planning task to select predicates to abstract over. The relevance of each predicate depends of different factors: (i) the type of predicate – if the predicate is added or deleted during search –; (ii) if the information that describes the predicate is relevant to achieve the goals; and (iii) the information on a predicate is relevant to select other predicates. This information is collected during the first planning step of the planning and execution cycle and stored in a file. This file is used during the rest of the planning and execution cycle in order to generate the predicate abstractions which are used during search. This information can be extracted from different data structures. Then, we have defined two different approaches which extract some of these features about some predicates of the planning task.

7.3.1.1 Landmarks based Selection

The landmark extraction process generates a landmark graph which stores the information of each landmark and the order relationship between them. This data structure can be used to obtain some relevant information about the predicates as: (1) the number or occurrences of an ungrounded predicate in the graph; (2) the partial order between the different ungrounded predicates; (3) the complexity of generating a predicate according to its position in the graph. Regarding the example shown on Figure 23, the landmark set L is composed of landmarks related to the location of the rover, the state of the rover store and the analysis of rock and soil.

$$
\begin{aligned}
L(\Pi) = \{ & \texttt{at(r1 w01), at(r1 w30), at(r1 w10), have\_soil\_analysis(r1,w22),} \\
& \texttt{full(rs1), at(r1 w22), have\_rock\_analysis(r1,w10),} \\
& \texttt{empty(rs1), communicated\_soil\_data(w22),} \\
& \texttt{communicated\_rock\_data(w10)}\}
\end{aligned}
$$

There are several algorithms to compute landmarks (Richter et al., 2008; Karpas and Domshlak, 2009; Helmert and Domshlak, 2009; Keyder et al., 2010; Pommerening and Helmert, 2012). In this thesis. we have computed landmarks using the exhaustively technique described in the previous section. Once landmarks are computed, we build a subset of landmarks composed of dynamic predicates removing goal predicates. It is not necessary remove static predicates from the landmark set, so these predicates are not considered as landmarks. See landmark definition in section 7.2.1.

Definition 23. (Predicates set extracted from the landmark set). Let $\Pi = (F, A, I, G)$ be a planning task, $L(\Pi)$ the set of fact landmarks of Π and let name(p) be a function which gets the ungrounded predicate of a grounded predicate p. The predicate set to be abstracted, extracted from the landmark set is:

$$ps(\Pi) = \{n \mid l \in L(\Pi), l \notin G, n = name(l), l \in g(n, F)\}.$$

According to the previous definition, the predicates set for the planning task depicted on Figure 23 is

$$ps(\Pi) = \{\texttt{at, empty, full, have\_rock\_analysis, have\_soil\_analysis}\}$$

Additionally, we obtain additional information for each element of the predicate set ps, although this information has not been used in this version of the technique. The additional information is related to the number of occurrences and the relative order of the ungrounded predicates in the landmark graph, which is computed as the first occurrence of a grounded predicate of the type of the predicate. Algorithm 7.1 shows the pseudo-code of the generation process based on Landmarks. The candidate list, candidates, stores the information about the predicates collected. For each ungrounded predicate a list of grounded predicates is stored and their relative location in the Landmarks graph. Initially, the algorithm computes the landmark graph of the planning task Π (line 2). Then, the algorithm iterates the landmarks graph as a list. At each iteration, the corresponding predicate is analyzed in order to extract its information. First, the algorithm checks if the predicate is a goal (line 6). If the predicate is not a goal predicate, it is considered as a candidate predicate. Then, the algorithm retrieves the name of the predicate (line 7) which is used to identify the type of the predicate and searches in the candidate list if there is an instance of this predicate type. If there is not an instance, a new instance (line 10) is added to the candidate list (line 11) and the information about the grounded predicate is added (line 14). Finally, the information of the candidate list is used to build the predicate set of the planning task (line 14).

7.3.1.2 Relaxed Plan based Selection

The relaxed plan extraction process generates a relaxed plan which stores a set of ordered relaxed actions. This data structure can be used to obtain some relevant

Algorithm 7.1: Pseudo-code of the generation process based on Landmarks

```
input : Planning task: Π = (F, A, I, G)
output : Predicate set computed from the landmark graph
1  begin
2      Set graph ← generateLandmarkGraph(Π)
3      Set fileName ← "abstractions.sas"
4      Set candidates ← empty
5      foreach predicate ∈ graph do
6          if predicate is not goal then
7              name ← getName(predicate)
8              candidate ← getCandidate(candidates, name)
9              if candidate is empty then
10                 candidate ← generateCandidate(name)
11                 candidates ∪ candidate;
12             position ← getRelativePosition(graph, predicate)
13             params ← getParams(predicate)
14             candidate.addOccurrence(params, position)
15     return buildPredicateSet(fileName, candidates)
```

information about the predicates which are part of the preconditions and the added effects of the action: (1) the number of occurrences of the ungrounded predicates in the preconditions and the added effects of the action; and (2) the partial order between the different predicates according to their position in the relaxed plan. Regarding the example shown on Figure 23, the relaxed plan RP is depicted in Figure 35.

The relaxed plan is iterated by extracting all predicates from the effects of each action. As described below, the relaxed plan is computed by removing the delete effects of the actions, so we collect all predicates from the effects of each action. Then, the predicate set is built according to definition 24.

Definition 24. (Predicates set extracted from the relaxed plan). Let $\Pi = (F, A, I, G)$ be a planning task, let $\pi_r = (a_0, \ldots, a_n)$ be the relaxed plan of Π and let name(p) be a function which gets the ungrounded predicate of a grounded predicate p . The predicate set extracted from the relaxed plan is:

$$ps(\Pi) = \{n \mid \forall a \in \pi_r, rp \in Add(a), rp \notin G, n = name(rp)\}.$$

1	navigate(r1, w10, w11)
2	navigate(r1, w11, w21)
3	navigate(r1, w21, w22)
4	sample_rock(r1, rs1, w22)
5	navigate(r1, w21, w20)
6	comunicate_rock_data(r1, l1, w22, w20, w30)
7	navigate(r1, w20, w10)
8	sample_soil(r1, w20, w10)
9	communicate_rock_data(r1, l1, w10, w20, w30)

Figure 35: Relaxed plan of the planning task depicted in Figure 32.

According to the previous definition, the predicates set for the planning task depicted on Figure 23 is

$$ps(\Pi) = \{\texttt{at, full, have\_rock\_analysis, have\_soil\_analysis}\}$$

As in the previous technique, we collect more information during the generation process of the predicate set. This information is related to the number of occurrences of the predicate in the effects of the actions, the relative order in the plan and the average distance of each predicate to the goal state according to the action that added it. The algorithm 7.2 shows the pseudo-code of the generation process based on the Relaxed Plan. The candidate list, candidates, stores the information about the predicates collected. For each ungrounded predicate a list of grounded predicate is stored as well as its position in the relaxed plan and the action name. Initially, the algorithm computes the relaxed plan of the planning task Π (line 2). Then, the algorithm iterates the relaxed plan. At each iteration, the algorithm extracts the added predicates of each action (line 6). Each predicate is analyzed in order to extract its information, starting to check if the predicate is a goal (line 8). If the predicate is not a goal predicate, it is considered as a candidate predicate. Then, the algorithm retrieves the name of the predicate (line 9) which is used to identify the type of the predicate and searches in the candidate list if there is an instance of this predicate type. If there is not an instance, a new instance (line 12) is added to the candidate list (line 13) and the information about the grounded predicate is added (line 16). Finally, the information of the candidate list is used to build the predicate set of the planning task (line 17).

Algorithm 7.2: Pseudo-code of the generation process based on Relaxed Plan

```
input : Planning task: Π = (F, A, I, G)
output : Predicate set computed from the Relaxed Plan
1  begin
2      Set relaxedPlan ← computeRelaxedPlan(Π)
3      Set fileName ← "abstractions.sas"
4      Set candidates ← empty
5      foreach action ∈ relaxedPlan do
6          predicates ← getAddedPredicate(action)
7          foreach predicate ∈ predicates do
8              if predicate is not goal then
9                  name ← getName(predicate)
10                 candidate ← getCandidate(candidates, name)
11                 if candidate is empty then
12                     candidate ← generateCandidate(name)
13                     candidates ∪ candidate;
14                 position ← getPosition(relaxedPlan, predicate)
15                 params ← getParams(predicate)
16                 candidate.addOccurrence(params, position, getName(action))
17     return buildPredicateSet(fileName, candidates)
```

7.4 Empirical Evaluation

The evaluation environment has been configured using the light version of the PELEA architecture described in chapter 5.3 which uses both the MDPSim to emulate the execution of plans and the Error Simulator which increases the variability of the execution dynamics introducing exogenous events. The VRP approach described in this chapter has been implemented over the previous version of AKFD. The source code, written in C++, has been built as an extension of the processor of the previous version of AKFD [2].

The PELEA architecture has been configured as shown in Figure 36. The configuration of the PELEA architecture is similar to the previous one except for the inputs of the planning system which receives two different inputs: (1) the horizon value; and (2) the predicate generation mechanism. All these experiments were conducted in a Intel Xeon 2.93 GHZ Quad Core processor (64 bits) running under Linux. The maximum available memory for the planners was set to 8 GB.

[2] https://bitbucket.org/momartin/akfd stores the source code

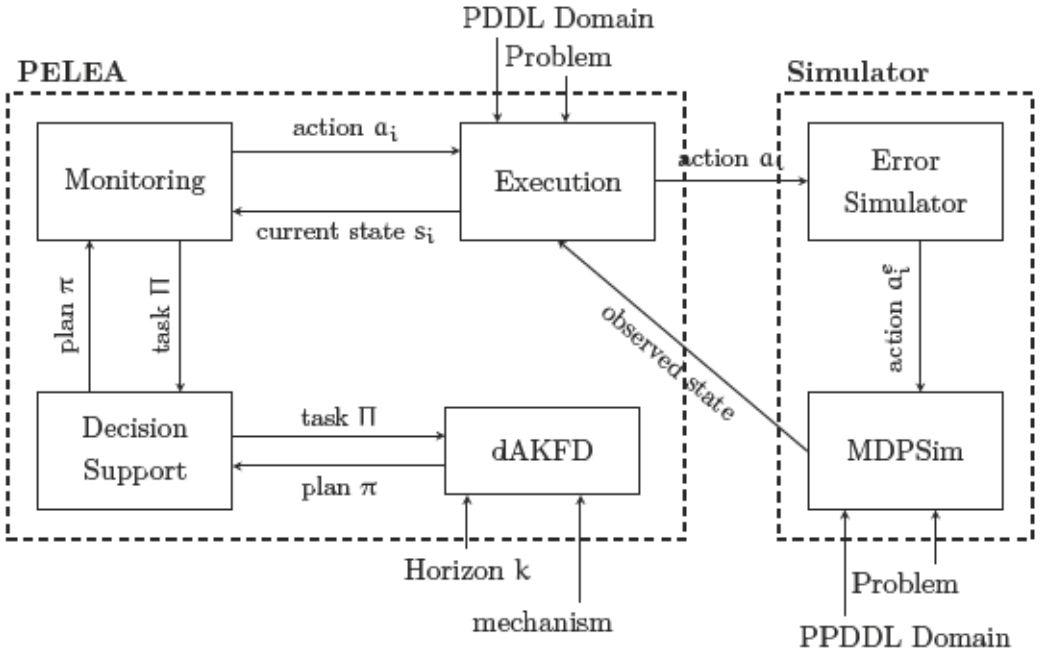


Figure 36: Light PELEA Architecture used to evaluate VRP with automatic predicate set generation.

Table 12 shows a summary for five different planning domains. In order to compare the performance of the different versions of VRP (Manual (M), Landmarks based (L) and Relaxed Plan based (R)) to LAMAF in different domains, we have computed the score using the equations described in section 6.5.1. We compute the first planning time (F) and the planning time (T) using equation 2 and the replanning steps (R), the number of executed actions (A) and the coverage (C) using equation 3. The maximum number of points for each metric is 75, because for each domain we have executed 75 planning tasks. In general, the manual version of VRP obtains better scores than the Landmarks version of VRP, the Relaxed Plan version of VRP and LAMAF. On one hand, if the horizon value is less than 5, VRP cannot solve many of the problems of the benchmarks, excepts for the Relaxed Plan version which can solve problems for all domains.

On the other hand, if the horizon value is greater than or equal to 5, the manual version of VRP can solve most of the problems of the benchmark decreasing the first planning time and the full planning time. Both the Landmarks version and the Relaxed Plan version of VRP obtain similar results in most of the domains, but the results get worse due to both the computation cost of the generation process of the predicate set and the excessive number of predicates included in the predicate set which can degenerate the sequential abstract plan. The Landmarks version improves

Planner	Met	Domain															Total		
		Rovers			Depots Robots			TidyBot			Port			Satellite					
		M	L	R	M	L	R	M	L	R	M	L	R	M	L	R	M	L	R
LAMAF	F(s)	5.4			22.3			20.7			25.8			0			74.3		
	T(s)	18.6			25.5			24.6			9.7			0			77.8		
	R	67.6			42.2			28.1			34.3			0			171.9		
	A	65.4			38.4			27.8			14.1			0			146		
	C	75			45			30			45			0			195		
AKFD (k=2)	F(s)	70.8	51.2	66.7	0	27.1	18.9	10	8.4	8	0	0	48.6	0	0	34.9	80.8	86.7	176.9
	T(s)	4.7	10.2	3.7	0	1.2	1.3	0.1	0.2	0.7	0	0	20.3	0	0	19.8	4.8	11.7	45.7
	R	18.1	26	13.9	0	3.8	6.6	0.3	0.5	2.2	0	0	46.8	0	0	40.4	18.5	30.3	110
	A	4.8	9.2	3.5	0	1.1	1.4	0.1	0.1	0.5	0	0	16.1	0	0	21.2	4.9	10.5	42.8
	C	75	75	75	0	30	30	10	15	12	0	0	65	0	0	44	85	120	226
AKFD (k=5)	F(s)	74.9	51.9	66.6	52.8	32.8	27.4	64.3	22.8	32.9	73.2	17.7	58.1	57.5	34.1	35.8	322.2	159.2	220.7
	T(s)	30.1	30.7	22.2	17.8	16.5	12.7	9.8	5.9	8	30.5	33.3	36.6	43.7	19	22.1	141	105.4	101.7
	R	49.9	47.7	42.3	20.3	22.3	17.4	18	9	14.7	54	55.4	55	68.1	39.6	33.9	210.3	173.9	163.3
	A	28.2	27.5	22.1	17	14	11	8.6	4.4	7	35.4	38.8	31.2	50.5	20.7	19.9	139.7	105.5	91.2
	C	75	75	75	55	70	41	65	43	47	75	75	75	70	43	36	340	306	274
AKFD (k=10)	F(s)	74.2	51.9	66	64.9	37.9	37.3	72.6	39.7	48.3	73	14.6	57.6	44.3	35.2	24	329.1	179.3	233.3
	T(s)	53.7	50.9	45.9	30.2	49.2	23.5	19.2	28.6	20.4	57	25.5	44.8	37.3	26.5	17.4	197.4	180.5	152
	R	64.1	58.1	61.4	33.8	46.6	24.1	29.4	34.6	25.5	52.5	42.5	58	54.6	46.8	24.3	234.5	228.6	193.3
	A	50.5	46.6	44.2	30.4	45	21.8	19.1	23.9	17.6	53.8	20.1	39.8	48.3	35	17.2	202.1	170.6	140.6
	C	75	75	75	71	75	59	75	73	69	75	60	75	59	53	25	355	336	303
AKFD (k=20)	F(s)	73.6	49.8	65.5	63.8	34.3	48.2	70.7	40.7	50.9	72.3	14.5	56.7	40.7	30	14.4	321.2	169.4	235.7
	T(s)	62	61.1	59.4	39.3	40.3	47.1	39.6	51.4	47.8	49.2	25	54.8	37	31.1	10.2	227.2	208.9	219.3
	R	63.3	64.3	65.7	44	44.1	45	49.5	59.9	49.3	57.6	41.9	55.6	50.4	44.1	14.5	264.9	254.3	230.1
	A	58.6	57.3	58.7	42.8	39.5	45.8	37	45.7	42.5	44.2	32	51.8	48.7	37	13.1	231.3	211.6	212
	C	75	75	75	68	60	75	75	75	75	75	60	75	52	45	15	345	315	315
AKFD (k=30)	F(s)	73.5	51.2	65.5	53.8	24.3	40.7	67.7	31.4	50.6	69.6	13.5	56.3	39.7	23.8	5.9	304.3	144.2	219
	T(s)	64.5	62.3	54.4	35.4	28.4	40	42	52.3	55.4	44.9	36.8	47.1	31.4	19.2	9.7	218.3	199	206.6
	R	66.8	64.1	58.4	41.9	34.7	47.1	53	51.4	52.7	58.8	57.7	58.1	48.3	37.9	14	268.8	245.9	230.3
	A	61.4	58.2	54.5	39.4	29.9	42.7	41.2	43.7	53.1	42.6	53	42.9	44	35.6	14.1	228.6	220.5	207.3
	C	75	75	75	59	45	64	75	75	75	75	75	75	50	40	15	334	310	304

Table 12: Results of planning and execution on five different planning domains. The first column (P) corresponds to the planners used to solve the benchmark. The second column (M) corresponds to the different metrics measured for each domain. The meaning of the metrics is the same as in Table 4. The next sixteen columns correspond to the score obtained for each domain, where M corresponds to the Manual selection, L corresponds to the Landmark generation technique and R corresponds to the Relaxed Plan generation technique. The last three columns correspond to the total scores of the three techniques. In bold, we highlight the best results per domain.

the score of the manual version in TidyBot domain and the Relaxed Plan version improves the score of the manual version in the Port domain. In conclusion, we show that both the Landmarks version and the Relaxed Plan version of VRP can generate automatically predicate abstractions which decrease the planning time in dynamic and stochastic environments. This is an important advantage over the manual version of VRP in which the predicate set is chosen manually. Additionally, we can observe that there are important differences between the results of each automatic technique. This means that the predicate set generated by each technique is different.

7.4.1 The generation time

In this section, we analyze the computational effort of generating the predicate set using the different techniques described in this chapter. Figure 37 shows the average time of the first planning time for all domains. On one hand, we can observe that the Landmarks based technique increases the time of the first planning time according to the complexity of the domains. Besides, there are some domains in which the generation process based on Landmarks needs excessive time to compute the landmark graph. As an example, the generation process consumes the maximum planning time allowed when AKFD tries to solve some problems of the Port domain. This fact is related to the complexity of the planning task making the planning task unsolvable.

On the other hand, the Relaxed Plan based technique increases the first planning time too. However, the increase in time is lower than the increase in time introduced by the Landmarks technique. This means that the generation process of the relaxed plan requires less time than the generation process of the landmark graph. Additionally, it is important to emphasize that in some domains, like Port and Satellite, the time needed to generate the first sequential abstract plan is shorter than the time needed for the other configurations of AKFD and LAMAF. Thus, the computational cost of generating the predicate set using the relaxed plan of the planning task is not very expensive. Finally, the results show that in some domains (Port and Satellite) the average time of generating the first sequential abstract plan using the relaxed plan is smaller than the average time of the other configurations of AKFD.

7.4.2 The predicate set

In this section, we analyze what predicates are chosen from the planning task using the different techniques described in this chapter. Table 13 shows the average

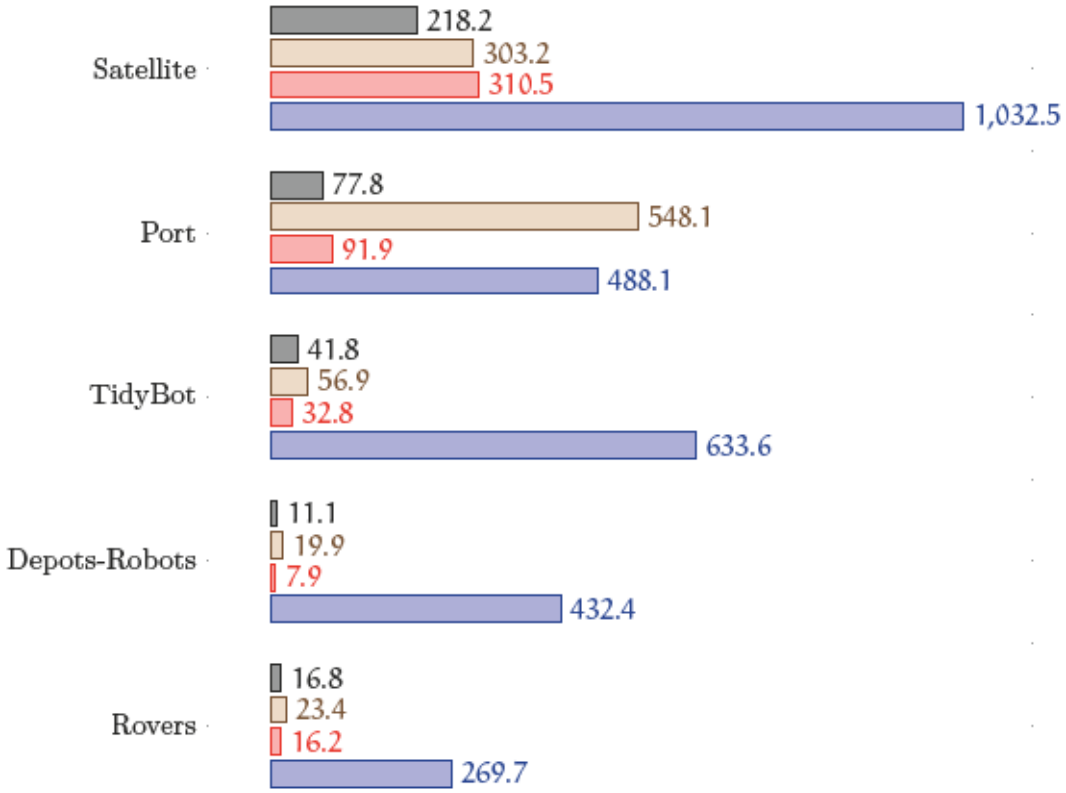


Figure 37: Average time of the first planning time in five different domains. The average time is computed using the first planning time of 75 runs for each domain. In blue the average of the first planning time of LAMAF is shown. In red the average of the first planning time of AKFD (k=10) using a manual technique to generate the predicate set is shown. In light brown the average of the first planning time of AKFD (k=10) using the Landmarks technique to generate the predicate set is shown. In grey the average of the first planning time of AKFD (k=10) using the Relaxed Plan technique to generate the predicate set is shown.

number of occurrences of each grounded predicate collected from the different planning tasks. The results show that both techniques generate similar predicate sets. However, the number of occurrences of each predicate collected for each technique is different. Interestingly in most domains, there are some predicates which are only collected using one of the techniques. This means that the predicate set generated is different depending of the technique chosen to generate it.

7.5 Summary

In this chapter, we have presented a variation of Variable Resolution Planning (VRP) which uses a generation technique that dynamically generates the predicate set which is used to build predicate abstractions. On one hand, the Landmarks based technique builds the landmark graph of the planning task by deploying an exhaustive algorithm which checks if each predicate of the search space is a landmark. The landmark graph is explored in order to collect the information about the predicates which is used to generate the predicate set. This process is only conducted during the first planning step of the planning and execution cycle in order to avoid excessive time effort during the different planning steps. The time needed to compute the landmark graph is excessive in some domains according to the maximum planning time. Besides, the computation effort increases according to the complexity of the planning task. In general, the Landmarks based technique is able to significantly decrease the computational effort of the search process if the time required to build the landmark graph is not very high.

On the other hand, the Relaxed Plan based technique builds the relaxed plan for the initial state of the planning task. The relaxed plan is explored in order to collect the information about the predicates added for each action of the plan. The generation process of the Relaxed Plan always needs much lees time than the Landmark based technique. This is an important feature in order to generate specific predicate abstractions automatically as quick as possible. Additionally, predicate abstractions decrease the computational effort of the planning task solving difficult problems automatically.

In conclusion, we can say that the automatic generation process of the predicate set offers an important advantage against manual generation. This process can generate a predicate set automatically according to the structure of the planning task. This is an important advantage in complex domains which are difficult to analyze. Additionally, the predicate sets built using both generation techniques can be used to generate predicate abstractions which can solve a large number of planning tasks of the benchmark with similar results than the manual version. However, the predicate sets are generated using a large number of predicates which

Domain	Predicates	Manual	Landmarks	Relaxed Plan
Rovers	have-rock-analysis	0	384 ± 159.6	20.8 ± 4.9
	at	1	14 ± 0	66.4 ± 13.1
	have-soil-analysis	0	337 ± 138.1	21.8 ± 8.3
	empty	0	11.6 ± 1.5	0
	calibrated	0	15.8 ± 1.3	3.8 ± 1.7
	have-image	0	284.4 ± 57.9	14.6 ± 2.1
	full	0	0	42.6 ± 10.3
Depots Robots	empty	0	76.4 ± 42.8	48.8 ± 24.4
	at-pod	0	22 ± 10.9	11 ± 2.5
	at-robot	1	3.8 ± 1.6	48.8 ± 24, 4
	free	0	3.8 ± 1.6	11 ± 2.5
	carries	0	0	13.8 ± 2.8
TidyBot	gripper-obstacle	0	104.6 ± 13.5	6.2 ± 0.9
	parked	0	2 ± 0	0
	base-pos	1	102.6 ± 14.1	19.2 ± 0.6
	object-pos	0	43.8 ± 10.8	3.4 ± 0, 57
	gripper-rel	0	9 ± 0	2.8 ± 0.5
	not-pushing	0	1 ± 0	0
	not-pushed	0	1 ± 0	0
	cart-pos	0	100.4 ± 14.2	0.8 ± 0.5
	base-obstacle	0	102.6 ± 14.2	20 ± 0.5
	holding	0	3.4 ± 0.5	3.4 ± 0.6
	on-cart	0	4 ± 0	0
	gripper-empty	0	1 ± 0	3.4 ± 0.5
	pushing	0	0	0.8 ± 0.5
Port	clear	0	112.8 ± 86.6	61.2 ± 40.6
	on-dock	0	528 ± 547.7	0
	available	0	9.6 ± 9.4	26 ± 16.7
	at	1	30 ± 24.5	26 ± 16.7
	lifting	0	15 ± 12.2	35.2 ± 23.8
	height	0	20, 6 ± 17.1	26 ± 16.7
Satellite	power-on	1	19.8 ± 2.1	11 ± 1.5
	power-avail	0	15 ± 1.2	0
	calibrated	0	19.8 ± 2.2	11 ± 1.5

Table 13: Average number of occurrences of each grounded predicate in the predicate set.

can introduce an excessive simplification of the planning task when abstractions are applied. This fact can degenerate the accuracy of the sequential abstract plans incurring in more replanning steps. Then, there is room to improve both techniques using the additional information collected about the predicates in order to build better predicate sets.

Part III

CONCLUSIONS AND FUTURE WORK

8

CONCLUSIONS

This thesis sets out to prove our initial hypothesis:

> It is possible to solve a planning task in dynamic and stochastic (real-world) environments using deterministic planning by generating k-bounded plans by means of abstractions which are built removing some predicates that represent future information about the environment

This chapter summarizes the contributions over the state-of-the-art that we have presented to prove this hypothesis.

8.1 Contributions

The contributions of this thesis are grouped in three classes:

1. Variable Resolution Planning Model

In this thesis, we introduced the concept of Variable Resolution Planning (VRP), a novel technique in which both the action space and the state space are pruned by removing some predicates from the description of the planning task. We have presented a theoretical model to build and represent predicate abstractions for deterministic planning. This work has been published in:

- Moisés Martínez, Fernando Fernández, and Daniel Borrajo (2012). "Variable resolution planning through predicate relaxation" In proceedings of the ICAPS workshop on Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx). Atibaia, Sao Paulo, Brazil, (pages 5–12). (Chapter 6)
- Moisés Martínez, Fernando Fernández, and Daniel Borrajo (2013). "Selective Abstraction in Automated Planning" In proceedings of Second Annual Conference

on Advances in Cognitive Systems (Cogsys). Baltimore, USA, (pages 133–147). (Chapter 6)

2. Variable Resolution Planning Planners

In this thesis, we introduced two approaches of Variable Resolution Planning based on different techniques to generate the predicate set which is used to build predicate abstractions. The first approach is based on generating the predicate set manually (Chapter 6). The second approach is based on generating the predicate set using the information about the planning task collected using two different data structures: (1) the landmark graph; and (2) the relaxed plan (Chapter 7). Both approaches are implemented by extending the Fast Downward planning system in the Abstract K Fast Downward (AKFD) planning system which implements VRP over SAS^+. This planner has two extensions: (1) a first version of AKFD which deploys predicate abstraction by choosing the predicate set and the horizon value manually; and (2) a second version which deploys predicates abstraction where the predicate set has been generated automatically and the horizon value has been chosen manually. This work has been published in:

- Moisés Martínez, Fernando Fernández, and Daniel Borrajo (2016). "Planning and Execution through Variable Resolution Planning" In Journal of Robotics and Autonomous Systems, volume 83, (pages 214-230). (Chapter 6 and Chapter 7)

3. Empirical Analysis

We provided a rich empirical evaluation of the proposed approaches for two different tasks: (1) to solve different planning and execution tasks where the predicate set is chosen manually; and (2) to solve different planning and execution tasks where the predicate set is chosen automatically. Besides, we have implemented a Error Simulator which can introduce both more complex failures and exogenous events in the environment during execution. Different domains were used in our empirical evaluation, including domains proposed in this thesis and benchmarks from the deterministic planning community. Besides, the PELEA architecture implemented to conduct the empirical evaluation has been deployed as control system in different real scenarios. This work has been published in:

- Luis J. Manso, Luis V. Calderita, Pablo Bustos, Javier García, Moisés Martínez, Fernando Fernández, Adrían Romero-Garcés and Antonio Bandera (2014), "A

general-purpose architecture to control mobile robots“, In proceedings of the 15th Workshop of physical agents (WAF), León, Spain, (pages 105–116). (Chapter 5)

- Moisés Martínez, Fernando Fernández, and Daniel Borrajo (In press). “Planning and Execution through Variable Resolution Planning” In Journal of Robotics and Autonomous Systems. (Chapter 6 and Chapter 7)

- Nerea Luis, Sofía Herreo and Moisés Martínez. “Robot Collaboration in a Warehouse Environment through Planning and Execution” In The IJCAI-2016 Workshop on Autonomous Mobile Service Robots, New York, USA. (Chapter 6)

9

FUTURE WORK

Throughout this thesis we have outlined ideas for future research. In this chapter, we identify more general ideas of future work that apply to the thesis as a whole. We have defined an approach that uses abstraction based on removing some predicates in order to simplify the planning task. These abstractions are built removing some predicates from the structure of the planning task after a horizon has been reached. There are a number of ways in which this work can be extended.

9.1 Improving the predicate set generation mechanism

Variable Resolution Planning can generate a predicate set using two different mechanisms. Both techniques are able to extract critical information about the predicates (number of occurrences, relative order with the other predicates, distance to the goals, etc), but we have only collected the fact name of the grounded predicates. This means that the predicate sets generated for both techniques build similar predicate sets for all problems of each domain. Besides, in some domains the time needed to generate the predicate set is huge depending of what generation technique is used.

A future direction is to improve the generation techniques in two ways: (1) decreasing the time needed to generate the predicate set by analyzing other different landmark graph generation algorithms, like the landmark generation method introduced by Keyder (Keyder et al., 2010); and (2) increasing the information used to generate the predicate set in order to choose a smaller predicate set where predicates are more related with the structure of the problem. It would be really interesting to analyze all information which can be collected from both data structures in order to choose the best predicate set. This fact can help to build better abstractions which decrease the number of replanning step during execution and increase the accuracy of the sequential abstract plan.

9.2 Automatically Changing the predicate set during execution

As we describe below, Variable Resolution Planning solves a planning task in a planing and execution cycle where the complexity of each planning step is decreased by using a predicate abstraction. VRP generates a sequential abstract plan for each planning step, but the accuracy of the plan can be degraded depending of which predicates are chosen to build the predicate abstractions. Our approach uses different automatic techniques to build the abstraction, but these mechanisms choose all ungrounded predicates of the candidate set. The selection technique does not perform any analysis of which predicates are better than others or any analysis in order to detect if the abstraction generated by one predicate dominates other abstractions. In addition, the structure of the planning task can change depending of the evolution of the planning and execution cycle. These changes can produce that the predicate set computed in the initial state could not be the most appropriate one in other steps of the planning and execution cycle.

An interesting improvement to Variable Resolution Planning consists of computing different predicate sets during planning and execution in order to generate a good abstraction according to the state of the planning task. This improvement should compute a new predicate set whenever the system detects that the planning task has changed enough. For instance, the predicate set generation process can be performed periodically if the generation time is not very high. In theory, this process should compute better abstractions which increase the accuracy of the sequential abstract plans decreasing the number of replanning steps.

9.3 Automatically Changing the horizon during execution

As we describe below, the horizon value k has an important influence on the quality of the plan before the horizon and even over the number of replanning steps. Small k values decrease the quality of the plan before the horizon, increasing the number of replanning steps. Meanwhile, large k values increase the quality of the plan before the horizon, but planning time increases too. Besides, we believe that the value of the horizon has some influence over the planning time, which is extremely important in robotic environments where robots cannot spend much time on reasoning. In general, smaller values of k decrease planning time, meanwhile higher values of k increase the accurate of the abstract plan increasing the planning time.

It would be interesting to analyze the behaviour of Variable Resolution Planning if the horizon value is changed dynamically during planning and execution. The horizon value can be computed using information about the current state, the size of the last plan and/or the number of goals that must be reached. In our opinion,

if the value of k is changed dynamically during execution, VRP could decrease the number of replanning steps and increase the quality of the plan before the horizon.

9.4 Heuristic functions based on Variable Resolution Planning

Heuristic functions are commonly used in Automated Planning to estimate the cost from a given state to a goal state. Red-black heuristic (Katz et al., 2013) tries to improve the delete relaxation heuristic selecting which variables are relaxed. This heuristic function divided the set of state variables into two disjoints sets (red and black). During the planning process, red variables are treated as in delete relaxation abstraction while black variables are not relaxed. Predicate abstractions can be tuned depending on what variables are marked as red or black. In Chapter 6, we described the concept of **Predicate Abstraction** which removes some predicates from the structure of the planning task in order to build an abstract space simpler than the original one.

A future direction consists of generating a heuristic function based on combining Variable Resolution Planning and GraphPlan. The predicate abstraction generates an abstract action space where some predicates are removed from both the preconditions and the effects of the actions. The Abstract Relaxation heuristic consists of generating a relaxed plan using two different action spaces in a Graphplan-style algorithm. Then, the relaxed plan is built using original actions in the first part of the relaxed plan and abstract actions in the final part of the relaxed plan. This approach decreases the information used in the final part of the relaxed plan, but preserves full information in the first part. In our opinion, this heuristic function could increase the information used to build the relaxed plan which can be useful to build a better heuristic function.

9.5 Variable Resolution Planning in real-world scenarios

Variable Resolution Planning has been designed to solve real-world problems. In this thesis, we have tried to simulate real-world environments by introducing two types of events: (1) probabilistic actions which simulate errors in the action's execution related to the structure of the environment or due to failures in the actuators of the automatic systems; and (2) exogenous events which try to simulate the effects produced by other agents and/or the process of collecting unknown information about the environment.

It would be interesting to deploy VRP in combination with LPELEA architecture as a deliberative controller of a real robotic system or an automatic agent for

a video-game. In both cases, the interaction with the environment is one of the most important processes, because the control system must face two important problems: (1) changes in the environment which are produced to other agents (robots, humans, agents, etc) which cannot be predicted, and (2) quick responses to offer a realistic interaction with the environment. Besides, new information can be discovered during execution changing the structure of the planning task and/or the goals of the problem. Finally, it is important to emphasize that real-world environments are described using a large amount of information which increases the complexity of the environment and therefore of the problems to be solved in these environments. In our opinion, VRP can be used to solve complex problems as we have shown previously.

Part IV

APPENDIX

A

THE ABSTRACT K FAST DOWNWARD PLANNER

The Abstract K Fast Downward (AKFD) planner is based on the Fast Downward planning system (Helmert, 2006). This planner accepts as input planning tasks encoded in the propositional fragment of PDDL 2.2 (Fox and Long, 2003), including ADL conditions and effects, derived predicates (axioms) and action costs (Helmert and Geffner, 2008). AKFD consists of four separate components:

- The translation module transforms a propositional planning task encoded in PDDL into a finite domain planning task.
- The preprocesor module builds a set of data structures using the finite-domain representation generated by the translation module. These data structures represent the domain transition graphs of the state variables and the successors generators, one for each action set. This module uses two input parameters: (1) the output file generated by the translation module and (2) the predicate set which can be generated manually or automatically.
- The predicate generator module is a variation of the search module that generates the predicate set used to build the abstract planning task. This module can build the predicate set using two different data structures: (1) the Landmarks graph and (2) the Relaxed Plan.
- The search module generates an sequential abstract plan using a greedy best-first search in order to find a solution as quickly as possible.

These components are implemented as separate programs which are executed in an specific sequence. The order in which these programs are executed defines the version of VRP. Figure 38 shows the execution sequence of the different versions of VRP. The manual version of VRP (AKFD) consists of a sequence of three modules. The translation and preprocesor modules correspond to the Predicate Generation phase and the search module corresponds to the Search Algorithm phase. Meanwhile, the dynamic version of VRP (AKFD) consists of a sequence of five modules. The translation, preprocesor and predicate generation modules correspond

to the Knowledge Gathering phase. The second preprocesor module corresponds to the Predicate Generation phase and the search module corresponds to the Search Algorithm phase.

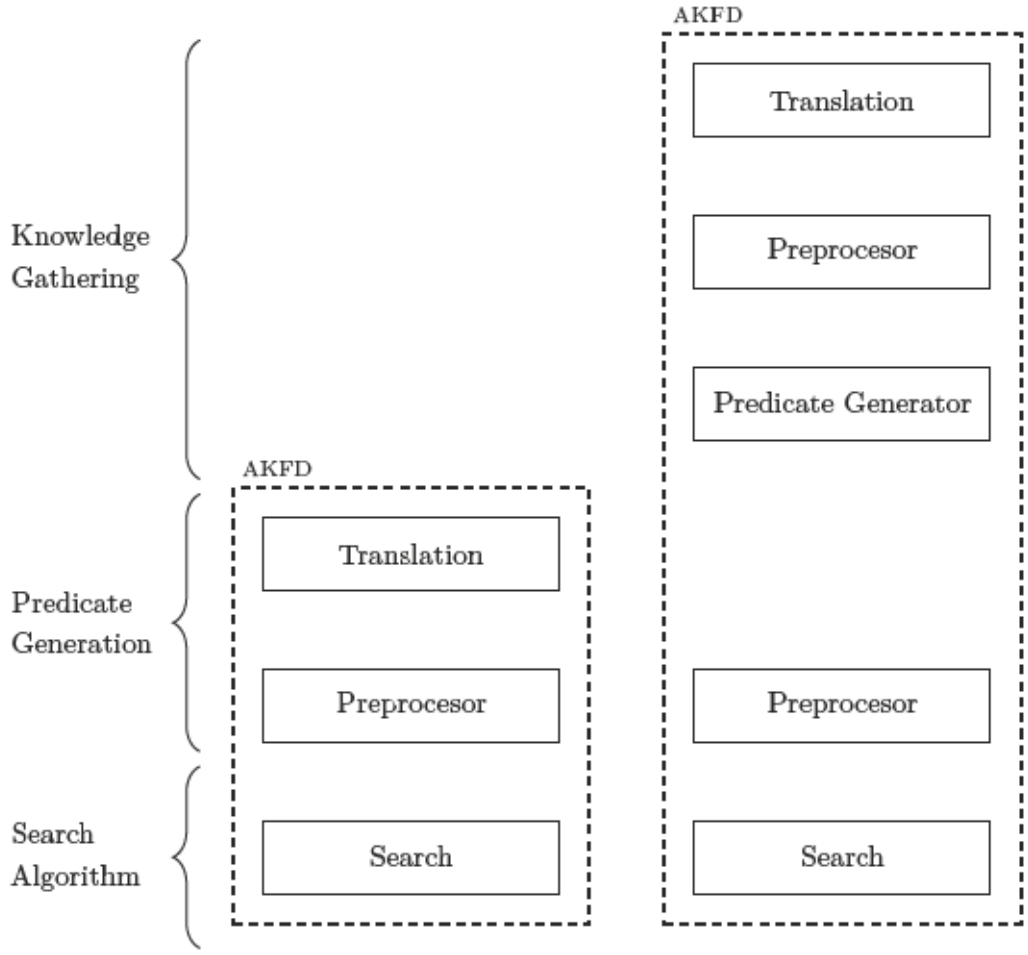


Figure 38: Execution sequence of the different version of VRP.

B

HOW TO DEPLOY THE LIGHT PELEA ARCHITECTURE

This Annex describes how to download, configure and use the LPELEA architecture described in this dissertation. The LPELEA architecture is made available under the GNU Public License (GPL). If you want to use the architecture in any way that is not compatible with the GPL, you will have to get permission from the author of this dissertation.

B.1 Installation guide

Then, it is possible to use the LPELEA architecture in any operative system. However, we recommended to use in Linux (Ubuntu 14.04 o higher) which is the main platform for which the architecture is developed and tested.

- Step 1: Installing Java

  The first step consists on installed Java version 1.7 or higher using the next commands (this sequence of commands are for Ubuntu 14.04 or Debian 8).

```
$ sudo echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu xenial
  main" | tee /etc/apt/sources.list.d/webupd8team-java.list
$ sudo echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu xenial
  main" | tee -a /etc/apt/sources.list.d/webupd8team-java.list
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
  EEA14886
$ sudo apt-get update
$ sudo apt-get install -y oracle-java7-installer
```

- Step 2: Installing dependencies

  The second step consists on installing some important dependencies which are necessary to download and compile the LPELEA architecture.

```
$ sudo apt-get install maven git
```

- Step 3: Download LPELEA source code

  The third step consists on downloading the source code. We recommended to define a destination folder (DIRNAME) which must be empty if exists. If the destination folder does not exist, it will be created by the clone command.

```
$ git clone https://bitbucket.org/momartin/pelea.git DIRNAME
```

- Step 4: Compiling the source code

  The four step consists on compiling the source code using the next commands.

```
$ cd DIRNAME
$ mvn compile
```

- Step 5: Integrating the AKFD planning system

  The five step consists on integrating the AKFD planning system into the LPELEA architecture (This step is optional, because you can use another planner developing if you implement a wrapper). We are going to download and compile the planning system. We commonly locate the different planning systems into the `planners` folder inside to the folder in which the architecture have been downloaded in the step 2. But, the planning system can be located in a different folder. Linux is the main platform for which the planner is developed. All features should work without restrictions under Linux. We recommended to used the decision support node in a Linux machine.

```
$ sudo apt-get install cmake g++ g++-multilib make python
$ flex bison
$ git clone https://bitbucket.org/momartin/akfd.git DIRNAME
$ cd ./planners/akfd
$ ./build.py
```

B.2 Configuring and running LPELEA

The LPELEA architecture is composed of a set of nodes which are interconnected between them. In order to run each node, it is necessary to define a XML configuration file. This file describes a set of properties which are used by the different nodes which are part of the architecture. These properties are defined using term nodes. Each term node has two attributes: (1) a name which describes the name of the property; and (2) a value which describes the value of the property. An example of a configuration file is depicted in Figure 39. This example shows a set of global properties and a set of properties which are specific for the node identified as **AKFD**. Commonly, each node has a set of specific properties.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
        <term value="192.168.1.100" name="IP"/>
        <term value="30520" name="PORT"/>
        <term value="NOT" name="TEMPORAL"/>
        <term value="/home/plg/pelea/problems/warehouse/domain.pddl" name="DOMAIN"/>
        <term value="/home/plg/pelea/problems/warehouse/p11.pddl" name="PROBLEM"/>
        <term value="/home/plg/pelea/experiment/" name="OUTPUT_DIR"/>
        <term value="/home/plg/pelea/temp/" name="TEMP_DIR"/>
        <term value="AKFD" name="NAME"/>
        <term value="1" name="ROUNDS"/>
        <term value="PARTIAL" name="STATE"/>
        <term value="ON" name="DEBUG"/>
        <nodes>
                <node type="planner" id="AKFD">
                        <term value="AKFD" name="PLANNER_NAME"/>
                        <term value="/home/plg/pelea/planners/akfd/" name="PLANNER_DIR"/>
                        <term value="org.pelea.planners.AKFD" name="PLANNER_CLASS"/>
                        <term value="0" name="PLANNER_MODE"/>
                        <term value="1000" name="MAX_PLANNING_TIME"/>
                </node>
                ...
        </nodes>
</configuration>
```

Figure 39: Example of a configuration file of LPELEA architecture.

B.2.1 Global Parameters

The global parameters define properties which are common to all nodes. Table 14 shows the meaning of the global properties.

B.3 How to run LPELEA

This section describes how to run a node in LPELEA architecture. The meaning of the options is:

- c: XML configuration file.
- n: Unique name of the node. This corresponds to the id of the node in the configuration file.
- p (optional): Numeric id for job generation. This parameters is only used to execution and monitoring nodes.

Name	Type	Default value	Description
IP	String		The host name in which the monitoring node is running.
PORT	Int		The port on which the monitoring node accepts client connections.
TEMPORAL	Enumerate	NOT	The type of actions. This means that each action has a maximum execution time. The available values are NOT (0) and YES (1).
DOMAIN	String		The location of the PDDL domain file.
PROBLEM	String		The location of the PDDL problem file.
OUTPUT_DIR	String	./temp	The directory in which LPELEA output files are stores.
TEMP_DIR	String	./temp	The directory in which LPELEA temporal files are stores.
NAME	String	LPELEA	the name used to identify information in output files.
ROUNDS	Int	1	The number of whole execution of the problems.
STATE	Enumerate	FULL	The structure of the state information collected to the execution nodes. LPELEA allows two types of states: (1) Full (FULL) which means that each execution node send global information about the state of the environment; (2) Partial (PARTIAL) which means that each execution node send partial information about the state of the environment.
DEBUG	Enumerate	OFF	Enable debug information in the nodes. The available values are OFF (0) and ON (1).

Table 14: Global properties of the LPELEA architecture

```
java -jar LPELEA.jar -c <file> -n <name> -p <numeric_id>
```

C

DETAILED RESULTS FROM CHAPTER 6

In this appendix, we show the detail results for the different planning domains used in the empirical evaluation of the Chapter 6.

C.1 Detailed results for Rovers domain

Planner	M	Problem				
		rovers 36 (14.80.40)	rovers 37 (14.80.40)	rovers 38 (14.85.57)	rovers 39 (14.95.62)	rovers 40 (14.100.68)
LAMAF	F(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	45.8 ± 5.4	**75.60 ± 7.3**	59.2 ± 7.3	**68.2 ± 5.1**	**70.1 ± 10.2**
	A	323.6 ± 24.2	**512.20 ± 28.9**	432.8 ± 42.3	**486.8 ± 21.2**	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 2)	F(s)	11.4 ± 0.1	18.9 ± 1.0	23.6 ± 0.1	19.1 ± 0.0	29.5 ± 0.2
	T(s)	1932.1 ± 656.3	4378.2 ± 2337.5	3289.7 ± 839.3	2458.8 ± 745.8	4731.7 ± 1109.3
	R	184.8 ± 58.0	284.0 ± 146.6	152.0 ± 38.3	142.8 ± 37.6	189.6 ± 44.5
	A	551.8 ± 114.9	909.4 ± 228.6	547.8 ± 50.2	645.2 ± 70.7	676.4 ± 66.2
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 5)	F(s)	11.4 ± 0.1	18.6 ± 0.1	24.2 ± 0.2	19.2 ± 0.1	29.0 ± 0.1
	T(s)	757.4 ± 114.6	1832.4 ± 220.4	1762.8 ± 164.3	1556.2 ± 217.9	2720.7 ± 415.7
	R	73.6 ± 11.0	123.2 ± 13.4	84.0 ± 7.8	98.2 ± 8.5	111.0 ± 17.5
	A	411.4 ± 30.4	666.4 ± 51.8	474.0 ± 22.2	600.8 ± 29.3	571.6 ± 45.5
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	F(s)	11.3 ± 0.2	18.6 ± 0.1	24.0 ± 0.1	20.6 ± 0.0	29.1 ± 0.2
	T(s)	594.6 ± 89.5	1313.0 ± 204.4	1517.2 ± 257.8	1289.1 ± 136.6	2050.4 ± 464.5
	R	58.3 ± 8.9	90.8 ± 14.6	73.2 ± 12.6	86.2 ± 10.1	84.4 ± 19.3
	A	377.7 ± 46.9	593.6 ± 44.7	475.8 ± 36.6	594.6 ± 63.3	524.2 ± 39.0
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	F(s)	11.4 ± 0.1	18.4 ± 0.1	25.4 ± 0.2	21.0 ± 0.1	29.9 ± 0.1
	T(s)	461.7 ± 44.9	1224.4 ± 207.4	1357.6 ± 138.7	1224.5 ± 166.7	2182.7 ± 283.8
	R	45.4 ± 4.5	86.6 ± 14.5	66.0 ± 6.8	83.2 ± 10.5	89.2 ± 11.5
	A	350.6 ± 12.7	602.2 ± 52.8	449.8 ± 41.7	568.4 ± 50.6	581.8 ± 34.6
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	F(s)	11, 5 ± 0, 1	19, 4 ± 0, 1	24, 9 ± 0, 1	20, 3 ± 0, 1	30.1 ± 0.4
	T(s)	440, 6 ± 30, 6	1130, 2 ± 144.8	1359.2 ± 144.5	1164.9 ± 178.0	21572.2 ± 262.1
	R	43.4 ± 2.9	81.6 ± 11.0	65.8 ± 7.0	79.9 ± 12.4	91.3 ± 9.3
	A	343.6 ± 28.6	572.0 ± 63.2	478.5 ± 12.2	570.9 ± 41.1	573.7 ± 42.9
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 15: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Rovers domain removing predicate `have_image` manually.

Planner	M	Problem				
		rovers 36 (14,80,40)	rovers 37 (14,80,40)	rovers 38 (14,85,57)	rovers 39 (14,95,62)	rovers 40 (14,100,68)
LAMAF	F(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	45.8 ± 5.4	75.60 ± 7.3	59.2 ± 7.3	68.2 ± 5.1	70.1 ± 10.2
	A	323.6 ± 24.2	512.20 ± 28.9	432.8 ± 42.3	486.8 ± 21.2	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=2)	F(s)	11.9 ± 0.5	22.2 ± 0.1	26.3 ± 0.1	26.7 ± 0.1	31.7 ± 0.2
	T(s)	469.1 ± 97.8	1199.3 ± 120.0	1516.9 ± 340.5	1260.5 ± 82.0	1796.3 ± 407.2
	R	46.5 ± 10.7	82.4 ± 9.7	72.2 ± 16.2	85.4 ± 3.4	73.2 ± 17.1
	A	354.8 ± 67.2	594.0 ± 88.6	483.8 ± 58.4	574.6 ± 26.0	538.8 ± 59.1
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=5)	F(s)	12.0 ± 0.1	21.2 ± 0.0	26.3 ± 0.2	26.7 ± 0.1	31.7 ± 0.1
	T(s)	444.6 ± 28.5	1076.9 ± 155.7	1228.6 ± 84.3	1177.7 ± 92.2	1851.7 ± 143.8
	R	44.0 ± 2.8	73.6 ± 9.3	59.8 ± 4.5	82.2 ± 7.1	74.8 ± 6.5
	A	350.2 ± 15.8	574.4 ± 29.5	463.5 ± 16.5	558.6 ± 51.1	520.8 ± 17.7
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=10)	F(s)	11.6 ± 0.1	20.3 ± 0.1	25.3 ± 0.1	20.1 ± 0.1	31.8 ± 0.1
	T(s)	535.4 ± 63.5	1205.5 ± 230.5	1437.1 ± 252.4	1112.2 ± 126.9	1693.9 ± 190.9
	R	52.3 ± 6.7	85.2 ± 16.0	68.6 ± 12.9	74.6 ± 10.5	69.6 ± 7.9
	A	353.0 ± 27.7	588.6 ± 79.9	486.0 ± 55.5	564.8 ± 35.8	549.2 ± 42.2
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=20)	F(s)	13.7 ± 0.0	19.7 ± 0.1	27.0 ± 0.1	21.6 ± 0.2	31.8 ± 0.1
	T(s)	526.2 ± 82.1	1074.3 ± 148.4	1302.2 ± 279.9	1184.3 ± 105.0	1918.1 ± 273.4
	R	49.8 ± 8.0	76.8 ± 11.4	62.0 ± 13.5	81.8 ± 7.7	77.6 ± 10.5
	A	374.4 ± 20.5	565.6 ± 40.3	445.8 ± 39.0	628.5 ± 31.1	552.2 ± 27.9
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=30)	F(s)	12.9 ± 0.1	21.7 ± 0.1	26.8 ± 0.1	21.7 ± 0.1	31.9 ± 0.2
	T(s)	510.5 ± 78.6	1135.8 ± 155.3	1478.2 ± 187.1	1180.6 ± 172.1	1834.5 ± 232.7
	R	50.0 ± 7.4	79.4 ± 10.8	72.0 ± 9.5	79.4 ± 11.2	74.2 ± 12.1
	A	348.6 ± 34.9	569.0 ± 28.4	503.0 ± 49.6	556.0 ± 46.2	551.6 ± 29.2
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 16: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Rovers domain removing predicate `have_rock_sample` manually.

Planner	M	Problem				
		rovers 36 (14,80,40)	rovers 37 (14,80,40)	rovers 38 (14,85,57)	rovers 39 (14,95,62)	rovers 40 (14,100,68)
LAMAP	P(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	45.8 ± 5.4	75.60 ± 7.3	59.2 ± 7.3	68.2 ± 5.1	70.1 ± 10.2
	A	323.6 ± 24.2	512.20 ± 28.9	432.8 ± 42.3	486.8 ± 21.2	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=2)	P(s)	10.8 ± 0.2	15.9 ± 0.1	23.6 ± 0.1	15.6 ± 0.1	28.9 ± 0.3
	T(s)	8602.7 ± 5780.1	16943.5 ± 2651.3	12166.4 ± 3790.5	8123.7 ± 2037.4	10560.4 ± 3309.0
	R	872.0 ± 574.3	1263.4 ± 189.7	616.6 ± 196.0	619.4 ± 156.1	437.2 ± 142.6
	A	1559.4 ± 896.1	2326.6 ± 352.7	1174.6 ± 322.8	1197.4 ± 310.0	920.4 ± 199.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=5)	P(s)	10.7 ± 0.0	16.3 ± 0.1	23.0 ± 0.1	15.6 ± 0.1	28.8 ± 0.1
	T(s)	1325.4 ± 140.5	2578.2 ± 83.1	2606.9 ± 301.7	2129.6 ± 156.8	3552.0 ± 323.7
	R	133.4 ± 14.1	189.8 ± 5.6	130.6 ± 15.8	158.8 ± 12.3	150.6 ± 13.5
	A	514.8 ± 65.7	745.2 ± 16.6	515.0 ± 43.1	680.2 ± 54.5	598.8 ± 52.0
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=10)	P(s)	10.9 ± 0.3	16.3 ± 0.1	23.7 ± 0.2	15.7 ± 0.2	28.4 ± 0.1
	T(s)	704.9 ± 62.1	1403.2 ± 171.3	1623.9 ± 115.3	1560.8 ± 63.9	2275.6 ± 174.7
	R	71.2 ± 7.4	103.6 ± 12.0	79.8 ± 5.9	116.2 ± 4.8	95.8 ± 6.8
	A	385.0 ± 23.3	596.2 ± 51.2	466.4 ± 30.2	664.0 ± 32.3	569.6 ± 12.5
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=20)	P(s)	10.8 ± 0.0	17.4 ± 1.6	23.4 ± 0.1	15.7 ± 0.1	30.5 ± 0.4
	T(s)	543.8 ± 95.7	1154.7 ± 217.0	1537.1 ± 107.2	1058.3 ± 147.0	1800.1 ± 130.1
	R	54.8 ± 9.8	85.4 ± 15.6	75.6 ± 5.6	78.4 ± 11.5	74.4 ± 6.0
	A	383.6 ± 33.4	579.0 ± 43.3	504.8 ± 42.1	554.0 ± 36.2	544.4 ± 24.0
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k=30)	P(s)	10.7 ± 0.0	16.3 ± 0.1	23.0 ± 0.1	15.6 ± 0.1	28.8 ± 0.1
	T(s)	1325.4 ± 140.5	2578.2 ± 83.1	2606.9 ± 301.7	2129.6 ± 156.8	3552.0 ± 323.7
	R	133.4 ± 14.1	189.8 ± 5.6	130.6 ± 15.8	158.8 ± 12.3	150.6 ± 13.5
	A	514.8 ± 65.7	745.2 ± 16.6	515.0 ± 43.1	680.2 ± 54.5	598.8 ± 52.0
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 17: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Rovers domain removing predicate `have_soil_sample` manually.

Planner	M	Problem				
		rovers 36 (14.80.40)	rovers 37 (14.80.40)	rovers 38 (14.85.57)	rovers 39 (14.95.62)	rovers 40 (14.100.68)
LAMAF	F(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	45.8 ± 5.4	75.60 ± 7.3	59.2 ± 7.3	68.2 ± 5.1	70.1 ± 10.2
	A	323.6 ± 24.2	512.20 ± 28.9	432.8 ± 42.3	486.8 ± 21.2	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 2)	F(s)	11.3 ± 0.1	17.6 ± 0.1	23.2 ± 0.2	21.8 ± 0.1	28.7 ± 0.1
	T(s)	870.3 ± 173.9	1424.2 ± 171.6	1861.5 ± 144.4	1636.7 ± 288.3	2958.4 ± 169.4
	R	85.2 ± 17.0	100.4 ± 9.9	89.8 ± 7.3	98.8 ± 12.4	123.2 ± 7.2
	A	353.4 ± 31.0	594.0 ± 69.2	447.4 ± 26.8	588.8 ± 8.9	553.8 ± 41.5
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 5)	F(s)	11.3 ± 0.1	16.9 ± 0.1	24.7 ± 2.5	21.8 ± 0.2	27.8 ± 0.1
	T(s)	730.9 ± 95.5	1374.0 ± 334.6	1637.0 ± 145.3	1170.8 ± 171.9	2314.4 ± 256.9
	R	73.2 ± 10.3	99.0 ± 24.4	78.8 ± 6.2	82.2 ± 11.1	96.0 ± 10.9
	A	391.6 ± 32.3	598.8 ± 77.6	462.5 ± 21.8	535.6 ± 18.8	522.6 ± 37.5
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	F(s)	11.2 ± 0.2	16.9 ± 0.1	23.4 ± 0.0	17.8 ± 0.1	27.9 ± 0.2
	T(s)	566.1 ± 79.0	1320.0 ± 170.8	1635.7 ± 191.9	1251.5 ± 211.7	1932.9 ± 188.3
	R	56.5 ± 8.2	94.8 ± 13.7	80.2 ± 9.5	90.0 ± 15.1	80.4 ± 8.2
	A	349.7 ± 34.6	600.8 ± 48.7	481.2 ± 34.3	558.0 ± 48.4	519.0 ± 36.6
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	F(s)	11.3 ± 0.1	17.3 ± 0.1	24.1 ± 0.1	19.1 ± 0.1	30.5 ± 0.2
	T(s)	548.0 ± 110.6	1156.9 ± 69.6	1325.8 ± 205.3	1165.7 ± 207.9	1808.3 ± 241.8
	R	54.2 ± 11.3	83.4 ± 5.6	65.0 ± 10.5	82.4 ± 14.7	75.6 ± 10.2
	A	359.2 ± 26.1	584.8 ± 45.2	450.4 ± 37.6	563.2 ± 37.8	549.6 ± 25.5
	C	15.0.0.0/5	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	F(s)	13, 0 ± 3, 2	18, 3 ± 0, 1	23, 5 ± 0, 2	18, 1 ± 0, 1	32.1 ± 0.3
	T(s)	406, 0 ± 58, 4	1187, 9 ± 97, 4	1243, 0 ± 91, 7	1118, 1 ± 146, 8	1843.2 ± 212.9
	R	39, 8 ± 5, 9	86, 2 ± 7, 5	59, 8 ± 4, 4	79, 0 ± 9.8	72.4 ± 12.4
	A	341.4 ± 24.3	620.0 ± 39.5	456.0 ± 19.6	567.5 ± 48.6	539.2 ± 21.8
	C	15.0.0.0/5	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 18: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Rovers domain removing predicate `calibrated` manually.

Planner	M	Problem				
		rovers 36 (14.80.40)	rovers 37 (14.80.40)	rovers 38 (14.85.57)	rovers 39 (14.95.62)	rovers 40 (14.100.68)
LAMAF	P(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	45.8 ± 5.4	75.60 ± 7.3	59.2 ± 7.3	68.2 ± 5.1	70.1 ± 10.2
	A	323.6 ± 24.2	512.20 ± 28.9	432.8 ± 42.3	486.8 ± 21.2	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 5)	P(s)	9.7 ± 0.1	19.5 ± 0.1	22.8 ± 0.3	14.6 ± 0.1	13.9 ± 0.1
	T(s)	1044.9 ± 115.8	2503.9 ± 395.1	3217.9 ± 128.2	3487.3 ± 1933.7	2562.6 ± 500.4
	R	114.6 ± 12.8	137.4 ± 22.2	151.4 ± 7.2	285.1 ± 163.6	207.8 ± 40.3
	A	444.4 ± 40.9	540.4 ± 51.8	602.2 ± 35.4	1057.5 ± 606.4	796.6 ± 132.6
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	P(s)	9.6 ± 0.1	19.9 ± 0.3	22.9 ± 0.2	14.4 ± 0.1	13.7 ± 0.1
	T(s)	570.8 ± 75.2	1711.6 ± 125.3	2084.5 ± 95.8	1322.3 ± 154.4	1358.1 ± 175.1
	R	62.2 ± 8.8	93.6 ± 7.5	98.4 ± 5.2	106.4 ± 12.6	111.6 ± 15.2
	A	347.5 ± 24.5	477.8 ± 30.5	557.8 ± 23.4	615.6 ± 49.6	635.00 ± 48.2
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	P(s)	9.8 ± 0.2	19.9 ± 0.2	22.9 ± 0.1	14.8 ± 0.1	13.9 ± 0.1
	T(s)	487.1 ± 72.3	1224.7 ± 160.7	1798.3 ± 325.1	1070.2 ± 113.6	995.8 ± 53.4
	R	53.5 ± 8.2	66.2 ± 8.9	84.2 ± 15.7	85.2 ± 9.3	81.4 ± 4.8
	A	341.2 ± 32.9	449.6 ± 21.3	540.8 ± 60.8	534.2 ± 29.9	563.6 ± 13.6
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	P(s)	9.7 ± 0.1	20.1 ± 0.3	23.2 ± 0.1	14.9 ± 0.1	14.3 ± 0.3
	T(s)	445.2 ± 44.5	1337.2 ± 173.7	1786.1 ± 206.2	1011.52 ± 89.4	975.5 ± 109.4
	R	48.2 ± 5.2	72.5 ± 9.2	84.2 ± 10.6	79.6 ± 6.9	79.5 ± 8.1
	A	346.8 ± 16.1	456.8 ± 41.2	561.8 ± 48.4	539.8 ± 16.3	530.8 ± 47.6
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 19: Comparing AKFD using different horizons to LAMAF over five problems from the Rovers domain removing 2 predicates chosen manually.

Planner	M	Problem				
		rovers 36 (14.80.40)	rovers 37 (14.80.40)	rovers 38 (14.85.57)	rovers 39 (14.95.62)	rovers 40 (14.100.68)
LAMAF	P(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	45.8 ± 5.4	**75.60 ± 7.3**	59.2 ± 7.3	**68.2 ± 5.1**	**70.1 ± 10.2**
	A	323.6 ± 24.2	**512.20 ± 28.9**	432.8 ± 42.3	**486.8 ± 21.2**	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 5)	PPT(s)	9.44 ± 0.13	19.08 ± 0.09	22.08 ± 0.19	13.56 ± 0.08	13.04 ± 0.13
	T(s)	1067.56 ± 132.95	2599.59 ± 237.26	3216.64 ± 188.49	2478.59 ± 208.80	2668.34 ± 412.49
	R	118.20 ± 14.09	144.00 ± 12.84	154.20 ± 9.85	201.60 ± 16.95	220.20 ± 35.21
	A	434.60 ± 42.55	560.40 ± 39.44	606.20 ± 21.06	780.20 ± 69.51	827.80 ± 143.65
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	PPT(s)	9.55 ± 0.05	19.14 ± 0.14	22.27 ± 0.26	13.58 ± 0.07	12.99 ± 0.11
	T(s)	605.74 ± 80.40	1518.06 ± 118.05	2226.64 ± 142.03	1325.84 ± 119.54	1151.09 ± 22.92
	R	66.60 ± 8.45	83.40 ± 6.92	105.60 ± 6.34	108.20 ± 9.37	95.00 ± 0.89
	A	355.80 ± 50.38	466.60 ± 23.22	589.80 ± 21.35	589.60 ± 17.73	549.40 ± 26.50
AKFD (k = 20)	P(s)	9.80 ± 0.34	19.40 ± 0.20	22.40 ± 0.25	13.76 ± 0.14	13.10 ± 0.16
	T(s)	433.86 ± 64.30	1270.05 ± 86.02	1582.58 ± 318.12	1016.10 ± 145.11	1044.44 ± 190.81
	R	47.40 ± 7.71	69.00 ± 4.52	74.80 ± 14.74	82.20 ± 11.89	86.00 ± 16.70
	A	321.00 ± 17.58	448.80 ± 22.05	518.60 ± 37.23	545.80 ± 47.63	566.00 ± 17.40
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	P(s)	9.60 ± 0.14	19.44 ± 0.16	22.48 ± 0.18	13.78 ± 0.10	13.30 ± 0.17
	T(s)	466.92 ± 47.04	1328.60 ± 95.15	1555.64 ± 238.69	954.34 ± 112.41	937.57 ± 122.36
	R	50.00 ± 4.98	72.60 ± 5.92	73.00 ± 11.52	77.00 ± 9.76	76.60 ± 10.44
	A	338.20 ± 13.06	442.60 ± 12.29	526.80 ± 42.86	541.80 ± 37.08	538.00 ± 48.80
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 20: Comparing AKFD using different horizons to LAMAF over five problems from the Rovers domain removing 3 predicates chosen manually.

Planner	M	Problem				
		rovers 36 (14.80.40)	rovers 37 (14.80.40)	rovers 38 (14.85.57)	rovers 39 (14.95.62)	rovers 40 (14.100.68)
LAMAF	F(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	45.8 ± 5.4	75.60 ± 7.3	59.2 ± 7.3	68.2 ± 5.1	70.1 ± 10.2
	A	323.6 ± 24.2	512.20 ± 28.9	432.8 ± 42.3	486.8 ± 21.2	530.8 ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 5)	F(s)	9.44 ± 0.13	19.08 ± 0.09	22.08 ± 0.19	13.56 ± 0.08	13.04 ± 0.13
	T(s)	1067.56 ± 132.95	2599.59 ± 237.26	3216.64 ± 188.49	2478.59 ± 208.80	2668.34 ± 412.49
	R	118.20 ± 14.09	144.00 ± 12.84	154.20 ± 9.85	201.60 ± 16.95	220.20 ± 35.21
	A	434.60 ± 42.55	560.40 ± 39.44	606.20 ± 21.06	780.20 ± 69.51	827.80 ± 143.65
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	F(s)	9.55 ± 0.05	19.14 ± 0.14	22.27 ± 0.26	13.58 ± 0.07	12.99 ± 0.11
	T(s)	605.74 ± 80.40	1518.06 ± 118.05	2226.64 ± 142.03	1325.84 ± 119.54	1151.09 ± 22.92
	R	66.60 ± 8.45	83.40 ± 6.92	105.60 ± 6.34	108.20 ± 9.37	95.00 ± 0.89
	A	355.80 ± 50.38	466.60 ± 23.22	589.80 ± 21.35	589.60 ± 17.73	549.40 ± 26.50
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	F(s)	9.80 ± 0.34	19.40 ± 0.20	22.40 ± 0.25	13.76 ± 0.14	13.10 ± 0.16
	T(s)	433.86 ± 64.30	1270.05 ± 86.02	1582.58 ± 318.12	1016.10 ± 145.11	1044.44 ± 190.81
	R	47.40 ± 7.71	69.00 ± 4.52	74.80 ± 14.74	82.20 ± 11.89	86.00 ± 16.70
	A	321.00 ± 17.58	448.80 ± 22.05	518.60 ± 37.23	545.80 ± 47.63	566.00 ± 17.40
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	F(s)	9.60 ± 0.14	19.44 ± 0.16	22.48 ± 0.18	13.78 ± 0.10	13.30 ± 0.17
	T(s)	466.92 ± 47.04	1328.60 ± 95.15	1555.64 ± 238.69	954.34 ± 112.41	937.57 ± 122.36
	R	50.00 ± 4.98	72.60 ± 5.92	73.00 ± 11.52	77.00 ± 9.76	76.60 ± 10.44
	A	338.20 ± 13.06	442.60 ± 12.29	526.80 ± 42.86	541.80 ± 37.08	538.00 ± 48.80
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 21: Comparing AKFD using different horizons to LAMAF over five problems from the Rovers domain removing 4 predicates chosen manually.

C.2 DETAILED RESULTS FOR OTHER DOMAINS

Planner	Metrics	Problem				
		DRobots 2 05 0 (5.25.5)	DRobots 2 10 2 (10.25.10)	DRobots 5 10 2 (10.100.10)	DRobots 5 10 3 (10.100.10)	DRobots 5 20 0 (10.100.20)
LAMAF	P(s)	**0.6 ± 0.3**	**1.6 ± 0.1**	-	171.8 ± 23.4	-
	T(s)	**4.9 ± 1.7**	**18.6 ± 2.30**	-	3604.3 ± 453.2	-
	R	**8.3 ± 3.3**	**16.4 ± 1.4**	-	40.4 ± 5.6	-
	A	**57.2 ± 2.2**	**122.7 ± 10.8**	-	215.6 ± 27.9	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 15, 0, 0/15	15.0.0.0/15	0, 15, 0, 0/15
AKFD (k = 2)	P(s)	-	-	-	-	-
	T(s)	-	-	-	-	-
	R	-	-	-	-	-
	A	-	-	-	-	-
	C	0, 0, 15, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15
AKFD (k = 5)	P(s)	**0,54 ± 0,00**	**0,90 ± 0,01**	-	**9,93 ± 0,02**	**15,14 ± 5, 18**
	T(s)	10, 74 ± 2, 78	108, 69 ± 122, 74	-	2797, 27 ± 791, 87	**6821,69 ± 5272, 96**
	R	19, 83 ± 5, 34	122, 17 ± 139, 05	-	281, 50 ± 80, 50	**377,32 ± 221, 92**
	A	80, 17 ± 23, 37	445, 50 ± 534, 96	-	984, 00 ± 257, 00	1300, 44 ± 790, 41
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15	10, 0, 0, 5/15	10, 0, 0, 5/15
AKFD (k = 10)	P(s)	0, 59 ± 0, 11	0, 93 ± 0, 01	**10,24 ± 0, 06**	10, 10 ± 0, 06	20, 60 ± 0, 09
	T(s)	7, 90 ± 2, 37	27, 74 ± 5, 31	7699, 20 ± 4796, 83	824, 25 ± 241, 38	10028, 73 ± 2340, 37
	R	13, 50 ± 4, 03	30, 00 ± 5, 97	772, 75 ± 482, 13	81, 17 ± 24, 30	470, 00 ± 109, 43
	A	69, 33 ± 6, 75	166, 00 ± 19, 60	3810, 75 ± 2483, 39	413, 67 ± 108, 65	**2427,67 ± 487, 69**
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	10, 0, 0, 5/15
AKFD (k = 20)	P(s)	0, 58 ± 0, 01	0, 96 ± 0, 01	10, 38 ± 0, 05	10, 28 ± 0, 04	22, 54 ± 0, 91
	T(s)	5, 96 ± 1, 15	21, 60 ± 3, 30	1297, 44 ± 739, 41	437, 54 ± 43, 26	50139, 74 ± 10345, 82
	R	10, 17 ± 2, 11	22, 50 ± 3, 69	100, 00 ± 41, 69	41, 17 ± 4, 52	1376, 00 ± 210, 08
	A	72, 67 ± 3, 09	133, 83 ± 11, 71	622, 33 ± 235, 05	251, 83 ± 17, 22	8511, 00 ± 953, 73
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	8, 7, 0, 0/15
AKFD (k = 30)	P(s)	0, 57 ± 0, 01	0, 97 ± 0, 01	10, 93 ± 0, 07	11, 27 ± 0, 22	-
	T(s)	6, 18 ± 1, 55	21, 67 ± 4, 22	**1133,78 ± 817, 93**	**410,63 ± 58, 70**	-
	R	10, 67 ± 2, 87	20, 67 ± 3, 77	**71,33 ± 31, 63**	**37,40 ± 5, 75**	-
	A	74, 83 ± 12, 72	140, 33 ± 11, 26	**477,33 ± 230, 47**	**236,60 ± 4, 32**	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	10, 5, 0, 0/15	12, 3, 0, 0/15	0, 15, 0, 0/15

Table 22: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Depots-robots domain manually removing predicate at-robot. The meaning of the rows is the same as in Table 4. In bold, we highlight the best results per row The problems' complexity has been defined as: (i) the number of robots; (ii) the number of cells of the grid; and (iii) the number of goals.

Planner	Metrics	Problem				
		TidyBot 01 (1. 18. 4)	TidyBot 09 (1. 20. 4)	TidyBot 11 (1. 20. 4)	TidyBot 16 (1. 20. 4)	TidyBot 17 (1. 22. 4)
LAMAF	PPT(s)	-	33.1 ± 0.3	-	69.1 ± 0.1	-
	T(s)	-	**153.1 ± 25.9**	-	949.3 ± 93.5	-
	R	-	**4.3 ± 0.8**	-	**18.3 ± 1.1**	-
	A	-	**38.5 ± 0.8**	-	**110.4 ± 14.1**	-
	C	0, 15, 0, 0/15	15, 0, 0, 0/15	0, 15, 0, 0/15	15, 0, 0, 0/15	0, 15, 0, 0/15
AKFD (k = 2)	P(s)	-	**31.2 ± 0.1**	-	-	-
	T(s)	-	6739.5 ± 4999.7	-	-	-
	R	-	219.4 ± 163.5	-	-	-
	A	-	467.6 ± 348.9	-	-	-
	C	0, 0, 0, 15/15	15, 0, 0, 0/15	0, 0, 0, 15/15	0, 0, 0, 15/15	0, 0, 0, 15/15
AKFD (k = 5)	P(s)	**20.5 ± 0.1**	31.9 ± 0.2	**30.7 ± 0.2**	**30.9 ± 0.2**	**45.7 ± 0.1**
	T(s)	2108.8 ± 278.3	714.8 ± 95.3	3163.4 ± 1096.6	7538.5 ± 5191.8	3735.5 ± 109.8
	R	103.5 ± 13.5	22.01 ± 3.2	102.8 ± 36.2	246.7 ± 168.3	81.3 ± 2.4
	A	345.4 ± 16.5	78.4 ± 9.1	356.8 ± 115.7	855.5 ± 546.9	302.7 ± 23.7
	C	11.0.0.4/15	15, 0, 0, 0/15	12, 0, 0, 3/15	11.0.0.4/15	13.0.0.2/15
AKFD (k = 10)	P(s)	21.1 ± 0.2	31.9 ± 0.2	32.2 ± 0.2	31.4 ± 0.9	46.4 ± 0.2
	T(s)	1383.9 ± 711.4	412.2 ± 67.2	1269.8 ± 455.1	2316.3 ± 543.2	1274.9 ± 842.9
	R	68.6 ± 34.9	12.2 ± 2.1	40.2 ± 14.7	75.4 ± 13.5	26.8 ± 18.5
	A	307.2 ± 16.5	49.6 ± 5.3	214.6 ± 97.1	343.1 ± 84.2	142.4 ± 107.4
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	3.0.0.7/10	15, 0, 0, 0/15
AKFD (k = 20)	P(s)	21.9 ± 0.2	32.4 ± 0.1	32.9 ± 0.3	31.7 ± 0.2	48.1 ± 0.4
	T(s)	**635.4 ± 237.3**	**246.7 ± 83.8**	**922.3 ± 413.4**	**835.7 ± 188.2**	**646.1 ± 214.1**
	R	**28.1 ± 10.6**	**6.8 ± 2.6**	**28.7 ± 12.7**	**25.2 ± 6.3**	**12.8 ± 4.5**
	A	**166.2 ± 39.5**	**42.2 ± 1.9**	**154.8 ± 53.9**	**163.3 ± 28.9**	**80.4 ± 17.3**
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	9.0.0.1/10	15, 0, 0, 0/15
AKFD (k = 30)	P(s)	**23.1 ± 0.12**	**33.1 ± 0.3**	**35.8 ± 0.3**	**32.4$_i$ ± 0.14$_i$**	**50.3 ± 0.3**
	T(s)	**748.5 ± 344.1**	**214.9 ± 86.1**	**1085.2 ± 288.1$_i$**	**810.2 ± 238.9**	**480.7 ± 113.1**
	R	**30.4 ± 13.7**	**5.8 ± 2.8**	**30.6 ± 8.1**	**21.2 ± 7.4**	**9.2 ± 2.5**
	A	**213.2 ± 120.4**	**44.2 ± 3.2**	**168.6 ± 27.9**	**133.2 ± 28.8**	**63.8 ± 2.9**
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 23: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the TidyBot domain manually removing predicate `base-pos`. The meaning of the rows is the same as in Table 4. In bold, we highlight the best results per row. The problems' complexity has been defined by: (i) number of robots; (ii) size of the environment and (iii) number of goals.

Planner	Metrics	Problem				
		ports 5 5 (5.20.10)	ports 05 30 (5.50.30)	ports 10 20 (10.80.20)	ports 10 40 (10.110.40)	ports 15 20 (15.120.20)
LAMAP	F(s)	1.2 ± 0.1	-	161.1 ± 2.2	-	135.7 ± 0.8
	T(s)	5.3 ± 1.1	-	1212.1 ± 382.9	-	953.1 ± 363.5
	R	3.5 ± 1.1	-	9.8 ± 2.4	-	5.8 ± 2.4
	A	14.8 ± 2.5	-	60.8 ± 5.8	-	44.9 ± 1.3
	C	15, 0, 0, 0/15	0,15,0,0/15	15, 0, 0, 0/15	0, 15, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 2)	F(s)	-	-	-	-	-
	T(s)	-	-	-	-	-
	R	-	-	-	-	-
	A	-	-	-	-	-
	C	0, 0, 0, 15/15	0, 0, 0, 15/15	0, 0, 0, 15/15	0, 0, 0, 15/15	0, 0, 0, 15/15
AKFD (k = 5)	F(s)	1.2 ± 0.1	68.9 ± 0.6	45.4 ± 0.1	283.1 ± 0.2	**62.5 ± 0.1**
	T(s)	4.8 ± 0.1	988.5 ± 303.7	397.5 ± 110.4	5221.9 ± 1301.1	599.9 ± 58.1
	R	3.1 ± 0.1	17.7 ± 5.9	8.3 ± 2.6	20.7 ± 5.2	8.7 ± 0.9
	A	**11.7 ± 0.9**	93.3 ± 10.5	49.7 ± 4.1	101.25 ± 0.8	42.7 ± 2.5
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	F(s)	1.2 ± 0.1	69.5 ± 0.1	44.9 ± 0.3	281.9 ± 1.3	63.8 ± 0.1
	T(s)	**2.1 ± 0.6**	662.5 ± 98.5	473.6 ± 159.3	4279.5 ± 134.9	481.2 ± 28.8
	R	**0.7 ± 0.5**	**11.7 ± 2.1**	10.2 ± 3.7	16.3 ± 0.45	6.7 ± 0.5
	A	12.7 ± 0.5	91.3 ± 15.7	53.5 ± 3.3	**100.1 ± 1.63**	43.3 ± 2.2
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	F(s)	1.2 ± 0.1	70.6 ± 0.4	**44.3 ± 5.2**	**275.1 ± 0.7**	65.7 ± 0.2
	T(s)	3.2 ± 1.1	725.8 ± 164.3	312.2 ± 41.6	4160.2 ± 864.9	491.3 ± 104.6
	R	1.7 ± 0.9	12.7 ± 3.3	6.3 ± 0.9	16.5 ± 3.4	6.7 ± 1.7
	A	12.7 ± 0.5	86.3 ± 3.3	**50.7 ± 1.3**	101.3 ± 1.9	44.7 ± 1.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	F(s)	**1.2 ± 0.2**	**65.7 ± 0.8**	45.2 ± 0.1	277.8 ± 2.3	67.5 ± 0.2
	T(s)	2.5 ± 1.8	794.3 ± 339.8	**231.9 ± 18.2**	**3849.9 ± 1310.8**	**476.3 ± 260.9**
	R	1.2 ± 1.4	14.3 ± 6.6	**4.3 ± 0.5**	**147 ± 5.2**	**6.3 ± 4.1**
	A	12.00 ± 1.32	**85.7 ± 2.5**	51.33 ± 0.9	100.4 ± 0.8	**43.1 ± 0.8**
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 24: Comparing AKFD using different horizons to FD and mGPT over five problems from the Port domain removing predicate at. The meaning of the rows is the same as in Table 4. In bold, we highlight the best results per row. Besides, three values have been defined to describe the complexity of the problem: (i) number of ships; (ii) number of pallet-ship and (iii) number of goals.

Planner	Metrics	Problem				
		satellite 1 (10, 132, 342)	satellite 2 (10, 140, 348)	satellite 3 (10, 132, 351)	satellite 4 (11, 132, 327)	satellite 5 (12, 132, 330)
LAMAF	P(s)	0	-	-	-	-
	T(s)	-	-	-	-	-
	R	-	-	-	-	-
	A	-	-	-	-	-
	C	0, 10, 0, 0/10	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15
AKFD (k = 2)	P(s)	-	-	-	-	-
	T(s)	-	-	-	-	-
	R	-	-	-	-	-
	A	-	-	-	-	-
	C	0, 0, 15, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15
AKFD (k = 5)	P(s)	258.2 ± 10.1	**526,6 ± 23.2**	248.8 ± 9.3	**216.9 ± 0.7**	**183.9 ± 0, 9**
	T(s)	12586.4 ± 2710.9	**24738.8 ± 3126.4**	11380.7 ± 2534.3	9480.5 ± 2173, 6	**7685.25 ± 368.29**
	R	124.5 ± 17.5	108.3 ± 22.4	119.4 ± 34.4	92.7 ± 18.6	**97.7 ± 12, 7**
	A	947, 9 ± 9, 2	1028, 5 ± 18, 7	977, 7 ± 26, 5	872, 3 ± 17, 2	**786, 7 ± 7, 1**
	C	15, 0, 0, 0/15	14, 0, 1, 0/15	13, 0, 2, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	P(s)	249.3 ± 2.2	602.8 ± 41.2	265.9 ± 15.4	244.7 ± 4.7	–
	T(s)	8965.4 ± 1390.1	27656.3 ± 4316.1	9838.3 ± 2253.5	**7523.9 ± 1725.1**	–
	R	100.7 ± 16.8	112.3 ± 19.5	104.9 ± 6.6	**89.7 ± 9.8**	–
	A	974.4 ± 20.8	1039.7 ± 37.2	1014.5 ± 32.5	893.2 ± 21.4	–
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	9, 1, 0, 0/10	15, 0, 0, 0/15	0, 15, 0, 0/15
AKFD (k = 20)	P(s)	248.1 ± 2.7	576.3 ± 6.1	**243.2 ± 16.5**	223.3 ± 4.1	–
	T(s)	8904.7 ± 418.1	28086.9 ± 934.9	**6681.8 ± 412.9**	10769.1 ± 3820.5	–
	R	**94.3 ± 6.5**	107.3 ± 8.2	100.2 ± 5.7	87.5 ± 3.5	–
	A	966.5 ± 14.5	1002.1 ± 12.7	1003.5 ± 13.8	**872.1 ± 11.2**	–
	C	14, 0, 1, 0/15	15, 0, 0, 0/15	14, 0, 15, 0/15	14, 0, 1, 0/150	0, 0, 15, 0/15
AKFD (k = 30)	P(s)	**246.5 ± 3.8**	564.3 ± 17.1	247.1 ± 19.9	218.1 ± 4.1	–
	T(s)	**8784.9 ± 432.1**	25452.7 ± 2354.9	7159.5 ± 396.5	14647.2 ± 5863.5	–
	R	95.1 ± 8.9	**108.1 ± 15.9**	**98.6 ± 8.9**	91.3 ± 3.1	–
	A	**998.2 ± 19.8**	**995.1 ± 19.6**	**960.9 ± 12.1**	**887.5 ± 33.4**	–
	C	13, 0, 2, 0/15	12, 0, 3, 0/15	14, 0, 1, 0/15	13, 0, 2, 0/15	0, 0, 15, 0/15

Table 25: Comparing AKFD using different horizons to FD and mGPT over five problems from the Satellite domain removing predicate `power-on`. The meaning of the rows is the same as in Table 4. For each problem has been defined three values which describe the complexity of the problem: (i) the number of satellite; (ii) the number or directions and (iii) the number of goals.

D

DETAILED RESULTS FROM CHAPTER 7

In this appendix, we show the detail results for the different planning domains used in the empirical evaluation of the Chapter 7.

D.1 The Rovers Domain

Planner	M	Problem				
		rovers 36 (14,80,40)	rovers 37 (14,80,40)	rovers 38 (14,85,57)	rovers 39 (14,95,62)	rovers 40 (14,100,68)
AKFD (k = 2)	F(s)	11.1 ± 0.0	15.3 ± 0.1	22.7 ± 0.1	16.1 ± 0.1	25.7 ± 0.0
	T(s)	6909.9 ± 4210.5	35157.7 ± 16548.4	14271.3 ± 4234.1	9201.1 ± 7227.4	14246.5 ± 1962.8
	R	737.0 ± 457.6	2846.0 ± 1365.3	749.3 ± 222.8	720.3 ± 567.1	642.3 ± 87.7
	A	1303.7 ± 826.4	4414.7 ± 1923.0	1257.7 ± 341.8	1265.0 ± 994.0	1152.3 ± 152.5
	C	3.0.0.0/3	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 5)	F(s)	11.2 ± 0.0	15.3 ± 0.1	22.7 ± 0.1	16.1 ± 0.1	25.7 ± 0.1
	T(s)	1199.9 ± 48.2	3744.2 ± 1909.8	3469.9 ± 172.8	1560.3 ± 1088.3	3926.8 ± 228.9
	R	126.3 ± 4.9	298.7 ± 156.0	180.0 ± 9.1	121.0 ± 85.4	176.3 ± 10.2
	A	466.0 ± 10.7	1084.3 ± 511.5	639.3 ± 21.6	444.3 ± 311.1	658.3 ± 19.6
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	F(s)	11.1 ± 0.0	15.3 ± 0.1	23.2 ± 0.5	16.1 ± 0.1	25.6 ± 0.1
	T(s)	681.0 ± 67.5	1394.2 ± 123.9	1608.7 ± 191.5	1323.6 ± 83.1	2478.6 ± 263.5
	R	71.3 ± 6.9	110.0 ± 9.8	83.0 ± 10.0	102.7 ± 6.8	111.0 ± 12.0
	A	418.0 ± 8.0	586.0 ± 27.6	480.0 ± 20.3	556.0 ± 8.8	587.7 ± 31.8
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	F(s)	11.3 ± 0.0	15.5 ± 0.0	23.1 ± 0.1	16.4 ± 0.1	25.9 ± 0.3
	T(s)	504.2 ± 50.9	1075.1 ± 97.6	1349.0 ± 185.5	983.9 ± 160.4	1650.7 ± 82.9
	R	52.3 ± 5.4	84.3 ± 7.8	69.3 ± 9.7	75.7 ± 13.0	73.3 ± 3.7
	A	349.7 ± 11.6	551.3 ± 20.0	446.3 ± 28.5	501.7 ± 70.6	506.3 ± 22.1
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	F(s)	11.3 ± 0.1	15.7 ± 0.1	23.0 ± 0.1	16.3 ± 0.1	26.1 ± 0.2
	T(s)	409.6 ± 58.6	1050.5 ± 120.0	1158.8 ± 248.0	601.4 ± 457.5	1262.9 ± 871.5
	R	42.3 ± 6.2	82.7 ± 9.7	59.0 ± 12.7	45.7 ± 35.9	56.0 ± 39.5
	A	343.7 ± 13.6	584.3 ± 26.9	412.3 ± 30.7	351.3 ± 244.5	383.7 ± 259.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 26: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Rovers domain removing a predicate set generated using the Relaxed Plan automatically.

Planner	M	Problem				
		rovers 36 (14.80.40)	rovers 37 (14.80.40)	rovers 38 (14.85.57)	rovers 39 (14.95.62)	rovers 40 (14.100.68)
LAMAF	P(s)	80.4 ± 0.1	412.1 ± 3.3	236.5 ± 1.8	257.1 ± 1.3	354.8 ± 2.2
	T(s)	1132.5 ± 202.9	7125.5 ± 1145.5	3205.1 ± 236.7	3342.1 ± 845.4	4406.12 ± 872.1
	R	**45.8** ± 5.4	**75.60** ± 7.3	**59.2** ± 7.3	**68.2** ± 5.1	**70.1** ± 10.2
	A	**323.6** ± 24.2	**512.20** ± 28.9	432.8 ± 42.3	**486.8** ± 21.2	**530.8** ± 23.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 2)	P(s)	13.41 ± 0.34	20.19 ± 0.53	**28.86** ± 0.22	**21.34** ± 0.16	**32.94** ± 0.43
	T(s)	2918.56 ± 1384.30	6814.19 ± 1634.96	5024.64 ± 947.92	6926.29 ± 2002.50	6420.82 ± 829.46
	R	326.88 ± 158.15	572.22 ± 136.70	280.00 ± 53.01	575.00 ± 171.50	311.42 ± 40.60
	A	802.25 ± 356.13	1432.00 ± 320.74	699.56 ± 119.63	1406.00 ± 407.93	802.50 ± 101.32
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 5)	P(s)	**13.20** ± 0.13	**19.83** ± 0.04	29.08 ± 0.27	21.42 ± 0.28	33.01 ± 0.45
	T(s)	937.10 ± 153.01	2250.78 ± 157.80	2128.08 ± 229.08	2061.12 ± 373.71	3250.35 ± 371.15
	R	103.33 ± 16.98	189.00 ± 12.79	117.11 ± 13.62	169.44 ± 31.75	156.43 ± 18.18
	A	441.78 ± 72.44	747.25 ± 37.22	500.44 ± 34.74	697.89 ± 119.12	656.00 ± 68.17
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	P(s)	13.11 ± 0.01	20.07 ± 0.25	29.02 ± 0.19	21.54 ± 0.19	33.14 ± 0.45
	T(s)	576.13 ± 81.27	1238.51 ± 108.79	1496.42 ± 144.35	1235.18 ± 91.91	2059.00 ± 319.36
	R	63.40 ± 9.35	103.10 ± 9.30	82.40 ± 8.26	100.20 ± 7.98	98.20 ± 15.89
	A	375.20 ± 43.74	606.70 ± 43.81	447.60 ± 29.57	565.60 ± 33.35	570.00 ± 73.31
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	P(s)	13.79 ± 0.77	20.28 ± 0.60	29.30 ± 0.39	21.68 ± 0.11	33.18 ± 0.42
	T(s)	**430.54** ± 84.49	1009.12 ± 108.95	1299.33 ± 143.21	1030.73 ± 90.42	1596.71 ± 172.46
	R	46.80 ± 9.28	83.00 ± 8.03	70.80 ± 8.33	82.80 ± 6.71	75.60 ± 8.60
	A	**323.00** ± 25.88	537.50 ± 25.62	476.80 ± 29.17	545.60 ± 28.36	533.80 ± 41.82
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	P(s)	13.28 ± 0.12	20.19 ± 0.14	29.59 ± 0.35	21.82 ± 0.31	33.30 ± 0.21
	T(s)	479.16 ± 77.98	**953.12** ± 124.90	**1243.65** ± 151.88	1023.01 ± 126.82	1522.30 ± 130.94
	R	52.18 ± 8.75	78.33 ± 10.58	67.22 ± 8.42	82.11 ± 10.15	72.22 ± 6.27
	A	354.24 ± 27.91	536.33 ± 39.42	446.56 ± 33.63	542.33 ± 24.64	**519.56** ± 25.10
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 50)	P(s)	13.66 ± 0.14	20.91 ± 0.49	29.53 ± 0.41	21.94 ± 0.25	33.96 ± 0.67
	T(s)	475.56 ± 93.04	998.06 ± 154.29	1242.64 ± 190.59	**992.75** ± 166.71	**1722.84** ± 287.97
	R	51.65 ± 10.27	81.11 ± 13.39	66.78 ± 10.70	78.92 ± 13.07	81.21 ± 14.47
	A	353.39 ± 33.16	559.56 ± 38.32	460.78 ± 47.70	555.86 ± 47.59	543.14 ± 27.65
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 100)	P(s)	14.29 ± 0.07	23.18 ± 0.11	30.84 ± 0.12	23.05 ± 0.10	35.60 ± 0.29
	T(s)	502.11 ± 90.60	1074.21 ± 178.51	1136.84 ± 170.31	1076.31 ± 154.89	1568.35 ± 216.49
	R	52.80 ± 9.71	83.40 ± 13.27	60.20 ± 9.31	82.60 ± 11.83	72.53 ± 10.07
	A	345.40 ± 31.67	558.00 ± 30.10	464.60 ± 22.20	535.90 ± 35.48	511.90 ± 24.34
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 27: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Rovers domain removing a predicate set generated using the landmark graph automatically.

D.2 The Depots Robots domain

Planner	M	Problem				
		DRobots 2 05 0 (5.25.5)	DRobots 2 10 2 (10.25.10)	DRobots 5 10 2 (10.100.10)	DRobots 5 10 3 (10.100.10)	DRobots 5 20 0 (10.100.20)
LAMAF	F(s)	0.6 ± 0.3	1.6 ± 0.1	-	171.8 ± 23.4	-
	t(s)	4.9 ± 1.7	18.6 ± 2.30	-	3604.3 ± 453.2	-
	R	8.3 ± 3.3	16.4 ± 1.4	-	40.4 ± 5.6	-
	A	57.2 ± 2.2	122.7 ± 10.8	-	215.6 ± 27.9	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 15, 0, 0/15	15.0.0.0/15	0, 15, 0, 0/15
AKFD (k = 2)	F(s)	0.7 ± 0.0	1.2 ± 0.0	-	-	-
	T(s)	273.9 ± 345.4	132.4 ± 17.3	-	-	-
	R	589.0 ± 744.7	160.7 ± 19.6	-	-	-
	A	847.0 ± 1029.0	285.3 ± 22.2	-	-	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15
AKFD (k = 5)	F(s)	0.7 ± 0.1	1.1 ± 0.1	-	13.9 ± 0.2	-
	T(s)	11.6 ± 1.6	47.2 ± 1.1	-	5975.9 ± 2545.6	-
	R	23.0 ± 3.3	54.5 ± 0.5	-	622.0 ± 266.0	-
	A	84.0 ± 17.7	198.5 ± 6.5	-	2030.0 ± 891.0	-
	C	15, 0, 0, 0/15	14, 0, 1, 0/15	0, 0, 15, 0/15	12, 0, 3, 0/15	0, 0, 15, 0/15
AKFD (k = 10)	F(s)	0.7 ± 0.1	1.2 ± 0.1	14.1 ± 0.1	15.5 ± 1.1	-
	T(s)	6.8 ± 0.5	69.8 ± 62.7	2025.5 ± 1236.4	678.4 ± 150.2	-
	R	12.2 ± 1.5	83.7 ± 76.7	207.0 ± 127.0	68.3 ± 15.4	-
	A	66.0 ± 4.3	436.3 ± 365.3	1017.5 ± 571.5	358.7 ± 65.4	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	14, 0, 1, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15
AKFD (k = 20)	F(s)	0.7 ± 0.0	1.2 ± 0.1	15.4 ± 1.8	14.5 ± 0.4	30.3 ± 0.6
	T(s)	6.6 ± 1.1	18.9 ± 2.3	2667.7 ± 1256.1	462.8 ± 39.5	10221.6 ± 4963.8
	R	12.3 ± 2.1	21.0 ± 2.9	266.3 ± 130.8	45.3 ± 4.1	363.0 ± 230.7
	A	71.3 ± 2.4	152.7 ± 5.0	1540.0 ± 763.8	263.7 ± 21.5	2262.0 ± 1465.5
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	12, 0, 3, 0/15
AKFD (k = 30)	F(s)	0.7 ± 0.1	1.2 ± 0.1	14.5 ± 1.1	15.0 ± 0.5	31.0 ± 0.8
	T(s)	7.0 ± 2.6	17.2 ± 1.2	1123.3 ± 734.10	371.9 ± 67.7	51595.6 ± 26543.3
	R	12.7 ± 5.3	18.3 ± 1.7	70.0 ± 23.1	35.3 ± 6.8	1650.0 ± 1276.3
	A	71.3 ± 3.3	142.0 ± 3.3	396.0 ± 53.21	257.0 ± 50.3	10281.0 ± 6321.3
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	10, 0, 5, 0/15	15, 0, 0, 0/15	4, 4, 7, 0/15

Table 28: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Depots-Robots domain removing a predicate set generated using the Relaxed Plan automatically.

Planner	M	Problem				
		DRobots 2 05 0 (5.25.5)	DRobots 2 10 2 (10.25.10)	DRobots 5 10 2 (10.100.10)	DRobots 5 10 3 (10.100.10)	DRobots 5 20 0 (10.100.20)
LAMAF	F(s)	0.6 ± 0.3	1.6 ± 0.1	-	171.8 ± 23.4	-
	t(s)	4.9 ± 1.7	18.6 ± 2.30	-	3604.3 ± 453.2	-
	R	8.3 ± 3.3	16.4 ± 1.4	-	40.4 ± 5.6	-
	A	57.2 ± 2.2	122.7 ± 10.8	-	215.6 ± 27.9	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 15, 0, 0/15	15.0.0.0/15	0, 15, 0, 0/15
AKFD (k = 2)	F(s)	0.6 ± 0.1	0.9 ± 0.1	-	-	-
	t(s)	49.8 ± 41.9	296.7 ± 217.1	-	-	-
	R	94.7 ± 79.9	338.6 ± 248.3	-	-	-
	A	177.3 ± 146.3	580.6 ± 413.1	-	-	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15
AKFD (k = 5)	F(s)	1.01 ± 0.2	1.3 ± 0.01	21.8 ± 0.19	21.6 ± 0.1	61.1 ± 7.93.5
	t(s)	16.03 ± 2.7	63.5 ± 9.40	13658.8 ± 2345.3	2804.9 ± 1997.7	14611.1 ± 1543.3
	R	30.3 ± 5.6	73.7 ± 10.8	1443.2 ± 104.2	294.7 ± 211.8	760.7 ± 183.4
	A	105.7 ± 16.2	264.3 ± 15.9	4763.8 ± 543.4	955.7 ± 613.6	2632.5 ± 435.2
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	4.0.0.6/10	15, 0, 0, 0/15	9, 0, 6, 0/15
AKFD (k = 10)	F(s)	0.7 ± 0.1	1.3 ± 0.1	21.2 ± 0.1	21.5 ± 0.1	66.2 ± 8.4
	t(s)	5.9 ± 1.6	24.1 ± 0.89	633.1 ± 7.03	867.7 ± 28.3	29541.5 ± 2843.8
	R	10.7 ± 3.3	26.7 ± 1.3	64.5 ± 0.5	89.4 ± 3.4	1447.7 ± 345.6
	A	69.7 ± 10.5	143.3 ± 10.2	360.5 ± 28.5	484.5 ± 38.50	7181.5 ± 1392.1
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	F(s)	0.8 ± 0.1	1.3 ± 0.1	21.9 ± 0.1	21.7 ± 0.2	-
	t(s)	7.4 ± 2.5	16.1 ± 3.8	604.4 ± 180.7	451.3 ± 46.5	-
	R	13.3 ± 5.1	17.6 ± 4.24	59.4 ± 18.4	44.7 ± 5.6	-
	A	74.3 ± 4.2	145.4 ± 11.6	435.3 ± 112.3	256.5 ± 31.5	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15
AKFD (k = 30)	F(s)	0.8 ± 0.1	1.4 ± 0.1	-	22.5 ± 0.2	-
	t(s)	8.2 ± 1.2	24.1 ± 6.7	-	387.8 ± 110.7	-
	R	11.9 ± 2.1	24.4 ± 6.4	-	36.3 ± 11.2	-
	A	74.9 ± 13.2	132.7 ± 15.2	-	233.3 ± 22.4	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15

Table 29: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Depots-Robots domain removing a predicate set generated using the landmark graph automatically.

D.3 The TidyBot domain

Planner	M	Problem				
		TidyBot 01 (1. 18. 4)	TidyBot 09 (1. 20. 4)	TidyBot 11 (1. 20. 4)	TidyBot 16 (1. 20. 4)	TidyBot 17 (1. 22. 4)
LAMAF	F(s)	-	33.1 ± 0.3	-	69.1 ± 0.1	-
	T(s)	-	**153.1 ± 25.9**	-	949.3 ± 93.5	-
	R	-	**4.3 ± 0.8**	-	**18.3 ± 1.1**	-
	A	-	**38.5 ± 0.8**	-	**110.4 ± 14.1**	-
	C	0, 15, 0, 0/15	15, 0, 0, 0/15	0, 15, 0, 0/15	15, 0, 0, 0/15	0, 15, 0, 0/15
AKFD (k = 2)	F(s)	-	44.3 ± 1.4	-	-	-
	T(s)	-	1725.0 ± 667.6	-	-	-
	R	-	58.0 ± 23.6	-	-	-
	A	-	115.0 ± 53.2	-	-	-
	C	-	12, 0, 2, 0/15	-	-	-
AKFD (k = 5)	F(s)	-	43.7 ± 0.5	45.8 ± 0.5	42.4 ± 0.0	63.1 ± 0.2
	T(s)	-	723.5 ± 236.9	2725.6 ± 667.7	5448.2 ± 2753.6	2103.2 ± 605.2
	R	-	23.3 ± 8.2	89.0 ± 22.0	189.0 ± 45.2	48.3 ± 14.3
	A	-	88.3 ± 20.7	262.0 ± 64.0	646.0 ± 127.1	165.7 ± 42.8
	C	-	15, 0, 0, 0/15	12, 0, 3, 0/15	5, 0, 10, 0/16	15, 0, 0, 0/15
AKFD (k = 10)	F(s)	29.5 ± 1.3	43.6 ± 0.1	44.4 ± 0.1	42.7 ± 0.1	63.6 ± 0.1
	T(s)	5442.3 ± 2811.6	395.5 ± 156.9	1082.3 ± 141.3	2537.7 ± 643.2	2059.4 ± 419.4
	R	282.5 ± 146.5	12.5 ± 5.4	34.3 ± 4.7	86.5 ± 22.5	47.0 ± 9.9
	A	1331.0 ± 650.0	60.7 ± 16.8	173.3 ± 16.8	423.5 ± 92.5	204.7 ± 54.9
	C	12, 0, 2, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	12, 0, 2, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	F(s)	29.0 ± 0.7	44.7 ± 0.6	46.4 ± 1.0	43.7 ± 0.8	64.4 ± 0.1
	T(s)	463.2 ± 8.3	252.2 ± 42.4	1342.5 ± 702.2	1007.8 ± 137.6	494.3 ± 187.4
	R	22.3 ± 0.5	7.4 ± 1.4	41.7 ± 22.2	32.5 ± 4.5	10.0 ± 4.3
	A	163.0 ± 11.0	48.0 ± 3.7	219.3 ± 90.3	216.0 ± 6.0	68.3 ± 8.2
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	11, 0, 4, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	F(s)	29.3 ± 0.2	44.3 ± 0.1	48.4 ± 0.9	44.2 ± 1.1	66.0 ± 0.2
	T(s)	626.5 ± 203.4	183.3 ± 15.3	713.2 ± 241.1	854.0 ± 51.1	514.2 ± 131.6
	R	30.0 ± 9.9	4.7 ± 0.5	20.7 ± 7.6	25.3 ± 1.2	10.3 ± 3.1
	A	198.3 ± 72.0	42.7 ± 1.2	149.3 ± 29.4	159.7 ± 18.0	79.0 ± 8.6
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 30: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the TidyBot domain removing a predicate set generated using the Relaxed Plan automatically.

Planner	M	Problem				
		TidyBot 01 (1. 18. 4)	TidyBot 09 (1. 20. 4)	TidyBot 11 (1. 20. 4)	TidyBot 16 (1. 20. 4)	TidyBot 17 (1. 22. 4)
LAMAF	P(s)	-	33.1 ± 0.3	-	69.1 ± 0.1	-
	T(s)	-	153.1 ± 25.9	-	949.3 ± 93.5	-
	R	-	4.3 ± 0.8	-	18.3 ± 1.1	-
	A	-	38.5 ± 0.8	-	110.4 ± 14.1	-
	C	0, 15, 0, 0/15	15, 0, 0, 0/15	0, 15, 0, 0/15	15, 0, 0, 0/15	0, 15, 0, 0/15
AKFD (k = 2)	P(s)	-	31, 23 ± 0, 05	-	-	-
	T(s)	-	6739, 46 ± 4999, 70	-	-	-
	R	-	219, 00 ± 163, 49	-	-	-
	A	-	467, 00 ± 348, 97	-	-	-
	C	0, 0, 0, 10/10	10, 0, 0, 0/10	0, 0, 0, 10/10	0, 0, 0, 10/10	0, 0, 0, 10/10
AKFD (k = 5)	P(s)	-	55, 99 ± 1, 12	53, 64 ± 1, 03	-	88, 60 ± 1, 18
	T(s)	-	662, 84 ± 57, 53	4407, 32 ± 893, 38	-	7475, 33 ± 7429, 99
	R	-	22, 00 ± 2, 16	161, 00 ± 63, 19	-	183, 60 ± 185, 02
	A	-	71, 67 ± 14, 82	511, 00 ± 213, 93	-	521, 80 ± 453, 84
	C	0, 0, 0, 10/10	10, 0, 0, 0/10	6, 0, 0, 4/10	0, 0, 10, 0/10	10, 0, 0, 0/10
AKFD (k = 10)	P(s)	34, 84 ± 0, 12	56, 29 ± 0, 92	55, 57 ± 1, 27	53, 94 ± 1, 23	88, 85 ± 0, 46
	T(s)	5134, 68 ± 3383, 40	306, 92 ± 46, 84	1395, 15 ± 187, 44	1394, 32 ± 234, 94	818, 00 ± 260, 35
	R	278, 50 ± 185, 25	9, 00 ± 1, 63	49, 00 ± 6, 68	47, 92 ± 3, 32	18, 00 ± 6, 48
	A	1365, 50 ± 903, 67	45, 67 ± 2, 87	253, 33 ± 29, 77	189, 34 ± 23, 94	98, 00 ± 26, 87
	C	8, 0, 0, 2/10	10, 0, 0, 0/10	10, 0, 0, 0/10	10, 0, 0, 0/10	10, 0, 0, 0/10
AKFD (k = 20)	P(s)	34, 19 ± 0, 98	56, 24 ± 0, 95	56, 34 ± 1, 11	54, 75 ± 0, 09	89, 31 ± 0, 23
	T(s)	471, 28 ± 43, 33	211, 72 ± 73, 13	834, 02 ± 390, 36	1299, 18 ± 114, 48	445, 32 ± 78, 11
	R	23, 17 ± 2, 11	5, 50 ± 2, 60	27, 67 ± 13, 89	44, 33 ± 4, 11	8, 67 ± 1, 89
	A	148, 83 ± 9, 99	41, 75 ± 2, 59	156, 00 ± 52, 36	191, 33 ± 29, 51	60, 00 ± 1, 63
	C	10, 0, 0, 0/10	10, 0, 0, 0/10	10, 0, 0, 0/10	10, 0, 0, 0/10	10, 0, 0, 0/10
AKFD (k = 30)	P(s)	34, 85 ± 1, 08	57, 03 ± 0, 91	57, 82 ± 1, 54	55, 09 ± 0, 33	91, 86 ± 1, 53
	T(s)	850, 87 ± 368, 55	243, 15 ± 76, 74	1012, 89 ± 149, 87	740, 15 ± 89, 13	592, 62 ± 33, 11
	R	41, 00 ± 17, 58	6, 50 ± 2, 69	32, 50 ± 5, 17	22, 00 ± 2, 94	10, 67 ± 0, 94
	A	220, 83 ± 103, 95	40, 50 ± 1, 66	181, 00 ± 32, 60	151, 33 ± 22, 98	68, 33 ± 8, 50
	C	10, 0, 0, 0/10	10, 0, 0, 0/10	10, 0, 0, 0/10	10, 0, 0, 0/10	10, 0, 0, 0/10

Table 31: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the TidyBot domain removing a predicate set generated using the landmark graph automatically.

D.4 The Port domain

Planner	M	Problem				
		ports 5 5 (5.20.10)	ports 05 30 (5.50.30)	ports 10 20 (10.80.20)	ports 10 40 (10.110.40)	ports 15 20 (15.120.20)
LAMAF	F(s)	1.2 ± 0.1	-	161.1 ± 2.2	-	135.7 ± 0.8
	T(s)	5.3 ± 1.1	-	1212.1 ± 382.9	-	953.1 ± 363.5
	R	3.5 ± 1.1	-	9.8 ± 2.4	-	5.8 ± 2.4
	A	14.8 ± 2.5	-	60.8 ± 5.8	-	44.9 ± 1.3
	C	15, 0, 0, 0/15	0,15,0,0/15	15, 0, 0, 0/15	0, 15, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 2)	F(s)	1.7 ± 0.1	65.9 ± 0.1	61.5 ± 0.3	325.1 ± 0.8	91.4 ± 0.2
	T(s)	5.7 ± 0.5	4034.7 ± 178.7	867.6 ± 70.8	13449.1 ± 553.5	1209.8 ± 49.1
	R	3.3 ± 0.5	89.0 ± 4.0	19.7 ± 1.7	61.7 ± 2.6	18.0 ± 0.8
	A	10.7 ± 0.5	164.5 ± 16.5	42.3 ± 5.7	113.3 ± 3.1	37.0 ± 1.6
	C	15, 0, 0, 0/15	5, 0, 10, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 5)	F(s)	1.7 ± 0.1	66.3 ± 0.2	61.8 ± 0.5	326.8 ± 0.7	91.9 ± 0.3
	T(s)	4.1 ± 1.2	1340.0 ± 118.7	638.8 ± 133.3	5963.5 ± 360.4	674.1 ± 30.1
	R	2.1 ± 0.8	28.3 ± 2.6	14.0 ± 3.3	26.3 ± 1.7	9.3 ± 0.5
	A	12.7 ± 0.8	93.7 ± 4.5	49.7 ± 4.5	101.7 ± 5.4	40.7 ± 1.2
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	F(s)	1.7 ± 0.1	68.2 ± 0.3	61.8 ± 0.3	329.1 ± 1.2	92.7 ± 0.3
	T(s)	4.5 ± 0.5	690.1 ± 92.3	503.8 ± 156.6	4214.0 ± 528.7	719.5 ± 89.9
	R	2.3 ± 0.5	13.7 ± 2.1	10.7 ± 3.8	18.0 ± 2.4	10.0 ± 1.4
	A	12.3 ± 0.5	83.7 ± 1.2	52.3 ± 1.2	102.3 ± 2.9	43.7 ± 1.7
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 20)	F(s)	1.7 ± 0.1	70.9 ± 0.1	62.4 ± 0.1	332.9 ± 0.4	93.9 ± 0.3
	T(s)	3.4 ± 0.5	765.7 ± 129.7	412.2 ± 20.9	3505.4 ± 465.9	431.6 ± 61.1
	R	1.3 ± 0.5	15.6 ± 2.8	8.3 ± 0.5	14.3 ± 2.1	5.3 ± 0.9
	A	12.0 ± 0.5	87.1 ± 5.9	54.0 ± 2.2	97.3 ± 2.1	44.3 ± 2.1
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 30)	F(s)	1.7 ± 0.0	72.1 ± 0.1	63.1 ± 0.3	334.9 ± 1.1	94.8 ± 0.3
	T(s)	4.3 ± 1.9	1012.3 ± 233.4	384.7 ± 38.9	3637.6 ± 671.7	390.9 ± 81.3
	R	2.2 ± 1.6	20.8 ± 4.9	7.7 ± 0.9	14.7 ± 3.1	4.7 ± 1.2
	A	12.7 ± 0.9	84.3 ± 4.9	51.0 ± 0.8	97.0 ± 1.4	45.8 ± 0.8
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15

Table 32: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Port domain removing a predicate set generated using the Relaxed Plan automatically.

Planner	Metrics	Problem				
		ports 5 5 (5.20.10)	ports 05 30 (5.50.30)	ports 10 20 (10.80.20)	ports 10 40 (10.110.40)	ports 15 20 (15.120.20)
LAMAF	F(s)	1.2 ± 0.1	-	161.1 ± 2.2	-	135.7 ± 0.8
	T(s)	5.3 ± 1.1	-	1212.1 ± 382.9	-	953.1 ± 363.5
	R	3.5 ± 1.1	-	9.8 ± 2.4	-	5.8 ± 2.4
	A	14.8 ± 2.5	-	60.8 ± 5.8	-	44.9 ± 1.3
	C	15, 0, 0, 0/15	0.15.0.0/15	15, 0, 0, 0/15	0.15.0.0/15	15, 0, 0, 0/15
AKFD (k = 2)	F(s)	-	-	-	-	-
	T(s)	-	-	-	-	-
	R	-	-	-	-	-
	A	-	-	-	-	-
	C	0, 0, 0, 15/15	0, 0, 0, 15/15	0, 0, 0, 15/15	0, 0, 0, 15/15	0, 0, 0, 15/15
AKFD (k = 5)	F(s)	1.9 ± 0.1	477.1 ± 4.1	373.8 ± 1.5	1362.4 ± 1.6	799.9 ± 5.3
	T(s)	3.8 ± 0.5	1370.3 ± 104.9	771.9 ± 49.4	7069.1 ± 78.4	1403.8 ± 31.6
	R	1.7 ± 0.5	21.3 ± 2.6	10.3 ± 1.3	23.7 ± 0.5	10.3 ± 0.5
	A	11.3 ± 0.5	89.7 ± 8.1	48.7 ± 2.1	109.3 ± 4.7	41.4 ± 2.5
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15
AKFD (k = 10)	F(s)	1.9 ± 0.1	478.5 ± 1.1	374.5 ± 7.2	-	794.6 ± 2.1
	T(s)	3.4 ± 1.1	1075.1 ± 137.4	788.9 ± 83.6	-	1215.2 ± 80.6
	R	1.3 ± 0.9	14.1 ± 3.1	10.6 ± 2.2	-	7.1 ± 1.4
	A	12.5 ± 0.8	84.6 ± 7.5	52.7 ± 1.9	-	43.3 ± 1.1
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	0.10.0.0/10	15, 0, 0, 0/15
AKFD (k = 20)	F(s)	1.9 ± 0.1	482.5 ± 3.9	381.9 ± 9.2	-	795.2 ± 1.4
	T(s)	3.5 ± 1.5	1081.5 ± 116.2	636.4 ± 91.2	-	1208.1 ± 57.9
	R	1.3 ± 1.4	13.8 ± 2.6	6.50 ± 2.4	-	7.1 ± 1.1
	A	12.7 ± 0.5	85.3 ± 3.9	52.3 ± 2.6	-	43.2 ± 3.4
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	0.10.0.0/10	15, 0, 0, 0/15
AKFD (k = 30)	F(s)	2.2 ± 0.4	484.3 ± 3.1	376.4 ± 1.1	1955.5 ± 345.9	809.8 ± 8.8
	T(s)	2.9 ± 0.8	1003.45 ± 114.8	586.8 ± 75.3	5453.3 ± 1216.3	1126.6 ± 69.4
	R	0.7 ± 0.5	11.7 ± 2.5	5.3 ± 1.9	16.7 ± 5.1	5.3 ± 1.25
	A	12.7 ± 0.5	83.1 ± 1.6	55.3 ± 2.62	96.7 ± 0.9	44.9 ± 1.6
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	4.10.0.0/10	15, 0, 0, 0/15

Table 33: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Port domain removing a predicate set generated using the landmark graph automatically.

D.5 The Satellite domain

Planner	M	Problem				
		satellite 1 (10. 132. 342)	satellite 2 (10. 140. 348)	satellite 3 (10. 132. 351)	satellite 4 (11. 132. 327)	satellite 5 (12. 132. 330)
LAMAF	F(s)	-	-	-	-	-
	T(s)	-	-	-	-	-
	R	-	-	-	-	-
	A	-	-	-	-	-
	C	0.15.0.0/15	0.15.0.0/15	0.15.0.0/15	0.15.0.0/15	0.15.0.0/15
AKFD (k = 2)	F(s)	237.6 ± 1.7	235.4 ± 1.4	227.0 ± 0.0	239.8 ± 5.0	-
	T(s)	25238.3 ± 1142.2	23968.8 ± 2572.2	21235.5 ± 0.0	31882.6 ± 5702.6	-
	R	274.0 ± 8.0	279.0 ± 17.2	251.0 ± 0.0	294.0 ± 53.0	-
	A	898.3 ± 22.4	1023.0 ± 5.4	974.0 ± 0.0	915.0 ± 111.0	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	15, 0, 0, 0/15	12, 0, 3, 0/15	0, 0, 15, 0/15
AKFD (k = 5)	F(s)	233.9 ± 2.1	233.0 ± 3.6	-	-	153.5 ± 0.4
	T(s)	17546.1 ± 266.9	13044.1 ± 564.2	-	-	17020.8 ± 783.1
	R	188.3 ± 6.9	167.7 ± 8.9	-	-	185.3 ± 1.0
	A	973.3 ± 8.2	999.6 ± 12.21	-	-	908.5 ± 1.5
	C	15, 0, 0, 0/15	10, 0, 5, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	11, 4, 0, 0/15
AKFD (k = 10)	F(s)	234.7 ± 0.9	-	-	237.7 ± 1.3	-
	T(s)	12296.0 ± 579.8	-	-	11433.7 ± 984.23	-
	R	137.7 ± 5.8	-	-	122.0 ± 3.4	-
	A	960.0 ± 12.1	-	-	865.0 ± 23.7	-
	C	15.0.0.0/15	0, 3, 12, 0/15	0, 0, 15, 0/15	10, 0, 5, 0/15	0, 0, 15, 0/15
AKFD (k = 20)	F(s)	245.1 ± 0.5	-	-	-	-
	T(s)	11469.1 ± 1157.1	-	-	-	-
	R	103.7 ± 5.3	-	-	-	-
	A	948.3 ± 22.3	-	-	-	-
	C	15, 0, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15
AKFD (k = 30)	F(s)	582.0 ± 6.1	-	-	-	-
	T(s)	12566.4 ± 854.8	-	-	-	-
	R	92.3 ± 7.8	-	-	-	-
	A	978.3 ± 23.5	-	-	-	-
	C	15, 0, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15

Table 34: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Satellite domain removing a predicate set generated using the relaxed plan automatically.

Planner	Metrics	Problem				
		satellite 1 (10, 132, 342)	satellite 2 (10, 140, 348)	satellite 3 (10, 132, 351)	satellite 4 (11, 132, 327)	satellite 5 (12, 132, 330)
LAMAF	P(s)	-	-	-	-	-
	T(s)	-	-	-	-	-
	R	-	-	-	-	-
	A	-	-	-	-	-
	C	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15	0, 15, 0, 0/15
AKFD (k = 5)	P(s)	313.5 ± 0.8	**309.5 ± 0.8**	**297.2 ± 1.04**	-	-
	T(s)	18676.5 ± 570.5	15686.2 ± 1753.5	**26229.1 ± 6419.6**	-	-
	R	182.3 ± 3.7	178.5 ± 28.2	194.7 ± 18.1	-	-
	A	979.3 ± 29.9	**995.2 ± 45.2**	**1045.3 ± 23.2**	-	-
	C	15, 0, 0, 0/15	14, 0, 1, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15	0, 0, 15, 0/15
AKFD (k = 10)	P(s)	**312.9 ± 0.8**	382.1 ± 56.2	303.7 ± 5.4	310.4 ± 52.11	-
	T(s)	12665.5 ± 1121.2	13469.9 ± 1274.4	28400.6 ± 4425.9	22002.71 ± 1938.4	-
	R	131.9 ± 5.9	133.1 ± 11.3	**152.3 ± 5.4**	134.4 ± 11.7	-
	A	973.7 ± 22.7	998.4 ± 83.6	1076.8 ± 9.20	941.8 ± 96.7	-
	C	15, 0, 0, 0/15	6, 0, 9, 0/5	15, 0, 0, 0/15	7, 0, 8, 0/15	0, 0, 15, 0/15
AKFD (k = 20)	P(s)	325.2 ± 3.3	369.1 ± 14.9	-	310.1 ± 3.1	-
	T(s)	**10140.3 ± 1023.7**	**12310.2 ± 1151.5**	-	**19899.1 ± 1580.7**	-
	R	105.1 ± 6.2	126.4 ± 5.10	-	103.7 ± 5.1	-
	A	**955.2 ± 8.6**	1005.7 ± 0.9	-	911.3 ± 7.7	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15
AKFD (k = 30)	P(s)	343.8 ± 15.5	367.2 ± 16.8	-	**307.7 ± 41.6**	-
	T(s)	10990.6 ± 1896.3	20993.3 ± 5935.8	-	22943.9 ± 1643.9	-
	R	102.3 ± 10.4	**103.7 ± 4.5**	-	**85.7 ± 12.2**	-
	A	957.5 ± 4.1	1028.7 ± 11.7	-	**894.4 ± 65.2**	-
	C	15, 0, 0, 0/15	15, 0, 0, 0/15	0, 0, 15, 0/15	8, 0, 7, 0/15	0, 0, 15, 0/15

Table 35: Comparing AKFD using different horizons to LAMAF and mGPT over five problems from the Satellite domain removing a predicate set generated using the landmark graph automatically.

BIBLIOGRAPHY

Aguirre, Eugenio and Antonio González (2003). "A Fuzzy Perceptual Model for Ultrasound Sensors Applied to Intelligent Navigation of Mobile Robots." In. *Applications Intelligence* 19.3, pp. 171–187 (page 3).

Ai-Chang, Mitchell et al. (2004). "MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission." In. *IEEE Intelligent Systems* 19.1, pp. 8–12. DOI: 10.1007/s10489-014-0542-0 (page 1).

Alami, Rachid, Raja Chatila, Sara Fleury, Malik Ghallab, and Félix Ingrand (1998). "An Architecture for Autonomy" In. *International Journal of Robotic Research* 17.4, pp. 315–337 (page 72).

Albore, Alexandre, Héctor Palacios, and Hector Geffner (2007). "Fast and Informed Action Selection for Planning with Sensing" In. *Proceedings of the Twelveth Conference of the Spanish Association for Artificial Intelligence, CAEPIA*. Salamanca, Spain, pp. 1–10 (page 45).

Amir, Eyal and Pedrito Maynard-reid (1999). "Logic-Based Subsumption Architecture" In. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 147–152 (pages 3, 76).

Asunción, Marc de la, Luis A. Castillo, Juan Fernández-Olivares, Óscar García-Pérez, Antonio González, and Francisco Palao (2005). "SIADEX: An interactive knowledge-based planner for decision support in forest fire fighting" In. *Artificial Intelligence Communications* 18.4, pp. 257–268 (pages 1, 11).

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer (2002). "Finite-time Analysis of the Multiarmed Bandit Problem" In. *Journal of Machine Learning* 47.2-3, pp. 235–256 (page 37).

Bäckström, Christer and Bernhard Nebel (1993). "Complexity Results for SAS+ Planning" In. *Computational Intelligence* 11, pp. 625–655 (pages 19, 53, 96).

Bellman, Richard (1957). "A Markovian Decision Process" In. *Indiana University Mathematical Journal* 6, pp. 679–684 (page 34).

Bernardini, Sara, Maria Fox, and Derek Long (2014). "Planning the Behaviour of Low-Cost Quadcopters for Surveillance Missions" In. *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*. Portsmouth, New Hampshire, USA (page 11).

Bertoli, Piergiorgio, Alessandro Cimatti, Ugo Dal Lago, and Marco Pistore (2003). "Extending PDDL to nondeterminism, limited sensing and iterative conditional plans" In. *Proceedings of ICAPS'03 workshop on PDDL*. Trento, Italy (page 43).

Blum, Avrim and Merrick L. Furst (1997). "Fast Planning Through Planning Graph Analysis" In. *Artificial Intelligence Journal* 90.1–2, pp. 281–300 (pages 26, 132).

Blum, Avrim L. and Merrick L. Furst (1995). "Fast Planning Through Planning Graph Analysis" In. *Artificial Intelligence* 90.1, pp. 1636–1642 (pages 45, 52).

Blythe, Jim (1994). "Planning with External Events" In. *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence*. Seattle, Washington, USA, pp. 94–101 (page 64).

Bonet, Blai and Hector Geffner (2000a). "Planning with Incomplete Information as Heuristic Search in Belief Space" In. *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*. Breckenridge, CO, USA, pp. 52–61 (page 38).

Bonet, Blai and Héctor Geffner (2000b). "Planning with Incomplete Information as Heuristic Search in Belief Space" In. *Proceedings of the fifth International Conference on Artificial Intelligence Planning Systems*, pp. 52–61 (pages 40, 43).

Bonet, Blai and Héctor Geffner (2001). "Planning as Heuristic Search" In. *Artificial Intelligence* 129, pp. 5–33 (pages 26, 45).

Bonet, Blai and Hector Geffner (2003). "Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming" In. *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*. Trento, Italy, pp. 12–21 (page 66).

Bonet, Blai and Hector Geffner (2005). "mGPT: A Probabilistic Planner Based on Heuristic Search" In. *Journal of Artificial Intelligence Research* 24, pp. 933–944. DOI: 10.1613/jair.1688 (pages 2, 28, 35, 106, 109).

Bonet, Blai and Hector Geffner (2006). "Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and Its Application to MDPs." In. *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*. Cumbria, UK, pp. 142–151 (page 35).

Borgo, Stefano, Amedeo Cesta, Andrea Orlandini, Riccardo Rasconi, Marco Suriano, and Alessandro Umbrico (2014). "A Cooperative Model-based Control Agent for a Reconfigurable Manufacturing Plant" In. *Proceedings of the second ICAPS Workshop on Planning and Robotics (PlanRob)*. Portsmouth, New Hampshire, USA (page 73).

Borrajo, Daniel and Manuela Veloso (2012). "Probabilistically Reusing Plans in Deterministic Planning" In. *Proceedings of ICAPS'12 workshop on Heuristics and Search for Domain-Independent Planning*. Atibaia Brazil (pages 2, 33, 69).

Borrajo, Daniel, Anna Roubíčková, and Ivan Serina (2014). "Progress in Case-Based Planning" In. *ACM Computing Surveys* 47.2, 35:1–35:39 (page 69).

Boutilier, Craig (2000). "Approximately Optimal Monitoring of Plan Preconditions" In. *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence*. Stanford, California, USA, pp. 54–62 (page 70).

Boutilier, Craig, Ray Reiter, and Bob Price (2001). "Symbolic Dynamic Programming for First-order MDPs" In. *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*. San Francisco, CA, USA, pp. 690–697 (pages 28, 35).

Bresina, John L., Ari K. Jónsson, Paul H. Morris, and Kanna Rajan (2005). "Mixed-Initiative Activity Planning for Mars Rovers" In. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 1709–1710 (page 34).

Britanik, J. and M. Marefat (2004). "CBPOP: A Domain-Independent Multi-Case Reuse Planner" In. *Computational Intelligence* 20.2, pp. 405–443 (page 69).

Brooks, Rodney A. (1986). "A Robust Layered Control System for a Mobile Robot" In. *IEEE Journal of Robotics and Automation* 2.10, pp. 14–23. DOI: 10.1109/JRA.1986.1087032 (pages 3, 76).

Brügmann, Bernd (1993). "Monte Carlo Go" In. *In Technical Report Max Planke Institute* (page 37).

Bryant, Randal E. (1992). "Symbolic Boolean Manipulation with Ordered Binary-decision Diagrams" In. *ACM Computing Surveys* 24.3, pp. 293–318. DOI: 10.1145/136035.136043 (page 41).

Bryce, Daniel and Subbarao Kambhampati (2004). "Heuristic Guidance Measures for Conformant Planning" In. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pp. 365–375 (page 66).

Bulitko, Vadim and Greg Lee (2006). "Learning in Real-time Search: A Unifying Framework" In. *Journal of Artificial Intelligence Research* 25.1, pp. 119–157 (pages 56, 58).

Bulitko, Vadim, Lihong Li, Russell Greiner, and Ilya Levner (2003). "Lookahead Pathologies for Single Agent Search" In. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*. Acapulco, Mexico, pp. 1531–1533 (page 59).

Burch, Jerry R., Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang (1990). "Symbolic Model Checking: 10^20 States and Beyond" In. *Proceedings of the fifth Annual Symposium on Logic in Computer Science*. Philadelphia, Pennsylvania, USA, pp. 428–439. DOI: 10.1109/LICS.1990.113767 (page 41).

Buro, Michael (2003). "ORTS: A Hack-Free RTS Game Environment" In. *Proceedings of the Third International Conference on Computers and Games*, pp. 156–161 (page 56).

Butler, Greg, Andrea Gantchev, and Peter Grogono (2001). "Object-oriented design of the subsumption architecture" In. *Software Practice & Experience* 31.9, pp. 911–923. DOI: 10.1002/spe.396 (page 3).

Cai, Shaowei, Zhong Jie, and Kaile Su (2015). "An effective variable selection heuristic in SLS for weighted Max-2-SAT" In. *Journal of Heuristics* 21.3, pp. 433–456 (page 28).

Carbonell, Jaime G., Oren Etzioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steve Minton, and Manuela Veloso (1991). "PRODIGY: An integrated architecture for planning and learning" In. *In Kurt VanLehn, Architectures for Intelligence*, pp. 241–278 (page 52).

Cassandra, Anthony R., Leslie Pack Kaelbling, and Michael L. Littman (1994). "Acting Optimally in Partially Observable Stochastic Domains" In. *Proceedings of the 12th National Conference on Artificial Intelligence*. Seattle, WA, USA, pp. 1023–1028 (page 28).

Castellini, Claudio, Enrico Giunchiglia, and Armando Tacchella (2003). "SAT-based planning in complex domains: Concurrency, constraints and nondeterminism" In. *Artificial Intelligence Journal* 147.1-2, pp. 85–117. DOI: 10.1016/S0004-3702(02)00375-2 (page 41).

Champandard, Alex J., Tim Verweij, and Remco Straatman (2009). "The AI for Killzone 2's multiplayer bots" In. *Proceedings of Game Developers Conference* (page 3).

Cimatti, Alessandro and Marco Roveri (2000) In. *Journal of Artificial Intelligence Research (JAIR)* 13, pp. 305–338. DOI: 10.1613/jair.774 (page 41).

Coddington, A. M., M. Fox, and D. Long (2001). "Handling Durative Actions in Classical Planning Frameworks" In. *Proceedings of the 20th Workshop of the UK Planning and Scheduling Special Interest Group*. Edinburgh, Scotland, pp. 44–58 (page 64).

Coles, Amanda and Andrew Coles (2012). "Branched Plans with Execution-Time Choice to Reduce the Costs of Conservative Planning" In. *Proceedings of ICAPS'12 workshop on Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx)*. Atibaia, São Paulo, Brazil (page 64).

Coles, Amanda Jane, Andrew Coles, Angel García Olaya, Sergio Jiménez, Carlos Linares López, Scott Sanner, and Sungwook Yoon (2012). "A Survey of the Seventh International Planning Competition" In. *AI Magazine* 33.1, pp. 83–88 (page 31).

Culberson, Joseph C. and Jonathan Schaeffer (1998). "Pattern databases" In. *Computational Intelligence* 14.3, pp. 318–334 (page 53).

Dean, Thomas and Keiji Kanazawa (1989). "A Model for Reasoning About Persistence and Causation" In. *Computational Intelligence* 5.3, pp. 142–150 (page 30).

Dräger, Klaus, Bernd Finkbeiner, and Andreas Podelski (2006). "Directed Model Checking with Distance-Preserving Abstractions" In. *Proceedings of the 13th International SPIN Workshop on Model Checking Software*. Vol. 3925. Lecture Notes in Computer Science. Berlin, Germany, pp. 19–36 (page 53).

Dräger, Klaus, Bernd Finkbeiner, and Andreas Podelski (2009). "Directed Model Checking with Distance-Preserving Abstractions" In. *Journal on Software Tools for Technology Transfer STTT* 11.1, pp. 27–37. DOI: 10.1007/s10009-008-0092-z (page 53).

Draper, Denise, Steve Hanks, and Daniel Weld (1994). "Probabilistic Planning With Information Gathering and Contingent Execution" In. *Proceedings of the Second International Conference on AI Planning Systems*. Menlo Park, California, USA, pp. 31–36 (page 66).

Dvorak, Filip, Arthur Bit-Monnont, Felix Ingrand, and Malic Ghallab (2014). "Plan-Space Hierarchical Planning with the Action Notation Modeling Language" In. *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*. Limassol, Cyprus (page 72).

Edelkamp, S. (2001). "Planning with pattern databases" In. *Proceeding of the sixth European Conference on Planning*, pp. 13–24 (page 53).

Edelkamp, Stefan (2002). "Symbolic Pattern Databases in Heuristic Search Planning" In. *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*. Toulouse, France, pp. 274–283 (page 24).

Fikes, Richard E. (1971). "Monitored Execution of robot plan produced by STRIPS" In. *Proceedings of the International Federation for Information Processing congress (IFIP)* (pages 65, 70).

Fikes, Richard E. and Nils J. Nilsson (1971). "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving" In. *Proceedings of the 2nd International Joint*

Conference on Artificial Intelligence. London, England, pp. 608–620 (pages 21, 45, 50, 65).

Fikes, Richard E., Peter E. Hart, and Nils J. Nilsson (1972). "Learning and Executing Generalized Robot Plans" In. *Artificial Intelligence Journal* 3, pp. 251–288 (pages 65, 76).

Finzi, Alberto, Félix Ingrand, and Nicola Muscettola (2004). "Model-based executive control through reactive planning for autonomous rovers" In. *Procedings of the International Conference on Intelligent Robots and Systems IEEE/RSJ*. Sendai, Japan, pp. 879–884 (page 73).

Fox, Maria and Derek Long (2003). "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains" In. *Journal of Artificial Intelligence Research* 20.1, pp. 61–124 (page 159).

Fox, Maria, Alfonso Gerevini, Derek Long, and Ivan Serina (2006). "Plan Stability: Replanning versus Plan Repair" In. *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*. Cumbria, UK, pp. 212–221 (pages 2, 33).

Francès, Guillem and Hector Geffner (2015). "Modeling and Computation in Planning: Better Heuristics from More Expressive Languages" In. *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*. Jerusalem, Israel, pp. 70–78 (page 24).

Fritz, Christian and Sheila A. McIlraith (2007). "Monitoring Plan Optimality During Execution" In. *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 144–151 (page 70).

García, Javier, José E. Flórez, Álvaro Torralba Arias de Reyna, Daniel Borrajo, Carlos Linares López, Angel García Olaya, and Juan Sáenz (2013). "Combining linear programming and automated planning to solve intermodal transportation problems." In. *European Journal of Operational Research* 227.1, pp. 216–226 (page 11).

Gat, Erann (1998). "On Three-Layer Architectures" In (page 72).

Geffner, H. (2000). "Functional Strips: a more flexible language for planning and problem solving" In, pp. 188–209 (page 24).

Gerevini, Alfonso and Ivan Serina (2000). "Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques" In. *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*. Breckenridge, CO, USA, pp. 112–121 (page 68).

Ghallab, Malik and Hervé Laruelle (1994). "Representation and Control in IxTeT, a Temporal Planner" In. *Proceedings of the Second International Conference on Artificial Intelligence and Planning Systems*. Chicago, Illinois, USA, pp. 61–67 (page 72).

Giacomo, Giuseppe De, Raymond Reiter, and Mikhail Soutchanski (1998). "Execution Monitoring of High-Level Robot Programs." In. *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 453–465 (page 69).

Goldman, Robert P. and Mark S. Boddy (1996). "Expressive Planning and Explicit Knowledge." In. *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, pp. 110–117 (page 37).

Gutmann, Jens steffen, Masaki Fukuchi, and Masahiro Fujita (2005). "Real-time path planning for humanoid robot navigation" In. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*. Edinburgh, Scotland, pp. 1232–1237 (page 56).

Haigh, Karen Zita and Manuela M. Veloso (1996). "Planning with Multiple Goals for Robot Execution" In. *Proceedings of the National Conference on Artificial Intelligence Fall Symposium "Plan Execution: Problems and Issues"*. New Orleans, LA, USA, pp. 65–71 (page 65).

Hansen, Eric A. and Shlomo Zilberstein (2001). "LAO*: A heuristic search algorithm that finds solutions with loops" In. *Artificial Intelligence* 129.1-2, pp. 35–62 (pages 35, 66).

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths" In. *IEEE Transactions on Systems Science and Cybernetics* 2, pp. 100–107 (pages 26, 56).

Helmert, Malte (2006). "The Fast Downward Planning System" In. *Journal of Artificial Intelligence Research* 26, pp. 191–246. DOI: 10.1613/jair.1705 (pages 19, 27, 159).

Helmert, Malte (2009). "Concise finite-domain representations for PDDL planning tasks" In. *Artificial Intelligence Journal* 173.5-6, pp. 503–535. DOI: 10.1016/j.artint.2008.10.013 (page 27).

Helmert, Malte and Carmel Domshlak (2009). "Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?" In. *Proceedings of the 19th International Conference on Automated Planning and Scheduling*. Thessaloniki, Greece (page 134).

Helmert, Malte and Hector Geffner (2008). "Unifying the Causal Graph and Additive Heuristics." In. *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*. Sydney, Australia, pp. 140–147 (page 159).

Helmert, Malte, Patrik Haslum, and Jörg Hoffmann (2007). "Flexible Abstraction Heuristics for Optimal Sequential Planning" In. *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*. Providence, Rhode Island, USA, pp. 176–183 (pages 27, 45, 53).

Helmert, Malte, Patrik Haslum, Jörg Hoffmann, and Raz Nissim (2014). "Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces" In. *Journal of the ACM (JACM)* 61.3, 16:1–16:63 (page 53).

Hernández, Carlos and Pedro Meseguer (2005). "LRTA*(k)" In. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*. Edinburgh, Scotland, UK, pp. 1238–1243 (page 58).

Hoffmann, Jörg and Ronen I. Brafman (2006). "Conformant Planning via Heuristic Forward Search: A New Approach" In. *Artificial Intelligence Journal* 170.6-7, pp. 507–541. DOI: 10.1016/j.artint.2006.01.003 (page 40).

Hoffmann, Jörg and Bernhard Nebel (2001). "The FF Planning System: Fast Plan Generation Through Heuristic Search" In. *Journal of Artificial Intelligence Research* 14, pp. 253–302. DOI: 10.1613/jair.855 (pages 26, 45, 52, 109, 132).

Hoffmann, Jörg, Julie Porteous, and Laura Sebastia (2004). "Ordered Landmarks in Planning" In. *Journal of Artificial Intelligence Research* 22.1, pp. 215–278. DOI: 10.1613/jair.1492 (pages 129, 130).

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press (page 35).

Howard, Ronald A. and James E. Matheson, eds. (1984). *Readings on the Principles and Applications of Decision Analysis*. Menlo Park, CA: Strategic Decision Group (page 31).

Hyafil, Nathanael and Fahiem Bacchus (2004). "Utilizing Structured Representations and CSPs in Conformant Probabilistic Planning" In. *European Conference on Artificial Intelligence (ECAI 2004)* (page 37).

Ihrig, Laurie H. and Subbarao Kambhampati (1997). "Storing and Indexing Plan Derivations through Explanation-based Analysis of Retrieval Failures" In. *Journal of Artificial Intelligence Research (JAIR)* 7, pp. 161–198 (page 69).

Ingrand, Felix, Raja Chatila, Rachid Alami, and Frederic Robert (1996). "PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots" In. *In Proceedings of the IEEE International Conference On Robotics and Automation (ICRA)*, pp. 43–49 (page 72).

Ishida, Toru and Richard E. Korf (1991). "Moving Target Search" In. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*. Sydney, Australia, pp. 204–211 (pages 56, 58).

Jimenez, Sergio, Andrew I. Coles, and Amanda J. Smith (2006). "Planning in Probabilistic Domains Using a Deterministic Numeric Planner" In. *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)* (page 36).

Jiménez, Sergio, Fernando Fernández, and Daniel Borrajo (2008). "The PELA Architecture: Integrating Planning and Learning to Improve Execution" In. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*. Chicago, Illinois, USA, pp. 1294–1299 (page 74).

Kalyanam, R. and R. Givan (2008). "LDFS with deterministic plan based subgoals" In. *Proceedings of the sixth International Planning Competition. Eighteenth International Conference on Automated Planning and Scheduling*. Freigburg, Germany (page 36).

Kambhampati, Subbarao, Eric Parker, and Eric Lambrecht (1997). "Understanding and extending graphplan" In. *Recent Advances in Planning: Fourth European Conference on Planning, ECP'97*. SpringerVerlag, pp. 260–272 (page 26).

Karpas, Erez and Carmel Domshlak (2009). "Cost-Optimal Planning with Landmarks" In. *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pp. 1728–1733 (pages 130, 134).

Katz, Michael, Jörg Hoffmann, and Carmel Domshlak (2013). "Red-Black Relaxed Plan Heuristics" In. *Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence*. Bellevue, Washington, USA (page 155).

Kautz, Henry and Bart Selman (1992). "Planning as satisfiability" In. *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*. Vienna, Austria, pp. 359–363 (page 27).

Keiji Nagatani, Howie Choset and Sebastian Thrun (1998). "Towards exact localization without explicit localization with the generalized Voronoi graph" In. *Proceeding of the*

IEEE International Conferente on Robotics and Automation (ICRA). Leuven, Belgium, pp. 342–348 (page 76).

Keller, Thomas and Patrick Eyerich (2012). "PROST: Probabilistic Planning Based on UCT" In. *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*. Atiba´i: AAAI Press, pp. 119–127 (page 37).

Keyder, Emil, Silvia Richter, and Malte Helmert (2010). "Sound and Complete Landmarks for And/Or Graphs" In. *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. Amsterdam, The Netherlands, pp. 335–340 (pages 134, 153).

Knoblock, Craig, Josh Tenenberg, and Qiang Yang (1990). "Learning abstractions hierarchies for problem solving" In. *Proceedings of the Eighth National Conference of Artifical Intelligence*. Boston, MA, pp. 223–228 (pages 6, 92).

Knoblock, Craig A. (1994). "Automatically Generating Abstractions for Planning" In. *Artificial Intelligence* 68.2, pp. 243–302 (page 93).

Knoblock, Craig A., Josh D. Tenenberg, and Qiang Yang (1991). "Characterizing Abstraction Hierarchies for Planning" In. *Proceedings of the nineth National Conference on Artificial Intelligence*. Vol. 2. Anaheim, CA, USA, pp. 692–697 (pages 51, 54).

Kocsis, Levente and Csaba Szepesvári (2006). "Bandit Based Monte-carlo Planning" In. *Proceedings of the 17th European Conference on Machine Learning*. Berlin, Germany, pp. 282–293 (page 37).

Koenig, Sven (1996). "Agent-Centered Search: Situated Search with Small Look-Ahead" In. *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*. Portland, Oregon, USA, pp. 13–65 (page 56).

Koenig, Sven, David Furcy, and Colin Bauer (2002). "Heuristic Search-Based Replanning." In. *Proceedings of the sixth International Conference on Artificial Intelligence Planning Systems*, pp. 294–301 (page 68).

Koenig, Sven, Maxim Likhachev, and David Furcy (2004). "Lifelong Planning A*" In. *Artificial Intelligence Journal* 155.1-2, pp. 93–146. DOI: 10.1016/j.artint.2003.12.001 (page 57).

Koller, Alexander and Jörg Hoffmann (2010). "Waking Up a Sleeping Rabbit: On Natural-Language Sentence Generation with FF" In. *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*. Toronto, Ontario, Canada, pp. 238–241 (page 11).

Kolobov, Andrey, Mausam, and Daniel S. Weld (2010). "Classical Planning in MDP Heuristics: with a Little Help from Generalization" In. *Proceedings of the Twentyth International Conference on Automated Planning and Scheduling, ICAPS 2010*. Toronto, Ontario, Canada, pp. 97–104 (page 2).

Korf, Richard E. (1985). "Depth-first Iterative-Deepening: An Optimal Admissible Tree Search" In. *Artificial Intelligence* 27, pp. 97–109 (page 56).

Korf, Richard E. (1990). "Real-time Heuristic Search" In. *Artificial Intelligence Journal* 42.2-3, pp. 189–211. DOI: 10.1016/0004-3702(90)90054-4 (pages 3, 56, 58).

Krogt, Roman Van Der and Mathijs De Weerdt (2005). "Plan Repair as an Extension of Planning" In. *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*. Monterey, CA, USA, pp. 161–170 (page 2).

Kushmerick, Nicholas, Steve Hanks, and Daniel Weld (1995). "An Algorithm for Probabilistic Planning" In. *Artificial Intelligence Journal* 76.1-2, pp. 239–286. DOI: 10.1002/spe.396 (page 33).

Likhachev, Maxim, Geoff Gordon, and Sebastian Thrun (2004). "ARA*: Anytime A* with Provable Bounds on Sub-Optimality" In. *Proceedings of the 16th conference in advances in neural information proccessing systems (NIPS)*, pp. 767–774 (page 57).

Likhachev, Maxim, David Ferguson, Geoffrey Gordon, Anthony Stentz, and Sebastian Thrun (2005). "Anytime Dynamic A*: An Anytime, Replanning Algorithm" In. *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)* (page 57).

Little, Iain and Sylvie Thiébaux (2006). "Concurrent Probabilistic Planning in the Graphplan Framework" In. *In Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, pp. 263–273 (page 33).

Littman, Michael L., Judy Goldsmith, and Martin Mundhenk (1998). "The Computational Complexity of Probabilistic Planning" In. *Journal of Artificial Intelligence Research* 9.1, pp. 1–36 (pages 4, 30, 55).

Luis, Nerea, Sofía Herreo, and Moisés Martínez (2016). "Robot Collaboration in a Warehouse Environment through Planning and Execution" In. *Proceedings of the first IJCAI Workshop on Autonomous Mobile Service Robots*. New York, USA (page 80).

Majercik, Stephen M. and Michael L. Littman (1998). "MAXPLAN: A New Approach to Probabilistic Planning." In. *The Fourth International Conference on Artificial Intelligence Planning Systems*. Pittsburgh Pennsylvania, USA, pp. 86–93 (page 36).

Martínez, Moisés, Fernando Fernández, and Daniel Borrajo (2012). "Variable resolution planning through predicate relaxation" In. *Proceedings of ICAPS'12 workshop on Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx)*. Atibaia, São Paulo, Brazil, pp. 5–12 (page 95).

Martínez, Moisés, Fernando Fernández, and Daniel Borrajo (2013). "Selective Abstraction in Automated Planning." In. *Proceedings of Second Annual Conference on Advances in Cognitive Systems (Cogsys)*. Baltimore, USA, pp. 133–147 (pages 95, 96).

Matthies, Larry H. et al. (2002). "A portable, autonomous, urban reconnaissance robot" In. *Robotics and Autonomous Systems* 40.2-3, pp. 163–172. DOI: 10.1016/S0921-8890(02)00241-5 (page 56).

Mausam and Daniel S. Weld (2008). "Planning with Durative Actions in Stochastic Domains" In. *Journal of Artificial Intelligence Research* 31.1, pp. 33–82 (page 64).

McAllester, David A. and David Rosenblitt (1991). "Systematic Nonlinear Planning" In. *Proceedings of the 9th National Conference on Artificial Intelligence*. Vol. 2. Anaheim, CA, USA, pp. 634–639 (page 44).

McCain, Norman and Hudson Turner (1997). "Causal Theories of Action and Change" In. *Proceedings of the 14th National Conference on Artificial Intelligence*. Providence, Rhode Island, USA, pp. 460–465 (page 41).

McCarthy, John and Patrick J. Hayes (1969). "Some Philosophical Problems from the Standpoint of Artificial Intelligence" In. *Machine Intelligence*. Edinburgh University Press, pp. 463–502 (page 21).

Mcdermott, Drew (2000). "The 1998 AI Planning Systems Competition" In. *AI Magazine* 21, pp. 35–55 (pages 14, 24).

McGann, Connor, Frederic Py, Kanna Rajan, John P. Ryan, and Richard Henthorn (2008a). "Adaptive Control for Autonomous Underwater Vehicles" In. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*. Chicago, Illinois, USA, pp. 1319–1324 (page 73).

McGann, Conor, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Robert S. McEwen (2008b). "A deliberative architecture for AUV control" In. *2008 IEEE International Conference on Robotics and Automation*. Pasadena, California, USA, pp. 1049–1054 (page 73).

Mendelson, Elliott (1987). *Introduction to Mathematical Logic*. Vol. 3. Monterey, CA, USA: Wadsworth and Brooks/Cole Advanced Books & Software (page 16).

Muscettola, Nicola (1994). "HSTS: Integrating planning and scheduling" In. *Intelligent Scheduling*, pp. 169–212 (page 70).

Nareyek, Alexander and Tuomas Sandholm (2003). "Planning in dynamic worlds: More than external events" In. *Proceedings of the Workshop on agents and automated reasoning (IJCAI)*. Acapulco, Mexico, pp. 30–35 (page 64).

Nau, Dana, Malik Ghallab, and Paolo Traverso (2004). *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (page 11).

Nebel, Bernhard and Jana Koehler (1995). "Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis" In. *Artificial Intelligence Journal* 76, pp. 427–454 (page 68).

Newell, A. and H.A. Simon (1972). *Human Problem Solving*. Upper Saddle River, NJ, USA: Prentice Hall, Englewood Cliffs (pages 12, 47, 50).

Onaindia, Eva, Oscar Sapena, Laura Sebastia, and Eliseo Marzal (2001). "SimPlanner: An Execution-Monitoring System for Replanning in Dynamic Worlds." In. *Proceedings of the Artificial Intelligence International Conference in Portugal (EPIA)*. Vol. 2258. Lecture Notes in Computer Science, pp. 393–400 (page 68).

Palacios, Héctor and Hector Geffner (2005). "Mapping Conformant Planning into SAT Through Compilation and Projection" In. *Proceedings of the 11th Conference of the Spanish Association for Artificial Intelligence*. Santiago de Compostela, Spain, pp. 311–320. DOI: 10.1007/11881216_33 (pages 2, 41).

Palacios, Héctor and Héctor Geffner (2006). "Compiling Uncertainty Away: Solving Conformant Planning Problems using a Classical Planner (Sometimes)" In. *Proceedings of the Twenty-first AAAI National Conference on Artificial Intelligence*. Boston, Massachusetts, USA, pp. 900–905 (page 40).

Palacios, Héctor and Héctor Geffner (2007). "From Conformant into Classical Planning: Efficient Translations that May Be Complete Too" In. *Proceeding of the seventeenth International Conference on Automated Planning and Scheduling*, pp. 264–271 (pages 40, 45).

Pearl, Judea (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (pages 26, 52).

Pednault, Edwin P. D. (1994). "ADL and the State-Transition Model of Action." In. *Journal of Logic and Computation* 4.5, pp. 467–512 (page 22).

Pemberton, Joseph and Richard E. Korf (1994). "Incremental Search Algorithms for Real-Time Decision Making" In. *Proceedings of the 2 nd Artificial Intelligence Planning Systems Conference (AIPS-94*, pp. 140–145 (page 56).

Peot, M. A. and David E. Smith (1992). "Conditional nonlinear planning" In. *Proceedings of the First International Conference on Artificial Intelligence (AAAI).* College Park, Maryland, pp. 189–197 (pages 2, 44, 66).

Pohl, Ira (1970). "Heuristic search viewed as path finding in a graph" In. *Artificial Intelligence Journal* 1.3-4, pp. 193–204 (page 26).

Pommerening, Florian and Malte Helmert (2012). "Optimal Planning for Delete-Free Tasks with Incremental LM-Cut" In. *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling ICAPS.* Atibaia, São Paulo, Brazil (page 134).

Porteous, Julie, Laura Sebastia, and Jörg Hoffmann (2001). "On the extraction, ordering and usage of landmarks in planning" In. *Proceedings of the European Conference of Planning.* Toledo, Spain, pp. 37–48 (pages 27, 129).

Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* 1st. New York, NY, USA (page 28).

Quintero, Ezequiel, Vidal Alcázar, Daniel Borrajo, Juan Fernández-Olivares, Fernando Fernández, Angel García Olaya, César Guzmán, Eva Onaindia, and David Prior (2011). "Autonomous Mobile Robot Control and Learning with the PELEA Architecture." In. *Proceedings of the AAAI-11 Workshop on Automated Action Planning for Autonomous Mobile Robots (PAMR).* San Francisco, CA, USA (page 76).

Rajan, K., C. McGann, F. Py, and H. Thomas (2007). "Robust Mission Planning using Deliberative Autonomy for Autonomous Underwater Vehicles" In. *Proceedings of the Workshop on Robotics in Challenging and Hazardous Environments, ICRA.* Rome, Italy (pages 1, 11).

Ramalingam, G. and Thomas Reps (1996). "An Incremental Algorithm for a Generalization of the Shortest-path Problem" In. *Journal of Algorithms* 21.2, pp. 267–305. DOI: 10.1006/jagm.1996.0046 (page 57).

Reiter, R. (1987). "Readings in Nonmonotonic Reasoning" In. Ed. by Matthew L. Ginsberg. San Francisco, CA, USA. Chap. On Closed World Data Bases, pp. 300–310 (page 21).

Reyna, Álvaro Torralba Arias de, Carlos Linares López, and Daniel Borrajo (2013). "Symbolic Merge-and-Shrink for Cost-Optimal Planning" In. *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI).* Beijing, China (page 53).

Richter, Silvia and Matthias Westphal (2010). "The LAMA Planner: Guiding Cost-based Anytime Planning with Landmarks" In. *Journal of Artificial Intelligence Research* 39.1, pp. 127–177. DOI: 10.1613/jair.2972 (pages 27, 105, 109).

Richter, Silvia and Mattias Westphal (2008). "The LAMA planner using landmark counting in heuristic search" In. *Proceedings of the sixth International Planning Competition. Eighteenth International Conference on Automated Planning and Scheduling.* Freigburg, Germany (page 45).

Richter, Silvia, Malte Helmert, and Matthias Westphal (2008). "Landmarks Revisited" In. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 975–982 (pages 130, 131, 134).

Rintanen, Jussi (1999). "Constructing Conditional Plans by a Theorem-Prover" In. *Journal of Artificial Intelligence Research (JAIR)* 10, pp. 323–352 (page 41).

Rintanen, Jussi, Keijo Heljanko, and Ilkka Niemelä (2006). "Planning as satisfiability: parallel plans and algorithms for plan search" In. *Artificial Intelligence Research* 170.12-13, pp. 1031–1080 (page 28).

Sacerdoti, Earl D. (1972). "Planning in a hierarchy of abstraction spaces" In. *Artificial Intelligence* 2.5, pp. 115–135 (page 50).

Sanner, Scott (2011). *Relational Dynamic Influence Diagram Language (RDDL): Language Description* (pages 31, 33).

Sanner, Scott and Craig Boutilier (2005). "Probabilistic planning via linear valueapproximation of first-order mdps" In. *In Proceedings of the fifth International Planning Competition. International Conference on Automated Planning and Scheduling* (page 35).

Sebastia, Laura, Eva Onaindia, and Eliseo Marzal (2006). "Decomposition of Planning Problems" In. *Artificial Intelligence Communications* 19.1, pp. 49–81 (page 3).

Shimbo, Masashi and Toru Ishida (2003). "Controlling the learning process of real-time heuristic search" In. *Artificial Intelligence Journal* 146.1, pp. 1–41. DOI: 10.1016/S0004-3702(03)00012-2 (page 58).

Simmons, Reid (1992). "Concurrent Planning and Execution for Autonomous Robots" In. *IEEE Control Systems* 12.1, pp. 46–50 (page 71).

Simmons, Reid G., Richard Goodwin, Karen Zita Haigh, Sven Koenig, Joseph O'Sullivan, and Manuela M. Veloso (1997). "Xavier: Experience with a Layered Robot Architecture" In. *ACM SIGART Bulletin* 8.1-4, pp. 22–33. DOI: 10.1145/272874.272878 (page 71).

Simon, Herbert and Allen H. Newell (1969). "Gps: A case study in generality and problem solving" In. *Artificial Intelligence* 2.5, pp. 109–124 (page 50).

Smith, David E. (2004). "Choosing Objectives in Over-Subscription Planning." In. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pp. 393–401 (page 24).

Smith, David E. and Daniel S. Weld (1998). "Conformant Graphplan" In. *Proceedings of the fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pp. 889–896 (pages 26, 39).

Son, Tran Cao, Phan Huy Tu, Michael Gelfond, and A. Ricardo Morales (2005). "Conformant Planning for Domains with Constraints: A New Approach" In. *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3.* Pittsburgh, Pennsylvania, pp. 1211–1216 (page 40).

Stentz, Anthony (1995a). "The Focussed D* Algorithm for Real-time Replanning" In. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA, pp. 1652–1659 (page 57).

Stentz, Anthony (1995b). "The Focussed D* Algorithm for Real-time Replanning" In. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Montreal, Quebec, Canada, pp. 1652–1659 (page 100).

Stentz, Anthony and Martial Hebert (1995). "A Complete Navigation System for Goal Acquisition in Unknown Environments" In. *Autonomous Robots* 2, pp. 127–145 (page 56).

Teichteil-königsbuch, Florent, Guillaume Infantes, and Ugur Kuter (2008). *RFF: A Robust, FF-Based MDP Planning Algorithm for Generating Policies with Low Probability of Failure* (page 36).

Tompkins, Dave A. D., Adrian Balint, and Holger H. Hoos (2011). "Captain Jack: New Variable Selection Heuristics in Local Search for SAT" In. *Proceedings of the 14th International Conference on Theory and Application of Satisfiability Testing*. Ann Arbor, MI, pp. 302–316 (page 28).

Trope, Yaacov and Nira Liberman (2010). "Construal-level theory of psychological distance." In. *Psychological Review* 117.2, pp. 440–463. DOI: 10.1037/a0018963 (page 85).

Veloso, Manuela M. (1993). "Prodigy/Analogy: Analogical Reasoning in General Problem Solving." In. *Proceedings of the first European Conferente on Case-Based Reasoning*. Vol. 837. Lecture Notes in Computer Science, pp. 33–52 (page 69).

Veloso, Manuela M., Martha E. Pollack, and Michael T. Cox (1998a). "Rationale-Based Monitoring for Continuous Planning in Dynamic Environments" In. *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*. Pittsburgh, PA, USA, pp. 171–179 (page 64).

Veloso, Manuela M., Martha E. Pollack, and Michael T. Cox (1998b). "Rationale-Based Monitoring for Planning in Dynamic Environments." In. *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 171–180 (page 70).

Wang, Chenggang, Saket Joshi, and Roni Khardon (2007). "First Order Decision Diagrams for Relational MDPs" In. *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 1095–1100 (page 35).

Washington, Richard (1995). "Incremental Planning for Truly Integrated Planning and Reaction" In. Vol. 28, pp. 28–40 (page 66).

Weld, Daniel S., Corin R. Anderson, and David E. Smith (1998). "Extending Graphplan to Handle Uncertainty & Sensing Actions." In. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 897–904 (page 44).

Wilkins, David E. (1990). "Can AI Planners Solve Practical Problems?" In. *Computational Intelligence* 6.4, pp. 232–246 (page 68).

Williams, Brian C. and Pandurang P. Nayak (1996). "A Model-based Approach to Reactive Self-Configuring Systems" In. *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. Vol. 2, 971—978 (page 70).

Wu, Jia-Hong, Rajesh Kalyanam, and Robert Givan (2011). "Stochastic Enforced Hill-climbing" In. *Artificial Intelligence Journal* 42.1, pp. 815–850. DOI: 10.1613/jair.3420 (pages 35, 45).

Wurman, Peter R., Raffaello D'Andrea, and Mick Mountz (2007). "Coordinating Hundreds of Cooperative, Autonomous Vehicles in **W**arehouses" In. *Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence*. Vol. 2 (page 107).

Yang, Qiang, Josh D. Tenenberg, and Steven **W**oods (1991). *Abstraction in Nonlinear Planning*. Tech. rep. University of **W**aterloo (page 6).

Yang, Simon X. and Max Meng (2003). "Real-time collision-free motion planning of a mobile robot using a Neural Dynamics-based approach" In. *International Journal of Robotics and Automation* 2.3, pp. 1541–1552 (page 3).

Yoon, Sung **W**ook, Alan Fern, and Robert Givan (2007). "FF-Replan: A Baseline for Probabilistic Planning" In. *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*. Providence, Rhode Island, USA, pp. 22–26 (pages 2, 34, 68).

Yoon, Sung Wook, Alan Fern, Robert Givan, and Subbarao Kambhampati (2008). "Probabilistic Planning via Determinization in Hindsight." In. *Proceedings of the Twenty-Third Conference on Artificial Intelligence*, pp. 1010–1016 (page 2).

Younes, Häkan L. S. and Michael L. Littman (2004). "PPDDL1.0: An extension to pddl for expressing planning domains with probabilistic effects" In. *In Technical Report CMU-CS-04-162* (page 30).

Younes, Häkan L. S., Michael L. Littman, David Weissman, and John Asmuth (2005). "The First Probabilistic Track of the International Planning Competition. International Conference on Automated Planning and Scheduling" In. *Journal of Artificial Intelligente Research* 24, pp. 851–887 (pages 30, 39, 44, 81, 104).

Zalama, Eduado, Jaime Gómez, Mariano Paul, and Perán José Ramón (2002). "Adaptive behavior navigation of a mobile robot" In. *IEEE International Conference on Systems, Man, and CybernAdaptive behavior navigation of a mobile robotetics, Part A: Systems and Humans*. Vol. 31. 3, pp. 160–169 (page 3).

Zettlemoyer, Luke S., Hanna Pasula, and Leslie Pack Kaelbling (2005). "Learning Planning Rules in Noisy Stochastic Worlds" In. *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pp. 911–918 (pages 2, 4, 65).

Zhu, Anmin and Simon X. Yang (2010). "A goal-oriented fuzzy reactive control for mobile robots with automatic rule optimization." In. *International Conference on Intelligent Robots and Systems*, pp. 3688–3693 (page 3).

Zhu, Lin and Robert Givan (2003). "Landmark Extraction via Planning Graph Propagation" In. *In ICAPS DOCTORAL Consortium* (page 131).

Zickler, Stefan and Manuela Veloso (2010). "Variable Level-of-Detail Motion Planning in Environments with Poorly Predictable Bodies" In. *Proceeding of the nineteenth European Conference on Artificial Intelligence*. Lisbon, Portugal, pp. 189–194 (page 3).