Plan Merging via Plan Reuse

By

NEREA LUIS MINGUEZA

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science and Technology

UNIVERSIDAD CARLOS III DE MADRID

Advisor(s):
DANIEL BORRAJO MILLÁN
SUSANA FERNÁNDEZ ARREGUI

MAY 20, 2019

This page has been intentionally left blank.

Humans are allergic to change.
They love to say:
"We've always done it this way."
I try to fight that. That's why I
have a clock on my wall that
runs counter-clockwise.

Grace Murray Hopper
(1906 - 1992)

This page has been intentionally left blank.

Acknowledgements

«Has llegado a la meta, Nerea».

«Haciendo una analogía con el área de tu tesis, se podría decir que el plan es válido ¿no?» — dijo el algoritmo de planificación clásica.

«Bueno, perdona pero un plan de cuatro años y medio no es tan sencillo de ejecutar a la primera... hay que replanificar unas cuantas veces» — replicó RRPT-PLAN, el algoritmo de reutilización de planes.

«¡Y no sólo eso! es una tarea de planificación excesivamente grande, habrá que dividirla y coordinarse con distintos "agentes"... ¡Ostras! dicho así hasta se parece a lo que has escrito en estas páginas, Nerea. » — respondió orgulloso PMR, el algoritmo de planificación multi-agente.

« Bueno, veréis... la verdad es que una tesis no sólo se hace siguiendo un plan o una fórmula... es algo más complicado, mucho más "humano". » — comentó Nerea. « Por el camino se cruzan ideas, personas, momentos, reuniones, eventos, conocimiento...».

«¿De verdad? ¿Pero no decís los humanos que los algoritmos y las máquinas somos buenos y eficientes? » — se preguntaron con intriga.

« ¡Ja, ja, ja! Claro que sí, pero a veces no es suficiente con eso, al menos no en 2019... veréis...» — comenzó a contar Nerea.

Si echo la vista atrás me doy cuenta de que han pasado tantas cosas... cuatro años y medio es mucho tiempo. Afortunadamente, hay muchas personas a las que agradecer y mencionar, pues sin ellas seguramente el camino hubiese sido aún más duro.

A finales de 2014 decidí que quería hacer un doctorado en inteligencia artificial. La primera persona que apoyó esa decisión fue Daniel, mi director de tesis. Poco después se unió Susana, mi directora. Gracias a los dos por guiarme y acompañarme en este camino; por la disciplina, la rigurosidad que me habéis inculcado. Gracias también al grupo PLG, que me acogió como una más desde que llegué en 2012 pidiendo trabajar con robots. En especial gracias a Ángel, por las innumerables horas compartidas de docencia, y a Raquel, porque hablar de cómo nos sentimos nos hace más fuertes y tú has sido clave.

Según llegué al laboratorio tuve la suerte de encontrarme con un grupo de gente excepcional. Tres de ellos me han acompañado a lo largo de estos años: Jesús, gracias por tu incansable sonrisa; Sergio, gracias por tu cariño, los consejos, abrazos y todos los momentos compartidos, sin excepción. Moisés, gracias por ser un gran compañero de despacho pero, ante todo, un gran amigo; sabes que nos queda mucho por recorrer.

Durante el doctorado he podido desarrollar paralelamente T3chFest, junto a un equipo increíble que se ha convertido mi segunda familia. Gracias Baldo, Mario, Áxel,

David; por muchos años más votando charlas en navidad y dando regalos de Reyes que hagan más diverso y accesible el sector tecnológico. Ali, Diego, Marina, Sara, María, Álvaro, Lisardo...sois tantos que no tengo espacio para nombraros uno a uno, pero ole ese equipazo de voluntarios y lo que os echen.

Con T3chFest llegaron los eventos y las comunidades, y así mucha gente con un gran potencial y corazón. Carlos, Alberto, Abraham, Teba, Aurora, María, Lacarra, Laura M., Laura V., Paola, Carlos A., Ulises, Susana, Almo,... sois solo algunos de los representantes. Inés, tú llegaste de imprevisto y ahora no me imagino un proyecto sin ti; gracias por la fuerza, las locuras y los ánimos. Por muchos más R-Ladies, Pyladies, OSW, GDGs,...

Si por algo hemos luchado ha sido por escuchar más voces femeninas y servir de altavoz: Alma, Ana, Marta, Patry, Sabela,... gracias por ser un gran ejemplo para mí. Y a mis queridas Ferminas, ¡ésto sólo acaba de empezar!

Let me switch to English to express my gratitude to Prof. Dr. Manuela Veloso. You will always be an incredible role-model to me. Thank you for giving me the chance of living Carnegie Mellon University at its fullest and encourage me to believe in my ideas. It was a chance to start from scratch. Fortunately, Lenka crossed in the middle. I learned so much from you, I miss those days together 24/7. You taught me the brightest side of research and helped me in so many ways, thank you and hope to meet you anywhere in the world. Tiago, thanks for all the hours spent at lab and post-CMU, we made it! and Kim,... I will never forget my last day in NY dancing together in Brooklyn, thank you. Also, thank you Bram, I would' ve never expected to connect with someone as fast as we both did. I am forever grateful for every minute we spent together, no matter we say good-bye 'before sunrise' or loose some train on the way.

Si algo me llevo de Pittsburgh es mi encuentro con Luz Rello y su emoción por hacer de Change Dyslexia una realidad. Los fried pickles nos unieron para siempre, gracias por tanto.

Volver después de la estancia a la UC3M también tuvo sus cosas buenas. Gracias Elena por vivir este último año y medio juntas. Gracias Álex por re-aparecer y sacarme las fuerzas necesarias para seguir adelante. Sabes lo mucho que te quiero a mi lado. Madrid, Braunschweig o Berlín está lleno de recuerdos de los dos... ¡y los que quedan!.

A la UC3M siempre le tendré especial cariño: Araceli, Calderón, Carbó, Belén,... todo el departamento en sí, gracias, así como a Celeste, Daniel y Paco. A Juan Margalef, ¡bendito momento en el que te escribí!. En 12 meses pasamos de tomar un café a hablar en el Congreso o con los Reyes,.. gracias por hacerme disfrutar cada momento.

A nivel profesional surgieron oportunidades que a día de hoy no puedo dejar de agradecer a Jorge de la Fundación COTEC, al equipo de Ciencia en el Parlamento/FECYT y a Maysoun de La Nave. Gracias por creer en mí.

Gracias al I.E.S El Espinillo por abrirme las puertas a volver y contar lo que soy gracias su profesorado y a la educación pública.

Gracias también a los que estuvieron en los inicios aunque ahora estén más lejos: Stefy, Lucía, Silvia, Andie, Dani. Y Rodry, porque este logro también lleva un gran pedacito de ti. Gracias por cuidar de mí.

Finalmente, gracias a mi familia por el incansable apoyo, independientemente de que los días sean buenos o malos. Gracias por todo el cariño y dedicación, os quiero mucho.

Published and submitted content

Each of the contributions of this Thesis has been published in journals, conferences or workshops. Here we specify (1) the papers that were published and are related to some contribution and (2) in which Chapters the explanations included in those papers are reused or rephrased.

- **Publication #1:**
  (2019) N. Luis, D. Borrajo, S. Fernández. *Plan Merging by Reuse for Multi-agent Planning*. Journal of Applied Intelligence.
  Role: First author.
  DOI: 10.1007/s10489-019-01429-0
  URL: *not available yet as in May 2019. Accepted for publication as in February 2019.*
  Journal Impact factor: 1.983 / Q2.
  Contributions: PMR and RRPT-PLAN.
  Statement: The content from this paper is partially included in the following referred Chapters and Sections:

  - Main Chapters: 5, 6, 7.
  - Sections: 2.3, 3.1, 3.2.4, 3.3.3, 3.7, 3.8.

  The material from this source included in this thesis is not singled out with typographic means and references.

- **Publication #2:**
  (2019) N. Luis, T. Pereira, S. Fernández, A. Moreira, D. Borrajo, M Veloso. *Using Pre-Computed Knowledge for Goal Allocation in Multi-Agent Planning*. Journal of Intelligent & Robotic Systems.
  Role: First author and coordinator of the international collaboration.
  DOI: 10.1007/s10846-019-01022-0.
  URL: https://link.springer.com/article/10.1007%2Fs10846-019-01022-0
  Journal Impact factor: 1.583 / Q3.
  Contribution: Application in Robotics.

Statement: The content from this paper is partially included in Chapter 8. The material from this source included in this thesis is not singled out with typographic means and references.

The following research papers have been published in conferences or workshops. They contain early-stage research of our contributions PMR and Application in Robotics.

- **Publication #3:**
  (2018) T. Pereira, N. Luis, A. Moreira, D. Borrajo, M Veloso, S. Fernández. *Heterogeneous Multi-agent Planning Using Actuation Maps*. IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC).
  Role: Second author.
  DOI: 10.1109/ICARSC.2018.8374186
  URL: `https://ieeexplore.ieee.org/document/8374186`

- **Publication #4:**
  (2014) N. Luis, D. Borrajo. *Plan Merging by Reuse for Multi-agent Planning*. 2nd Workshop on Distributed and Multi-Agent Planning (ICAPS DMAP-2014). Pages 38-44.
  Role: First author.
  URL: `http://icaps14.icaps-conference.org/proceedings/dmap/DMAP_proceedings.pdf`

Other research merits

During PhD a research stay has been done, different awards and scholarships have been obtained and research content unrelated to this Thesis has been published. Here it is the list of merits:

Awards and scholarships

These are awards and scholarships obtained during PhD. They are divided into research stay, awards, scholarships, travel grants and appreciation awards.

Research stay:

- (2016) 4 months at Carnegie Mellon University (Pittsburgh, PA, USA) under the supervision of Prof. Dr. Manuela Veloso. Partially funded by Universidad Carlos II de Madrid through “Ayudas para estancias doctorales”.

National awards:

- (2019) Premio de Excelencia, category: Alumni, Universidad Carlos III de Madrid.
- (2018) Innovadoras TIC award by Fundacion Cibervoluntarios.

International Scholarships:

- (2017) Google scholarship for Google I/O.
- (2017) Google scholarship for Grace Hopper Conference.
- (2016) Women Techmakers’ scholarship, category: Europe, Middle East and Africa, Google.
- (2016) Anita Borg Institute scholarship for Grace Hopper Conference.

Travel grants to attend research conferences:

- (2018) 13th Workshop of Women in Machine Learning, NeurIPS edition, Montréal. Poster presentation.

- (2018) ACM WomENcourage, Belgrade. Poster presentation.
- (2016) International Joint Conference on Artificial Intelligence, New York. Workshop presentation.

Appreciation awards:

- (2018) Nomination as Emerging Talent, Top100 Mujeres Líderes in Spain.
- (2018) Selected as expert on technology, talent and gender (ODS #5) by Fundacion COTEC.
- (2018) Selected as scientific advice technician by Ciencia en el Parlamento to present scientific-based evidences in Congress.

Publications unrelated to this Thesis

- (2018) M. Martínez, N. Luis. *Goal-Reasoning in StarCraft: Brood War through Multilevel Planning.* XVIII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA).
  URL: `https://sci2s.ugr.es/caepia18/proceedings/docs/CAEPIA2018_paper_171.pdf`
- (2018) N. Luis, M. Martínez. *Reinforcement Learning over Starcraft: Brood War*. 13th Women in Machine Learning Workshop (WiML 2018). Poster presentation.
- (2018) N. Luis, M. Martínez. *STAICRAFT: a Starcraft-based educational platform for Artificial Intelligence.* ACM WomENcourage. Poster presentation.
- (2016) N. Luis, S. Villaroya, M. Martínez. *Robot Collaboration in a Warehouse Environment through Planning and Execution.* Workshop on Autonomous Mobile Service Robots, IJCAI.
- (2015) A. Baldominos, N. Luis, M.C. García del Pozo. *OpinAIS: An Artificial Immune System-based Framework for Opinion Mining*. International Journal of Interactive Multimedia and Artificial Intelligence.
  URL: `https://www.ijimai.org/journal/node/764`

Abstract

Multi-agent planning deals with the task of generating a plan for/by a set of agents that jointly solve a planning problem. One of the biggest challenges is how to handle interactions arising from agents' actions. There exist some other relevant challenges such as planner's scalability when agents, goals or resources increase; or improving the makespan of the resulting plan. In this Thesis, we present Plan Merging by Reuse, PMR, a multi-agent planner that automatically adjusts its behaviour to the level of interaction. Given a multi-agent planning task, PMR decides which agents will try to achieve each goal. The chosen agents solve their individual planning tasks. The resulting plans are merged and PMR checks the plan's validity. Given the potential interactions, merged plans are not always valid. When that happens, PMR performs planning by reuse to generate a valid plan. In order to deal with the problem of agents' coordination, another contribution presented on this Thesis is RRPT, a stochastic plan-reuse planner. RRPT is able to adjust itself to two different cases: when the input invalid plan is very similar to the final valid solution (classic plan reuse scenario) and also when the input plan is completely different from the final solution. RRPT combines plan reuse and standard search. It will decide stochastically on each iteration which technique to run. Thus, RRPT adapts itself to a wide variety of scenarios. We have performed extensive sets of experiments in order to analyze when to use the different PMR variants, as well as which tasks are more appropriate to be solved by PMR. Our contributions obtain solutions to multi-agent planning tasks where PMR and RRPT can successfully adapt their behavior to the particularities of the problems.

This page has been intentionally left blank.

Table of Contents

List of Tables

List of Figures

Glossary

Actuation Map (AM). Image representation of the reachability limit of a robot's motion and actuation over a map. 120

Actuation Space (AS). Set that contains all the waypoints that can be actuated by a robot. Its representation as an image is called Actuation Map. 126

Actuation Task Performing an operation that results in some task being executed in the environment. 120

Automated Planning (AP). Field of Artificial Intelligence that studies the deliberative processes of reasoning. 5

Competition of Distributed and Multi-Agent Planning (CoDMAP). Preliminary version of a Multi-Agent Planning systems competition, which took place in 2015. 32

Coverage Planning metric that represents the number of problems solved. 20

Domain Describes the environment in which the problem has to be solved and the set of actions that can be performed. Thus the domain contains the facts and actions and may also contain types, fluents, constants, etc. 12

International Planning Competition (IPC). Empirical evaluation of state-of-the-art planning systems on a number of benchmark problems. Besides the results, the aim is to identify challenges and future directions in the Automated Planning research area. The competition is organized and celebrated in the International Conference on Planning and Scheduling (ICAPS). 18, 23

Makespan Planning metric that represents the number of steps of a plan that can be executed in parallel. 22

Multi-Agent Planning (MAP). Subarea of Automated Planning that deals with multiple agents and might solve interactions, coordination and privacy issues. 5

Plan Cost Planning metric that represents the sum of associated costs to each of the actions of a given plan. 21

Plan Length Planning metric that represents the number of actions of a plan. 21

Planning by Reuse Tries to fix a given (usually invalid) plan instead of planning from scratch a new solution. Belongs to the subarea of Automated Planning called plan repair. 55

Planning Domain Description Language (PDDL). It is the standard language in Automated Planning to model domains and problems. 14

Problem Contains the information of the environment, the initial state and the set of goals to achieve. 12

Quality Score that represents the cost of executing a valid plan. It is obtained by the sum of the costs associated to each action of a plan. 21

Part I

Introduction

CHAPTER 1

INTRODUCTION

When diving into this Thesis you will find a work that joins two worlds: Artificial Intelligence and Multi-Agent Systems. From theory to practice, we have been exploring different approaches to come up with a solution based on Automated Planning that efficiently solves problems where multiple agents are involved i.e. agents' goals are reached while coordination and interactions are handled. In a world where people are more connected and move faster each day, we are constantly surrounded by multi-agent scenarios in our daily life: robots that solve tasks on-demand inside a building, luggage routing inside an airport, food/package delivery, warehouse organization, cars on a parking lot, drone swarms, etc. Through this Thesis we present a solid multi-agent approach that can be applied to a broad set of environments. Therefore, this chapter is included for the readers to fully understand the aim of this Thesis, its scope and the challenges we have faced to accomplish each of the objectives. Every idea or hypothesis has always some motivation behind. Thus, that will be our starting point.

1.1 Motivation

Since Artificial Intelligence was born in the early 50s, one of the challenges the researchers have been facing is how to replicate the act of human reasoning. Understanding how our brain works is still not easy nor obvious. We humans make decisions daily in order to perform "actions" that will affect -usually in a positive way- to ourselves and/or the environment that surrounds us. Some decisions will require more effort and coordination to be successfully performed than others, especially if they involve several people. For instance, when ordering a pizza using a smartphone you will usually do the following:

1. Click on the *app* of the pizza restaurant of your choice and log in
2. Customize your pizza
3. Choose your payment method
4. Confirm and wait for your pizza at home

Although the sequence of steps might be tricky if you are not familiar with ordering online, the reasoning process is trivial. We call deliberative reasoning to the process of inference that selects which set of ordered actions will be performed in order to reach some objective. Deliberative reasoning has multiple levels of abstractions i.e customize your pizza involves to look for available ingredients, to choose your favourite ones, to choose the kind of dough etc. The deeper the abstraction, the less deliberative is the behaviour to undertake. Low-level actions, such as picking up an ingredient, belong to the reactive reasoning, which means that they barely need of any inference or thinking component to be performed.

Automated planning is the field of Artificial Intelligence that studies the processes of deliberative reasoning. The planning algorithms are able to automatically generate, synthesize and organize a set of actions (plan) that, if applied in order, make the system transit from an initial state to a state where goals (objectives) are achieved. Regarding the previous example, as the plan is performed by one human (agent) only, it belongs to the subfield of single-agent planning.

Meanwhile, in the pizza restaurant, the situation is more complicated to handle. From the point of view of deliberative reasoning, a team of humans has to coordinate several times to:

1. Complete each received order
2. Delivery them to the correct destination address
3. Replenish ingredients/doughs/resources when needed

One of the first ways to tackle this problem would be to have a team in charge of doughs, another team for customizing each pizza and one last team for delivery. Each of the teams would have its own plan to carry out its mission. There is still one last issue: communication. Teams could decide to either communicate everything they do or only communicate when something unexpected happens (usually conflicts). In Automated Planning, when multiple agents are involved we talk about Multi-Agent Planning (MAP). This subfield studies not only the reasoning process but also the coordination/cooperation among agents, the communication (especially if some information is private) and the conflicts' resolution. In a world where the digital components and the autonomous/intelligent systems are scaling quickly, multi-agent scenarios are given more often. Here we present some real-life examples where MAP could be applied:

- Logistics problems such as warehouse management where robots help workers to complete the customers' orders e.g. Amazon's Kiva Robots [Wurman et al., 2007] or Alibaba's Quicktrons.
- Transportation problems such as delivery or school bus routes [Kozlak et al., 2004].
- Surveillance tasks in buildings with multiple robots [Zlot and Stentz, 2006] or in open environments using Unmanned Air Vehicles [Nigam, 2014]
- Service robots to assist people in public/private buildings e.g. Cobots [Veloso et al., 2015] or STRANDS [Santos et al., 2017].
- Bots behavior in games [Brafman et al., 2009] simulations [Kohan Marzagão et al., 2017] or real-life games [Kok et al., 2005].

One of the problems that MAP faces is scalability. Having a considerable amount of agents and goals produces bottlenecks on task assignment, planning and communication. Depending on the level of interaction among agents, communication could be avoided as long as there exists some algorithm to deal with interactions/coordination.

In big-size scenarios, exploring the search space implies to spend a considerable amount of planning time. Therefore, skipping communication might alleviate the cost of solving the planning task. Also, dividing the goals among the agents in an efficient way e.g. minimizing potential interactions, will make the planning tasks easier to solve. Thus, if after goal-assignment agents can plan individually, the set of plans will be more likely to be parallelized. There is an Automated Planning metric that measures the number of execution steps of a plan, which is called makespan. Minimizing the makespan in this kind of environments will result into having efficient parallel plans that, when executed jointly, will solve the planning task.

The main idea of this Thesis consists on extending some previous works on multi-agent and single-agent planning. Specifically, we want to focus on solving the MAP scalability problem mentioned above. We will focus on big-size multi-agent tasks in terms of number of agents, goals and search space, where agents have little or no interaction among them and the goal is to improve the makespan.

1.2 Objectives

Multi-agent planning (MAP) aims at solving planning tasks for/by a set of agents. Dealing with a set of agents instead of planning for a single agent involves some issues that directly affect to the design of the algorithm. Following the classification proposed by Torreño et al. [2017] they can be grouped into six features: agent distribution (which are the agents or executors?), computational process (centralized or distributed), agents' coordination (before, during or after planning), communication (internal, no communication), heuristic search (per agent or global) and privacy preservation (private information, cipher communication, obfuscation).

This Thesis aims to solve the MAP scalability problem in domains that have little or no interaction where the search space is big and the number of agents and goals are considerable. We aim to skip agents' communication by solving interactions that might arise using plan-merging and plan-reuse after the planning phase. Also, in order to obtain efficient parallel plans, our proposal will focus on minimizing the makespan metric.

Specifically, we contribute with (1) a MAP approach that is able to automatically adapt itself to different multi-agent scenarios using plan-merging and plan-reuse; (2) a new single-agent planning approach that combines heuristic search and plan-reuse, which can be integrated into the former and (3) an adaptation of our approach for robotics environments based on path-planning tasks.

In order to achieve that, the particular steps which we aim to accomplish are:

- Design a new MAP approach that automatically adjusts to the interaction level among agents and goals. We will focus on combining distributed and centralized MAP techniques to solve agents' interactions.

- Explore plan-reuse algorithms to fix agents' interactions. We will focus on post-planning coordination of agents to avoid agents' communication during planning. Thus, a new plan-reuse approach will be proposed.

- Make an analysis on efficiency, scalability, modelling tasks and environment, and agents' features to identify the strengths and weaknesses of our proposal. We will specially focus on the size of the planning task, measured as the number of agents and goals.
- Study a real-world use case to apply our approach. In order to explore and contribute to other areas of knowledge, we will study the robotics surveillance problem.

1.3 Thesis Outline

Previous sections have briefly summarized the motivation behind this Thesis, the main objectives and some observations about the research background. In order to close this introductory Chapter, we present the outline of this Thesis. It is organized as follows:

- Chapter 2 contains a state-of-the-art of Automated Planning, where the key concepts, formalizations, languages and state-of-the-art planners are presented.
- Chapter 3 contains a state-of-the-art of Multi-Agent Planning, the field of Automated Planning that deals with multiple agents. Key concepts, formalizations, languages and state-of-the-art planners are presented.
- Chapter 4 contains a state-of-the-art of Plan Reuse, where key concepts, different algorithms and plan-reuse planners are presented.
- Chapter 5 contains the first contribution of this Thesis, the multi-agent planner Plan Merging by Reuse (PMR).
- Chapter 6 contains the second contribution of this Thesis, the plan reuse planner Reuse Randomly-exploring Planning Tree (RRPT-PLAN).
- Chapter 7 contains the empirical evaluation of each of the contributions first separately and then joined. They were tested against some other state-of-the-art planners.
- Chapter 8 contains a joint work with Manuela Veloso's CORAL lab, where our first contribution was combined with Automated Maps to speed up the planning process and work efficiently on big multi-agent environments.
- Chapter 9 contains some final conclusions and future directions of this Thesis.

This page has been intentionally left blank.

Part II

State of the art

CHAPTER 2

Automated Planning

ARTIFICIAL INTELLIGENCE (AI) was officially born in 1956 with two ambitious objectives. First, to understand intelligent entities. Researchers were interested on studying animals and humans' brains, their behaviour and reasoning. Second, to build intelligent entities by recreating intelligence in machines using computational-based models. One of the subfields of AI that helps to reach both objectives is Automated Planning. In this Chapter, an overview of Automated Planning is given focusing on key concepts, planning models, languages and metrics.

2.1 Introduction

As it was previously said, Artificial Intelligence aims at creating intelligent entities. In order to define "entity" we follow the definition given by Russell and Norvig [2003]:

> *Anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.*

An intelligent entity needs to be rational, which means to acquire knowledge about the world and to reason about possible solutions when acting and performing tasks over the environment. Agents and entities are synonyms in this context. Thus, for the sake of simplicity, we will refer to agents instead of entities.

Any computational-based agent can manifest different degrees of autonomy. Thus, there exist different classifications. Here, we are mentioning the one established by Geffner [2010] where three big groups can be identified:

- *Program-based agents*, which are mainly developed using *if-then-else* sentences. Their knowledge and their ability to react to the environment is limited to the situations anticipated by the developer.
- *Model-based agents*, which solve a given problem using specific knowledge (actions, goals, initial state) that has been previously modelled and received as input. They are more flexible than program-based agents as the solver is domain-independent.
- *Learning-based agents*, which do not need prior knowledge to react to the environment. Since the beginning they are able to learn and understand what it is happening. The ability to learn depends on the quality of the processed information and the algorithm employed.

We mentioned on Chapter 1 that Automated planning is the field of Artificial Intelligence that studies the processes of deliberative reasoning. Agents that are built by using Automated Planning belong to the second group. They are considered model-based and goal-oriented agents. Model-based agents are usually integrated into real systems by following the architecture depicted on Figure 2.1. The Domain contains the representation of the environment and the set of actions that can be performed. The Problem contains the information of the environment, the initial state and the set of goals to achieve. Using that information, the solver computes a plan. It usually contains a sequence of ordered actions. That plan is later executed by a controller over the environment.

On this Thesis, we are focused on the first part of the architecture. The Execution side is out of its scope. As we are presenting a work that lies on Multi-Agent Planning, our approach will be able to coordinate multiple agents to solve what is called a Multi-Agent

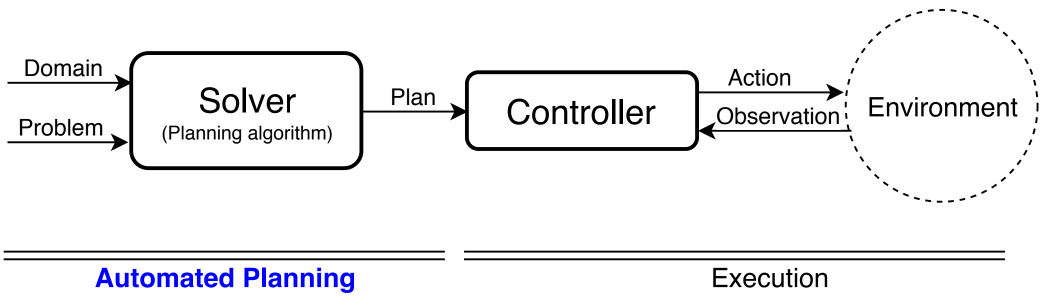


Figure 2.1: Model-based agent's architecture

Planning task. However, in order to fully understand the Multi-Agent Planning approach we first need to present the key concepts of Automated Planning.

The following sections explain the different planning models, the most popular modelling languages employed by the planning community and the formalization of the Single-Agent Planning task. Afterwards, we define the popular metrics and evaluation and present the State-of-the-art of Single-Agent Planning.

2.2 Planning models and modelling languages

Most Automated Planning algorithms are considered domain-independent, which means that the same planning algorithm can solve planning problems from different domains. While the algorithm remains intact, the input sent to the algorithm is the definition of the planning problem and the domain, which are the elements that vary. As it is impossible that any planner works efficiently in every existing domain, the domain-independent feature is bounded to the planning model that the algorithm supports. Thus, there exist different planning models that vary depending on the configuration of the following set of features.

- *Partial or full observability*. When the algorithm has access to all the information about the current state of the environment we refer to full observability. Otherwise, it will have partial observability.

- *Presence of uncertainty*. Uncertainty can be present in the planning actions and in the environment. When the environment is deterministic, any action applied to the environment will only lead to a concrete state. On the contrary, when the same action might lead to different states, the environment presents uncertainty. Another situation that can be given regarding uncertainty is the partially-mistaken perception of the world e.g. noisy sensors. The environment presents uncertainty, as observations cannot be fully trusted.

- *Actions' duration*. Actions can have an assigned duration when executed in the environment. Thus, we can reason about actions' time windows when they are being executed or actions' concurrency.

- *Goal reachability*. A problem can have hard-goals, which are mandatory to reach to solve the problem successfully or soft-goals, which are optional to reach and do not affect to the success of the plan.

- *Offline or online planning*. Online planning takes into account the dynamics of the environment (observations might change) and returns an action on each iteration. Offline planning returns the full plan.

Besides choosing a planning model, we also need a language to model the problem. The basic language for modelling planning problems is STRIPS [Fikes and Nilsson, 1971], the STanford Research Institute Problem Solver. STRIPS employs first-order logic to represent a problem and declares operators (actions) using preconditions and effects. On the other hand, the Planning Domain Description Language (PDDL) [Fox and Long, 2003] is based on the STRIPS language and provides more flexibility and expressiveness for modelling planning problems. It was an attempt to establish a standard language to represent propositional planning tasks. Through the years, PDDL has been extended and has gained even more expressiveness to represent more complex structures such as numerical fluents, conditional effects, different types of constraints and goals etc. It became the standard language to represent a problem in Classical Planning. There exist some other approaches such as the state-variable model SAS+ [Bäckström and Nebel, 1995], which is later explained in Section 2.5 or the Action-Description Language, ADL [Pednault, 1994], which was included into the first version of PDDL.

On this Thesis we are interested on the Classical Planning model, in which there is a deterministic environment, full observability, no uncertainty, duration-less actions and hard goals. Planning tasks will be represented in PDDL. The following Section explains the conceptual model of Classical Planning.

2.3 The propositional formalization

The propositional formalization of a Classical Planning is as follows:

Definition 2.3.1. *Single-agent planning task*. A single-agent planning task is a tuple $\Pi = \langle F, A, I, G \rangle$, where:

- F stands for the set of propositions,
- A stands for the set of instantiated actions,
- $I \subseteq F$ stands for the initial state,
- and $G \subseteq F$ stands for the set of goals

Each action $a \in A$ is described by

- a set of preconditions (pre(a)) that represent literals that must be true in a state to execute the action;
- and a set of effects (eff(a)), which are literals that are expected to be added (add(a) effects) or removed (del(a) effects) from the state after the execution of the action.

The definition of each action might also include a (positive) cost $c(a)$, where the default cost is one.

A state $s \subseteq F$ describes the current situation of the environment. Instantiated literals from F describe s. The state space S contains every possible situation that can be given on the environment. In Classical Planning, S is finite and discrete. Also, $I \in S$ and all goal states $s_G \in S$.

The solution of a planning task is a plan, which is a sequence of actions $\pi = (a_1, \ldots, a_n)$ that, if executed in order from the initial state, reaches another state s_G where all the goals in G are satisfied, $G \subseteq s_G$.

In order to transit to a different state s', an action a must be applicable to s. Therefore, the planning task could also be represented as a State-Transition Graph [Jonsson and Bäckström, 1998]. The application of an action a in a state s is defined by the function:

$$\gamma(s,a) = \begin{cases} (s \backslash \text{del}(a)) \cup \text{add}(a) & \text{if pre}(a) \subseteq s \\ s & \text{Otherwise } a \text{ cannot be applied in } s \end{cases} \tag{2.1}$$

Thus, the execution of a plan π from a state s can be defined as:

$$\Gamma(s,\pi) = \begin{cases} \Gamma(\gamma(s,a_1),(a_2,...a_n)) & \text{if } \pi \neq \emptyset \\ s & \text{if } \pi = \emptyset \end{cases} \tag{2.2}$$

Next, the lifted representation of the Classical Planning task model is presented.

2.4 The PDDL representation

Usually, the Classical Planning task is encoded in the propositional fragment of the standard Planning Domain Definition Language (PDDL) [Fox and Long, 2003]. A planning task Π is generated by the description of a PDDL *domain* and *problem*.

2.4.1 The PDDL domain

The domain represents the general knowledge of the environment. It contains a set of PDDL requirements; a domain name; a hierarchy of types to characterize the problem objects; a set of definitions of predicates and a set of definitions of functions, whose instantiations generate the facts in F; and a set of generalized actions. The instantiations of those actions with the problem objects generate the set A.

There exist multiple domains in the planning community. For instance, domains that simulate the behavior of the space *Rovers*, the behavior of a *Satellite*, the management of an *Elevator*, the *Logistics* of package-delivery, etc. In order to fully understand the modelling process, through this Section we will show a modelling example of a typed version of the Logistics environment, where trucks and airplanes cooperate to deliver packages from the city of origin to the destination. Next, we present the hierarchy of types.

```
(:types    truck
           airplane - vehicle
           package
           vehicle - physobj
           airport
           location - place
           city
           place
           physobj - object)
```

Figure 2.2: Hierarchy of types of the Logistics domain. Types on the left are children of the types situated on the right.

Once the hierarchy of types is defined, the next step is declaring the set of predicates. A predicate is composed of a name plus a list of arguments of some type. Sometimes it is enough to declare a predicate just by its name e.g. (handEmpty), (full). Figure 2.3 shows the predicates that usually define a Logistics domain.

```
(:predicates   (in-city ?loc - place ?city - city)
               (at ?obj - physobj ?loc - place)
               (in ?pkg - package ?veh - vehicle))
```

Figure 2.3: Predicates of the Logistics domain

Finally, actions are composed of parameters, preconditions and effects. The parameters are object types, the preconditions and effects come from the previous declared predicates.

```
(:action LOAD-TRUCK
   :parameters     (?pkg - package ?truck - truck ?loc - place)
   :precondition   (and (at ?truck ?loc) (at ?pkg ?loc))
   :effect         (and (not (at ?pkg ?loc)) (in ?pkg ?truck)))
```

Figure 2.4: Action *Load-truck* of the Logistics domain

2.4.2 The PDDL problem

A planning problem in PDDL contains a set of objects, which are instances of the types in the domain; the initial state I; the set of goals G; and an optional metric to define the optimization criteria. The initial state and the set of goals are composed of instantiated predicates. Figure 2.5 presents a simple Logistics problem.

```
(define (problem logistics-4-0)
(:domain logistics)
(:objects
     apn1 - airplane
     apt1 apt2 - airport
     pos2 pos1 - location
     cit2 cit1 - city
     tru2 tru1 - truck
     obj23 obj22 obj21 obj13 obj12 obj11 - package)

(:init (at apn1 apt2) (at tru1 pos1) (at obj11 pos1) (at obj12 pos1)
       (at obj13 pos1) (at tru2 pos2) (at obj21 pos2) (at obj22 pos2)
       (at obj23 pos2) (in-city pos1 cit1) (in-city apt1 cit1)
       (in-city pos2 cit2) (in-city apt2 cit2))

(:goal (and (at obj11 apt1) (at obj23 pos1) (at obj13 apt1)
            (at obj21 pos1)(at obj12 apt1)(at obj22 apt2)))
)
```

Figure 2.5: Example of a Logistics problem encoded in PDDL

2.4.3 PDDL official versions

Since PDDL was born in 1998, new versions of the language have been defined in order to expand its expressiveness and flexibility. Here we present a brief summary of each PDDL version:

- PDDL 1.2: it is the first version and was released in 1998. This version already divides and models a planning task into the domain and the problem. The language also includes some of the *ADL formalization features* and *conditional effects*. It was the official language of the 1st and 2nd International Planning Competition (IPC).

- PDDL 2.1: it is the second version and was released in 2002. It introduced *durative actions*, *numeric-fluents* (e.g. to represent numeric quantities) and *plan metrics* to optimize the plan regarding different objectives and not only goal-driven. It was the official language of the 3rd IPC.

- PDDL 2.2: it is the third version and was released in 2004. It included the concept of dependency between predicates through a new element called *derived predicate or axioms*. It also included a way to model exogenous events, which are external events that happen at some point in the problem, through a new element called *timed initial literals*. They have assigned a concrete moment of time to be enabled. It was the official language of the 4rd IPC.

- PDDL 3.0: it is the fourth version of the language and was released in 2006. Now the language is able to model *preferences* in the form of hard and soft goals. The former are mandatory to solve the problem, the second are optional. It also included *state-trajectory constraints* to model hard constraints.

- PDDL 3.1: it is the latest version of the language and was released in 2008. It introduced object-fluents and supports functional STRIPS.

There also exist variations of PDDL to work with the different planning models e.g. such as PPDDL to model probabilistic environments and uncertainty.

Finally, most of the current planners do not support PDDL 3.1, instead they usually support PDDL 2.2, which includes enough expressiveness and flexibility to model most of the real-world environments.

2.5 The multi-valued formalization

There are multiple ways to formalize a planning task depending on the features of the planning model. In Classical Planning, there exist two well-known formalizations: the propositional and the multi-valued ones. The previous Sections described the propositional formalization of the planning task, which is usually encoded in PDDL. In this

Section we present SAS+ [Bäckström and Nebel, 1995], as an alternative to define a planning task by following the multi-valued formalization. Both representations will be employed along this Thesis.

Definition 2.5.1. *SAS+ planning task*. A SAS+ planning task is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$, where:

- $\mathcal{V}$ stands for a set of state variables $\mathcal{V} = \{v_1, ..., v_m\}$. Each variable $v \in \mathcal{V}$ is associated to an extended domain in the form $D_v^+ = D_v \cup \{\mathbf{u}\}$, where D_v represents the domain of variable v and $\mathbf{u}$ represents the undefined/unknown value. The total state-space is defined as $S_{\mathcal{V}} = D_1 \times \ldots \times D_m$ and $s[v]$ denotes the value of the variable v in the state s (fluent). The partial state-space is defined as $S_{\mathcal{V}}^+ = D_1^+ \times \ldots \times D_m^+$, in which at least one fluent $s[v_i] = \mathbf{u}$.

- $\mathcal{O}$ stands for the set of operators (actions). Each $o \in \mathcal{O}$ is represented by a tuple $o = \langle pre, post \rangle \in S_{\mathcal{V}}^+$ where each element represents pre- and post- conditions respectively.

- $s_0 \subseteq \mathcal{V}$ stands for the initial state, such that $s_0[v_i] \neq \mathbf{u}$ $\forall v_i \in \mathcal{V}$.

- $s_* \subseteq \mathcal{V}$ stands for a goal state, such that $s_* = [v_i] \in D_{v_m}^+$ $\forall v_i \in \mathcal{V}$.

Given a state $s \in \mathcal{S}_{\mathcal{V}}^+$, the preconditions $pre(o) \in \mathcal{O}$ are fluents that must be true before they are applied to s and become not true right after. The postconditions $post(o) \in \mathcal{O}$ are fluents that are not true before being applied to s and become true right after their application in s.

Thus, an action $o \in \mathcal{O}$ is applicable in a state s iff: $\forall v_i \in \mathcal{V} : (pre(o)[v_i] = \mathbf{u} \lor pre(o)[v_i] = s[v_i])$.

As a result, a solution plan π is a sequence of actions $\pi = (o_1, \ldots, o_k)$ such that if applied in order starting in s_0 will result in a state s_k where $s_* \subseteq s_k$.

2.6 Metrics and evaluation

In the middle 90s, the Automated Planning community became aware of the lack of benchmarking platforms or systems in the research area. Researchers found hard testing their algorithms against other state-of-the-art approaches.

Thus, this situation led to the creation of the International Planning Competition (IPC)[1] in 1998. The aim of the competition was to provide an empirical comparison of the state of the art planning algorithms. Also there were other objectives such as analyzing the community trends, proposing new directions for research, highlighting new challenges and providing new domains and problems sets as benchmarks. The last edition of the IPC took place in 2018.

Due to the success of the IPC, it became the reference for benchmarking any new planner approach. Thus, their evaluation metrics are commonly used by almost every researcher of this area. For a given domain and problem, the metrics to evaluate the solution plan are its coverage, quality and the time spent on obtaining the solution. In order to compute those metrics, first the soundness of the plan (or plan validity) is evaluated using VAL, the validator from the IPC [Howey et al., 2004]. The time spent is evaluated in seconds. The quality of a plan can be evaluated by measuring the plan cost, which is the sum of costs of all the actions contained in the plan. Sometimes, actions do not have any associated costs. Therefore, the quality of a plan would be equivalent to the number of actions of the plan.

However, when comparing different planners, the individual values obtained for quality and time are compared globally as follows.

First, let's say we have a set of planners $PA = \{P_1, P_2 ... P_n\}$ and a set of domains $D = \{D_1, D_2 ... D_k\}$. Each domain D_i is composed of a set of problems $D_i = \{Pr_1, Pr_2, ... Pr_m\}$. Usually, when running experiments, each domain contains the same number of problems. The time bound to solve each problem on Classical Planning is usually 1800s. Thus, each planner runs each domain and problem obtaining a maximum of $k \times m$ plans if every problem is solved in the allotted time. It was mentioned before that three metrics are computed for each plan: coverage (c_{ij}), quality (q_{ij}) and time (t_{ij}); so we can say that for a given planner P_X and a domain D_i, $P_{X_{ij}} = \{c_{ij}, q_{ij}, t_{ij}\}$, where i is the fixed index of the domain in D and j is the index of the problem in D_i. In order to compare the set of planners PA, scores from each of those metrics are computed per planner and domain.

The Coverage metric represents the number of problems solved per domain. Thus, c_{ij} will take the value 0 (not solved) or 1 (solved). The Coverage score (C) per planner and domain is defined in Equation 2.3.

$$C_{P_{X_i}} = \sum_{j=1}^{m} c_{ij} \tag{2.3}$$

[1] ipc.icaps-conference.org

The global Coverage score ($\mathcal{C}$) for each planner is computed as in Equation 2.4.

$$\mathcal{C}_{P_X} = \sum_{i=1}^{k} C_{P_{X_i}} \tag{2.4}$$

The Quality metric (q) measures either the Plan Length or the Plan Cost of a given plan. Each planner P_X obtains a q_{ij} value as long as the problem is solved. Otherwise $q_{ij} = 0$. For every problem, the lowest (best) value obtained from the set of planners is identified as Q_{best}. Equation 2.5 shows the Quality computed for each planner and problem.

$$Q_{P_{X_{ij}}} = \begin{cases} \dfrac{Q_{best}}{q_{ij}} & \text{if problem is solved} \\ 0 & \text{if not solved} \end{cases} \tag{2.5}$$

The Quality score per planner and domain is computed as in Equation 2.6. The global Quality score ($\mathcal{Q}$) for each planner is computed as in Equation 2.7.

$$Q_{P_{X_i}} = \sum_{j=1}^{m} Q_{P_{X_{ij}}} \tag{2.6}$$

$$\mathcal{Q}_{P_X} = \sum_{i=1}^{k} Q_{P_{X_i}} \tag{2.7}$$

The Time metric (t) measures the seconds spent by a planner on solving a given problem. Each planner from the set PA obtains a t_{ij} value as long as the problem is solved. Otherwise $t_{ij} = 0$. For every problem, the lowest value obtained from the set of planners is identified as T_{best}. Equation 2.8 shows the Time computed for each planner and problem solved.

$$T_{P_{X_{ij}}} = \begin{cases} \dfrac{1}{\left(1 + log_{10}(\frac{t_{ij}}{T_{best}})\right)} & \text{if problem is solved} \\ 1 & \text{if } t_{ij} \leq 1s. \\ 0 & \text{if not solved} \end{cases} \tag{2.8}$$

The Time score per planner and domain is computed as in Equation 2.9. The global Time score ($\mathcal{T}$) for each planner is computed as in Equation 2.10.

$$T_{P_{X_i}} = \sum_{j=1}^{m} T_{P_{X_{ij}}} \tag{2.9}$$

$$\mathcal{T}_{P_X} = \sum_{i=1}^{k} T_{P_{X_i}} \tag{2.10}$$

Finally, in Multi-Agent Planning, researchers also use as quality metric the makespan of the plan. Makespan refers to the number of execution steps, where several actions can be executed at each execution step. It is computed using the same formulas of the Quality metric: Equations 2.5, 2.6 and 2.7, unless the planner cannot return a parallel plan.

2.7 Automated Planning algorithms

So far we have explained that solving a Classical Planning task involves modelling the environment (domain, problem) and choosing a description language. There is still a third essential element, which consists on choosing the planner to find a solution. A planning algorithm usually runs a heuristic search-based algorithm to explore the search space in a way that a solution can be provided. This can be done in three different ways according to the direction of search:

- Forward search, which starts from the initial state and searches towards a goal state. When a state s is expanded, its successors are generated. In forward search, as it starts exploring from the initial state, every precondition or effect is known. Thus, states are completely defined.
- Backward search, which starts from the goal state and advances towards the initial state. Backward search advances from a goal state towards the initial state. In backward search, goal states are defined as partial states. Thus, only part of those states (facts in G) is known. As a result, there are multiple states that can serve as goal states and search algorithms will only obtain new partial states after each expansion.
- Bidirectional search, which interleaves forward search and backward search. When both meet, which is usually called the frontier, a solution for the problem has been found.

The first heuristic search approaches explored both forward and backward search, such as HSP and HSPr [Bonet and Geffner, 2001]. However, dealing with backward search was harder than expected. There were two main problems: branching factor is usually higher due to partial states and duplicates and also it is most likely to explore states that are not reachable from the initial state. Thus, at the beginning, researchers mainly worked on forward search.

The work by Bonet and Geffner [2001] was also one of the first planning approaches that explored domain-independent heuristics. They proposed to *relax* the planning problem, which consists on ignoring concrete parts of it during planning search e.g. delete effects. A relaxed plan is solved when every goal proposition has been added by some action. Later, Hoffman presented Fast-Forward (FF) [Hoffmann and Nebel, 2001], which is an improvement over HSP. FF computes a relaxed plan for each search state ignoring deletes. The main difference regarding HSP is that, besides estimating goal distance, it also identifies the successors that seem to be most useful and detects goal ordering information.

Currently, one of the State-of-the-art planning systems is still Fast Downward [Helmert, 2006]. The algorithm is a heuristic search planner, which contains a great amount of parameters to configure a wide variety of versions in order to run different search algorithms and heuristics. Fast Downward was the base planner of seven planners of the 2011 International Planning Competition. LAMA-2011 [Richter and Westphal, 2010] was the Fast Downward configuration that won the competition. It is a Classical Planning system based on heuristic forward search. The search algorithm is guided by heuristics based on landmarks, propositional formulas that must be true in every solution of a planning task.

Another Fast-Downward-based planner is RPT [Alcázar et al., 2011]. As we have been inspired by RPT in our second contribution and also we have run it in the experiments, it is explained in detail in Section 2.9.

Besides, YASHP [Vidal, 2004] is a forward search planner that employs a lookahead policy to choose the next node to be expanded. The lookahead policy consists on performing an analysis of different relaxed plans. An improved version of the algorithm called YAHSP3 was the Agile track winner of the IPC 2011. The Agile Track consists on solving problems in 500s instead of 1800s. Thus, the main features of the planner are its greediness and speed to return a solution plan.

Finally, there is a small subset of planners that encodes the planning task as a satisfability problem, commonly known as SAT solvers in the literature. SAT [Cook, 1971] was used in classical propositional logic to test the satisfability of the formulas as NP-complete problems. Madagascar [Rintanen, 2014] is a SAT-based planning algorithm that aims to solve planning problems encoded in SAT. It is one of the few state-of-the-art

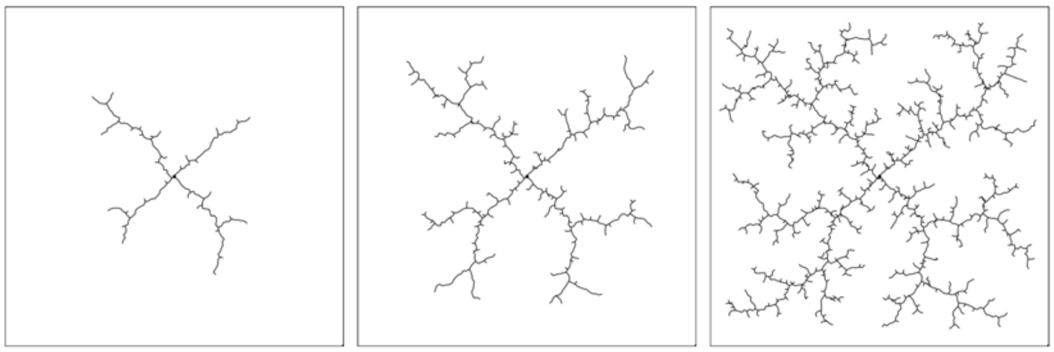


Figure 2.6: Progressive evolution of a RRT [LaValle and Kuffner, 2001].

planners that minimizes the makespan of the resulting plans.

We are using this set of diverse planners to compare against our contributions in Section 8.9.

2.8 Rapidly-exploring Random Trees

Rapidly-exploring Random Trees (RRTs) [LaValle and Kuffner, 2001] are a data structure combined with a search algorithm whose aim is to solve fast and efficiently continuous path planning problems.

Given two points, A (start) and B (end), the aim of any path planning problem is to find the solution path that connects both. RRTs drive the search towards unexplored regions by randomly changing the direction of the next node expansion. Thus, any implementation of RRT will grow in a similar manner to Figure 2.6.

As a result, RRTs were designed to allow fast searches in high-dimensional spaces. At the beginning, the algorithm creates the root node of the solution tree. On each iteration, the algorithm modifies the solution tree by adding a new node.

The algorithm is composed of two main steps:

- The local search phase. During this phase, the algorithm selects an unexplored random node of the search space. Then, the algorithm performs local search towards that direction from the closest node of the solution tree. There is a parameter ϵ which limits the number of expanded nodes per iteration.

- The extend phase. If the random node has not been reached after ϵ expanded nodes, the last expanded node is stored into the solution tree (Extend phase). Local search algorithms do not usually keep track of visited nodes. Thus, the solution tree is

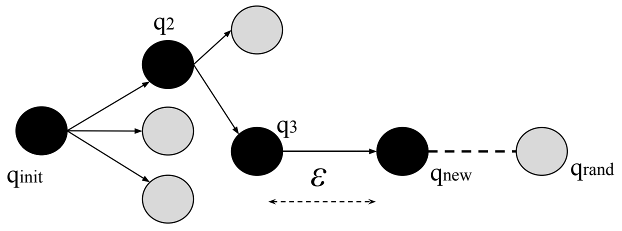


Figure 2.7: Extend phase of the RRT algorithm

useful to keep track of the explored path in an efficient way. Instead of storing the every visited node, only the state of the last expanded node on each iteration is stored into the tree.

2.9 Randomly-exploring Planning Tree

RPT [Alcázar et al., 2011] is a stochastic planner which combines forward search and sampling to solve a planning task represented in SAS+. RPT builds the solution inspired on the structure of RRTs. The algorithm decides on each iteration of the planning loop which kind of search runs. That decision depends on the values associated to a random number and the parameter p:

- p represents the probability of executing local search towards the nearest goal.
- *1-p* represents the probability of executing local search towards a sampled state.

Each iteration is controlled by a second parameter ϵ, which limits the number of expanded nodes of the local search. Once the number of expanded nodes reaches ϵ, a new random number is generated and the process will start again. Our second contribution, RRPT-PLAN is inspired on the structure of this planner.

As a result, RPT is able to combine two different types of search: one towards the nearest goal and another one towards a sampled state. There are two processes that are not trivial in RPT: sampling a state and find the closest node to it.

In order to sample a state, RPT considers that both, the initial state and the goal, must be reachable from the sampled state. Identifying unreachable states is strictly related to the mutual exclusivity computation between propositions [Haslum and Geffner, 2000].

In Automated Planning, a set of propositions $M = (f_1, f_2, ...f_k) \in F$ is mutually exclusive (mutex) if there is no reachable state $s \in S$ such that every proposition in M is true. These are known as static mutex. Once the unreachable sets of propositions are identified, unreachable states from the search space can be pruned to make the search process more efficient. Static mutexes are the most common type of mutex and are able to prune most of the unreachable states in many domains. However, computing all the static mutex from a given problem is PSPACE-complete. RPT performs an extensive computation of mutexes to identify as many as possible. Thus, the algorithm employs two state-of-the-art techniques: invariant synthesis [Helmert, 2009] and h^m [Haslum and Geffner, 2000]. The former is able to find set of propositions in which only one of them can be true at the same time. On the other hand, h^m gives an estimated distance (lower bound) from the initial state to a state in which any set of m propositions are true. If it is infinite, those propositions are mutex. Invariants are faster to compute than h^m, whose complexity grows exponentially when increasing m. RPT employs $m = 2$ for mutex computation.

In summary, RPT's sampling method works as follows:

1. a SAS+ invariant is chosen at random
2. every proposition mutex with that invariant is discarded
3. a proposition from that invariant is randomly chosen and added to the state
4. repeat until all SAS+ invariants have been chosen

The partial state is unreachable if no propositions are left when excluding propositions from some invariant that were mutex with the already chosen propositions. In that case, the process is restarted from scratch.

Then, RPT performs a reachability analysis from the initial state towards the sampled state and from the sampled state towards the goal state. If some proposition is still unreachable, the state is discarded.

Once a state is sampled, the next step is to find the closest node to it. In order to do that, planners usually compute a distance/cost estimation using heuristic functions. The values obtained are used to guide search algorithms towards the objective (usually the goal). However, the process is costly in terms of time and efficiency. We also mentioned that RPT needs to perform a reachability analysis to perform sampling. Thus, RPT saves time by caching on each node of the tree the *best supporters*, which are the actions that first achieve a given proposition in the reachability analysis.

As a result, a node of RPT's solution tree is defined as follows:

Definition 2.9.1. RPT *node*. A node q of the solution tree T can be defined as a tuple $\mathcal{N} = \langle s_i, \rho_i, \tau_i, cbs \rangle$, where:

- s_i represents the current state
- ρ_i contains a pointer to the previous node
- τ_i stores the sequence of actions that reaches s_i from the parent
- cbs stores the cached best supporters for every proposition $q \in F$.

Therefore, this information helps RPT to faster compute different heuristic functions such as h_{add}, which returns the combined cost of achieving every sampled proposition, and h_{FF}'s relaxed plans, by tracing back the relaxed plans from each node using the best supporters.

This page has been intentionally left blank.

CHAPTER 3

MULTI-AGENT PLANNING

MULTI-AGENT PLANNING lies in the middle of the research areas of Automated Planning and Multi-Agent Systems. Classical Planning approaches did not consider the concept of having multiple planning agents to solve a given planning task. Multi-agent planners offer more flexibility to solve any task that can be divided among a set of agents but also arise coordination and communication issues. In this Chapter, an overview of Multi-Agent Planning (MAP) is given by focusing on key concepts, representation languages, factorization, coordination, privacy issues and most popular approaches.

3.1 Introduction

Consider a warehouse where everyday hundreds of orders are received and have to be delivered into the shortest period of time. First, the warehouse workers need to get the corresponding products to complete each order. In order to speed up the process, some robots are available to fetch the products for them. Workers will only focus on receiving the products and checking that the order has been completed. After that, orders need to be delivered to the clients' addresses using trucks. When reasoning about this scenario, some coordination systems are needed: (1) a system should divide up the work into the workers; (2) also robots should move autonomously and individually through the warehouse avoiding collisions; and (3) each driver needs to be assigned a set of deliveries based on the deliveries' location or some other minimization cost feature to improve efficiency.

MAP is the subarea of Automated Planning that reasons on how to synthesize sequences of actions for solving tasks where multiple agents are involved (e.g. robots, workers, drivers) and/or some coordination is needed.

MAP aims at solving planning tasks for/by a set of agents. In collaborative environments, these agents collaborate to reach common goals. Two main approaches have been commonly used: centralized and distributed. The former builds a common plan for all agents by merely considering the agents as another planning resource. This was the usual way of dealing with MAP tasks in the Automated Planning community until recently. MAP complexity was studied by Brafman and Domshlak [2013]. The authors showed that MAP's complexity depends on the number of agents, the difficulty of their individual planning tasks and the number of interactions between agents (points where some coordination is needed). Centralized planning is usually more efficient when computing a plan with a reduced number of agents and goals. On the other hand, in distributed planning, agents generate their plans either synchronously with the rest of agents as in FMAP [Torreño et al., 2014] or MA-FS [Nissim and Brafman, 2014], or independently [Mudrova et al., 2016]. When planning synchronously, agents need to share information during planning. Thus, these approaches incur in a high communication cost.

Regarding MAP in real world scenarios, in most domains the solution should be executed in the shortest possible time. Hence, concurrent execution of agents' actions is needed. One way to deal with the task of finding the plan that minimizes the number of concurrent execution steps (*makespan*) consists of planning explicitly taking into account that minimization criteria. Another alternative consists of generating a sequential plan (totally-ordered plan) and transform it into a parallel one. The parallel plan is a restricted version of a partially-ordered plan, which means that the parallelization algorithm assures that actions assigned to the same plan step can be executed at the same time. By doing so, we can use any standard total-order planner in the state-of-the-art to generate the set of sequential MAP plans, and then apply some parallelization algorithm to improve the *makespan*.

This Chapter is structured as follows: Section 3.2 describes the MA-STRIPS formalization, the MA-PDDL language and a different approach to obtain the MAP task through factorization and goal-assignment; Section 3.4 describes how to implement privacy in different MAP approaches, Section 3.5 explains the different ways of handling agents' coordination in MAP; Section 3.6 contains the general remarks of the CoDMAP competition, Section 3.7 describes how to transform a partial-order plan into a parallel plan and finally 3.8 explains the State-of-the-art of multi-agent planners.

3.2 MAP task formalizations

The standard language to represent planning tasks is PDDL. However, Multi-Agent Planning does not have any official language yet to represent MAP tasks. In recent years, there has been several attempts to establish an official a multi-agent extension. Some examples are the Multi-Agent Planning Language (MAPL) in [Brenner, 2003] and the Multi-Agent PDDL (MA-PDDL) in [Kovacs, 2012]. However, neither of those was openly accepted by the MAP community. On the contrary, there was a formalization that has been widely accepted since it was published, which is MA-STRIPS [Brafman and Domshlak, 2008]. In the following sections, the MA-STRIPS formalization and the MA-PDDL language are explained in detail. Also, we describe the FMAP formalization, which defines fully-distributed environments and the MAPR formalization, which decomposes the MAP task into individual tasks regarding agents and goals.

3.2.1 MA-STRIPS

MA-STRIPS [Brafman and Domshlak, 2008] is an extension of the STRIPS formalism to work on cooperative multi-agent systems in which agents act under complete information and use deterministic actions. The definition of the MA-STRIPS task is as follows:

Definition 3.2.1. The MA-STRIPS task for a set of k agents, $\Phi = \{\phi_1, \ldots, \phi_k\}$, is given by the 4-tuple $\Pi = \langle F, \{A_i\}_{i=1}^{k}, I, G \rangle$ where:

- F is a finite set of facts, $I \in F$ encodes the initial state and $G \in F$ encodes the goal state.
- For $1 \leq 1 \leq k$, A_i is the set of actions that the agent ϕ_i is capable of performing.
- Each action $a \in A = \bigcup A_i$ has the standard STRIPS syntax and semantics. Thus, $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$.

As a result, MA-STRIPS is equivalent to the Single-agent planning task Definition 2.5.1 when k=1.

Agents in MA-STRIPS have disjoint sets of actions. This also implies that agents cannot share actions. Collaborative actions, which require at least two agents to be carried out, are widely known as joint actions. One of the advantages of MA-STRIPS is the generation of the agent interaction (di)graph. It depicts the level of interaction between agents so that the algorithm can easily deduce the facts from F that belong to either agents' internal information. Facts that do not belong to any agent are considered public information. Two agents are connected if one agent's action affects the functionality of the other agent. Formally, $F_i = \bigcup_{a \in A_i} \mathrm{pre}(a) \cup \mathrm{eff}(a)$ represents the set of facts affecting or affected by ϕ_i's actions. The set is divided into internal information (F_i^{int}) or public information (F_i^{pub}). Thus, $F_i^{int} \cup F_i^{pub} = F_i$ and $F_i^{int} \cap F_i^{pub} = \emptyset$. Authors declare $F_i^{int} = F_i \setminus \bigcup_{\phi_j \in \Phi \setminus \{\phi_i\}} F_j$, which means that a private fact $f \in F_i^{int}$ can only be achieved, erased or required by ϕ_i. As a result, $F_i^{pub} = F_i \setminus F_i^{int}$, which can be generalized to F^{pub} because the set of agents Φ have to share the same set of public facts by following previous definitions.

The easy process to infer the public and private information from a given MAP task has made MA-STRIPS a strong reference to follow when dealing with privacy schemas. More information related with privacy will be given in Section 3.4. Also, MA-STRIPS facilitates the different perspectives of a given MAP problem, which authors called projections. The current formalization is able to compute a projection of the original MAP task that only shows a concrete part of the problem e.g. the centralized view of the problem, the individual agent's view or the public part view. This aspect is interesting for real-world domains where different sets of agents have access to different levels of restricted information.

3.2.2 MA-PDDL

MA-PDDL is an extension of PDDL for multi-agent planning and includes the MA-STRIPS formalism. The first version of MA-PDDL was published in 2012 [Kovacs, 2012] as a pure extension of PDDL 3.1. When those authors decided to organize Competition of Distributed and Multi-Agent Planning (CoDMAP) at the end of 2013, the MAP community was focused on developing MA-STRIPS-based planners. Thus, they realized that MA-PDDL did not have yet a definition to describe the privacy of facts and actions. Therefore, Kovacs et al. prepared a new version of MA-PDDL[1], which extended the concept of privacy defined by MA-STRIPS and also would serve as the official language

[1]BNF definition of MA-PDDL http://agents.fel.cvut.cz/codmap/MA-PDDL-BNF-20150221.pdf

of the competition.

One of the advantages of MA-PDDL is the flexibility to model the MAP task as factored and unfactored. Factored MA-PDDL allows the definition of separate domain and problem descriptions for each planning agent. Each one represents the local view of each agent. Unfactored MA-PDDL allows the definition in a single domain and problem description, which includes task partitioning and privacy. Therefore, factored planning refers to “many for many” multi-agent planning (architecture is distributed, as well as the planning) and unfactored planning refers to “one to many” multi-agent planning (architecture is centralized; planning can be distributed or centralized). Kovacs et. al. utilized this criteria to organize CoDMAP planners into two independent tracks.

Most MAP planners are unfactored, which means that after receiving the whole MAP task they usually apply their own factorization techniques to divide the task among the agents. On the other hand, factored tasks are already divided among the agents and receive as input a set of domain and problem files, one for each agent. Factored planners need to establish some communication protocol for the agents to exchange essential information. Even though MA-PDDL supports MA-STRIPS, the former does not define private information nor the agents. Both elements are explicitly included into the domain and problem description files. Kovacs et. al. suggested new tags to complement the existing ones from PDDL:

- *:multi-agent*, it is included in the requirements of the domain to indicate that the description is a MAP task.
- *:unfactored-privacy :factored-privacy* they are included in the requirements of the domain to indicate if the file contains global or partial information of the MAP task.
- *:agent*, it is included on each action of the domain to indicate the type-object that performs the action.
- *:private*, it is included on the predicates of the domain and problem to explicitly indicate which are the private predicates.

Finally, in order to elaborate a privacy definition, MA-PDDL authors enunciate the following rules:

- Facts and actions can be private to particular agents or public among all agents
- A public predicate instantiated using public objects is a public fact.

- A public predicate instantiated with at least one private object to agent ϕ_i is considered private information of ϕ_i.
- A private predicate always grounds to a private fact, regardless of privacy of the objects used for instantiating.

3.2.3 The FMAP formalization

When a multi-agent planner follows a fully distributed configuration, agents have to communicate and plan by only using its own information. Therefore, each agent has a local view of the MAP problem. In FMAP [Torreño et al., 2014], authors proposed a MAP formalization where agents consider that the information that is not represented in each agent's model is unknown to the agent. Therefore, it allows to model fully distributed MAP tasks. The following paragraphs contain the description of this multi-valued formalization.

The FMAP task is defined by a 5-tuple $\Pi = \langle \Phi, \mathcal{V}, I, G, A \rangle$. where:

- Φ is a finite non-empty set of agents.
- $\mathcal{V}$ is the set of state variables known to an agent ϕ_i.
- $I = \bigcup_{\phi_i \in \Phi} I^{\phi_i}$ is the initial state.
- G is the set of goals.
- $A = \bigcup_{\phi_i \in \Phi} A^{\phi_i}$ is the set of planning actions of the agents.

The set of state variables is associated to objects in the world represented in the domain $\mathcal{D}_v$. Thus, each variable $v \in \mathcal{V}$ has assigned a value $d \in \mathcal{D}_v$, which is represented in a tuple called *fluent* $\langle v, d \rangle$. Fluents can be positive (v takes the value d) or negative (v does not take the value d). Therefore, a state S is represented by a set of instantiated positive or negative fluents.

This formalization employs the term *specialized agents* to define agents in Φ that have different knowledge and capabilities. Those agents might only now a subset of the initial state I. Regardless of specialization, all agents in Φ are cooperative, so they all are aware of the whole set of goals G to achieve. However, depending on the specialization, some agents might not understand some of the goals.

The view of an agent is defined by a 4-tuple $\Pi_i = \langle \mathcal{V}_\phi, A_{\phi_i}, I_{\phi_i}, G \rangle$. where:

- V_{ϕ_i} is the set of state variables known to agent ϕ_i;
- $A_{\phi_i} \in A$ is the set of agent's planning actions;
- I_{ϕ_i} is the subset of fluents of the initial state I that are visible to agent ϕ_i;
- and G is the set of global goals of the MAP task.

Having specialized agents also leads to a definition of privacy. This is later explained in Section 3.4.

3.2.4 The MAPR formalization

In order to work on MAP environments using unfactored planners, Borrajo and Fernández [2018] proposed the MAPR formalization, which is more general than MA-STRIPS and works with PDDL and MA-PDDL languages. This formalization is the one we are employing in our first contribution PMR. Thus, in order to prevent confusions, we will refer to the MAPR task as the MAP task.

Definition 3.2.2. The MAP Task. The MAPR task is a multi-agent setting where a set of m agents, $\Phi = \{\phi_1, \ldots, \phi_m\}$, has to solve the task Π. The MAPR task M is defined as a set of planning subtasks, Π_i, one for each agent. Thus, $M = \{\Pi_1, \ldots, \Pi_m\}$ being $i = \{1...m\}$. For representation convenience, an alternative equivalent lifted representation of each single-agent planning task in PDDL would be a pair (domain, problem): $\Pi_i = \langle D_i, P_i \rangle$.

As the planning task is an unfactored task, MAPR employs two methods -agentification and factorization- to generate each agent's individual task. Both methods are later explained in Section 3.3.

Once the MAP task has been defined, agents' and public information described and the final set of planning agents chosen (Φ'), the definition of the MAPR individual planning task is as follows:

Definition 3.2.3. Individual agent task (Π_i). For each agent $\phi_i \in \Phi'$ a specific task Π_i is generated, which is described as a tuple $\Pi_i = \langle F_i, A_i, I_i, G_i \rangle$ where:

- $F_i = (F_{\phi_i} \cup F_{pub}) \subseteq F \mid F_{\phi_i}$ and F_{pub} are disjoint sets, $F_{\phi_i} \cap F_{pub} = \emptyset$.
- $A_i = (A_{\phi_i} \cup A_{pub}) \subseteq A \mid A_{\phi_i}$ and A_{pub} are disjoint sets, $A_{\phi_i} \cap A_{pub} = \emptyset$.
- $I_i = (I_{\phi_i} \cup I_{pub}) \subseteq I$ is the initial state of agent ϕ_i, where $I_{\phi_i} \subseteq F_{\phi_i}$ and $I_{pub} \subseteq F_{pub}$. I_{ϕ_i} and I_{pub} are disjoint sets.

- $G_i \subseteq G$ is the set of ϕ_i's assigned goals.

In Definition 3.2.3, the set of agent's propositions is identified as F_{ϕ_i} and the set of agent's actions as A_{ϕ_i}.

F_{ϕ_i} includes ϕ_i's propositions that have been instantiated through the generic agent's predicates, i.e. the predicates parameterized as private and the predicates that include as argument either the agent or one of the object types parameterized as private that belongs to the agent. For example, agent's location, agent's features, agent's instrument, or agent's instrument's features.

A_{ϕ_i} represents ϕ_i 's instantiated actions. They were generated from the generic actions that included the agent itself and/or the agent's objects as argument. For instance, in the Rovers domain, they could include actions as "take a picture" that requires the rover, or "camera calibration" that only mentions the camera of a given rover.

The set of public propositions is identified as F_{pub} and the set of public actions as A_{pub}. We assume that both the complete initial state, $I = \cup_{i=1}^{m} I_i$, and the set of goals, $G = \cup_{i=1}^{m} G_i$ are consistent; that is, they are conflict-free (there are no mutexes). Information related with preservation of privacy will be given in Section 3.4. Finally, we enunciate two different representations of the MAP task, as the set of single-agent tuples and as a lifted representation, which contains a pair domain-problem for each agent.

Definition 3.2.4. Factorized MAP task (M'). Being Π the tuple described in Definition 4 and $n = |\Phi'|$, the M' task is defined as:

$$M' = \langle \Pi_1,..,\Pi_n \rangle = \left\{ \langle F_1, A_1, I_1, G_1 \rangle, ..., \langle F_n, A_n, I_n, G_n \rangle \right\}$$

Definition 3.2.5. Lifted representation of the M' task. Considering the lifted representation of $\Pi_i^{PDDL} = \langle D_i, P_i \rangle$ and being $n = |\Phi'|$, the M' task is defined as:

$$M' = \langle \Pi_1^{PDDL}, ..., \Pi_n^{PDDL} \rangle = \left\{ \langle D_1, P_1 \rangle, \langle D_2, P_2 \rangle, ..., \langle D_n, P_n \rangle \right\}$$

3.3 Factorization and Agentification

Agentification consists on identifying which are the agents of the given problem. This identification can be done in many different ways. Inside MA-STRIPS there is no information that explicitly indicates which type of agent can execute which action. Thus, if the planning task is defined using the MA-STRIPS formalization, agents are given as input to the planner along with the lifted representation of the MAP task (domain and problem). Alternatively, if the planning task is defined using MA-PDDL, the agents are explicitly included in the domain and problem files. Each action has a keyword *:agent* in order to know which type of agent can execute the action. It is important to choose as agents the elements of the domain that suit best the division of tasks. An advantage of dealing with MAP domains is that, in most of them, agentification comes naturally and the agents-to-be are immediately identified.

Factorization is the ability to divide a planning problem into subproblems using some criteria. For instance, it is common to apply factorization dividing the problem regarding goals [Borrajo, 2013; Crosby et al., 2013]. The aim is to transform the unfactored planning task into easier tasks so that each agent can solve its factorized problem individually. In the following sections, the agentification and factorization processes applied by SIW, ADP and MAPR are explained in detail.

3.3.1 SIW

Serialized Iterative Width (SIW) [Lipovetzky and Geffner, 2014] is a centralized algorithm that transforms a planning problem into subproblems. Thus, this approach fits with the purpose of unfactored MAP tasks. Authors of SIW employ the word serialization when referring to factorization. The reason behind this term is that they are not directly dividing the task among the agents. Actually, the algorithm is choosing only one atomic goal to solve at a time, which is the purpose of serialization.

SIW calls the algorithm Iterative Width (IW) [Lipovetzky and Geffner, 2014] to solve the subproblems. The aim is to achieve one atomic goal at a time, greedily, until all atomic goals are achieved jointly. Another remark is that SIW performs a blind-search, there is not a heuristic guidance during search. However, SIW does not loose performance in comparison to other greedy best-first algorithms guided by delete-relaxation heuristics, as the serialized tasks are very easy to solve.

Thus, the steps of the SIW algorithm can be summarized as follows: Given a problem $P = \langle F, I, O, G_i \rangle$, the algorithm performs a sequence of calls to IW over the serialized problems $P_k = \langle F, A, I_k, G_k \rangle, k = 1, ..., |G|$, where:

1. $I_1 = I$, $G_0 = \emptyset$.
2. G_k is the first consistent set of atoms achieved from I_k such that $G_{k-1} \in G_k \subseteq G$ and $|G_k| = k$. This means that G_{k-1} must hold in G_k.

3. I_{k+1} represents the state where G_k is achieved, $1 < k < |G|$.

SIW is written in C++ and employs Metric-FF to transform propositional facts into state variables.

Finally, we give a glimpse of the main remarks of the algorithm IW:

- First, the algorithm is composed of a sequence of calls IW(i) for $i = 0, 1, ..., |F|$, where F is the set of atoms of a given planning problem.
- Each iteration runs a breadth-first search that prunes newly generated states when novelty $> i$.
- The novelty is computed as the size of the smallest tuple of atoms t that is true in a state s and false in all previously generated states s'. The higher the novelty measure, the less novel the state.

3.3.2 ADP

The Agent Decomposition-based Planner (ADP) [Crosby, 2015] is a fully centralized planning algorithm that consists of an automated decomposition process and a heuristic search method designed specifically for decomposed domains. Factorization in ADP is first performed from the point of view of the domain and then from the problem. ADP is built on top of Fast Downward and instead of using the formalization of a classical planning task, it employs a simplification of the multi-valued planning task proposed by Edelkamp and Helmert [1991]. This representation allows to transform the information of a planning task into a set of fluents that represent different elements of the domain e.g. location, availability, data received etc.

On the first step, ADP performs a static analysis of the causal graph generated by Fast Downward. The causal graph is composed by nodes, which contain each of the values of the set of fluents, and edges, that show the dependencies between them. After obtaining the graph, a decomposition of the domain can be easily carried out by following three indicators: (1) dependencies are reduced, (2) goals can be achieved independently and (3) coordination between agents is minimized.

The aim is to find the most suitable agentification. If an agentification cannot be found because of the features of the domain, the search algorithm of ADP resorts to a standard single-agent algorithm. If some agentification was found, ADP employs the decomposition of the problem to find the solution plan. First, it checks that potential agents can work individually. In case this condition is not satisfied, tasks from these agents will be merged and both agents will count as one.

On the second phase, ADP starts a loop that runs on each iteration a goal assignment process plus a forward search guided by h_{FF}. The search is only focused on solving the goals of one agent per iteration. The loop ends when the solution plan is found or a dead-end is reached.

3.3.3 MAPR

MAPR [Borrajo and Fernández, 2018] supports both language definitions, either by explicitly receiving the list of potential agents or by getting the information through the *:agent* keyword of MA-PDDL to create the agents' list afterwards. In MAPR, the problem is first factorized from the agents' point of view. Definition 3.2.2 describes the factorization of the Planning Task in terms of agents, which gives as a result the Multi-Agent Planning Task. Regarding goals, current MAP techniques in the Automated Planning community employ the MAP task M where goals are considered achievable by the collaboration of all agents [Nissim and Brafman, 2014; Torreño et al., 2014]. So goals are pursued by all agents and no further factorization is applied. In other areas, as in Multi-Agent Systems or robotics, some approaches first perform task allocation (assignment of each public goal to a single agent) to improve the efficiency of problem solving [Conitzer, 2010; Gerkey and Matarić, 2004]. Inspired by that, MAPR has a second factorization step, which consists on dividing the goals among the agents by following some strategy. Some of the goal assignment (GA) strategies are:

- *All*, that assigns all goals to all agents;
- *Best-cost* (BC), where each goal $g \in G$ is assigned to the agent $\phi_i \in \Phi$ that could achieve it with the least cost; and
- *Load-balance* (LB) that first calculates $k = \left\lceil \frac{|G|}{|\Phi|} \right\rceil$, which will be the average number of goals per agent. Then it assigns each goal $g \in G$ to the best agent $\phi_i \in \Phi$ as in BC. This strategy avoids, if possible, assigning more goals than k to each agent.
- *Contract-net* (CN), which assigns on each iteration each goal to the best agent by taking into account all previous assignments of goals. On the first iteration it computes an estimation of the cost for agent ϕ_i to achieve the first goal, g_1, $c(g_1, \phi_i)$. Then, CN assigns the goal g_1 to the agent that returned the least cost. As long as an agent does not have previous goals assigned, the goal strategy will

compute the cost of achieving a new goal as $c(g,\phi_i)$. However, if the agent has already at least one goal assigned, the estimation cost of a new goal g is performed as $accum(g,\phi_i) = c\Big([g_1^{\phi_i},..,g_m^{\phi_i},g],\phi_i)\Big)$. This strategy is inspired in the homonym negotiation scheme commonly employed in Multi-Agent systems [Smith, 1980]. The aim is to choose the lowest $accum(g,\phi_i)$ on each goal assignment iteration.

As in [Borrajo, 2013], in order to assign a goal $g \in G$ to an agent $\phi_i \in \Phi$, a relaxed plan is computed using the FF heuristic [Hoffmann and Nebel, 2001]. Depending on the strategy selected, GA might leave some agents without any assigned goal. Hence, the output of goal allocation is a new MAP task, M', with goals assigned to a set of n agents $\Phi' \subseteq \Phi$. If the agent's cost of a goal is infinite (it cannot be reached), the goal is not assigned to that agent $\phi_i \in \Phi$ but if there is some goal that cannot be reached individually by any agent, the previous process assigns that goal to all agents. Then, $\Phi' = \Phi$ since all agents will have at least one goal assigned.

After the second factorization step has been applied, each Π_i of the MAP task M' is formally described by Definition 3.2.3. Also, M' can be defined in terms of Π_i or using the lifted representation.

Now, we describe through a simple example how agentification and factorization work before generating and delivering to any planner the input M'.

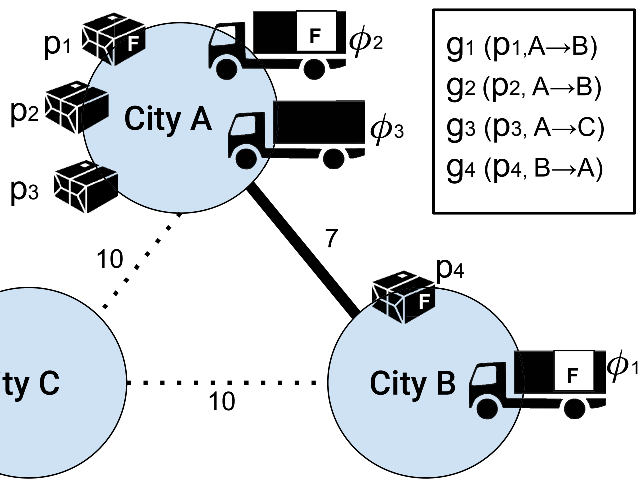


Figure 3.1: Example of a simple Logistics problem where trucks (ϕ_1, ϕ_2, ϕ_3) have to deliver some packages to the destinations (g_1,g_2,g_3,g_4) specified on the square box. Trucks marked with “F” (ϕ_1, ϕ_2) are the only ones allowed to transport fragile packages (p_1,p_4). Trucks that start in CityA can traverse any kind of road (lined, dotted). Trucks from other cities can only traverse lined roads.

Example (Logistics domain). Given a Logistics domain where trucks need to deliver packages to a set of locations, our example contains three trucks and four goals (package-delivery to its destination). Trucks initially located at CityA can traverse any kind of road (lined, dotted). Trucks from other cities can only traverse lined roads. Also, only trucks marked with "F" can deliver fragile "F" packages. The initial state of the problem is shown in Figure 3.1. The destination of each package is described inside the square box of the figure. Costs of driving between each pair of cities are shown over the lined/dotted city connections. There is also a cost of 1 for loading and unloading a package. Agentification comes naturally on this example, as trucks are the ones performing the actions. As a result, the first factorization generates the MAP task M, which contains an individual planning task for each truck.

Table 3.1: Example of an estimated-cost matrix from the problem pictured in Figure 3.1. Each number represents the cost returned by h_{FF} for an agent ϕ_i when reaching goal g_m. c(g_m,ϕ_i)=∞ means goal g_m is not reachable by agent ϕ_i.

Agents \ Goals	g_1	g_2	g_3	g_4
ϕ_1	9	9	∞	2
ϕ_2	2	2	2	9
ϕ_3	∞	2	2	∞

Table 3.2: Example of the estimated-cost computation for Contract-net from the problem pictured in Figure 3.1. Columns represent the estimated costs computed on each iteration. Bold numbers represent that the goal has been assigned to the agent placed in the row.

Agents	iter. #1	iter. #2	iter. #3	iter. #4
ϕ_1	$c(g_1,\phi_1) = 9$	$c(g_2,\phi_1) = 9$	$c(g_3,\phi_1) = \infty$	$c(g_4,\phi_1) = \mathbf{2}$
ϕ_2	$c(g_1,\phi_2) = \mathbf{2}$	$c([g_1,g_2],\phi_2) = 3$	$c([g_1,g_3],\phi_2) = \mathbf{3}$	$c([g_1,g_3,g_4],\phi_3) = 9$
ϕ_3	$c(g_1,\phi_3) = \infty$	$c(g_2,\phi_3) = \mathbf{2}$	$c([g_2,g_3],\phi_3) = 3$	$c([g_2,g_4],\phi_3) = \infty$

The second factorization of the MAP task M varies depending on which GA is chosen. Table 3.1 shows the estimated costs computed for GA strategies *All*, BC and LB. Table 3.2 shows the estimated costs computed for CN, as the process is slightly different than in the other strategies. Both compute the cost per truck and goal when delivering each package to its destination. The GA strategy *All* assigns every goal to every agent even if the goal cannot be reached (represented with ∞ cost). There are three cases where ∞ cost is returned by h_{FF}: c(g_3,ϕ_1) because the truck ϕ_1 cannot traverse dotted roads as it does not start at CityA; c(g_1,ϕ_3) and c(g_4,ϕ_3) because the truck ϕ_3 cannot transport fragile packages (marked with "F"). The BC strategy does not assign any goal to ϕ_3 even though it can reach two of the four goals. Instead, they are assigned to ϕ_2, who is the first on the list of agents that can reach (g_2,g_3) with an estimated cost of 2. LB returns an average of *k=2*. Thus, it assigns two goals per agent. After assigning g_1 and g_2 to ϕ_2, ϕ_2's goal capacity has been reached. Then, g_3 is assigned to ϕ_3 even though ϕ_2 could achieve it with less cost. Finally, CN starts assigning g_1 to ϕ_2, then g_2 to ϕ_3. In order to assign g_3, it analyzes first the accumulated cost from ϕ_2 and ϕ_3. As the accumulated cost for both agents, so far, it is the same (3), it will assign the goal to ϕ_2 and will increment its accumulated cost. Last, g_4 is assigned to ϕ_1.

To sum up, taking the information presented on Tables 3.1, 3.2 and the four GA strategies mentioned before, goals would be assigned to each agent as follows:

- *All*: $\phi_1(g_1,g_2,g_3,g_4)$, $\phi_2(g_1,g_2,g_3,g_4)$, $\phi_3(g_1,g_2,g_3,g_4)$
- *Best-cost*: $\phi_1(g_4)$, $\phi_2(g_1,g_2,g_3)$, $\phi_3(\emptyset)$
- *Load-balance*: $\phi_1(g_4)$, $\phi_2(g_1,g_2)$, $\phi_3(g_3)$
- *Contract-net*: $\phi_1(g_4)$, $\phi_2(g_1,g_3)$, $\phi_3(g_2)$

After this process is completed, the M' task is generated and any MAP planner would receive the following as input: (1) if LB, CN or All were chosen, the MAP task $M' = \{\Pi_1, \Pi_2, \Pi_3\}$ (2) if BC was chosen, $M' = \{\Pi_1, \Pi_2\}$, as ϕ_3 has no assigned goals. We are using this process and the M' task as input to our first contribution PMR.

3.4 Modelling Privacy in Multi-Agent Planning

Privacy preservation in Multi-Agent environments is a common feature that some MAP approaches are studying and implementing in their algorithms. Agents can have sensitive information that should be hidden from public knowledge. Thus, agents can share the public information but should keep the private information to themselves. MAP privacy models should offer a solid communication approach that deals with the privacy requirements of the agents. Also, privacy models are useful to avoid bottlenecks on excess of information sent through the network or on processing huge amount of unmeaningful data.

At this point, there are not any standards supported by the entire community on how to model privacy for MAP. MA-STRIPS < was one of the first and most supported approaches that mentions some directions in order to model privacy in MAP. Concretely they separate internal and public atoms (facts) and actions. The directions can be summarized in the following two rules:

- An atom is public if it is either achieved, required or modified by at least two actions of different agents.
- An action is public if it modifies or requires at least one public atom.

Even though MA-STRIPS offers these directions, it does not implement any privacy model. Thus, we have selected three different models of privacy that represent the wide variety of existing approaches. Section 3.4.1 explains the privacy model of the MA-PDDL language. Section 3.4.2 explains the privacy model of FMAP, as an example of dealing with privacy in distributed MAP architectures. Finally, Section 3.4.3 explains the privacy model of MAPR, which is one of the works that inspired our first contribution and serves as an example of privacy in distributed planning but centralized architectures.

3.4.1 MA-PDDL

As it was previously said, authors of MA-PDDL [Kovacs, 2012] were the same ones that organized CoDMAP. Thus, they found that the definition of privacy of MA-STRIPS was too rigid regarding the current and future versions of the competition.

The MA-PDDL approach consists mainly on declaring explicitly the private components in the domain and problem descriptions and relaxing the definition of privacy from MA-STRIPS. Thus, privacy related to MA-PDDL objects can be found in the problem description and privacy related to MA-PDDL predicates, functions and constants, in the domain. Authors suggest to start with the entire domain and problem description being private and then to identify the public parts; rather than doing the opposite, as MA-STRIPS would suggest. Also, authors established some rules in order to identify the public and private components of the domain and problem:

- A declared public predicate in the domain that only contains public objects results in a public fact.
- A declared public predicate in the domain which contains at least one private object, results in an agent's private fact.
- An object cannot be private to multiple agents.

- A declared private predicate in the domain results in a private predicate independently of the nature of its objects.

3.4.2 FMAP

Torreño et al. [2014] proposed a specific privacy model that allows agents to maintain hidden their private information while only communicating the relevant part of the refinement plans that need to be strictly shared among the planning agents. Thus, each agent declares which information is going to be shared.

As a result, FMAP minimizes the amount of information shared among agents. This issue will be directly reflected on the efficiency of communication.

As mentioned before in Section 3.2, FMAP employs a formalization based on fluents. Thus, authors define the privacy model as follows:

- If an agent refines the partial plan by introducing a new action, it communicates the resulting plan to the rest of the planning agents.
- An agent will only communicate the fluents in the new action if they both share those variables.
- Each planning agent has a partial view of the problem, which is updated when a refinement plan is received.
- There exist three types of fluents: public, private and partially private to a planning agent. A fluent is composed of predicates, causal links and effects. When two agents share a public fluent they exchange every component of it. If the fluent is private, only causal links and ordering constraints are shared between both agents. Finally, if a fluent is partially private, all components are shared but the private information is obfuscated, which the recipient agent reads as resource not available to use or not accessible.

3.4.3 MAPR

MAPR [Borrajo and Fernández, 2018] follows a distributed MAP approach in a single-core architecture where the set of agents solves the planning task iteratively. Agents do not communicate with each other during the individual planning phase.

MAPR cannot infer the private components from the planning task as in FMAP. Thus, the privacy approach of MAPR starts with the user declaring as input which are the agents and their privacy requirements. Then, MAPR applies factorization and agentification as explained in Section 3.3 to generate an individual domain and problem for each agent. Private components of these two files are obfuscated. Also, agents can have their own private goals.

Agents solve the task iteratively. As a result, the first agent plans for its own set of public and private goals using the available information from its specific domain and problem. Once the current agents finishes planning, it obfuscates part of the sensitive (private) information. Then, plans, goals and states are sent to the next agent, which will only perceive as available the information that is not obfuscated. Therefore, agents share both, their public and private knowledge, but the latter is obfuscated and will only be understandable by its owner.

As privacy is defined at the lifted representation level (usually PDDL), the property is attached to predicate and types and inherited by their instantiations i.e. atoms, objects.

In order to generate the individual domain and problem for each agent, the compiler takes as input the agents' types, the private predicates and the private types. Then, it assigns values to instantiate those predicates and types. Now the compiler can separate on each of the files the concrete information associated to each agent. Public knowledge is included on every file. Actions that have associated an agent type are also instantiated by removing the type from the parameters and adding it to the general name of the action. Thus, the domain file of an agent contains only the actions performed by that agent (private actions) or the actions that do not have any associated agent (public actions). The problem file of an agent contains the public predicates and goals and its own private predicates and goals.

MAPR assumes that the privacy definition of types is consistent. Thus, if two agents can modify the value of the same grounded literal, the corresponding predicate cannot be private. However, this is easy to infer in the majority of domains modelled for MAP.

3.5 Coordination techniques

When solving Multi-Agent Planning Tasks where multiple agents are involved, actions need some coordination in order to execute the plan successfully.

Each agent's individual plan should be: free of negative interactions among the different agents' plans (one agent deletes some literal that another agent plan needs); free of redundant actions, that achieve goals unnecessarily (two agents achieve the same goal twice); and valid (the plan achieves the goals from the initial state). Interactions among agents' plans are closely related to the (positive or negative) interaction level reflected on predicates (preconditions and effects) and actions of the planning domain. As a result, planning domains can vary from loosely-coupled, where there is almost no interference among the agents' plans to tightly-coupled, with higher interactions [Brafman and Domshlak, 2013].

Torreño et al. [2017] affirms that the capability and efficiency of a MAP solver is determined by the coordination strategy that governs its behaviour. Thus, there are different classifications that could be done depending on how each solver deals with coordination. In that work, authors suggest to explain coordination in terms of unthreaded coordination or interleaved planning and coordination.

Unthreaded coordination refers to agents performing individual planning after receiving an individual planning task. Coordination is performed before or after the individual planning process. Thus, communication between agents is not needed.

- When coordination is performed before the planning process starts, the MAP algorithm should guarantee that the division in subtasks yields into a consistent solution plan when combining the solution plans obtained from each agent. As the main task is decomposed into independent subtasks, agents plan individually and there should not be coordination problems. The MAP algorithm ADP [Crosby, 2015] follows this approach.

- When coordination is performed after the planning process starts, agents plan independently and do not employ any communication among agents while planning. Therefore, they have to later merge their plans. In that case, some merging function is applied to the set of plans to generate a joint plan [Foulser et al., 1992; Mali, 2000]. Plan merging has been shown to work best in loosely-coupled domains. In relation to plan merging, Mali devised algorithms for performing plan merging by removing or rearranging actions [Mali, 2000] Britanik and Marefat proposed to perform plan merging within HTN planning [Britanik and Marefat, 1995]. Merging appears at different levels of abstraction by decomposing a plan into subplans. In the field of temporal planning, Mudrova et al. [2016] propose an algorithm that merges partial order plans with durative actions for solving robotic tasks. A different approach close to plan merging is conflict solving. Jordán et al. [2018] propose one such approach, where conflicts are identified while agents generate the solution. The algorithm works at the same time on a joint plan and penalizes the agents that generate a conflict.

On the other hand, interleaved planning and coordination refers to planning synchronously. Agents need to share information during planning. Thus, these approaches incur in a high communication cost. Also, they require to modify the code of an existing planner in order to accommodate the communication among agents. FMAP [Torreño et al., 2014] and MAFS [Nissim and Brafman, 2014] are MAP algorithms that follow this approach.

There is a middle-way approach between unthreated and interleaved coordination which can be called the iterative planning and coordination process. For instance, in MAPR [Borrajo and Fernández, 2018], goals are distributed among the agents and each agent sends its solution plan and planning task to the next one in the sequence. Thus, agents do not communicate with each other but as the final solution is built on top of each individual agent's plan, coordination issues are fixed.

3.6 CoDMAP: The first competition on Distributed and Multi-Agent Planners

The field of MAP is very recent in Automated Planning. The most remarkable advances in the field have been performed during the last ten years. There was not an easy way to benchmark and evaluate different MAP approaches. There was not even a standard description language. Therefore, MAP Researchers found the same issues as those of the beginnings of Classical Planning. Thus, Stolba and Komenda organized in 2016 the first simulation of a MAP competition. It was called CoDMAP, the competition on Distributed and Multi-Agent Planners [Štolba et al., 2016a]. The MAP domains and problems were written in MA-PDDL. Most of the planners that participated in CoDMAP ended up using parsers that transformed MA-PDDL files into PDDL.

The competition was divided into two tracks, the centralized track (unfactored), where the planners were allowed to run on a single computer for all agents and the distributed track (factored), where each agent had to run on a separate machine and communicate over TCP/IP.

The centralized track allowed researchers to explore the MAP field without dedicating much effort. The planning task was received as in any classical planner (domain, problem). Authors should include a factorization step on their classical planners to divide the planning task into multiple agents' tasks. On the other hand, planners from the distributed track received the task already divided. Each machine, which represented an agent, received its part of the whole task.

The domains were in most cases multi-agent variants of the classical IPC domains: Blocksworld, Depot, Driverlog, Elevators08, Logistics00, Rovers, Satellites, Sokoban, Woodworking, and Zenotravel. There were two new domains called Taxi and Wireless. Each domain had 20 problem instances, with increasing complexity. The biggest problems had 10 agents and about 100 objects/constants. Even though some planners could deduce the agents of each MAP task, concrete agent types were specified by the organizers. It was mandatory to follow their instructions. MAP problems with private goals or joint actions were not considered for the competition.

The planning systems were evaluated following the metrics from IPC (coverage, plan cost, time). They did not include makespan.

The centralized track was composed of 12 planners in 17 configurations. For the distributed track, organizers received 3 planners in 6 configurations. Results are available in the official website [2]

3.7 Paralellization algorithms

In Automated Planning, plan ordering refers to how the actions are ordered inside a given plan. Plan actions are always assigned to a concrete step of the plan. Thus, there exist two popular orderings in the planning community: total-order planning and partial-ordered planning.

Total-order planning refers to any solution plan that maintains a totally ordered list of actions (every action is ordered with respect to every other action). In total-order planning only one action is assigned to each plan step.

On the other hand, partial-order planning relaxes some of the total-ordering constraints. Now the set of actions can be partially unordered. A partial plan, as generated by most partial-order planners, POP (e.g. UCPOP [Penberthy and Weld, 1992]), is a tuple $\pi_p = \langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$,[3], where:

- $\mathcal{A}$ is the set of instantiated actions in the plan
- $\mathcal{O}$ is the set of ordering constraints (each element of the set is a pair of actions (a_i, a_j), $a_i, a_j \in \mathcal{A}$, such that a_i should be executed before a_j),

[2]http://agents.fel.cvut.cz/codmap/

[3]We left out a fourth component, the set of bindings, since it is not needed for our purposes.

- and $\mathcal{L}$ is a set of causal links (each element is a tuple $\langle a_p, l, a_c \rangle$, such that the action a_p adds/deletes proposition l that is a precondition of a_c)

The plan steps of a partial-ordered plan are still ordered and so on, but each step can have assigned more than one action. We say that the set of actions assigned to the same plan step are unordered. A set of unordered actions indicates that those concrete actions can be executed sequentially regardless of the order in which they are executed. However, this does not imply that said set of actions are executed in parallel.

In Multi-Agent Planning, parallel plans help to improve the makespan of a given solution. If a plan is parallelized regarding the number of agents that can perform those actions, plan execution will be more efficient. Thus, some works on Automated Planning have explored different procedures in order to transform a partially ordered plan into a fully parallelized plan. For instance, Veloso et al. [1990] presented an algorithm that achieves that transformation. Also, Borrajo and Fernández [2015] implemented a new parallelization algorithm based in [Veloso et al., 1990]. For the sake of clarity, we are explaining the later, as it will be used inside our contribution.

From a total-order plan, the algorithm transforms the plan first into a partial-order plan. Thus, $\mathcal{A}$ is generated as the set of all actions in the totally-ordered plan, π, given as input, plus two fictitious actions: a_0 that is a virtual action that adds the initial state; and a_∞, that has as preconditions the goals of the planning task. $\mathcal{O}$ and $\mathcal{L}$ are dynamically generated by the algorithm. The algorithm also keeps track of open goals (as in the agenda of POP planners). Open goals are either the top level goals, at start, or preconditions of actions added to the partial plan such that there is not yet an action that achieves them (creating a causal link).

Once the partial-order plan is built, the algorithm iterates over all actions in the plan in reverse order, performing three operations: creating one causal link between the corresponding action a_i and all actions later in the plan (that were analyzed before by the algorithm) that included a precondition that belongs to the effects of a_i; that is, a_i is selected to add/delete that precondition. Next, it adds as open goals its preconditions. Finally, it checks and solves threats to existing causal links; an action threats a causal link if it deletes/adds the proposition of the causal link and can potentially be executed between a_p and a_c of the causal link. This process assures that each unordered set of actions is studied to check whether they all can be executed in parallel or not.

Then, the parallel plan is generated from the partial plan, $\pi_p = \langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$. Two variables are used: $\mathcal{O}$ (from the partial plan) and $\mathcal{T}$. $\mathcal{T} = \{(a_i, t_i)\}$ is the list of pairs representing the time instant t_i when a_i can be executed. $\mathcal{T}$ initially contains a single pair $(a_0, 0)$.

Then, for every action in the sequential plan $a_k \in \pi$ (in the order they appear in the sequential plan), we define $\mathcal{B}_k$ as the subset of actions that must be executed before a_k; this is computed from $\mathcal{O}$:

$$\mathcal{B}_k = \{a_i \mid (a_i, a_k) \in \mathcal{O}\}$$

For all these actions, their execution time is obtained from T. Then, a new pair (a_k, t_k) is added to T, where t_k will be the maximum of the execution times of the operators in B_k plus one:

$$t_k = 1 + \max_{t_i \mid (a_i, t_i) \in \mathcal{T}, a_i \in \mathcal{B}_k} t_i$$

When all actions in the plan have been visited, $\mathcal{T}$ contains the time instants at which every action in the plan can be executed. The parallel plan is obtained ordering $\mathcal{T}$ by the ascending values of t_k.

Regarding the Logistics example from 3.1, to load or unload different packages do not present any ordering constraints, so both could be executed in parallel. However, to load/unload and drive a truck have an ordering constraint because the location changes. As agents come naturally on this example (trucks), the resulting parallel plan could be as the right-side plan from Figure 3.2.

1: Load–package p_1, ϕ_2	1: Load–package p_1, ϕ_2
2: Load–package p_2, ϕ_2	1: Load–package p_4, ϕ_1
3: Load–package p_3, ϕ_3	1: Load–package p_3, ϕ_3
4: Load–package p_4, ϕ_1	2: Drive–truck ϕ_1, A
5: Drive–truck ϕ_3, C	2: Load–package p_2, ϕ_2
6: Drive–truck ϕ_2, B	2: Drive–truck ϕ_3, C
7: Drive–truck ϕ_1, A	3: Drive–truck ϕ_2, B
8: Unload–package p_1, ϕ_2	3: Unload–package p_4, ϕ_1
9: Unload–package p_2, ϕ_2	3: Unload–package p_3, ϕ_3
10: Unload–package p_3, ϕ_3	4: Unload–package p_2, ϕ_2
11: Unload–package p_4, ϕ_1	5: Unload–package p_1, ϕ_2

Figure 3.2: Example of a total-order plan (left) and a parallel plan (right) obtained from a Logistics problem. The parallel plan takes into account that there are only three trucks available to perform each action.

The Logistics problem is implicitly assuming that there are enough humans to perform the acts of driving/loading/unloading packages. However, if the problem were considering explicitly this issue, agents would potentially be the humans. Thus, agents' availability could also change. As a result, the parallel plan could differ from the current version and more steps could be added to the plan due to ordering constraints and availability.

3.8 Multi-Agent Planning Algorithms

MAP lies between the Automated Planning and Multi-Agent communities, with strong implications in other areas, as Robotics. As it was discussed in the Introduction, approaches range from centralized to distributed planning. In case of distributed planning, some papers employ a distributed coordinated approach when generating plans [Jonsson and Rovatsos, 2011; Nissim and Brafman, 2013; Štolba and Komenda, 2017; Torreño et al., 2018], while others delay coordination and perform plan merging after generating the individual plans [Foulser et al., 1992; Ephrati and Rosenschein, 1993]. The planners also vary from strong privacy preservation to no privacy preservation. As a result, there are many ways of classifying MAP algorithms. Different classifications are explained in the following survey [Torreño et al., 2017].

The CoDMAP competition showed the variation of MAP features among the wide set of planners that participated [Štolba et al., 2015]. Regarding CoDMAP results, the most relevant planners are explained in the following paragraphs. They are divided into factored and unfactored planners, respectively.

- Planning State Machines (PSM) [Tozicka et al., 2015], as the name suggests, employs non-deterministic finite state machines (NFM) to represent a set of plans of a given problem. Each machine belongs to a different agent. Thus, each one represents the agent's set of local plans. In summary, an agent ϕ_i has to solve its local task and has to store it into the machine. Then, agents exchange the public information of their NFMs. This process is repeated iteratively until a solution is found. Privacy is preserved, as agents only communicate public information.

- MAPlan [Fišer et al., 2015] can be configured with different search algorithms and heuristic functions and can be run either centralized or distributed. Distributed MAPlan employs the TCP/IP protocol to share agents' public information. Regarding privacy, the algorithm encrypts the private data of the states shared during the exchange of information. They utilize obfuscation, so the algorithm grants weak privacy.

- FMAP [Torreño et al., 2014] is a fully distributed algorithm that preserves the agents' privacy. It contains the same number of planning and execution agents. Planning agents are in charge of solving its individual tasks. In order to provide a messaging infrastructure, FMAP is built upon the MAS platform Magentix2 [Such et al., 2013].

Now we describe the set of relevant unfactored planners:

- Serialized Iterated Width (SIW) [Muise et al., 2015] employs the algorithm Iterated Width from [Lipovetzky and Geffner, 2014] to factorize the MAP problem into subproblems; also the algorithm solves the subproblems. Even though SIW has no heuristic guidance, it achieves good performance because it greedily solves only one atomic goal at a time. The algorithm will not stop until all atomic goals are achieved jointly. On each iteration, SIW merges the plan to achieve the new goal with the previous plan that reached the previous goals.

- Agent Decomposition Planner (ADP) [Crosby, 2015] is able to divide a single-agent planning task into a set of individual tasks, one for each agent. Tasks are solved individually and then pieced together. The strength of ADP is the computation of an agent factorization that results into a set of decoupled tasks, which completely avoid interactions between the agents. Thus, individual plans are sound and plan-merging techniques are not necessary.

- Multi-Agent Planning by plan Reuse (MAPR) [Borrajo and Fernández, 2015] builds a solution iteratively. The first agent solves its obfuscated planning task and sends the result to the next agent; then, the second agent it merges the solution plan as well as the domain and problem, sends them to the next agent so on. Private information is obfuscated.

- Centralized Multi-Agent Planning (CMAP) [Borrajo and Fernández, 2015] is based on MAPR following a slightly different configuration. First, each agent obfuscates its initial planning task. Then, there exists a central agent that assigns a subset of public goals to some of the agents. After that, agents that have been assigned goals send its obfuscated plan to a common agent. The central agent is in charge of merging each agent's domain and problem and also on performing a centralized planning phase to build the final solution, taking into account the information generated into the individual plans.

- Greedy Privacy Preserving Planner (GPPP) [Maliah et al., 2014] did not participate during the official competition but it was included into the summer run (post-competition). GPPP is built on top of MAFS, a distributed privacy preserving algorithm. The main improvement is related to the global landmark-based heuristic function. During the first phase of GPPP, the planner performs global planning. Thus, agents solve a relaxed MAP task that only contains public actions. After that, the agents run an individual planning phase to compute the private plans that build the final solution. If there exist some conflicts among the individual plans, the global planning phase is launched again to find a solution that satisfies every precondition. Regarding privacy, GPPP implements weak privacy, as they obfuscate the information using private state identifiers.

This page has been intentionally left blank.

CHAPTER 4

Plan reuse

There is a field of Automated Planning called Plan Reuse which, as the name suggests, tries to re-utilize the knowledge from previous or similar plans to fix a given plan. In this Chapter, an overview of Plan Reuse is given focusing on key concepts and explanation of different approaches.

4.1 Introduction

Planning by Reuse has been widely employed in areas such as Case-Based Planning [Borrajo et al., 2015], or replanning when plan execution fails [Fox et al., 2006]. Thus, planning by reuse belongs as well to plan repair, the area of Automated Planning that tries to fix a plan without having to plan from scratch a new solution. Mainly, it consists on adapting an existing plan to a new context while modifying the original plan as little as possible. Some authors have considered the relative benefits of replanning and plan repair from the point of view of efficiency [Nebel and Koehler, 1995; Gerevini and Serina, 2000]

A planning-by-reuse planner works best when the invalid plan and the final plan are similar, as only a small set of actions needs to be changed or added to the invalid plan for it to become valid. This situation is usually given on easy-to-solve interactions e.g. grabbing the same resource or passing through the same door at the same time step. As a result, the planner will be able to efficiently generate a valid solution without generating an entire valid plan from scratch.

However, depending on the features of the given problem, interactions might be harder to fix regarding plan immutability e.g. when an agent drives to pick up a package but another agent has already picked it up or when a resource has been consumed and is not available anymore. As a result, new actions are applied to fix the current state of the problem, as agents' original plans have been forcedly changed. Usually, an alternative path has to be found for those agents that still need to achieve some goals. Thus, the final valid plan turns out to be completely different from the invalid plan. Plan-reuse planners' performance noticeably decreases on this second scenario. They cannot reuse most of the actions so they look up for new actions on the search space closer to the invalid plan, but at the same time, they are still reusing the old actions whenever possible.

4.2 Plan Reuse in Automated Planning

Plan Reuse has been studied in recent works of Automated Planning. Gerevini and Serina [2000] define a new domain-independent planning system. They use two techniques: the first one divides the actions of the plan to repair in subgroups to later solve the conflicts with a Planning Graph; the second technique uses Action Graphs to reduce the number of inconsistencies reflected in the plan that needs to be adapted. Related with MAP, van der Krogt and de Weerdt [2005] apply a plan repair system to a MAP problem so that agents can adapt their plans iteratively by exchanging goals in an auction until all plans generated are valid. Krogt and Weerdt [2005] also presented an extension of the VHPOP planner called POPR. Their approach computes a set of partial plans similar to the given input plan. Then, it analyzes the dependencies of predicates and actions of those plans by generating removal trees and uses a heuristic to compute the most promising candidate. Finally, once the candidate plan is selected, plan reuse is applied to it. The first plan reuse planner that incorporated heuristic search inside the replanning process was SHERPA [Koenig et al., 2002]. It stores knowledge about both previous plans and previous plan-construction processes.

Currently, one of the state-of-the-art plan-reuse planners is LPG-ADAPT [Fox et al., 2006]. The aim of LPG-ADAPT was to adapt an existing plan to a new context while modifying the original plan as little as possible. As we have used LPG-ADAPT in this Thesis, a detailed explanation is given in Section 4.2.1. Also, a slightly different approach was followed by Borrajo and Veloso [2012], which has also been studied in this Thesis. Their algorithm called ERRT-PLAN is explained in Section 4.2.2.

4.2.1 LPG-ADAPT

LPG-ADAPT [Fox et al., 2006] is an stochastic plan-reuse planner that employs plan repair to fix an invalid plan. This planner is built on top of LPG [Gerevini and Serina, 2002], which is inspired by random walk search algorithms. The strategy of LPG consists on performing local search in the state of plans. Thus, the algorithm builds first a structure commonly known as action graph. LPG looks for neighbors that offer a solution similar to the initial plan thanks to an evaluation function that guides the planner through the space regarding that similarity.

In LPG-ADAPT, authors define first a term called “plan stability” to establish an explicit measure of similarity between the source and the target plan. Also, the planner uses PDDL 2.1 rather than STRIPS as in LPG.

Plan repair is applied on each iteration of LPG-ADAPT as in LPG, which means that first a local search is performed in order to identify the neighbors that could potentially fix part of the invalid plan. As a result, candidates will usually contain small permutations over the original plan actions.

Candidates are evaluated using a modified version of the LPG function. It originally had three terms related to: increment of the plan execution cost, estimating the moment of time when every unsupported precondition is satisfied and increment of the number of search steps that are still needed to reach a solution. In LPG-ADAPT they added a fourth term to estimate the distance of the current plan from the original plan (plan stability). As a result, this evaluation function aims to penalize any dramatic or meaningful changes over the original plan and benefits including sequences of small changes instead. Thus, LPG-ADAPT has a good and fast performance in domains where the output plan is expected to be very similar to the invalid input plan, which is the classic plan-reuse scenario.

4.2.2 ERRT-PLAN

Borrajo and Veloso [2012] explored in ERRT-PLAN the behavior of a plan-reuse planner and proposed a solution to adapt the algorithm to a broader set of plan reuse scenarios. Thus, ERRT-PLAN is a stochastic plan-reuse planner that can run up to three different versions of plan reuse.

First, the algorithm builds a solution tree inspired on the Rapidly-exploring Random Trees (RRTs) [LaValle and Kuffner, 2001]. As a result, ERRT-PLAN has a probability of p to expand the tree towards the goal, a probability of r to expand the tree towards an action of the input plan and a third probability to expand the tree towards one of the weakest preconditions. Given a plan π, the weakest preconditions of any action $a_i \in \pi$ represent the set of propositions that are required to be true before applying a_i in the current state s_i. As a result, the goals are achieved from a_i when applying the remaining actions of the plan; the weakest preconditions act as subgoals of π.

The algorithm employs the work by De la Rosa et al. [2006], which is a reimplementation of METRIC-FF [Hoffmann, 2003] as the heuristic planner and EHC [Hoffmann and Nebel, 2001] as the search algorithm.

ERRT-PLAN receives as input the domain and problem descriptions, the probabilities for node expansion, the solution (plan) represented as an ordered set of pairs of actions and their weakest preconditions. On each iteration, a random number between 0.0 and 1.0 is generated. Depending on its value, one of the following three scenarios can be executed:

- If $(0 < n < p)$, ERRT-PLAN calls EHC and performs local search towards the goal.
- If $(p < n < (p + r)$, ERRT-PLAN retrieves first open node in which an action a from the past plan can be reused.
- If $(1 - (r + p))$, ERRT-PLAN selects one of the goals of the previous solution. Then, it obtains the weakest preconditions and selects one to perform search towards that goal.

The main difference of ERRT-PLAN with respect to LPG-ADAPT can be found on the way of guiding the local search. ERRT-PLAN uses previous plans but do not necessarily focuses on minimizing plan stability. Also, ERRT-PLAN considers goal-reuse, which is not implemented in LPG-ADAPT.

Part III

Contributions

CHAPTER 5

PMR: Plan Merging by Reuse

Plan Merging by Reuse (PMR) is the first contribution of this Thesis. In this Chapter, we present our MAP algorithm in detail. A general description as well as the description of each of its components are included in the following sections.

5.1 Introduction

PMR focuses on solving classical deterministic planning tasks, where a set of agents should find a common plan. The objectives are to:

- Efficiently solve MAP tasks by combining distributed and centralized MAP techniques.
- Focus on big-size multi-agent tasks in terms of number of agents, goals and search space, where agents have little or no interaction.
- Directly reuse existing planning techniques without further code modification.
- Effectively apply factorization to divide the main task into subtasks regarding some minimization criteria such as *plan length* or *makespan*.
- Automatically adjust to the interaction level among agents and goals.

In order to satisfy those objectives, we run up to three different planning processes inside our first contribution: a MAP algorithm called Plan Merger by Reuse (PMR). In summary, our approach work as follows. First, a preprocessing step is performed to divide the MAP task among the agents. Then, PMR performs an individual planning phase run by each agent (distributed part), a plan merging phase to generate a joint plan and the validation of that plan. When validation fails, PMR runs a plan-reuse episode. Only when the factorization or the individual planning phase fail, PMR runs a centralized planning phase. Regardless of the chosen path, PMR parallelizes the resulting plan on the last step and the output of this process is returned as the solution. As a result, PMR is able to adjust to different MAP situations. For the sake of clarity, we are not interested in privacy preserving nor joint tasks.

5.2 Algorithm

Plan Merging by Reuse (PMR) receives as input a MAP task (M'). As shown in the pseudocode in Algorithm 1, M' is formed by a number of Π_i tasks, each of them containing the information included in Definition 3.2.3. In the first step, each agent in $\phi_i \in \Phi'$ builds its plan individually (line 1). Then, we find three different scenarios:

- If all agents failed at generating a plan (lines 2-4), a centralized planner solves the MAP task M'_{joined} from Definition 5.3.1.
- Otherwise, the plans are merged. If the merged plan is valid, PMR returns it as the solution to the MAP task (lines 6-7).
- If the merged plan is invalid, PMR calls a plan repair planner that can perform plan reuse, sending the merged plan and M'_{joined} as input. The plan repair planner will try to find a solution based on the actions of the input plan (lines 9-10).

Given that we are dealing with MAP tasks, it is expected that agents can execute the actions in their plans in parallel when possible. Thus, the aim of PMR is to minimize the makespan. Therefore, in any of the three scenarios, if the plan is valid, it is parallelized to improve the makespan of the solution as explained later on Subsection 5.6.

PMR contains three off-the-shelf planners: one for each agent to plan individually, P (it can be the same or a different planner); another one capable of applying plan repair by reusing an invalid/incomplete plan (R); and the third one, which is employed by PMR when all the individual agents' planning tasks fail. In this work we use the same planner,

Algorithm 1 High level description of the PMR algorithm.

```
Algorithm PMR ($M'$,$P$,$R$)

Inputs: $M'$: MAP task
        $P$: planner
        $R$: plan-reuse planner
Output: $\pi_{\text{PMR}}$: plan

1  Forall $\phi_i \in \Phi'$ do $\pi_i$ =plan($M'_i$,$P$)
2  If $\forall \phi'_i \in \Phi'$ $\pi_i$=fail
3  Then $M'_{joined}$ = join-task(M')
         $\pi_{cen}$ =plan-centralized($M'_{joined}$,$P$)
5        $\pi_{\text{PMR}}$ =parallelize($\pi_{cen}$)
6  Else $\pi_{seq}$ =merge($\pi_1,\ldots,\pi_n$)      /* where $n = |\Phi'|$ */
7       If valid($\pi_{seq}$)
8       Then $\pi_{\text{PMR}}$ =parallelize($\pi_{seq}$)
9       Else $M'_{joined}$ = join-task(M')
10           $\pi_{reuse}$ =plan-reuse($M'_{joined}$,$\pi_{seq}$,$R$)
11           $\pi_{\text{PMR}}$ =parallelize($\pi_{reuse}$)
12 If valid($\pi_{\text{PMR}}$) Then return $\pi_{\text{PMR}}$
13 Else return no-solution
```

P, used by individual agents to run the centralized phase. In order to check the validity of the plans, VAL, the validator from the International Planning Competition, has been used [Howey et al., 2004]. The following sections explain in detail the main steps and the properties of PMR.

5.3 Planning

In this first step, each agent $\phi_i \in \Phi'$ receives as input the description of its domain and problem. The problem includes the facts, actions and goals assigned to ϕ_i. Each agent invokes a planner P to solve its planning task. As a result, a partial solution π_i to the overall MAP task is obtained per agent. Any state-of-the-art planner can be used for this task and each agent could use a different planner.

Before the distributed phase starts. If all agents fail to generate a solution, it means that more than one agent is needed in order to achieve the goals. In this case, PMR resorts to a centralized planner. Centralized planners usually receive as input the lifted representation of the planning task (one domain and one problem file). Thus, the elements of the MAP task M' should be first joined.

Definition 5.3.1. The M'_{joined} task. The merging of each $\phi_i \in \Phi'$ task at the planning level is defined in the following tuple:

$$M'_{joined} = \left\langle \bigcup_{i=1}^{n} F_{\phi_i} \cup F_{pub}, \bigcup_{i=1}^{n} A_{\phi_i} \cup A_{pub}, \bigcup_{i=1}^{n} I_{\phi_i} \cup I_{pub}, \bigcup_{i=1}^{n} G_i \right\rangle$$

Regarding the lifted representation of the planning task, the merging can be performed as stated in the following tuple:

$$M'_{joined}{}^{PDDL} = \{\bigcup_{i=1}^{n} \Pi_i^{PDDL}\} = \{D_{joined}, P_{joined}\} = \{\bigcup_{i=1}^{n} D_i, \bigcup_{i=1}^{n} P_i\}$$

The centralized planner receives $M'_{joined}{}^{PDDL}$ as input and finds a solution from scratch to the MAP problem (π_{cen}).

In Algorithm 1, we have used for the centralized phase (C), the same planner employed for individual planning (P). However, since PMR is planner-independent, we could have used any other planner.

When the distributed phase works. If at least one agent generates a solution to its task, PMR merges all the solutions. We have implemented a basic merge strategy, which is a simple concatenation. Other more elaborated techniques could be used to improve the performance of PMR. The output of the merge process is the plan π_{seq}. PMR checks if that plan is valid. If so, PMR will parallelize it as explained below. Finally, if π_{seq} was invalid, the plan reuse phase will be executed, providing $M'_{joined}{}^{PDDL}$ and π_{seq} as input.

Example (Logistics domain). Following the example explained at the end of 3.2.4, we illustrate the result after running PMR's individual planning phase. The goal strategy chosen is still LB. The assignment of agents and goals was $GA = \phi_1(g_4)$, $\phi_2(g_1, g_2)$, $\phi_3(g_3)$. Agents are ordered by name. ϕ_1 is the first one to start planning.

```
The resulting individual plans obtained from the agents are the following
    .
π1 = (load φ1 g4 B)(drive φ1 B A)(unload φ1 g4 A)
π2 = (load φ2 g1 A)(load φ2 g2 A)(drive φ2 A B)(unload φ2 g1 B)
     (unload φ2 g2 B)
π3 = (load φ3 g3 A)(drive φ3 A C$)(unload φ3 g3 C)
```

Figure 5.1: Individual planning phase to solve the problem described in Figure 3.1

```
After concatenation
 $\pi_{seq} = \{\pi_1 \oplus \pi_2 \oplus \pi_3\} =$
        (load $\phi_1$ $g_4$ $B$)(drive $\phi_1$ $B$ $A$)(unload $\phi_1$ $g_4$ $A$)
        (load $\phi_2$ $g_1$ $A$)(load $\phi_2$ $g_2$ $A$)(drive $\phi_2$ $A$ $B$)
        (unload $\phi_2$ $g_1$ $B$)(unload $\phi_2$ $g_2$ $B$)(load $\phi_3$ $g_3$ $A$)
        (drive $\phi_3$ $A$ $C$)(unload $\phi_3$ $g_3$ $C$).

The plan $\pi_{seq}$ is valid but from the point of view of MAP this sequence of
    actions is not efficient.
```

Figure 5.2: Sequential plan (π_{seq}) that solves the problem described in Figure 3.1

5.4 Plan Reuse

We assume that the current invalid plan usually includes most of the strong actions that would make it a valid plan. Thus, by using plan repair techniques we are expecting PMR to generate a plan faster than planning from scratch. In the worst case, plan repair is PSPACE-complete [Nebel and Koehler, 1995]. But, in practical terms and under our assumption of closeness between the invalid plan and the valid one, plan repair techniques have shown good results [Fox et al., 2006].

Usually, planners that perform plan repair receive three inputs: a domain, a problem and a plan. Examples of plan-repair planners are LPG-ADAPT [Fox et al., 2006], ERRT-PLAN [Borrajo and Veloso, 2012] and our contribution RRPT-PLAN, that will be explained on Chapter 6. We use such a planner to transform an invalid input plan π_{seq} into a valid plan. In case the plan-repair planner solves the planning task and the plan is valid (π_{reuse}), PMR parallelizes it to improve the makespan and returns it.

Example (Logistics domain). Following the example explained at the end of Section 3.2.4, two scenarios could trigger PMR's plan reuse phase. The first scenario is given when solving the problem using the goal strategy *All*. This implies that each agent has to plan individually to deliver the four packages marked as goals. After concatenation (π_{seq}), ϕ_2 will not find any of the packages at the original locations, as ϕ_1 has already moved them. The same happens to ϕ_3. Thus, π_{seq} is invalid and needs to be fixed. PMR will call the plan-reuse phase. The second scenario is given when two agents need the same resource. For instance, suppose that trucks need drivers in order to transport the packages. If there was only one driver per city, trucks would need that driver to move from one city to another. Each agent would need to pick up a driver, and independently of the goal strategy selected, two trucks could pick the same driver during the individual planning phase. When concatenating the plans, ϕ_2 could have picked up the same driver as ϕ_3 (as they are located at the beginning at the same city). Thus, a failure would arise when ϕ_3 needs the driver as she is not at CityA.

5.5 Centralized planning

Centralized planning within PMR is only used when the individual planning phase fails. For instance, a situation where PMR would fail using the same example presented at the end of Section 3.2.4 is the following: if the *All* GA strategy was selected instead of LB or BC, as agents have to achieve every goal from the problem individually, ϕ_1 and ϕ_3 would not be able to make it because they cannot achieve all goals (see Table 3.1). ϕ_1 has an estimated cost of ∞ for *g3* as well as ϕ_3 for *g1* and *g4*. Additionally, if ϕ_2 was not able to transport fragile packages, *g1* and *g4* could not be delivered either. As a result, π_{seq} would not be generated and the centralized planner would be called instead.

5.6 Parallelization

Most state-of-the-art planners return sequential plans, since they do not usually consider minimizing the temporal execution window. As said before, in order to benefit from the existence of multiple agents executing in parallel a plan, we parallelize either the sequential plans generated by merging, the centralized plan, or the repaired plan. This function transforms the plan received as input into a parallel one.

Parallelization is performed in two steps: (1) converting the input total-order plan into a partial-order one and (2) parallelizing the resulting plan by ordering actions in the first time step that satisfies all ordering constraints in the partial-order plan. Details of the parallelization process are described in Section 3.7. The cost of the parallelization is quadratic in the number of actions in the plan.

Due to the parallelization process, more than one action might be executed at each plan step, as long as they are not mutex. In loosely-coupled domains, parallelization usually reduces the makespan of the solution plan proportionally to the number of agents. As actions and predicates of one agent are independent of those of the rest of agents, a considerable amount of actions can be executed in the same time step. However, as the level of interaction increases (e.g tightly-coupled domains), parallelization will not cause such an impact in terms of makespan.

Example (Logistics domain). Following the example, the plan π_{seq} described in Subsection 5.3 is validated by VAL. As this was a simple example with no interaction among agents, the plan is valid and the parallelization phase starts by receiving π_{seq} as input. The resulting plan, π_{PMR} is shown below in Figure **??**.

π_{PMR} =
1: (load $\phi_1\, g_4\, B$) (load $\phi_2\, g_1\, A$) (load $\phi_3\, g_3\, A$)
2: (drive $\phi_1\, B\, A$) (load $\phi_2\, g_2\, A$) (drive $\phi_3\, A\, C$)
3: (drive $\phi_2\, A\, B$) (unload $\phi_1\, g_4\, A$) (unload $\phi_3\, g_3\, C$)
4: (unload $\phi_2\, g_1\, B$) (unload $\phi_2\, g_2\, B$)

Figure 5.3: Parallel plan (π_{PMR}) that solves the problem described in Figure 3.1

5.7 Properties

In this Section we analize the properties of PMR in terms of optimality, soundness and completeness, respectively. There are three different flows in PMR (merging, reuse, centralized) so properties are analyzed on each particular case.

As most work on plan merging, PMR performs suboptimal planning.

- *Case Merging*: Agents' plans can be locally optimal but the simple merging of individually generated plans cannot ensure optimality, even if optimal planners were used.
- *Case Reuse*: As the merging phase cannot ensure optimality, neither can reuse, as the merged plan is received as input.
- *Case Centralized*: Optimality could be ensured only if an optimal planner is used in combination with the Goal-Assignment strategy ALL. Using any other strategy does not guarantee that assigning one goal to a certain agent and not to a different other makes the resulting plan the optimal solution. Specially, as some of the agents in those strategies can be left out of the planning phase.

In relation to soundness, we have three different possibilities. As PMR is configured with off-the-shelf planners, soundness will depend on the selected planners for P and R or C. Any sound planner satisfies Definition 2.2.

- *Case Merging:* If sound planners are employed, each individual plan is valid. After merging, soundness cannot be ensured because the merged plan might not be valid due to the agents' interactions.
- *Case Reuse:* PMR checks for validity, and if the plan is invalid, the soundness depends on the plan repair step. In summary, PMR is sound if R is sound.
- *Case centralized:* If the centralized planner is invoked, soundness is ensured by using a sound planner in C.

In relation to completeness, PMR is complete assuming that all goals are assigned to all agents (ALL Goal Strategy) and the selected off-the-shelf planners (P, R and C) are complete too. Under those conditions:

- *Case Merging:* Individual planning is globally incomplete since there might be a joint plan that is not contained in the space of each agent working separately. However, it is locally complete for each agent.
- *Case Reuse:* Once the plans are generated, PMR calls the plan repair strategy, which is complete.
- *Case Centralized:* When individual planning fails, the planner associated to C is called, which is complete.

The parallelization step does not affect the completeness. Finally, PMR has the same complexity as Automated Planning and MAP, which is PSPACE [Brafman and Domshlak, 2008; Bylander, 1994].

5.8 Dealing with Privacy inside PMR

Maintaining agents' privacy is one of the recent research lines in MAP. Although we have not considered private information in our approach, PMR supports weak privacy [Nissim and Brafman, 2014]. In this Section we briefly discuss some privacy-related aspects. Section 3.4.3 contains a summary of the privacy model that could be implemented in our contributions, which is MAPR [Borrajo and Fernández, 2018]. In that paper, its authors included a detailed explanation about the way weak privacy can be applied to similar contributions. Therefore, next paragraphs contain the process on how PMR could support weak privacy.

First of all, PMR receives the M' task, which is already factorized. Previously, during the first factorization -from the planning task to the MAP task- the private components would be identified and encrypted.

When the planning task is given in MA-PDDL (alternatively, as a PDDL description plus some privacy related information), the parser generates the PDDL versions. Agents' privacy could be preserved if the parser uses an encryption algorithm for the private components of each agent . In that case, subsequent steps can benefit from the level of encryption of that algorithm i.e. obfuscation for weak privacy.

The encryption algorithm should maintain some properties related to later planning tasks:

- First, VAL [Howey et al., 2004] should be able to validate an encrypted plan against an encrypted domain and problem files.
- Second, the centralized and the plan repair planners should be able to perform planning with an encrypted version of the domain and problem files.
- Third, the parallelization step must access the encrypted actions' models, since it reasons about preconditions and effects to detect mutex information.

As an example of a simple encryption mechanism, the parser can substitute private predicate, action, type and object names for randomly generated names for each agent. This was already suggested in [Borrajo, 2013]. As a result, when maintaining privacy, PMR would receive the same MAP task M' but with the agent's information (predicates, actions) obfuscated. Details about how to encrypt each agent's task are given in [Borrajo and Fernández, 2018].

Recent works in MAP present stronger and more robust privacy preserving planning (PPP) algorithms. MAFS, SECURE-MAFS [Brafman, 2015] and GPPP [Maliah et al., 2014] are state-of-the-art PPP algorithms. There exist new variations of MAFS such as MAFBS [Maliah et al., 2017], which presents a new way of reducing the number of messages exchanged among agents while preserving strong privacy; also, a recent formalization that supports joint/collaborative actions for PPP algorithms [Shekhar and Brafman, 2018]. On the other hand, another state-of-the-art PPP algorithm is PSM [Tozicka et al., 2015]. Their authors recently explored privacy leakage on MAP algorithms and proposed a new class of secure MAP algorithms (SECMAP) [Štolba et al., 2016b] that can avoid those leaks. Also, they studied privacy preserving from the point of view of criptography, concretely a subfield called secure multiparty computation [Tozicka et al., 2017] .

This page has been intentionally left blank.

CHAPTER 6

RRPT-PLAN: REUSE RANDOM PLANNING TREE

REUSE & RANDOMLY EXPLORING PLANNING TREE (RRPT-PLAN) is the second contribution of this Thesis. We have developed a stochastic plan-reuse planner that combines search, sampling and plan reuse to solve a planning task. In this Chapter, we present the contribution in detail. A general description as well as the description of each of its components are included in the following sections.

6.1 Introduction

Among the spectrum of different scenarios that can be given in trying to repair a plan, the most frequent one is when the invalid input plan is very similar to the final solution. Plan reuse will be very efficient as most of the final plan's actions are already on the invalid input plan. Thus, the planner reuses most of the invalid plan actions and has only to include a small set of new actions to transform it into a valid one. However, sometimes, the solution plan should be completely changed, as when trying to solve the Rovers scenario described on Figure 6.1. As both problems are very similar, any plan-reuse planner would usually choose the solution obtained from problem 6.1a to solve problem 6.1b. Unfortunately, that would not be efficient, as the plan-reuse planner would ignore the existence of the new path 0-10. Thus, the plan-reuse planner will return as a solution the same one obtained from problem 6.1a. In order to increase efficiency, combining

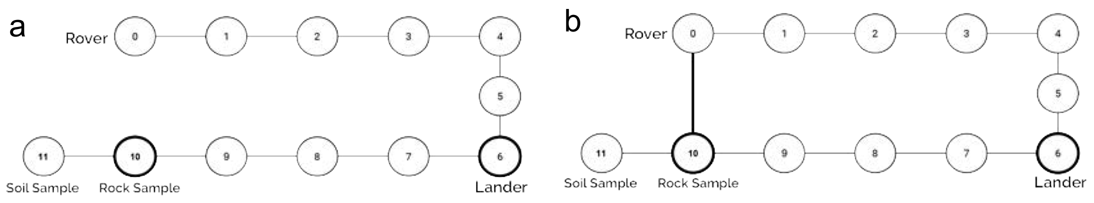


Figure 6.1: Example of a simple Rovers problem. (a) depicts the initial state of a Rovers problem where the Rover agent has to take the soil and rock samples to later send them to the lander. (b) depicts the same problem as on the left, but now the Rover agent can directly traverse 0-10 to reach the samples.

search and plan-reuse would help any plan-reuse planner to better solve a wide variety of scenarios: from the ones that are very similar to the ones that look similar but they are not.

Our approach RRPT-PLAN combines search, sampling and plan reuse. We were inspired by two previous works that are explained as follows.

The first previous work is ERRT-PLAN [Borrajo and Veloso, 2012], which was explained in Section 4.2.2. The second work, RPT [Alcázar et al., 2011], was taken as a basis for developing our contribution. RPT was explained in Section 2.7.

Our contribution, RRPT-PLAN, emulates the ERRT-PLAN behavior by receiving as input the invalid plan, the domain and problem descriptions and the set of probabilities; the weakest preconditions that ERRT-PLAN stored are not considered on our implementation. RRPT-PLAN combines search, plan reuse and sampling using the state-of-the-art planner FAST DOWNWARD [Helmert, 2006], which was the base planner used by RPT. Details on RRPT-PLAN and the main differences regarding RPT and ERRT-PLAN are explained later on Section 6.7.

As a result, RRPT-PLAN has three parameters:

- ϵ: limits the number of expanded nodes of the local search.
- p: probability of executing local search towards the nearest goal.
- r: probability of executing plan reuse.

Depending on the initial values of the set of parameters (explained later on detail), RRPT-PLAN performs either search, sampling or plan reuse on each iteration. The result will be a valid plan based on the input plan. RRPT-PLAN works as outlined in Algorithm 2. The main steps are: preprocessing, search-reuse-sampling and tracing back the solution.

Algorithm 2 Description of the RRPT-PLAN planning algorithm.

```
Algorithm RRPT-PLAN

Parameters: $\epsilon$, $p$, $r$
Inputs: domain, problem, input-plan
Output: Plan (solution)

1   $sas\_description$ ← translate(domain, problem)
2   $sas\_operators$ ← translate(input-plan, $sas\_description$)
3   $tree$ ← $q_{init}$
4   $first\_iteration$ ← $true$
5   while $not$ goalReached() do
6     $n$ ← random() /*between 0 and 1*/
//Search towards the goal
7     if ($not(first\_iteration) \wedge (n < p)$) then
8       $tree$ ←search( $q_{goal}$, $tree$, $\epsilon$)
//Plan reuse
9     else if (($first\_iteration$)$\vee$ ($n \geq p \wedge n < (p+r)$)) then
10      $first\_iteration$ ← $false$
11      $q_{reuse}$ ← $unreutilized\_nodes$.pop()
12      $tree$ ←reuse($q_{reuse}$, $sas\_description$, $sas\_operators$, $tree$)
//Sampling
13    else
14      $q_{rand}$ ← sampleSpace($\mathbb{S}$)
15      $tree$ ←search( $q_{rand}$, $tree$, $\epsilon$)
16      $tree$ ←search( $q_{goal}$, $tree$, $\epsilon$)
17  end while
//Tracing back the solution
18  $solution$ ← traceBack($q_{goal}$)
19  return $solution$
```

The following subsections explain in detail first the current configuration of RRPT-PLAN and then each step of the algorithm. Subsection 6.6 describes the planning properties of RRPT-PLAN. Finally, subsection 6.7 explains the differences between RRPT-PLAN, ERRT-PLAN and RPT.

6.2 Configuration

RRPT-PLAN follows the same configuration as RPT. Thus, the planner FAST DOWNWARD was configured with greedy best-first search choosing lazy evaluation as its local search algorithm. The heuristic used is the h_{FF} heuristic [Hoffmann and Nebel, 2001].

As both RPT and RRPT-PLAN are inspired on RRTs, a tree structure is built during the planning process. RPT originally had local search and sampling phases. We have added the plan reuse phase. Details on the implementation of the tree are explained in [Alcázar et al., 2011].

In order to build the solution tree, we have to describe the node structure that stores the information on each iteration:

Definition 6.2.1. RRPT-PLAN *node*. A node q of the solution tree T can be defined as a tuple $\mathcal{N} = \langle s_i, \rho_i, \tau_i, ra, cbs \rangle$, where:

- s_i represents the current state
- ρ_i contains a pointer to the previous node
- τ_i stores the sequence of actions that reaches s_i from the parent
- ra represents the index of the last reused action of the input plan
- cbs stores the cached best supporters for every proposition $q \in F$. We refer to cached best supporters as the implementation previously included in RPT [Alcázar et al., 2011] where the actions that first achieve a given proposition in the reachability analysis are stored in order to compute h_{FF} efficiently.

The following subsections explain in detail the preprocessing, the loop search-reuse-sampling of the RRPT-PLAN algorithm and some key details.

6.3 Preprocessing

The first step of the preprocessing translates the PDDL domain and problem to the SAS+ language [Bäckström and Nebel, 1995] by calling and using FAST DOWNWARD's Translate module [Helmert, 2006]. This module also returns a list of operators, which contains every valid combination of actions and parameters that can be generated from the given domain and problem. Our algorithm RRPT-PLAN uses that list to translate the input plan to SAS+. Thus, for each action of the plan, the algorithm looks for the equivalent SAS+ operator on that list (line 1, Algorithm 2). As a result, the input plan is transformed into a sequence of SAS+ operators instead of instantiated PDDL actions. For simplicity, along the following sections we will use the word *action* when referring to these SAS+ operators. SAS+ was explained in Section 2.5.

6.4 Search-Reuse-Sampling

After preprocessing is completed, RRPT-PLAN executes a loop (lines 4-16, Algorithm 2) that launches either the search, reuse or sampling phase until a valid solution is found. The algorithm only returns that no solution was found after all the search space has been explored. At each iteration, a random number (n) is generated. Depending on the value of the random number and the values set on p and r, one of the following scenarios is executed.

6.4.1 Search

When $n < p$, RRPT-PLAN runs the local search scenario (lines 6-7, Algorithm 2). Algorithm 3 describes the local search algorithm and Figures 6.2 and 6.3 show the process inside the tree. The algorithm receives as input the node (identified as q). The local search can only expand a maximum of ϵ nodes. In order to differentiate each node on Figures 6.2, 6.3 and Algorithm 3, a brief description of them is given below:

- q_{init}: first node on the tree. It contains the initial state of the problem.
- q_{near}: last expanded node of the tree before applying local search towards the goal.
- q_{new}: last expanded node after applying local search towards the goal.
- q_{goal}: node that contains the goal state of the problem. When it is reached, it means that a plan has been found.

Before explaining the process, some general remarks and data structures are presented:

Algorithm 3 Description of the search process of RRPT-PLAN. Last expanded node (q_{new}) is added to the *tree* and q_{near} is set as its parent.

```
Function search of RRPT-PLAN
Inputs: $q$, $tree$, $\epsilon$
1    $q_{near} \leftarrow$ findNearest($tree$, $q$)
2    $q_{new} \leftarrow$ join($q_{near}$, $q$, $\epsilon$)
3    $tree \leftarrow$ addNode($tree$, $q_{near}$, $q_{new}$)
4    return $tree$
```

- After running local search, RRPT-PLAN only adds a single node to the final tree structure, which is the last expanded node of the local search.
- A node can be expanded during search only once.
- There is an open list to store the unexpanded nodes for the local search. That list is ordered: nodes closer to the goal first, for efficiency reasons. The algorithm always extracts the first node on the list.

The *search* function is called (line 7, Algorithm 2), so the local search is performed towards the goal state, q_{goal}. The algorithm takes the first node of the open list (q_{near} - Algorithm 3, line 1), which is the closest expanded node found towards the goal state (q_{goal}). Thus, the initial state of the local search is the one stored on q_{near}. Local search is then executed until the solution is found or ϵ number of nodes are expanded. Finally, the last expanded node from the local search (q_{new} - Algorithm 3, line 2) is stored on the *tree* (Figure 6.3). As it was previously said, this node contains a pointer to its parent, q_{near}, and also the subset of actions that were instantiated during the local search to reach the node's current state from its parent q_{near}.

6.4.2 Reuse

When $p \le n < (p+r)$, RRPT-PLAN runs the plan reuse scenario (lines 8-11, Algorithm 2). During the first iteration, the algorithm always performs plan reuse regardless of the values of p and r. This decision is further justified in experiments from Section 7.4. Algorithm 4 describes the steps to run plan reuse. Figures 6.4 and 6.5 show the process inside the tree. In order to differentiate each node the Figures and Algorithm 4, a brief description of them is given below.

- q_{init}: first node on the tree. It contains the initial state of the problem.

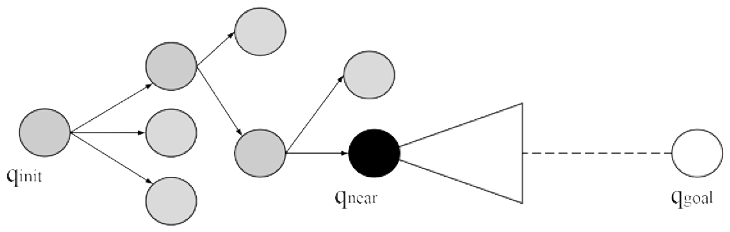


Figure 6.2: First step of RRPT-PLAN search towards the goal. Local search is run from q_{near}, which is the closest expanded node found to the goal so far.

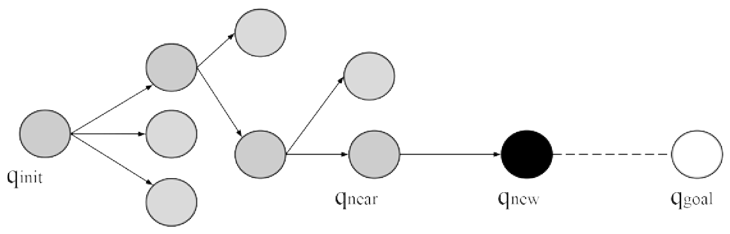


Figure 6.3: Second step of RRPT-PLAN search towards the goal. After expanding ϵ number of nodes, q_{new} is the closest node to q_{goal}. The node q_{new} stores the plan to reach q_{new}'s state from q_{near}. Finally, q_{new} is stored into the tree.

- q_{reuse}: first node of the plan reuse open list. Its state is checked when trying to reuse the first action.
- $q_{reuse'}$: node created after the first action is reused. Its state and its plan are being updated during the plan reuse process as long as new actions can be reused. At the end of the process it is added to the tree as child of q_{reuse}.
- q_{goal}: node that contains the goal state of the problem. When it is reached, a plan has been found.

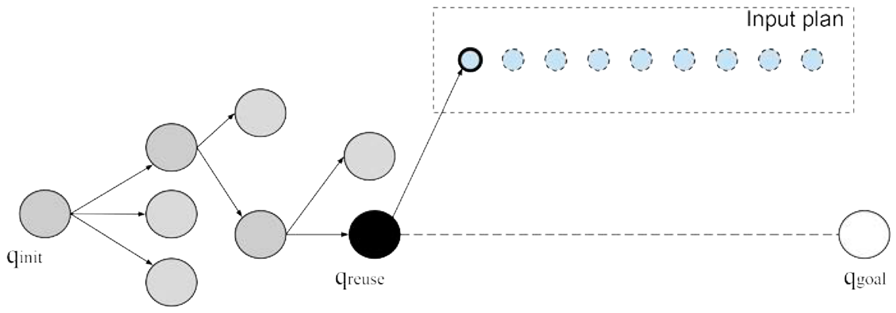


Figure 6.4: The first step of plan reuse in RRPT-PLAN. The state of q_{reuse} is evaluated to see if the first action from the input plan can be directly applied.

Before explaining the process, some general remarks and data structures are presented:

- Plan reuse can be applied to a node as long as the last reused action is not the last action of the input plan.
- There is an open list to store and provide the nodes to which plan reuse is applied. The algorithm always extracts the first node on the list. When local search or

Algorithm 4 Description of the plan reuse process of RRPT-PLAN. The node $q_{reuse'}$, which contains the reused actions from $sas\_operators$, is added to the tree and q_{reuse} is set as its parent.

```
Function reuse of RRPT-PLAN

Inputs: $q_{reuse}$, $sas\_description$, $sas\_operators$, $tree$

$i \leftarrow$ last_action_reused($q_{reuse}$)
$state \leftarrow$ getCurrentState($q_{reuse}$)
$q_{reuse'} \leftarrow$ createNewNode($state$)
$reuseSuccess \leftarrow false$
$applicable \leftarrow true$
while ($i$ <sizeof($sas\_operators$) $\wedge$ $applicable$) do
    $current \leftarrow sas\_operators$.get($i$)
    $operators \leftarrow$ getApplicableOperators($state$, $sas\_description$)
    if isApplicable($current$, $operators$) then
        $reuseSuccess \leftarrow true$
        $state' \leftarrow$ updateState(state, $current$, $q_{reuse'}$)
        $q_{reuse'} \leftarrow$ insert($state'$,$current$,$i$, $q_{reuse'}$)
        if (goalReached()) then
            $tree \leftarrow$ addNode($tree$,$q_{reuse}$,$q_{reuse'}$)
            return $tree$
        $i \leftarrow i + 1$
        $state \leftarrow$ ($q_{reuse'}$).$state$
    else $applicable \leftarrow false$
if $reuseSuccess$ then
    $tree \leftarrow$ addNode($tree$,$q_{reuse}$,$q_{reuse'}$)
return $tree$
```

sampling add new nodes to the tree, they are also automatically added to the plan-reuse open list.

The algorithm takes the first node of the plan reuse open list (q_{reuse}, line 1 Algorithm 4) and gets the position of the last action reused (stored on the node; by default 0 when none of the actions has been yet reused). Then, it iterates over the sequence of actions of the input invalid plan. For each one of them it checks if the action can be applied to the current state of the node (Figure 6.4). If this is true, the action ($current$) is added into the plan and the current $state$ and index i are updated (lines 10-11, Algorithm 4). In addition, when an action is added to the new plan, the algorithm checks if the goal state has been reached as well (line 12, Algorithm 4). The reuse process will be repeated

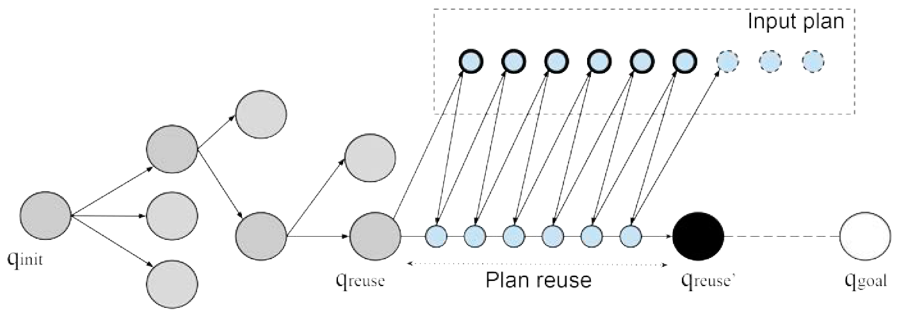


Figure 6.5: The second step of plan reuse in RRPT-PLAN. As long as there are actions that can be applied to the current state, they will be stored inside a new node $q_{reuse'}$ which at the end of the process, when no more actions can be reused, will be the child of q_{reuse}.

until an action cannot be applied (Figure 6.5). In that case, the position of the last reused action, the current state and the sequence of the reused actions are stored into the node $q_{reuse'}$. Also, the previous node q_{reuse} is set as parent.

6.4.3 Sampling

When $(p+r) \leq n < 1$, RRPT-PLAN runs the sampling scenario. Lines 13-15 of Algorithm 2 describe the steps to run the sampling process. Figures 6.6, 6.7 and 6.8 show the process inside the tree. In order to differentiate each node on the Figures, a brief description of them is given below.

- q_{init}: first node on the tree. It contains the initial state of the problem.
- q_{rand}: node that contains a random valid state from the search space.
- $q_{sampling}$: closest node from the tree to the sampled state.
- q_{new}: last expanded node towards q_{rand}.
- $q_{neargoal}$: last expanded node towards the goal.
- q_{goal}: node that contains the goal state of the problem. When it is reached, it means that a plan has been found.

Before explaining the process, some general remarks and data structures are presented:

- A new node is added to the tree at every iteration. If the solution is not reached during sampling, the last expanded node is added to the tree.

- After sampling, when the new node is obtained (q_{new}), a new local search is performed towards the goal until the limit ϵ is reached. This is equivalent to the *Extend* phase of RRTs.

First, the algorithm obtains a random valid state (q_{rand}) after sampling the search space ($\mathbb{S}$). Then, the closest node to the sampled state is found ($q_{sampling}$), Figure 6.6 by computing h_{FF}. Details about how the random state and the distance are computed can be found in [Alcázar et al., 2011]. After $q_{sampling}$ is identified, a local search is performed from there towards q_{rand}. However, q_{rand} might not be reached because of the limit ϵ. If this happens, the last expanded node is stored on the tree (q_{new}). The next step is to perform a new local search from q_{new} towards the goal (q_{goal}), Figure 6.7. This is the *Extend* phase, already implemented in RPT [Alcázar et al., 2011]. The last expanded node from the *Extend* local search is then stored into the tree ($q_{neargoal}$), Figure 6.8.

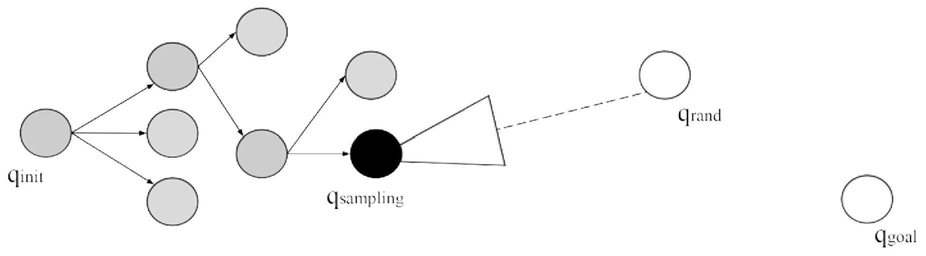


Figure 6.6: The first step of sampling in RRPT-PLAN is shown on the left. Once the sampled node q_{rand} is obtained, a local search is run from $q_{sampling}$ towards that node. $q_{sampling}$ represents the closest node from the tree to the sampled node.

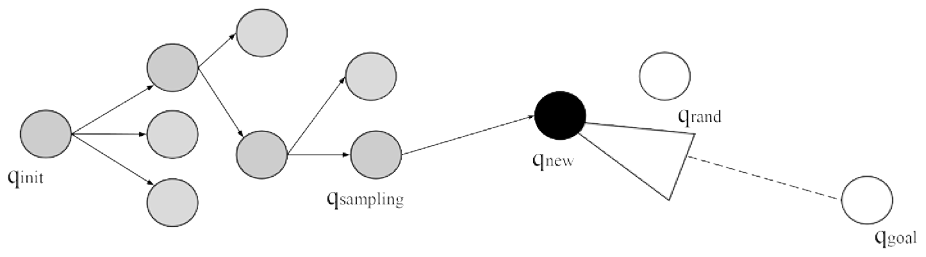


Figure 6.7: The second step of sampling in RRPT-PLAN is shown on the right. After local search is performed, q_{new} is added into the tree. Now a local search towards the goal (q_{goal}) is run.

The search-reuse-sampling phase is repeated once per iteration, independently of the strategy (search, reuse, sampling) used. The following subsection explains how the algorithm is capable of tracing back the solution through the tree.

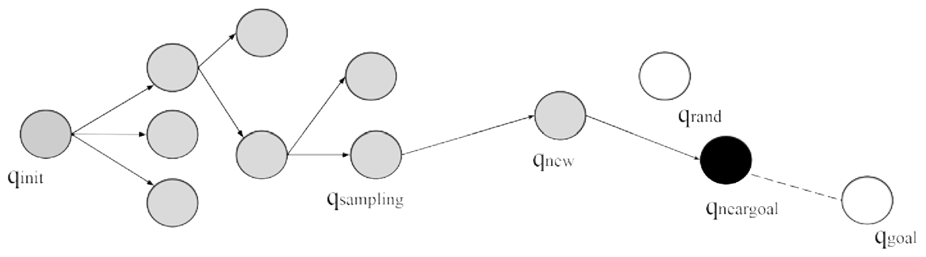


Figure 6.8: The third step of sampling in RRPT-PLAN is shown below the previous steps. After expanding ϵ number of nodes, $q_{neargoal}$ is the closest node to q_{goal} so it is stored into the tree.

6.5 Tracing back the solution

In order to retrieve the solution plan π, the algorithm has to check if q_{goal}'s state s_{goal} satisfies every proposition in G, so that $G \subseteq s_{goal}$. If so, RRPT-PLAN starts tracing back the solution plan from the node q_{goal}. From q_{goal}, RRPT-PLAN obtains the link (ρ_{goal}) to the parent's node and repeats the process until the initial node (q_{init}) has been reached. Figure 6.9 illustrates the process. Solution nodes are stored into a list called $Tree_{sol} = \{q_{init},...,q_{goal}\}$ where q_{init}'s state $s_{init} = I$ and q_{goal}'s state $G \subseteq s_{goal}$. Nodes' subplans are concatenated to obtain $\pi = \{\tau_{init} \oplus \tau_2 \oplus ... \oplus \tau_{goal}\}$. Each τ_i contains at least one action of the solution plan. The concatenation gives as a result the solution plan $\pi = \{a_1, a_2, ..., a_m\}$.

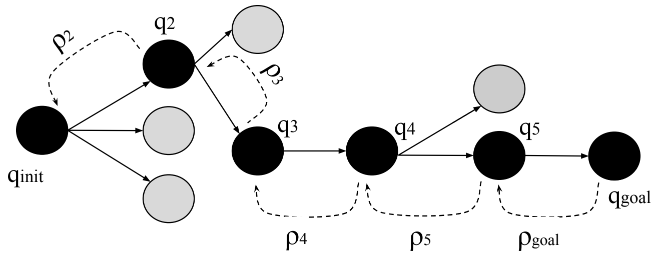


Figure 6.9: Tracing back the solution in RRPT-PLAN. The algorithm starts on q_{goal} and goes backwards on the *Tree*, obtaining in each step a new solution node through the link ρ_i. Black nodes of the Figure represent the solution nodes. They are stored in order on a list called $Tree_{sol}$.

6.6 Properties

RRPT-PLAN is an algorithm that performs suboptimal planning. Optimality is out of the scope of this work. Also, RRPT-PLAN is incomplete, as it cannot assure that a solution will always be found. For instance, extreme cases where $p = 1.0$ (only search) or $r = 1.0$ (only plan-reuse) or both have 0.0 values (only sampling); these are some configurations where RRPT-PLAN could fail to find a solution - specially if there exists some timeout. Regarding only search ($p = 1.0$ $r = 0.0$), RRPT-PLAN can get stuck after exploring and expanding every existing node of the search space and still not being able to obtain a solution. Regarding only plan reuse ($p = 0.0$ $r = 1.0$), if no action of the input plan can be reused or every action has been reused but it is not enough to solve the problem, RRPT-PLAN will constantly enter into the reuse phase and will never obtain a solution to the problem. Regarding only sampling ($p = 0.0$ $r = 0.0$), it is the same case as only applying search. Sampling explores the search space randomly, but after exploring and expanding every node of the search space, the solution might still not be found.

Finally, the plans generated by RRPT-PLAN are sound. First, when performing local search, actions are only applied to valid planning states. Second, when reusing actions, the successors of the current state are generated to obtain the list of applicable operators that reach those successors. Before including the action into the plan, it is verified that the equivalent operator of the action-to-be-reused appears on the list of applicable operators. Third, when performing sampling, the random node will be valid, as RPT included a procedure to only consider valid states. Fourth, we formally demonstrate $\Gamma(s_{init}, \pi) \models s_{goal}$ to proof the soundness of RRPT-PLAN. Given q_{init} as the first node on the *Tree*, which contains $s_{init} = I$; and q_{goal} as the last expanded node which contains the goal state $G \subseteq s_{goal}$. Given the list $Tree_{sol}$, which contains the sequence of nodes starting on q_{init} that reaches the goal state; and given plan π which contains the sequence of actions to reach s_{goal} from s_{init}, where $\tau_{init} = \emptyset$ as the initial node does not contain any action of the plan yet.

$$\pi = \{\tau_{init} \oplus \tau_2 \oplus \ldots \oplus \tau_{goal}\} = \{\overbrace{a_1, a_2, \ldots, a_k}^{\tau_2}, \overbrace{a_{k+1}, \ldots a_p}^{\tau_3}, \ldots, \overbrace{a_{r+1}, \ldots, a_m}^{\tau_{goal}}\} \tag{6.1}$$

From the set of nodes of $Tree_{sol}$ and the actions from π, using the function $\gamma(s_i, a_i) = s_{i+1}$ we can generate every intermediate state (s_i^j) between the q_i nodes as follows.

$$S = \left\langle \underbrace{s_{init}}_{q_{init}}, \overbrace{(s^1_{init}, s^2_{init} ... s^{k-1}_{init}), \underbrace{s_2}_{q_2}}^{\tau_2}, \overbrace{(s^1_2, s^2_2 ... s^{p-1}_2), s_3}^{\tau_3}, ..., \overbrace{(s^1_r, s^2_r ... s^{m-1}_r), \underbrace{s_{goal}}_{q_{goal}}}^{\tau_{goal}} \right\rangle \tag{6.2}$$

Thus, when the subplan τ_2 is applied to s_{init} following Equation 2.2, the resulting state is s_2.

$$\Gamma(s_{init}, \tau_2) \models s_2 \tag{6.3}$$

Theorem 6.1. RRPT-PLAN *is sound.*

Proof. By induction:
Base case: $s_{init} = I$ *is reachable from the initial state by* q_{init} *construction.*
Inductive step: *if* s_i *is reachable from* s_{init}, s_{i+1} *is a reachable state from* s_{init}.
By construction:

$$\Gamma(s_{init}, \tau_2 \oplus \tau_3 ... \oplus \tau_i) \models s_i \tag{6.4}$$

$$\Gamma(s_i, \tau_{i+1}) \models s_{i+1} \tag{6.5}$$

$$\Gamma(s_{goal-1}, \tau_{goal}) \models s_{goal} \tag{6.6}$$

Thus, s_{goal} is reachable from the initial state s_{init} and G is satisfied.

$$\Gamma(s_{init}, \underbrace{\tau_2 \oplus ... \oplus \tau_{goal}}_{\pi}) \models s_{goal}, \text{where } G \subseteq s_{goal} \quad \square \tag{6.7}$$

6.7 Differences of RRPT-PLAN regarding previous works

Apart from the obvious difference in implementation (ERRT-PLAN code was based on a reimplementation of Metric-FF in Lisp), RRPT-PLAN presents some differences with respect to the previous works.

- First, RRPT-PLAN presents a more clear bias towards search than ERRT-PLAN. While ERRT-PLAN considered the search step as adding one more node to the tree, RRPT-PLAN search algorithm works by expanding ϵ nodes in the same step, where we have found that ϵ should take big values.

- Second, ERRT-PLAN sampling of goals was directed by the computation of weakest preconditions from the input plan, while RRPT-PLAN sampling uses RPT sampling procedure instead. However, RRPT-PLAN does not use RPT's computation of *h2* mutexes for that task. By avoiding that computation, the aim is to speed up the process.

- Third, ERRT-PLAN and RPT considered the goal sampling step as equally relevant. On the contrary, RRPT-PLAN assigns a very small role to goal sampling by assigning a very low probability of using sampling.

All these differences are due to the diverse uses of ERRT-PLAN, RPT and RRPT-PLAN. In the case of ERRT-PLAN, authors were studying the effects of different strategies of plan reuse (replanning from scratch vs. eager use of the previous plan) in a wide variety of scenarios. In RRPT-PLAN we are interested in a very particular kind of plan reuse scenario, where in most cases, the reuse strategy is a mixture between the two extremes. Finally, the obvious difference with respect to RPT is that RRPT-PLAN can partially reuse a previous plan and that the sampling phase is not considered as important as the other two.

Chapter 7

Empirical evaluation

This Chapter presents an empirical evaluation on the performance of our two contributions, PMR and RRPT-PLAN. We compare them with other state-of-the-art planners.

7.1 Introduction

We have divided the experiments and results in six different sections structured as follows. First, Section 7.2 describes metrics and configuration environments. Section 7.3 presents some experiments specifically designed to explain the flexible behavior of RRPT-PLAN in different plan-reuse scenarios. Section 7.4 analyzes RRPT-PLAN in terms of parameters to select the best configuration. Section 7.5 shows the results of running the CoDMAP competition with different configurations of PMR and other state-of-the-art, multi-agent and centralized, planners. Then, Section 7.6 analyzes the results after changing the CoDMAP agentification. Also, Section 7.7 shows the performance of PMR when scaling the number of agents. Section 7.8 shows the performance of PMR against the same set of planners previously used on Section 7.5. However, the set of problems is harder to solve, containing a reasonable amount of agents and goals to reach. We have also included three new domains specifically designed to evaluate the makespan metric in order to show PMR's potential. Finally, Section 7.9 describes some general remarks and conclusions extracted from the extensive set of experiments.

7.2 Experimental setup

For each set of experiments described in the following sections, results of coverage (number of solved problems), quality (cost and makespan) and planning time are shown. In order to compute these metrics, we have used the scores of the International Planning Competition (IPC)[1]. For computing those metrics, time, makespan and cost, we use the formulas described in Equations 2.5 and 2.8. The time bound to solve each problem is 1800s. For every domain presented on the tables, 20 problems were run, except for the ones of Section 7.4 where 15 problems per domain were run instead (the reason is explained on that Section). All the experiments were run on an Intel(R) Xeon(R) X3470 2.93GHz with 8 GB RAM.

In order to distinguish among the different configurations of PMR that appear on the experiments, the notation used in the next sections is the following.

- Every configuration of PMR is using LAMA-FIRST as the planner P of the algorithm. LAMA-FIRST corresponds to the first search that LAMA performs, using greedy-best-first with unit costs for actions [Richter and Westphal, 2010]. We have also used LAMA-FIRST for the centralized planning step.

- We have used LPG-ADAPT [Fox et al., 2006] and RRPT-PLAN for the plan reuse experiments. When they have been used inside PMR we refer to them as PMR-LPG-ADAPT or PMR-RRPT-PLAN. Otherwise it means they were executed outside PMR. LPG-ADAPT has always been run in the *speed* mode.

- RRPT-PLAN has three additional parameters set on each configuration. We refer to them as p, r and ϵ. The way these parameters affect RRPT-PLAN is explained on Chapter 6.

- Our configurations of PMR have been evaluated using different goal assignment strategies. We refer to them as BC (*Best-cost*), LB (*Load-balance*) and ALL i.e. PMR-LPG-ADAPT-BC means that PMR was executed using *Best-cost* as goal assignment and LPG-ADAPT as plan reuse planner.

- We have tested our PMR configurations against two centralized planners, LAMA-FIRST and YAHSP [Vidal, 2004]. YAHSP was the winner of the Agile track in IPC 2014. We have also run RPT [Alcázar et al., 2011] in Section 7.4 and MADAGASCAR

[1] ipc.icaps-conference.org

[Rintanen, 2014] in Section 7.8, which was the runner-up of the Agile track in IPC 2014.

- Also, we have compared our configurations with three multi-agent planners (CMAP-T [Borrajo and Fernández, 2018], ADP-L [Crosby, 2015] and SIW [Muise et al., 2015]). They will be later explained in Section 7.5. They were chosen because of their remarkable performance on some metrics of CoDMAP.

7.3 Results of PMR-RRPT-PLAN and PMR-LPG-ADAPT when solving different plan-reuse scenarios

The aim of this section is to analyze in detail the behavior of RRPT-PLAN. A simple multi-agent scenario was designed to force PMR to use its plan reuse planner to fix the plan (either RRPT-PLAN or LPG-ADAPT on these experiments). This experiment serves as a proof-of-concept to show why a mixed search-reuse planner like RRPT-PLAN is useful in the kind of scenarios described below. The Load-Balance (LB) strategy was chosen for this experiment in order to assign goals using as many agents as possible and balancing the number of assigned goals per agent at the same time.

The designed scenario contains robots (agents), hammers, nails and paintings. The robot needs first to find and grab a hammer and a nail. It is not allowed for robots to grab more than one hammer and nail at the same time. Once the robot has grabbed the pair hammer, nail it should go to some room that contains a painting and hang it up. Each scenario is completely solved when all paintings are hanged up.

The ideal multi-agent situation (reflected on Figure 7.1a) has as many robots, hammers and nails as paintings. Thus, each robot will be in charge on hanging one painting up when goals are divided among the set of agents. However, the bottleneck of the problem is the number of hammers; if the robots need to share the hammer(s), many interactions arise and need to be fixed by a plan reuse planner. As agents (robots) plan individually on PMR's first step, all of them are forced to use a hammer on their individual plans.

Through these experiments we are able to explain the potential of our plan reuse contribution, RRPT-PLAN, when used inside PMR. We compare PMR-RRPT-PLAN configuration results with the ones obtained with PMR-LPG-ADAPT. Also we show a comparison with PMR-LAMA at the end in order to show how PMR-RRPT-PLAN performs better in these situations as it is in the middle of two extremes: PMR-LPG-ADAPT (plan-reuse) and

PMR-LAMA (centralized planning).

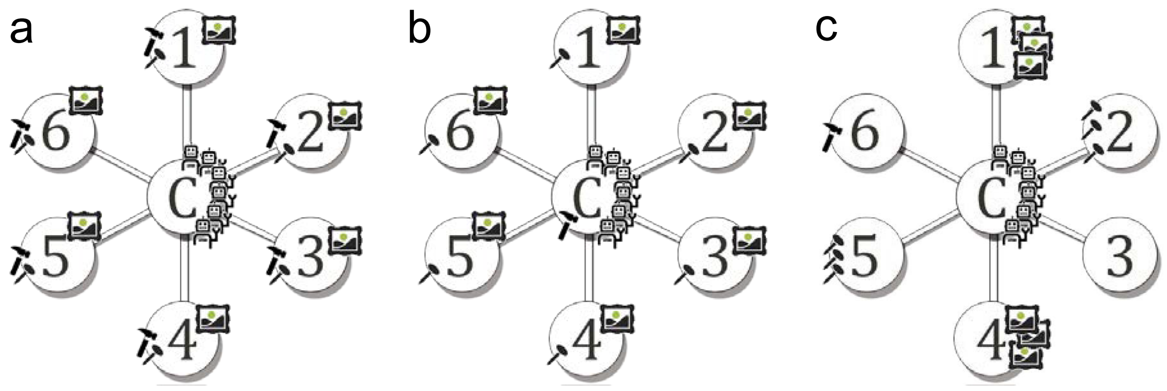


Figure 7.1: (a) presents the first scenario (6 robots, 6 nails, 6 hammers, 6 paintings). This problem is an ideal multi-agent planning scenario. There are as many hammers as robots. Thus, they do not need to share any resource during planning. In the second scenario (b) there is only one hammer for six robots. Paintings and nails are placed as in case (a). Interactions arise when merging the robots' plans as they all use the hammer in their individual plans. The third scenario (c) not only shows hammer interaction issues, but also interactions caused by nails. Nails are grouped into two rooms. Thus, multiple robots could have used the same nail in their individual plans.

Next, the three designed scenarios are described as well as the behaviour of PMR-LPG-ADAPT and PMR-RRPT-PLAN on each of them. Resulting plans from both planners per scenario can be found in Annex B. The first scenario presents six robots starting in a common room, C, plus a hammer and a nail per room (Figure 7.1a). When computing this ideal scenario for MAP with either PMR-RRPT-PLAN or PMR-LPG-ADAPT using the LB strategy, the planner will assign to each robot one of the goals. Thus, each robot will move to a different room, pick up the hammer and the nail and hang the painting on the wall. As a result, after the individual planning step, the concatenation and parallelization of the resulting plan will return a valid plan. There is no need for replanning in this first case as there is no interaction among agents.

The second scenario is the same as the previous one but the difference lies in the number of hammers: it has just one hammer in C (Figure 7.1b). This reflects a common MAP issue on how agents deal with a shared resource (hammer) and how the planner deals with agents' interactions.

Each robot will plan individually to take first the hammer from C and then move to some room to grab the nail and hang the painting up. As a result, the concatenation of plans is not entirely valid. Some actions need to be included into the plan such as dropping the hammer so that the next robot can pick it up later, if necessary. This issue is fixed on the plan reuse step of both PMR-LPG-ADAPT and PMR-RRPT-PLAN but following different approaches. The LPG-ADAPT version takes as input plan the non-valid concatenation of plans and adds the following actions between each robot's set of actions: *move robot to C and drop hammer*. Thus, from the planning point of view, the actions from the input invalid plan were easily reused, as only some new actions were added to the final plan to share the hammer. On the other hand, RRPT-PLAN directly reuses the set of robot1's actions (first agent on the list). Then, when it is time to reuse actions from robot2, the process fails because the hammer is not in C anymore. Thus, RRPT-PLAN changes to the search phase and decides to finish the plan using only robot1. As a result, PMR-RRPT-PLAN returns a plan of length 24 while PMR-LPG-ADAPT returns a plan of length 34. As there is only one hammer available, agents cannot solve their goals in parallel. Therefore, the makespan metric on both configurations has the same value as plan length.

Table 7.1: Besides the three problems previously generated to explain three different cases of plan-reuse, a set of five problems were generated based on Figure 7.1c's distribution to show the impact of increasing the number of paintings and hammers while having the same number of rooms and agents. The main difference between a and b versions is where the hammers are placed: (i) hammers are equally divided in the set of rooms 1, 4, 6; (ii) all hammers are placed in room 6. Paintings are equally divided between rooms 1 and 4 as in Figure 7.1c. Nails are also grouped into two rooms (rooms 2 and 5).

	Agents	**Paintings**	**Hammers**	**Rooms**	**Hammers' Distribution**
Fig 7.1a	6	6	6	7	1 per room
Fig 7.1b	6	6	1	7	Same room
Fig 7.1c	6	6	1	7	Same room
Prob 1a	12	12	2	7	Room 1, 4
Prob 1b	12	12	2	7	Room 6
Prob 2a	12	24	3	7	Rooms 1, 4, 6
Prob 2b	12	24	3	7	Room 6
Prob 3a	12	36	4	7	Room 1, 4, 6
Prob 3b	12	36	4	7	Room 6
Prob 4a	12	48	5	7	Rooms 1, 4, 6
Prob 4b	12	48	5	7	Room 6
Prob 5	36	48	4	7	Rooms 1, 3, 4, 6

We can see in this example that calling a pure plan-reuse planner to fix a plan might not always be a good choice. It will always either reuse as many actions as possible from the input invalid plan, or, when this is not possible, it generates new ones to be later reused. This issue can generate noise on the plan's actions, later explained on the third scenario. If the aim is to obtain an efficient valid plan fast, plan reuse needs to be combined with search to solve these specific multi-agent scenarios. Also, when a shared resource is limiting the number of agents that can perform actions, the best solution will probably be to involve a number of agents close to the number of available shared resources.

The third case presents six robots starting in C, six nails and paintings equally divided among two rooms and one hammer at Room 6 (Figure 7.1c). After obtaining the concatenated plan and checking that it is not valid, several parts of the plan need to be fixed. Not only the actions related with grabbing and dropping the hammer must be fixed, but also the ones related with looking for nails. In Annex B, Listing B.4 contains the 72 actions that compose the sequential merged plan.

As nails are grouped into two different rooms that have the same distance to C and robots plan individually, the robots pick up the hammer and the same nail on its individual plan. Thus, to transform these sequences into a valid plan, robots only need to look for the nails that are still available and the hammer. PMR-LPG-ADAPT reuses as many actions as possible from the invalid plan. It also decides to use the set of six robots to hang up all the paintings (as suggested in the input invalid plan). As a result, the final plan has redundant actions. PMR-LPG-ADAPT adds redundancy when reusing actions that are always valid regardless of the current state of the problem. They are supposed to help each agent to reach its goal; e.g (1) a robot moves to a room, (2) a robot grabs a nail, and (3) a robot drops a nail. These situations are given quite often while the set of six robots is being forced to look for the same hammer in order to hang up the assigned painting. Some of them even grab and drop several nails until they finally can hang the painting up. For instance, it might happen once a robot grabs a nail, as heuristics consider it is one step closer towards the goal. However, the robot might enter into some room where there is another robot that already grabs the hammer, but it does not grab any nail. Usually, the first robot drops the nail when there are no other available nails in the room. As a consequence, the other robot can grab it and hangs the painting up. From the point of view of planning, even though there were parts of the input plan that could be reused, they only cause PMR-LPG-ADAPT to return a worse plan (more actions).

On the other hand, when using the PMR-RRPT-PLAN approach, the set of actions performed by robot1 will be directly reused until it finishes hanging the first painting. When moving robot2 to room6 and realizing that the hammer is not there, the plan-reuse phase ends and the search phase starts, discarding the rest of the actions from the input plan. As a result, again only robot1 is used to hang all the paintings up and redundant actions are avoided except for the one action from robot2 moving to room6. Even though only one robot is used to execute the whole plan, the plan length is 43. That value is still lower than the one from PMR-LPG-ADAPT, which returns a plan of length 159. Regarding makespan, PMR-LPG-ADAPT returns 118 and PMR-RRPT-PLAN, 42.

Table 7.2: This table shows the plan length (L) and makespan (M) obtained for every scenario of the Hammer domain with multi-agent planners PMR-RRPT, PMR-LPG-AD, PMR-LAMA, CMAP-T, ADP-L, SIW and centralized planners LAMA, YAHSP and RPT. Results of YAHSP were not included as it was not able to solve any problem. Makespan values of ADP-L and SIW were obtained after paralellizing their resulting plans with our parallelization algorithm.

	PMR-RRPT		PMR-LPG-AD		PMR-LAMA		CMAP-T		ADP-L		SIW		LAMA		RPT	
	L	**M**	**L**	**M**	**L**	**M**	**L**	**M**	**L**	**M**	**L**	**M**	**L**	**M**	**L**	**M**
7.1a	24	3	24	3	24	3	24	3	24	23	24	3	24	3	24	3
7.1b	24	24	34	34	24	24	24	24	24	24	24	24	24	24	24	24
7.1c	43	42	159	133	43	38	43	38	38	38	34	32	43	38	43	38
1a	76	46	122	119	85	74	-	-	74	74	96	49	-	-	85	76
1b	89	84	121	85	-	-	-	-	74	74	96	49	-	-	79	32
2a	148	82	235	235	157	146	157	146	146	146	134	91	157	146	157	146
2b	161	150	-	-	-	-	-	-	146	146	95	68	-	-	157	146
3a	221	118	380	371	229	218	229	218	218	218	184	45	229	218	229	218
3b	233	223	-	-	-	-	-	-	218	218	180	74	-	-	241	111
4a	294	154	497	491	301	290	301	290	-	-	237	30	301	290	303	292
4b	299	163	-	-	-	-	-	-	290	290	226	51	-	-	280	131
5	297	140	-	-	-	-	-	-	-	-	211	52	-	-	-	-

Table 7.2 and Table 7.3 show the plan length, makespan and time obtained for each PMR configuration, other multi-agent and centralized planners on each Hammer problem. The first three rows are related to the three cases explained above. The following rows represent new problem-variations generated from the Figure 7.1c scenario where paintings and nails are equally divided into two sets of different rooms and hammers' location vary. The aim is to show the evolution of plan length, makespan and time when increasing number of paintings and hammers. Due to this fact, the complexity of the problem increases depending on where the hammers are initially placed and the number of interactions that have arisen.

Table 7.3: Time in seconds that each configuration of PMR, CMAP-T, ADP-L, SIW, LAMA and RPT took to solve the Hammer domain scenarios. YAHSP results are not shown as it was not able to solve any problem. Time limit per problem was 1800s. When (-) appears on a cell means that the problem was not solved by the planner on time. Problems solved in less than 1 second are all considered as 1.00s.

	PMR-RRPT	PMR-LPG-AD	PMR-LAMA	CMAP-T	ADP-L	SIW	LAMA	RPT
7.1a	1.80	2.25	1.79	1.00	1.00	1.00	1.00	1.00
7.1b	1.00	1.66	1.64	1.00	1.00	1.00	1.00	1.00
7.1c	1.08	1.63	1.12	1.00	1.00	1.00	1.00	1.00
1a	43.84	4.40	40.05	2.83	1.24	2.48	2.07	2.67
1b	4.01	39.20	-	-	1.23	4.54	-	3.91
2a	107.15	16.81	115.97	13.16	6.47	14.18	10.55	16.71
2b	14.37	-	-	-	6.37	14.27	-	22.11
3a	159.07	43.89	140.42	48.30	22.43	43.00	39.02	91.75
3b	59.81	-	-	-	30.69	65.66	-	110.62
4a	173.96	90.48	182.78	136.98	-	94.81	110.20	329.29
4b	126.81	-	-	-	77.31	103.73	-	329.18
5	659.55	-	-	-	-	1447.2	-	

As Table 7.2 shows, PMR-LAMA and PMR-LPG-ADAPT are not able to solve most of the problems where all hammers are placed on the same room (version *b* problems). PMR-LAMA gets lost on the search space as there is a huge number of possible combinations of movements with the same heuristic estimation (known as plateaux) to solve the problem. On the other hand, PMR-LPG-ADAPT gets lost when reusing the actions of the invalid plan, as most of them are valid, but they still do not solve the problem. This ends up generating redundancy and causes the planner to search for new actions, while always looking up at the ones on the invalid plan again on each iteration. However, PMR-RRPT-PLAN performs better than the other two configurations, as it has the opportunity to change between plan reuse, search and sampling to solve the problem and this has an impact on the number of problems solved as well as on the makespan metric. PMR-RRPT-PLAN is more flexible and in summary obtains the best performance on the three metrics regarding the three PMR configurations. Regarding multi-agent planners CMAP-T, ADP-L and SIW, the best configuration is SIW. SIW's serialization of goals allows the planner to use efficiently most of the agents, resulting in the improvement of makespan at the same time. ADP-L and CMAP-T behave very similar to LAMA, the centralized planner. These planners (except for SIW) try to solve the problem with the smallest possible number of agents, as a consequence of minimizing the plan length. ADP-L is the only one able to avoid the plateaux of the search space and does not get lost on version *b* problems. The centralized planner YAHSP was run as well, but it was not able to solve any of the problems.

When hammers are distributed among rooms 1, 4 and 6 (version "a" problems) the interactions are easier to solve by all planners. On PMR configurations, when agents plan individually, they choose the hammer that is closer to the set of paintings they have to hang up. Thus, less number of interactions arises and agents are better self-organized. The same effect is given on the rest of the planners.

Regarding times on Table 7.3, PMR-LPG-ADAPT is faster when it is capable of solving the problems but PMR-RRPT-PLAN is more regular on performance.

7.4 Analyzing the impact on performance of RRPT-PLAN's parameters

After explaining the difference between the performance of RRPT-PLAN and a classical plan reuse planner, such as LPG-ADAPT, we wanted to test both of them outside of PMR against a centralized planner, LAMA-FIRST. This experiment is divided into two parts. Firstly, we present an analysis of RRPT-PLAN's performance depending on a set of values assigned to its parameters p,r and ϵ. The aim is to find the best parameter configuration for RRPT-PLAN. Secondly, after choosing the best configuration for RRPT-PLAN, we compare the results obtained in coverage, quality and time against the ones obtained by LPG-ADAPT and LAMA-FIRST. Both parts of the experiment share the same benchmark, which was created as follows: first, a set of hard planning problems was generated (Rovers, Zenotravel, Driverlog, Depots, Elevators and Logistics) per domain. These domains were chosen because they have different levels of interaction and dependency among the different elements of the domain, which is a feature that directly affects the difficulty of reusing a previous plan. Any agent's decision from Rovers or Zenotravel seldom interfere the decisions taken by the other agents. There are not common resources either. These domains are commonly called loosely-coupled domains. This is the easiest scenario to be solved by a plan reuse planner, as interactions, if any, are easy to solve. However, Driverlog and Depots' agents share partially the domain resources, as most of them need to be delivered to some concrete places. On Elevators and Logistics the level of interaction is similar but dependency increases, as the problem goals usually need collaboration among two agents besides dealing with the sharing of common resources. These domains are commonly called tightly-coupled domains.

We made three versions of each problem; the first one contains one more goal than the original problem; the second contains five more and ten more goals were added to the third. For each domain we took three original problems per domain so we created nine new problems based on the originals.

The original problems were first run with LAMA-FIRST in order to obtain the resulting plans. These plans were later used as input plans for RRPT-PLAN, ERRT-PLAN and LPG-ADAPT for each one of the modified problems. As the number of added goals gets increased, the resulting plans should be very similar at the beginning but more different as more goals are added to the original version. Also, as LAMA-FIRST and RPT are not able to reuse, they had to run each modified problem from scratch.

As it was mentioned in Chapter 6, RRPT-PLAN has three parameters that change the behaviour of the algorithm. Parameters p and r control the probability of running local search, plan reuse or sampling, e.g. values $p = 0.6, r = 0.3$ cause RRPT-PLAN to have a probability of 0.6 to run the local search phase, 0.3 to run plan-reuse and 0.1 to run the sampling one. In addition, the parameter ϵ limits the number of expanded nodes per iteration during local search, e.g a value of $\epsilon = 1000$ means that 1000 nodes will be expanded at most per local search iteration.

We have tested eight different configurations of RRPT-PLAN in this experiment explained as follows.

1. $p = 0.3$, $r = 0.3$; this configuration gives equal probability to search and reuse and 0.4 to sampling.

2. $p = 0.3$, $r = 0.6$; this configuration benefits plan-reuse over search and leaves a probability of 0.1 to sampling.

3. $p = 0.3$, $r = 0.7$; same to the previous one but RRPT-PLAN will not perform the sampling phase.

4. $p = 0.6$, $r = 0.3$; this configuration benefits local search over plan-reuse and leaves a probability of 0.1 to sampling.

Each of these parameter configurations was tested for $\epsilon = 1000$ and $\epsilon = 10000$ to analyze the impact on the solution when allowing a smaller or greater number of nodes to be expanded during search. For this experiment, the execution of plan reuse on the first iteration of RRPT-PLAN is avoided in order to focus on the impact of these probabilities. We also show the comparison against LAMA-FIRST and LPG-ADAPT, ERRT-PLAN and RPT. Results of ERRT-PLAN are not shown on the tables neither in the heatmaps below, as it was not able to solve any of the problems in 1800 seconds. ERRT-PLAN uses Enforced Hill Climbing (EHC) as the search method [Hoffmann and Nebel, 2001]. Since EHC performs a form of hill climbing without backtracking, the heuristic can guide the search towards dead-end states, big plateaux, or very long paths. Therefore, ERRT-PLAN could not solve any problem.

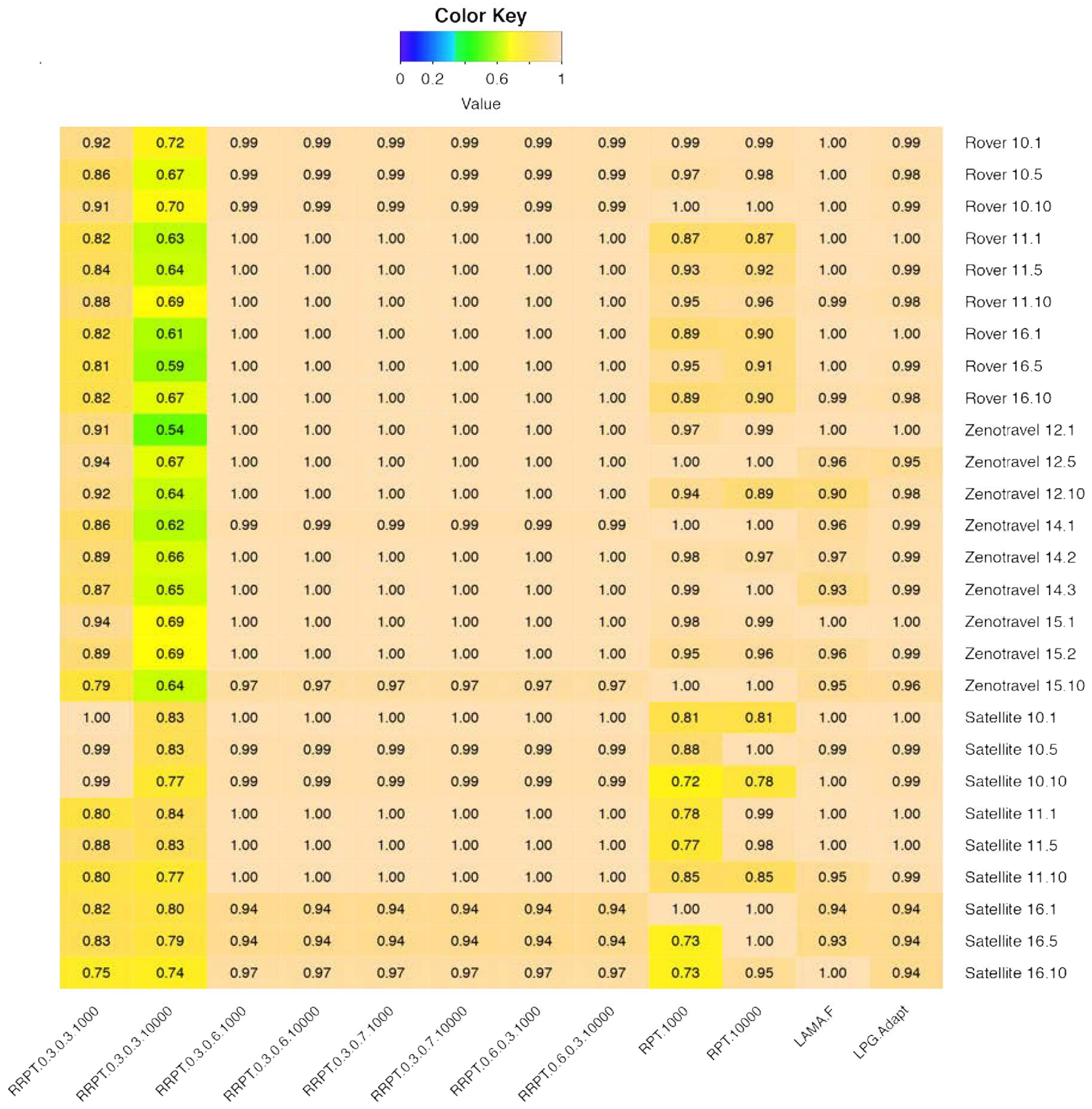
First, we show the individual results obtained by the eight different configurations of RRPT-PLAN plus the individual results of RPT, LAMA-FIRST and LPG-ADAPT depicted in four heatmaps. Each one corresponds to either domains with low level of interaction/dependency or to the ones in which the solution plan is more difficult to reuse. Heatmap values are ranged between 0 and 1, being 0 the worst (blue) and 1 the best (red) value. Figure 7.2 shows quality scores for Rovers, Zenotravel and Satellite domains, respectively. Figure 7.3 shows quality scores as well but for Driverlog, Elevators, Logistics and Depots. Figures 7.4 and 7.5 have the same division of domains but show time scores instead.

By observing the four heatmaps in both quality and time scores, the worst configuration is $p = 0.3$, $r = 0.3$ independently of the value of ϵ. Giving equal probability to each RRPT-PLAN phase, and specially assigning to sampling a medium-high probability, causes the planning process to explore many different paths over the search space. Thus, it ends up obtaining a bad solution and usually wastes more time than the other RRPT-PLAN configurations. Another general conclusion that can be extracted is that the limit number of expanded nodes (ϵ) does not have a big impact in efficiency; $\epsilon = 1000$ configurations usually perform slightly better.

Results are also shown on Tables 7.4, 7.5 and 7.6. Table 7.4 shows the number of problems solved per configuration. Table 7.5 shows the summary of the obtained quality score per configuration. Finally Table 7.6 shows the summary of the time scores per configuration.

Table 7.4 shows that coverage is very similar between RRPT-PLAN and LPG-ADAPT. LAMA-FIRST and RPT perform a bit worse, especially on domains where the interaction is higher (e.g. Depots, Logistics, Driverlog). Neither of them is able to employ plan reuse. Thus, they had to run each modified problem from scratch and those domains are harder to solve. RRPT-PLAN ($p = 0.6$, $r = 0.3$) is the best configuration followed by RRPT-PLAN ($p = 0.3$, $r = 0.6$).

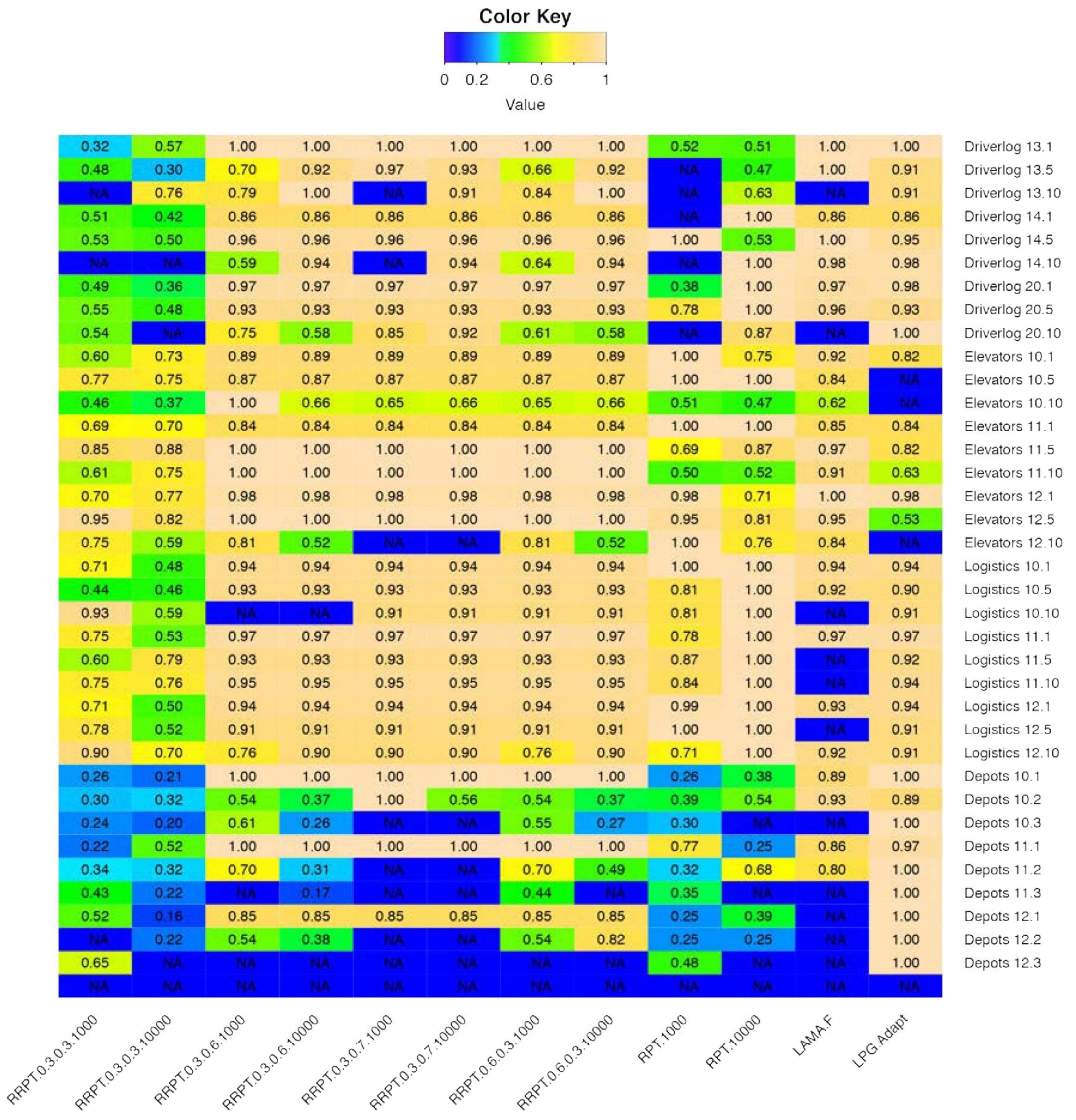
Quality results show that RRPT-PLAN and LPG-ADAPT results are similar. The difference between them can be found on the results from the Depots domain, where LPG-ADAPT performs slightly better (Figure 7.3). The best configuration is LPG-ADAPT followed closely by RRPT-PLAN ($p = 0.6$, $r = 0.3$) and ($p = 0.3$, $r = 0.6$).



	RRPT.0.3.0.3.1000	RRPT.0.3.0.3.10000	RRPT.0.3.0.6.1000	RRPT.0.3.0.6.10000	RRPT.0.3.0.7.1000	RRPT.0.3.0.7.10000	RRPT.0.6.0.3.1000	RRPT.0.6.0.3.10000	RPT.1000	RPT.10000	LAMA.F	LPG.Adapt
Rover 10.1	0.92	0.72	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1.00	0.99
Rover 10.5	0.86	0.67	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.98	1.00	0.98
Rover 10.10	0.91	0.70	0.99	0.99	0.99	0.99	0.99	0.99	1.00	1.00	1.00	0.99
Rover 11.1	0.82	0.63	1.00	1.00	1.00	1.00	1.00	1.00	0.87	0.87	1.00	1.00
Rover 11.5	0.84	0.64	1.00	1.00	1.00	1.00	1.00	1.00	0.93	0.92	1.00	0.99
Rover 11.10	0.88	0.69	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.96	0.99	0.98
Rover 16.1	0.82	0.61	1.00	1.00	1.00	1.00	1.00	1.00	0.89	0.90	1.00	1.00
Rover 16.5	0.81	0.59	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.91	1.00	0.99
Rover 16.10	0.82	0.67	1.00	1.00	1.00	1.00	1.00	1.00	0.89	0.90	0.99	0.98
Zenotravel 12.1	0.91	0.54	1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.99	1.00	1.00
Zenotravel 12.5	0.94	0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96	0.95
Zenotravel 12.10	0.92	0.64	1.00	1.00	1.00	1.00	1.00	1.00	0.94	0.89	0.90	0.98
Zenotravel 14.1	0.86	0.62	0.99	0.99	0.99	0.99	0.99	0.99	1.00	1.00	0.96	0.99
Zenotravel 14.2	0.89	0.66	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.97	0.97	0.99
Zenotravel 14.3	0.87	0.65	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	0.93	0.99
Zenotravel 15.1	0.94	0.69	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.99	1.00	1.00
Zenotravel 15.2	0.89	0.69	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.96	0.96	0.99
Zenotravel 15.10	0.79	0.64	0.97	0.97	0.97	0.97	0.97	0.97	1.00	1.00	0.95	0.96
Satellite 10.1	1.00	0.83	1.00	1.00	1.00	1.00	1.00	1.00	0.81	0.81	1.00	1.00
Satellite 10.5	0.99	0.83	0.99	0.99	0.99	0.99	0.99	0.99	0.88	1.00	0.99	0.99
Satellite 10.10	0.99	0.77	0.99	0.99	0.99	0.99	0.99	0.99	0.72	0.78	1.00	0.99
Satellite 11.1	0.80	0.84	1.00	1.00	1.00	1.00	1.00	1.00	0.78	0.99	1.00	1.00
Satellite 11.5	0.88	0.83	1.00	1.00	1.00	1.00	1.00	1.00	0.77	0.98	1.00	1.00
Satellite 11.10	0.80	0.77	1.00	1.00	1.00	1.00	1.00	1.00	0.85	0.85	0.95	0.99
Satellite 16.1	0.82	0.80	0.94	0.94	0.94	0.94	0.94	0.94	1.00	1.00	0.94	0.94
Satellite 16.5	0.83	0.79	0.94	0.94	0.94	0.94	0.94	0.94	0.73	1.00	0.93	0.94
Satellite 16.10	0.75	0.74	0.97	0.97	0.97	0.97	0.97	0.97	0.73	0.95	1.00	0.94

Figure 7.2: Quality per problem and configuration on loosely coupled domains (Rovers, Zenotravel, Satellite). The original problem was added one, five and ten goals respectively. The plan that returned the original problem was sent to RRPT-PLAN and LPG-ADAPT as input for plan reuse.

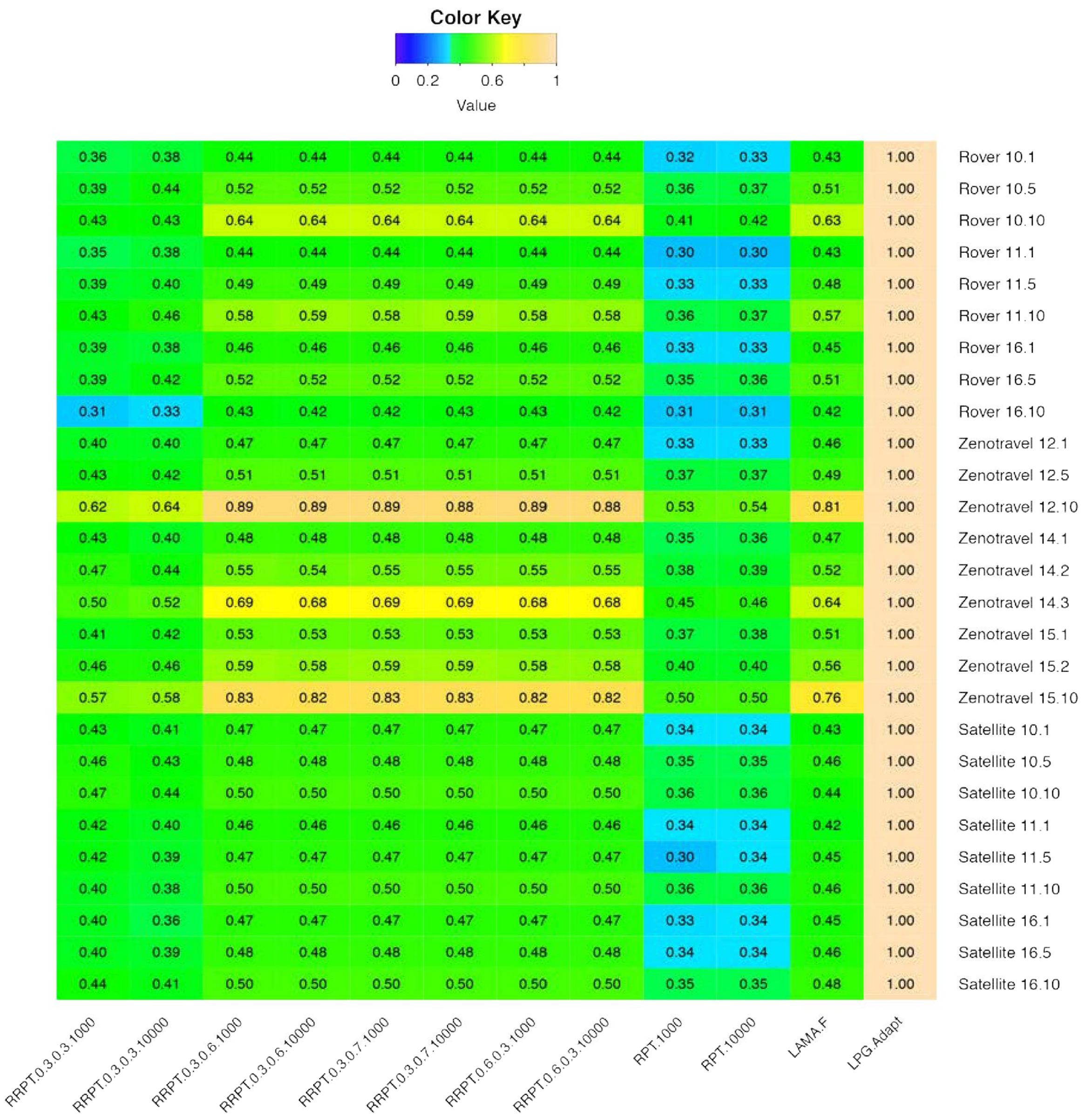
On the other hand, regarding time, the fastest configuration is LPG-ADAPT. If we considered only planning time, RRPT-PLAN results would be similar to those of LPG-ADAPT. Before starting to plan, our planner, first translates the domain and the problem and computes mutexes and disambiguation (which is useful for search and sampling). Thus, our planner's performance is worse than LPG-ADAPT in time (not in quality).



RRPT.0.3.0.3.1000	RRPT.0.3.0.3.10000	RRPT.0.3.0.6.1000	RRPT.0.3.0.6.10000	RRPT.0.3.0.7.1000	RRPT.0.3.0.7.10000	RRPT.0.6.0.3.1000	RRPT.0.6.0.3.10000	RPT.1000	RPT.10000	LAMA.F	LPG.Adapt	
0.32	0.57	1.00	1.00	1.00	1.00	1.00	1.00	0.52	0.51	1.00	1.00	Driverlog 13.1
0.48	0.30	0.70	0.92	0.97	0.93	0.66	0.92	NA	0.47	1.00	0.91	Driverlog 13.5
NA	0.76	0.79	1.00	NA	0.91	0.84	1.00	NA	0.63	NA	0.91	Driverlog 13.10
0.51	0.42	0.86	0.86	0.86	0.86	0.86	0.86	NA	1.00	0.86	0.86	Driverlog 14.1
0.53	0.50	0.96	0.96	0.96	0.96	0.96	0.96	1.00	0.53	1.00	0.95	Driverlog 14.5
NA	NA	0.59	0.94	NA	0.94	0.64	0.94	NA	1.00	0.98	0.98	Driverlog 14.10
0.49	0.36	0.97	0.97	0.97	0.97	0.97	0.97	0.38	1.00	0.97	0.98	Driverlog 20.1
0.55	0.48	0.93	0.93	0.93	0.93	0.93	0.93	0.78	1.00	0.96	0.93	Driverlog 20.5
0.54	NA	0.75	0.58	0.85	0.92	0.61	0.58	NA	0.87	NA	1.00	Driverlog 20.10
0.60	0.73	0.89	0.89	0.89	0.89	0.89	0.89	1.00	0.75	0.92	0.82	Elevators 10.1
0.77	0.75	0.87	0.87	0.87	0.87	0.87	0.87	1.00	1.00	0.84	NA	Elevators 10.5
0.46	0.37	1.00	0.66	0.65	0.66	0.65	0.66	0.51	0.47	0.62	NA	Elevators 10.10
0.69	0.70	0.84	0.84	0.84	0.84	0.84	0.84	1.00	1.00	0.85	0.84	Elevators 11.1
0.85	0.88	1.00	1.00	1.00	1.00	1.00	1.00	0.69	0.87	0.97	0.82	Elevators 11.5
0.61	0.75	1.00	1.00	1.00	1.00	1.00	1.00	0.50	0.52	0.91	0.63	Elevators 11.10
0.70	0.77	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.71	1.00	0.98	Elevators 12.1
0.95	0.82	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.81	0.95	0.53	Elevators 12.5
0.75	0.59	0.81	0.52	NA	NA	0.81	0.52	1.00	0.76	0.84	NA	Elevators 12.10
0.71	0.48	0.94	0.94	0.94	0.94	0.94	0.94	1.00	1.00	0.94	0.94	Logistics 10.1
0.44	0.46	0.93	0.93	0.93	0.93	0.93	0.93	0.81	1.00	0.92	0.90	Logistics 10.5
0.93	0.59	NA	NA	0.91	0.91	0.91	0.91	0.81	1.00	NA	0.91	Logistics 10.10
0.75	0.53	0.97	0.97	0.97	0.97	0.97	0.97	0.78	1.00	0.97	0.97	Logistics 11.1
0.60	0.79	0.93	0.93	0.93	0.93	0.93	0.93	0.87	1.00	NA	0.92	Logistics 11.5
0.75	0.76	0.95	0.95	0.95	0.95	0.95	0.95	0.84	1.00	NA	0.94	Logistics 11.10
0.71	0.50	0.94	0.94	0.94	0.94	0.94	0.94	0.99	1.00	0.93	0.94	Logistics 12.1
0.78	0.52	0.91	0.91	0.91	0.91	0.91	0.91	1.00	1.00	NA	0.91	Logistics 12.5
0.90	0.70	0.76	0.90	0.90	0.90	0.76	0.90	0.71	1.00	0.92	0.91	Logistics 12.10
0.26	0.21	1.00	1.00	1.00	1.00	1.00	1.00	0.26	0.38	0.89	1.00	Depots 10.1
0.30	0.32	0.54	0.37	1.00	0.56	0.54	0.37	0.39	0.54	0.93	0.89	Depots 10.2
0.24	0.20	0.61	0.26	NA	NA	0.55	0.27	0.30	NA	NA	1.00	Depots 10.3
0.22	0.52	1.00	1.00	1.00	1.00	1.00	1.00	0.77	0.25	0.86	0.97	Depots 11.1
0.34	0.32	0.70	0.31	NA	NA	0.70	0.49	0.32	0.68	0.80	1.00	Depots 11.2
0.43	0.22	NA	0.17	NA	NA	0.44	NA	0.35	NA	NA	1.00	Depots 11.3
0.52	0.16	0.85	0.85	0.85	0.85	0.85	0.85	0.25	0.39	NA	1.00	Depots 12.1
NA	0.22	0.54	0.38	NA	NA	0.54	0.82	0.25	0.25	NA	1.00	Depots 12.2
0.65	NA	NA	NA	NA	NA	NA	NA	0.48	NA	NA	1.00	Depots 12.3
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	

Figure 7.3: Quality per problem and configuration on tightly coupled domains (Driverlog, Elevators Logistics, Depots). The original problem was added one, five and ten goals respectively. The plan that returned the original problem was sent to RRPT-PLAN and LPG-ADAPT as input for plan reuse.

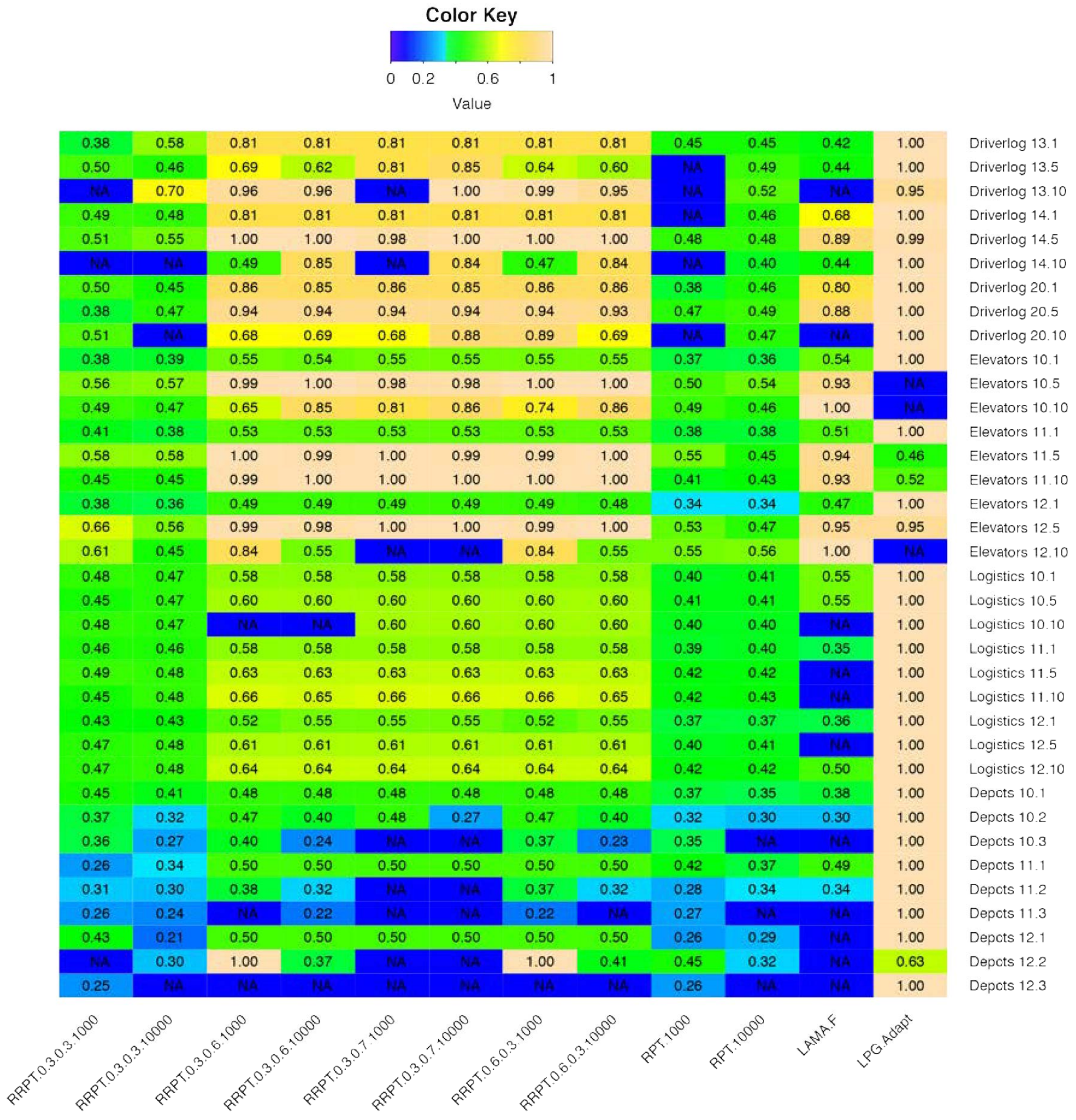
LPG-ADAPT's preprocessing phase only translates the domain and problem. LAMA-FIRST applies the same translation process as RRPT-PLAN but it does not compute mutexes and disambiguation. As it can be seen on Table 7.7, the fastest preprocessing is performed by LPG-ADAPT. Also, the impact on computing mutexes and disambiguation can be seen by comparing RRPT-PLAN and LAMA-FIRST preprocessing results. On the other



	RRPT.0.3.0.3.1000	RRPT.0.3.0.3.10000	RRPT.0.3.0.6.1000	RRPT.0.3.0.6.10000	RRPT.0.3.0.7.1000	RRPT.0.3.0.7.10000	RRPT.0.6.0.3.1000	RRPT.0.6.0.3.10000	RPT.1000	RPT.10000	LAMA.F	LPG.Adapt
Rover 10.1	0.36	0.38	0.44	0.44	0.44	0.44	0.44	0.44	0.32	0.33	0.43	1.00
Rover 10.5	0.39	0.44	0.52	0.52	0.52	0.52	0.52	0.52	0.36	0.37	0.51	1.00
Rover 10.10	0.43	0.43	0.64	0.64	0.64	0.64	0.64	0.64	0.41	0.42	0.63	1.00
Rover 11.1	0.35	0.38	0.44	0.44	0.44	0.44	0.44	0.44	0.30	0.30	0.43	1.00
Rover 11.5	0.39	0.40	0.49	0.49	0.49	0.49	0.49	0.49	0.33	0.33	0.48	1.00
Rover 11.10	0.43	0.46	0.58	0.59	0.58	0.59	0.58	0.58	0.36	0.37	0.57	1.00
Rover 16.1	0.39	0.38	0.46	0.46	0.46	0.46	0.46	0.46	0.33	0.33	0.45	1.00
Rover 16.5	0.39	0.42	0.52	0.52	0.52	0.52	0.52	0.52	0.35	0.36	0.51	1.00
Rover 16.10	0.31	0.33	0.43	0.42	0.42	0.43	0.43	0.42	0.31	0.31	0.42	1.00
Zenotravel 12.1	0.40	0.40	0.47	0.47	0.47	0.47	0.47	0.47	0.33	0.33	0.46	1.00
Zenotravel 12.5	0.43	0.42	0.51	0.51	0.51	0.51	0.51	0.51	0.37	0.37	0.49	1.00
Zenotravel 12.10	0.62	0.64	0.89	0.89	0.89	0.88	0.89	0.88	0.53	0.54	0.81	1.00
Zenotravel 14.1	0.43	0.40	0.48	0.48	0.48	0.48	0.48	0.48	0.35	0.36	0.47	1.00
Zenotravel 14.2	0.47	0.44	0.55	0.54	0.55	0.55	0.55	0.55	0.38	0.39	0.52	1.00
Zenotravel 14.3	0.50	0.52	0.69	0.68	0.69	0.69	0.68	0.68	0.45	0.46	0.64	1.00
Zenotravel 15.1	0.41	0.42	0.53	0.53	0.53	0.53	0.53	0.53	0.37	0.38	0.51	1.00
Zenotravel 15.2	0.46	0.46	0.59	0.58	0.59	0.59	0.58	0.58	0.40	0.40	0.56	1.00
Zenotravel 15.10	0.57	0.58	0.83	0.82	0.83	0.83	0.82	0.82	0.50	0.50	0.76	1.00
Satellite 10.1	0.43	0.41	0.47	0.47	0.47	0.47	0.47	0.47	0.34	0.34	0.43	1.00
Satellite 10.5	0.46	0.43	0.48	0.48	0.48	0.48	0.48	0.48	0.35	0.35	0.46	1.00
Satellite 10.10	0.47	0.44	0.50	0.50	0.50	0.50	0.50	0.50	0.36	0.36	0.44	1.00
Satellite 11.1	0.42	0.40	0.46	0.46	0.46	0.46	0.46	0.46	0.34	0.34	0.42	1.00
Satellite 11.5	0.42	0.39	0.47	0.47	0.47	0.47	0.47	0.47	0.30	0.34	0.45	1.00
Satellite 11.10	0.40	0.38	0.50	0.50	0.50	0.50	0.50	0.50	0.36	0.36	0.46	1.00
Satellite 16.1	0.40	0.36	0.47	0.47	0.47	0.47	0.47	0.47	0.33	0.34	0.45	1.00
Satellite 16.5	0.40	0.39	0.48	0.48	0.48	0.48	0.48	0.48	0.34	0.34	0.46	1.00
Satellite 16.10	0.44	0.41	0.50	0.50	0.50	0.50	0.50	0.50	0.35	0.35	0.48	1.00

Figure 7.4: Time (s) per problem and configuration on loosely coupled domains (Rovers, Zenotravel, Satellite). The original problem was added one, five and ten goals respectively. The plan that returned the original problem was sent to RRPT-PLAN and LPG-ADAPT as input for plan reuse.

hand, even though it is not explicitly reflected on Tables 7.6 and 7.7, the harder the problem, the closer the time score gets RRPT-PLAN to the one of LPG-ADAPT. The reason is the following: when the number of added goals of the problem increases, plan reuse performance decreases, as the input plan is not similar anymore. On the other hand,



	RRPT.0.3.0.3.1000	RRPT.0.3.0.3.10000	RRPT.0.3.0.6.1000	RRPT.0.3.0.6.10000	RRPT.0.3.0.7.1000	RRPT.0.3.0.7.10000	RRPT.0.6.0.3.1000	RRPT.0.6.0.3.10000	RPT.1000	RPT.10000	LAMA.F	LPG.Adapt
Driverlog 13.1	0.38	0.58	0.81	0.81	0.81	0.81	0.81	0.81	0.45	0.45	0.42	1.00
Driverlog 13.5	0.50	0.46	0.69	0.62	0.81	0.85	0.64	0.60	NA	0.49	0.44	1.00
Driverlog 13.10	NA	0.70	0.96	0.96	NA	1.00	0.99	0.95	NA	0.52	NA	0.95
Driverlog 14.1	0.49	0.48	0.81	0.81	0.81	0.81	0.81	0.81	NA	0.46	0.68	1.00
Driverlog 14.5	0.51	0.55	1.00	1.00	0.98	1.00	1.00	1.00	0.48	0.48	0.89	0.99
Driverlog 14.10	NA	NA	0.49	0.85	NA	0.84	0.47	0.84	NA	0.40	0.44	1.00
Driverlog 20.1	0.50	0.45	0.86	0.85	0.86	0.85	0.86	0.86	0.38	0.46	0.80	1.00
Driverlog 20.5	0.38	0.47	0.94	0.94	0.94	0.94	0.94	0.93	0.47	0.49	0.88	1.00
Driverlog 20.10	0.51	NA	0.68	0.69	0.68	0.88	0.89	0.69	NA	0.47	NA	1.00
Elevators 10.1	0.38	0.39	0.55	0.54	0.55	0.55	0.55	0.55	0.37	0.36	0.54	1.00
Elevators 10.5	0.56	0.57	0.99	1.00	0.98	0.98	1.00	1.00	0.50	0.54	0.93	NA
Elevators 10.10	0.49	0.47	0.65	0.85	0.81	0.86	0.74	0.86	0.49	0.46	1.00	NA
Elevators 11.1	0.41	0.38	0.53	0.53	0.53	0.53	0.53	0.53	0.38	0.38	0.51	1.00
Elevators 11.5	0.58	0.58	1.00	0.99	1.00	0.99	0.99	1.00	0.55	0.45	0.94	0.46
Elevators 11.10	0.45	0.45	0.99	1.00	1.00	1.00	1.00	1.00	0.41	0.43	0.93	0.52
Elevators 12.1	0.38	0.36	0.49	0.49	0.49	0.49	0.49	0.48	0.34	0.34	0.47	1.00
Elevators 12.5	0.66	0.56	0.99	0.98	1.00	1.00	0.99	1.00	0.53	0.47	0.95	0.95
Elevators 12.10	0.61	0.45	0.84	0.55	NA	NA	0.84	0.55	0.55	0.56	1.00	NA
Logistics 10.1	0.48	0.47	0.58	0.58	0.58	0.58	0.58	0.58	0.40	0.41	0.55	1.00
Logistics 10.5	0.45	0.47	0.60	0.60	0.60	0.60	0.60	0.60	0.41	0.41	0.55	1.00
Logistics 10.10	0.48	0.47	NA	NA	0.60	0.60	0.60	0.60	0.40	0.40	NA	1.00
Logistics 11.1	0.46	0.46	0.58	0.58	0.58	0.58	0.58	0.58	0.39	0.40	0.35	1.00
Logistics 11.5	0.49	0.48	0.63	0.63	0.63	0.63	0.63	0.63	0.42	0.42	NA	1.00
Logistics 11.10	0.45	0.48	0.66	0.65	0.66	0.66	0.66	0.65	0.42	0.43	NA	1.00
Logistics 12.1	0.43	0.43	0.52	0.55	0.55	0.55	0.52	0.55	0.37	0.37	0.36	1.00
Logistics 12.5	0.47	0.48	0.61	0.61	0.61	0.61	0.61	0.61	0.40	0.41	NA	1.00
Logistics 12.10	0.47	0.48	0.64	0.64	0.64	0.64	0.64	0.64	0.42	0.42	0.50	1.00
Depots 10.1	0.45	0.41	0.48	0.48	0.48	0.48	0.48	0.48	0.37	0.35	0.38	1.00
Depots 10.2	0.37	0.32	0.47	0.40	0.48	0.27	0.47	0.40	0.32	0.30	0.30	1.00
Depots 10.3	0.36	0.27	0.40	0.24	NA	NA	0.37	0.23	0.35	NA	NA	1.00
Depots 11.1	0.26	0.34	0.50	0.50	0.50	0.50	0.50	0.50	0.42	0.37	0.49	1.00
Depots 11.2	0.31	0.30	0.38	0.32	NA	NA	0.37	0.32	0.28	0.34	0.34	1.00
Depots 11.3	0.26	0.24	NA	0.22	NA	NA	0.22	NA	0.27	NA	NA	1.00
Depots 12.1	0.43	0.21	0.50	0.50	0.50	0.50	0.50	0.50	0.26	0.29	NA	1.00
Depots 12.2	NA	0.30	1.00	0.37	NA	NA	1.00	0.41	0.45	0.32	NA	0.63
Depots 12.3	0.25	NA	NA	NA	NA	NA	NA	NA	0.26	NA	NA	1.00

Figure 7.5: Time(s) per problem and configuration on tightly coupled domains (Driverlog, Elevators Logistics, Depots). The original problem was added one, five and ten goals respectively. The plan that returned the original problem was sent to RRPT-PLAN and LPG-ADAPT as input for plan reuse.

search becomes more useful. Results from Table 7.6 show the poor performance of a centralized planner (LAMA-FIRST) in comparison with those that employ plan-reuse. When the problems are similar, reusing input plans is faster than planning from scratch. Table 7.7 shows the average time employed on preprocessing by each planner on each domain.

Table 7.4: Coverage score obtained on each domain. Planners: RRPT-PLAN with eight different configurations, RPT (centralized), LAMA-FIRST (centralized) and LPG-ADAPT (plan reuse). The columns represent the values obtained using the set of probabilities (p,r) and ϵ limit of expanded nodes. The 'Total' row refers to the addition of the scores of each configuration. Nine problems per domain were run.

Problem	**RRPT-PLAN**					**RPT**	**LAMA-FIRST**	**LPG-ADAPT**
	ϵ	**0.3 0.3**	**0.3 0.6**	**0.3 0.7**	**0.6 0.3**			
Elevators	**1000**	9	9	8	9	9	9	6
	10000	9	9	8	9	9		
Logistics	**1000**	9	8	9	9	9	5	9
	10000	9	8	9	9	9		
Depots	**1000**	8	7	4	8	9	4	9
	10000	8	8	4	7	6		
Zenotravel	**1000**	9	9	9	9	9	9	9
	10000	9	9	9	9	9		
Rovers	**1000**	9	9	9	9	9	9	9
	10000	9	9	9	9	9		
Satellite	**1000**	9	9	9	9	9	9	9
	10000	9	9	9	9	9		
Driverlog	**1000**	7	9	7	9	4	7	9
	10000	7	9	9	9	9		
Total	**1000**	60	60	55	**62**	58	52	60
	10000	60	61	57	61	60		

Table 7.5: Quality score obtained on each domain. Planners: RRPT-PLAN with eight different configurations, RPT (centralized), LAMA-FIRST (centralized) and LPG-ADAPT (plan reuse). The 'Total' row refers to the addition of the scores of each configuration.

Problem	**RRPT-PLAN**					**RPT**	**LAMA-FIRST**	**LPG-ADAPT**
	ϵ	**0.3 0.3**	**0.3 0.6**	**0.3 0.7**	**0.6 0.3**			
Total	**1000**	42.88	55.26	52.75	56.11	45.28	49.26	**56.85**
	10000	36.16	54.49	54.19	55.84	51.73		

In order to choose the best configuration of RRPT-PLAN to later compare itself against other planners in the following experiments, we decided to choose ($p = 0.3$, $r = 0.6$, $\epsilon = 1000$). There was a minimal difference regarding the $p = 0.6$, $r = 0.3$ configuration. We realized that it was due to the execution of the plan reuse phase on the first iteration on both configurations. The stochasticity of RRPT-PLAN on this experiment turned out to discover the following: in order to stabilize the performance and commitment of RRPT-PLAN inside and outside of PMR, it is essential to always first execute plan reuse on the first iteration. This is also why we chose a higher probability on r (0.6 instead of 0.3).

Table 7.6: Time score obtained on each domain. Planners: RRPT-PLAN with eight different configurations, RPT (centralized), LAMA-FIRST (centralized) and LPG-ADAPT (plan reuse). The 'Total' row refers to the addition of the scores of each configuration.

Problem	**RRPT-PLAN**					**RPT**	**LAMA-FIRST**	**LPG-ADAPT**
	ϵ	**0.3 0.3**	**0.3 0.6**	**0.3 0.7**	**0.6 0.3**			
Total	**1000**	26.25	37.18	34.02	38.22	22.33	29.33	**58.50**
	10000	26.01	36.71	35.93	37.12	23.83		

Table 7.7: Average time (in seconds) spent on preprocessing each problem per planner and domain.

Problem	**RRPT-PLAN**	**RPT**	**LAMA-FIRST**	**LPG-ADAPT**
Elevators	38.22	217.41	42.12	2.96
Logistics	107.93	667.03	121.82	14.43
Depots	3.38	6.51	3.43	0.2
Zenotravel	86.21	484.21	102.49	4.01
Rovers	63.68	463.38	70.14	2.2
Satellite	23.81	103.56	32.33	1.2
Driverlog	18.58	173.05	24.09	1.59

7.5 Results in CoDMAP problems

In this Section the results of the CoDMAP benchmark are shown. CoDMAP was a preliminary version of what could be a multi-agent planning competition in the future, that took place in 2015. Here we have rerun the competition with our contributions to compare ourselves against two centralized planners (LAMA-FIRST, YAHSP) and three of the best multi-agent planners that participated in the competition (ADP-L, CMAP-T, SIW). There are 12 domains with 20 problems each. The time limit to solve each problem was 1800 seconds. We have also used the same agentification as it was explicitly noted on the MA-PDDL files (official language of the competition). Tables 7.8 and 7.9 show the obtained results in coverage (number of problems solved).

We have used the three different goal allocation strategies: *Best-cost* (BC), *Load-balance* (LB) and *All*. Since PMR can solve each problem in three different ways, we show the coverage obtained on each one separately: merge (M, when plans are valid after merging), centralized planning (C, when no agent could generate any plan) or plan reuse (R, when the merged plan was invalid). The configuration PMR ALL was added in order to make PMR complete. It shows a similar behavior as a centralized algorithm. Thus, when PMR ALL fails in the M and R steps, it usually solves the problems in the C step. On the other hand, PMR LB and BC solve more problems in the M and R steps than

Table 7.8: PMR with LPG-ADAPT and RRPT-PLAN configuration in combination with up to three goal assignments: BC (*Best-cost*); LB (*Load-balance*) and *ALL*. In PMR, *M*: merging; *R*: plan-reuse; *C*: centralized. Partial is the total score in coverage of each step in PMR. Each domain has 20 problems. It is the same set of problems used on CoDMAP.

	PMR-LPG-ADAPT									**PMR-RRPT-PLAN**								
	BC			**LB**			**ALL**			**BC**			**LB**			**ALL**		
	M	**R**	**C**	**M**	**R**	**C**	**M**	**R**	**C**	**M**	**R**	**C**	**M**	**R**	**C**	**M**	**R**	**C**
Driver.	19	1	0	8	12	0	0	20	0	19	1	0	8	12	0	0	20	0
Zenotra.	20	0	0	20	0	0	0	20	0	20	0	0	20	0	0	0	20	0
Elevators	1	0	19	1	0	19	0	0	20	1	0	19	1	0	19	0	0	20
Logistics	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20
Rovers	20	0	0	19	0	0	9	3	8	20	0	0	19	0	0	9	3	8
Satellites	20	0	0	20	0	0	16	0	4	20	0	0	20	0	0	16	0	4
Sokoban	1	3	7	0	4	7	0	0	17	1	3	7	0	2	7	0	0	17
Taxi	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20
Blocks	20	0	0	0	20	0	0	20	0	20	0	0	0	20	0	0	20	0
Wireless	0	0	2	0	0	2	0	0	5	0	0	2	0	0	2	0	0	5
Depots	0	0	17	0	0	17	0	0	17	0	0	17	0	0	17	0	0	17
Woodw.	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20
Partial	101	4	105	68	36	105	25	63	131	101	4	105	68	34	105	25	63	131
Total	210			209			**219**			210			207			**219**		

PMR ALL. The aim of PMR is to solve as many problems as it can executing the *M* and *R* phases.

The best configurations of our contributions in coverage are PMR-LPG-ADAPT-ALL and PMR-RRPT-PLAN-ALL followed by PMR-RRPT-PLAN-BC. In general, all our contributions are similar in terms of coverage, they all have passed the barrier of 200 problems solved (over 240). Our contributions had very good coverage in all domains, except for Wireless that was the hardest domain in CoDMAP; none of the planners obtained good results on it.

Analyzing the coverage results of PMR in relation to which phase solved the problems allows us to classify the domains in three groups. This classification reflects in turn the interaction level among agents and goals: low, medium and high.

Low interaction domains: Zenotravel, Rovers and Satellites. PMR often solves problems in these domains by merging the individual agents' plans, because these plans are mostly independent. However, it depends on the type of goal allocation strategy selected. Take, for instance, the Zenotravel domain. Both BC and LB assign only one agent to each goal. Since the plans generated by agents can solve all their individual

Table 7.9: Coverage score per configuration in CoDMAP domains.

	PMR-RRPT-PLAN			MULTI-AGENT			SINGLE-AGENT	
	BC	LB	ALL	CMAP-T	ADP-L	SIW	LAMA-FIRST	YAHSP
Driverlog	20	20	20	20	20	18	20	16
Zenotravel	20	20	20	20	20	20	20	12
Elevators	20	20	20	20	20	20	20	2
Logistics	20	20	20	20	20	20	20	20
Rovers	20	19	20	20	20	20	19	4
Satellites	20	20	20	20	20	20	19	10
Sokoban	11	9	17	13	17	17	17	7
Taxi	20	20	20	20	20	20	20	20
Blocks	20	20	20	20	20	20	20	17
Wireless	2	2	5	5	9	7	5	18
Depots	17	17	17	17	17	19	16	14
Woodworking	20	20	20	17	20	18	20	2
Total Cover.	210	207	219	212	**223**	219	216	142
Total Cost	181.61	165.33	181.87	184.17	188.34	179.05	**189.22**	128.27
Total Mkspn	127.60	139.02	136.24	140.12	146.66	-	**149.72**	140.90
Total Time	186.87	173.48	175.86	201.72	**207.59**	191.84	197.14	126.69

goals and do not interfere with other agents' plans, the merge step solves all problems. But, in the case of the ALL strategy, it will assign each passenger to all airplanes. The first agent will move all passengers, the second agent will also move all of them, and so on. Therefore, the merged plan will be invalid (all airplanes will try to move all passengers). These problems are easily solved then by plan repair.

Medium interaction domains: Driverlog, Blocks and Depots. In these domains, when the goal assignment strategy selects several agents for planning, problems are mostly solved in the plan repair step. So, individual agents are able to solve their problems, but the merged plans are invalid due to interaction among the plans to achieve the goals. Then, the plan repair step can generate a valid plan by solving the negative interactions.

High interaction domains: Elevator, Logistics, Sokoban, Taxi, Wireless and Woodworking. In these domains, a single goal might need the collaboration among two or more agents. For instance, in the Logistics domain, most packages need at least two trucks and one airplane to reach their destination. In those cases, PMR individual plans will fail and the centralized planning solves most problems.

A key related issue is how domains are modeled. For instance, if the only agents considered in the Logistics domain are the airplanes, then BC and LB would solve all problems just by either merging the resulting individual plans or by plan repair. Similarly, in the case of Elevator and Taxi, one could define fast elevators or taxis respectively as the only agents and PMR would solve all problems without the centralized planner.

We have compared our contributions against the winner of CoDMAP in coverage, ADP-L [Crosby, 2015]. Note that ADP-L does not preserve privacy and it uses a different agents' configuration than the one proposed in the competition. We also compare against CMAP-T [Borrajo and Fernández, 2018] that obtained the best coverage and time score from the planners that preserved privacy. Finally we also included SIW [Muise et al., 2015] as it was one of the best planners of the competition. Results of this comparison are shown on Table 7.9.

Regarding Table 7.9, ADP-L obtains the best coverage followed up by SIW and PMR-ALL (both with LPG-ADAPT and RRPT-PLAN). YAHSP coverage is the worst. Related to coverage in the official CoDMAP, after the summer-run[2] PMR-ALL would share the first position with ADP-L, which was 219. However, in our CoDMAP rerun, ADP-L solved 223 problems.

Regarding time scores (Table C.1 in the Appendix B), the fastest planner is ADP-L followed very closely CMAP-T. The performance on time of PMR configurations is homogeneous. The time score in PMR was computed using the total amount of time spent by the whole process. Since the individual agents' planning processes were executed in sequence, the total time is computed as the sum of all these processes. However, considering that the merging phase could be implemented fully distributed, the total time would be the maximum planning time among all agents, instead of the sum, plus the time spent on plan reuse or centralized, if needed.

In relation to quality scores, Tables C.2 and C.3 (in the Appendix B) show the results of makespan and cost of plans, respectively. As LAMA and ADP-L did not compute the makespan metric, our parallelization algorithm was applied to their resulting plans to fairly compare the results of makespan. SIW makespan results are not shown in the table because our parallelization algorithm could not support the use of constants that they include in the PDDL domain and problem to respect the privacy of objects and fluents after they transform the MA-PDDL files into PDDL [Muise et al., 2015]. This issue does not happen when the domain and problem are directly given in PDDL.

Regarding PMR configurations, it can be seen how BC is better in cost and LB in makespan. The difference between these numbers lies in the number of agents involved in the planning process. BC often includes the minimum necessary number of agents to

[2] http://agents.fel.cvut.cz/codmap/results-summer/

plan (the ones that expectedly achieve goals with the minimum cost), so the plans' cost will usually be low (good score). In the extreme, some problems were solved using only one agent. However, the makespan is penalized given that the same agent is achieving all goals, so many actions cannot be executed in parallel. On the other hand, LB tries to include as many agents as possible, as long as they can solve at least one of the goals of the problem. The makespan will be better than that of BC, because actions can be easily parallelized. But, the cost is penalized when choosing LB, as it uses agents whose plans are worse in terms of cost. Also, potentially more interactions need to be solved (causing an increase in the number of plans solved by R instead of M).

PMR-RRPT-PLAN configurations perform better on quality and PMR-LPG-ADAPT configurations, better on time. Moreover, PMR-LPG-ADAPT-ALL, which was one of the best configurations in coverage, is the worst of our configurations in quality scores. The ALL strategy uses all agents in the problem to plan leading to long merged plans. PMR-RRPT-PLAN-ALL performs better in quality than PMR-LPG-ADAPT-ALL because it obtains better results mainly in Driverlog and Blocks thanks to the combination of plan-reuse and search. Although ADP-L obtains a better value in makespan than all PMR-LB configurations, it is mainly due to the difference in solved problems in Wireless and Sokoban. It is also remarkable the result obtained by LAMA-FIRST (best configuration in cost and makespan) because it is a centralized planner. This clearly shows that the CoDMAP problems are not hard to solve, even for classical planners. As a result, even when agents try to solve all goals at once in centralized planning, the impact in time is minimal.

7.6 Results changing agentification

After running the CoDMAP experiments and identifying three different levels of interaction, we propose a new set of experiments focused on the high level group of interaction domains. The aim is to analyze the planner's performance regarding different agentifications.

As it was mentioned before, CoDMAP organizers specified which were the agents on each domain of the competition. This directly affected the performance of the participating planners, especially in our case, as we factorize the problem regarding agents and goals. In some domains, such as Depots or Logistics, our contributions were not able to generate the MAP task for the set of problems. Thus, the centralized planner solved the task instead.

Here we want to evaluate (1) if by just changing the agents of the problem, PMR is

Table 7.10: Makespan score in domains solved during plan reuse phase. Depots, Logistics and Elevators are CoDMAP versions in MA-PDDL. Depots-truck, Logistics-airplanes, Elevators-fast are modelled in PDDL and indicate that the agentification has been changed with respect to CoDMAP. Makespan from SIW and ADP-L is obtained through our parallelization algorithm. SIW values marked with (*) mean that the makespan could not be obtained because of a parsing problem.

	PMR-RRPT-PLAN-BC	**PMR-RRPT-PLAN-LB**	**SIW**	**CMAP-T**	**ADP-L**
Depots	16.10	16.18	0.00*	12.04	16.18
Logistics	6.78	5.08	0.00*	8.65	12.27
Elevators	13.66	19.92	0.00*	7.21	6.49
Depots-trucks	10.12	6.56	17.10	12.41	8.06
Logistics-air	18.56	16.12	9.89	17.27	18.24
Elevators-fast	9.37	13.20	8.41	19.25	17.05
Total Mkspn	74.61	77.06	35.40	76.83	**78.28**
Total Cost	87.63	84.25	87.61	**100.59**	90.28

capable of generating the MAP task correctly and solve the problem using the plan reuse phase instead and (2) if the plan-reuse performance increases or decreases.

In order to carry out this analysis we run PMR-RRPT-PLAN with strategies LB and BC and compare our results against SIW, CMAP-T and ADP-L.

Table 7.10 summarizes the IPC scores obtained in makespan. Last two row also reflect the score obtained in cost. The detailed table of cost is placed in Annex B (C.4.)

In Depots-trucks we changed agents from drivers and places to trucks; Elevators-fast considers fast elevators as agents and Logistics just the airplanes instead of airplanes and trucks. As goals are previously assigned to agents in PMR, these agentifications allow to compute the estimation costs per agent and goal.

The aim is also to evaluate what would happen in these domains if a centralized planner is run after the individual planning phase instead of running plan-reuse. By doing this we are also evaluating the complexity of the given problems.

As it can be seen in the results of makespan, ADP-L obtains the best score, followed closely by PMR-RRPT-PLAN-LB. Bearing in mind that these problems are not difficult to solve, multi-agent centralized planners obtain good results. New agentifications benefit BC and LB configurations in Logistics. However, the planner that takes the greatest advantage using the new agentification is CMAP-T. It is an example of how much the planner performance can vary when choosing one agentification or another.

PMR-BC scores are very similar to those obtained by PMR-LB. This is due to the fact that the LB strategy tries to use as many agents as possible. As in these problems the interactions between agents are higher, it penalizes the makespan metric. BC configurations are not designed to optimize makespan but in this case they indirectly take advantage of choosing the minimum number of agents.

7.7 Results scaling the number of agents

In order to analyze how PMR configurations scale with the number of agents regarding makespan and cost, we have generated three medium-sized problems in Zenotravel, Driverlog and Logistics domains respectively.

The Zenotravel problem has 63 goals. Then, we increased the number of agents. For each instance of the Zenotravel problem, agents were increased by 10 starting on 10 agents and stopping at 70. The configuration used for this experiment was PMR-RRPT-PLAN. Zenotravel belongs to the low interaction group identified in Section 7.5. The domains that belong to this group are the ones where generally PMR performs better, as most problems are solved during merging or plan-reuse phases, avoiding the centralized planning step. Thus, we were interested in exploring in detail the evolution of makespan and plan length when increasing the number of agents.

Results are shown in Figure 7.6. X axis represents the number of available agents per

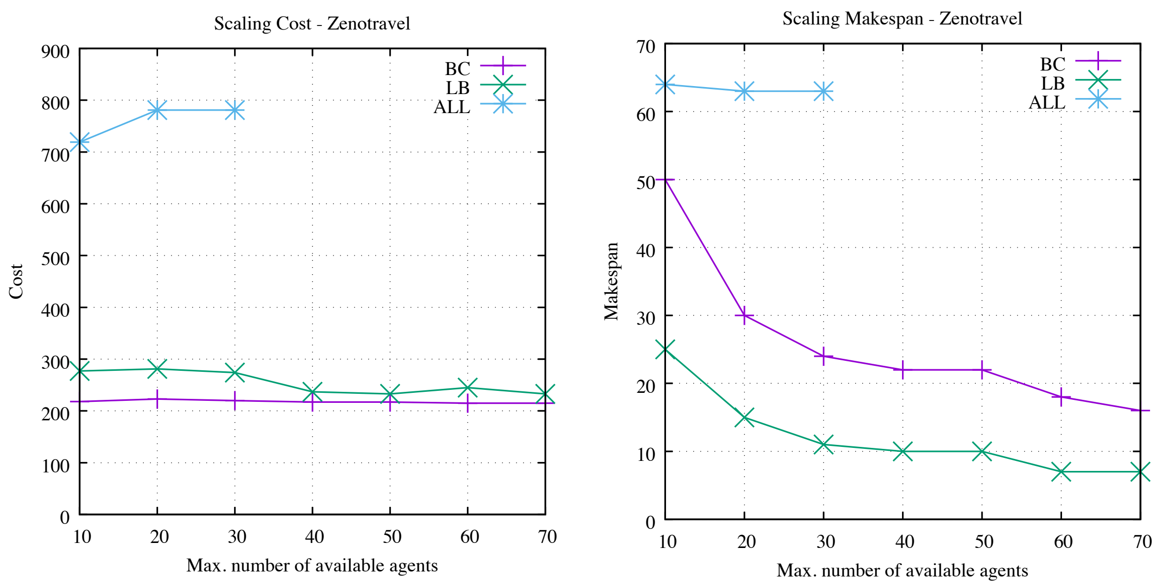
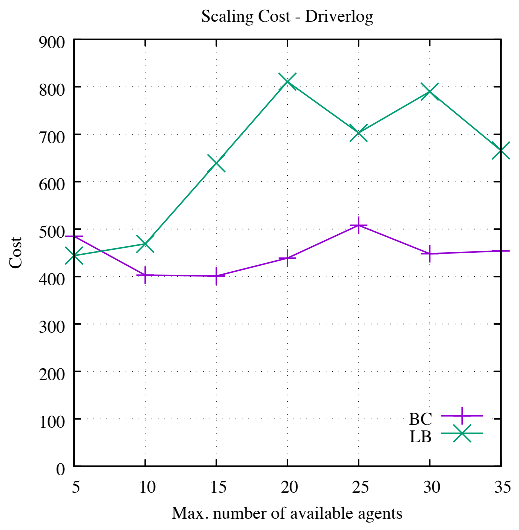


Figure 7.6: Evolution of Cost and Makespan in a Zenotravel problem when increasing the number of agents gradually (x axis). The configuration used was PMR-RRPT-PLAN with *Best-cost* (BC), *Load-balance* (LB) and ALL.

problem that can be selected to plan by each Goal-Assignment strategy. In Zenotravel, BC uses 10, 16, 18, 21, 21, 22 and 23 agents per problem while LB uses 10, 20, 30, 34, 37, 59 and 62. As Zenotravel belongs to the low interaction level group, the bigger the number of agents is, the better the makespan obtained by LB. The performance decreases drastically using ALL (only 3 problems out of 7 were solved in 1800s), because all goals are assigned to all agents in the problem, which makes the process of solving the interactions harder during plan-reuse.

When solving a MAP problem with PMR, our goal is to obtain the makespan effect shown on Zenotravel. This will not be possible in every domain, as it will mainly depend on the number of interactions (coupling), but it is the ideal scenario to follow regarding the potential of our contributions.

Driverlog belongs to the medium interaction level group of domains and the selected problem contains 64 goals. Thus, as the level of interaction increases, the problems are solved in the plan-reuse step. The aim of Driverlog is to deliver a set of packages by using drivers to drive trucks. BC uses 5, 9, 12, 16, 20, 23 and 26 agents (drivers) per problem; meanwhile LB employs 5, 10, 15, 20, 23, 30 and 35. Even though the number of agents increases, the number of trucks is fixed to 8. Thus, here the cost and the makespan are closer in both LB and BC. In this domain it is more difficult to paralellize actions due to the higher level of interaction between agents. Also, the LB strategy tends to use as many



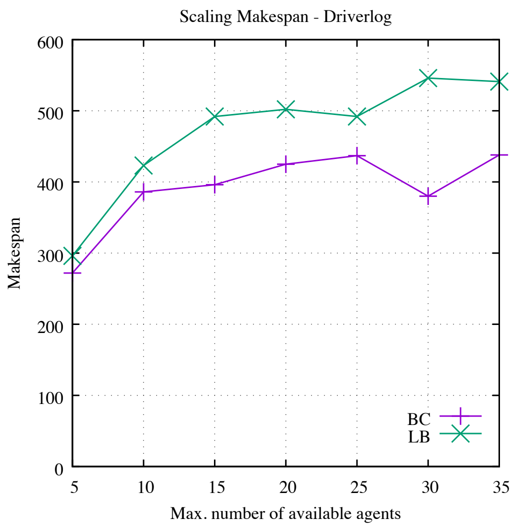


Figure 7.7: Evolution of Cost and Makespan in a Driverlog problem when increasing the number of agents gradually (x axis). The configuration used was PMR-RRPT-PLAN with *Best-cost* (BC), *Load-balance* (LB) and ALL. However, ALL results are not shown because none of the problems were solved when using that goal strategy.

agents as possible, which could add redundant actions to the plan. As a consequence, the BC strategy has better results in both makespan and cost. It uses a smaller number of agents, so less redundant actions are included. Also parallelization has partially lost the impact it had in Zenotravel's makespan. Finally, ALL could not solve any problem in 30 minutes, thus the results do not appear in the two bottom figures of Figure 7.7.

Logistics belongs to the high interaction level group and the chosen problem has 16 goals (Figure 7.8). The main difference regarding previous problems is that Logistics has two potential agent types: trucks and airplanes. Here, we have chosen airplanes as agents to fix interactions in plan-reuse. We wanted to avoid the centralized stage. The topology of the Logistics problem is a grid of cities where trucks can drive inside them and airplanes can fly everywhere. We have increased the number of agents in five per problem. The maximum number of needed airplanes would be equivalent to the number of packages to be delivered (in this case, 16). Thus, 16 agents are employed on ALL configurations except on the first problem, in which PMR uses 15, which is the maximum number available. LB employs 10 agents in the first problem and 16 in the rest. However, BC uses 5, 6, 8, 9, 9, 10 and 11 agents, respectively. Having fewer agents than goals in Logistics makes a package to be exchanged more times between airplanes and trucks until it reaches the final destination. Thus, fewer actions can be parallelized. This issue provokes the opposite effect as in the BC configuration from Driverlog. Now,

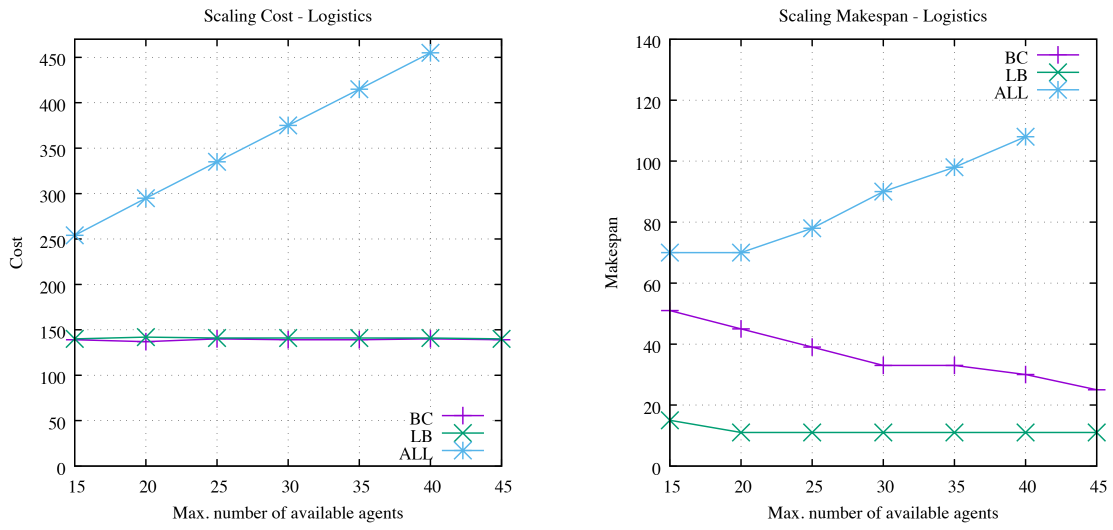


Figure 7.8: Evolution of Cost and Makespan in a Logistics problem when increasing the number of agents gradually (x axis). The configuration used was PMR-RRPT-PLAN with *Best-cost* (BC), *Load-balance* (LB) and ALL.

the makespan decreases gradually as the number of planning agents increases. In ALL results this effect is not appreciated. The policy forces that all goals are assigned to all agents. Thus, redundancy and package-exchanges between trucks and airplanes result in gradually obtaining worse makespan.

7.8 Hard Multi-Agent Planning problems

One of the key goals when working on multi-agent environments consists on improving the distribution of the work load among agents, which directly improves the makespan of the resulting plans. Also, ideally, one would expect multi-agent planners that maximize the work-load distribution to scale up as the number of agents increases.

In this Section we show some experiments on eight domains whose problems are harder to solve than the ones used on previous experiments. Table 7.11 shows the number of agents, goals and main features of those problems. For these experiments we have used the same planners and configurations shown on the CoDMAP experimentation (Section 7.5). We have additionally added Madagascar [Rintanen, 2014], the runner-up of the Agile track in IPC 2014. This centralized planner optimizes the minimum horizon length of plans, which sometimes results in obtaining more parallel actions. We refer to Madagascar as MAD in the Tables below.

In order to select the domains on this experimentation, we decided to include some domains from the group of low interaction (Zenotravel, Satellite, Rover), and some from the group of medium interaction (Driverlog, Blocks). We excluded the ones that had higher interaction (Elevator, Logistics) as they were only solved on the centralized phase, so the work-load distribution of PMR would not affect the result.

Table 7.11: Maximum number of agents, goals and main problem features in hard problems.

	#Agents	**#Goals**	**Features**
Zenotravel	65	104	21 cities, 6 levels of fuel
Satellite	29	114	446 instruments, 85 planets, 86 stars, 87 phenomenons
Rover	135	95	59 waypoints, 81 cameras, 14 objectives
Driverlog	68	131	85 packages, 19 locations, 37 path nodes
Blocks	4	28	30 blocks
Rover-graph	4	268	400 waypoints, 60 cameras, 20 objectives
VRP	5	79	80 packages, 159 locations
Depots-robots	2	3	5 humans, 9 pods, grid 4x5, 4 directions

Additionally, we show on the same tables the results obtained on three domains that were specifically chosen to show the strength of PMR on makespan. Two of those, Rover-graph and the new version of Depots-robots, are contributions of this Thesis. After all the previous experiments, we realized that the MAP problems that fitted PMR best were those where the initial state can be easily divided in regions for the agents to "work" on a specific part of the search space. The lower the number of interactions between them the better.

The Rover-graph domain is an evolution of the usual Rover where two huge grids of waypoints are generated independently and then joined by an edge between those grids. The aim was to test how the planners' performance evolves when a different configuration of the environment penalizes the use of a single agent to solve the whole problem. The number of goals of the set of problems oscillates between 150-200. The minimum number of propositions per problem is 5100. The goals are the same as in the classical Rover domain (e.g. communicate soil, image, rock data etc.). The number of agents varies from 2 to 6. The two grids contain around 400 waypoints in total. On this variation of the usual Rover problem, planners (centralized-based ones specially) get lost because of the size of the search space. Instead of applying factorization to alleviate the agents' individual planning process, those planners employ the smallest possible number of agents to solve the problem. As a result, an agent has to deal with almost all the goals, which makes its planning task harder to solve.

Depots-Robots is the planning version of a warehouse environment inspired on the Kiva Robots [Wurman et al., 2007] where robots have to deliver to humans a list of products from the storage pods to complete a list of delivery orders. This new version

Table 7.12: Coverage results in hard and specific problems.

	PMR-RRPT-PLAN							
	BC	**LB**	**CMAP-T**	**ADP-L**	**SIW**	**LF**	**YAHSP**	**MAD**
Zenotravel	20	20	20	20	1	20	0	18
Satellite	20	20	20	20	1	20	0	20
Rover	20	20	20	20	0	20	0	0
Driverlog	0	3	6	6	0	8	0	0
Blocks	16	16	16	15	13	14	14	0
Rover-graph	18	18	20	20	19	8	0	0
VRP	20	19	17	20	5	17	0	12
Depots-robots	13	13	11	9	12	11	0	5
Total	127	129	**130**	**130**	51	118	14	55

works over a grid of waypoints where pod storages are placed in columns, leaving one column empty between each pair of them for robots to move. The first and last row of the grid are empty. The last row is where humans are situated for the reception of products. The original version, where restrictions on the placements of pods and empty rows are not applied, is described in [Borrajo and Fernández, 2018]. Robots are spread through the grid. Thus PMR can indirectly assign to a robot a specific zone of the grid to pick up the nearby packages. As a result, planners that factorize the problem for each robot regarding human goals and pods' locations will obtain better results.

VPR is the usual Vehicle Routing Problem where trucks need to deliver packages to some cities. The aim is to reduce the cost as much as possible. Here, again the problem can be easily divided as trucks will only care of delivering the packages by themselves. Usually, when goals are well-balanced among the agents, a different portion of the grid of waypoints is assigned to each of them. Thus, the problem is easier to solve for PMR; factorization again is key to simplify the planning tasks.

Coverage results on Table 7.12 show how a planner like SIW that had promising results on CoDMAP now cannot solve more than half of the problems due to their complexity. The rest of the planners, including our configurations, except for YAHSP have a similar coverage.

Table 7.13 shows that our LB configuration outperforms the rest of the planners regarding makespan. These planners were again multi-agent (CMAP-T, ADP-L, SIW) and centralized (LAMA, YAHSP, MADAGASCAR) planners. Hence, even though our configurations might be slower, they are still capable of solving harder problems by involving multiple agents. Dividing the number of goals as much as possible among the agents has

Table 7.13: Makespan score in hard and specific problems

	PMR-RRPT-PLAN							
	BC	**LB**	**CMAP-T**	**ADP-L**	**SIW**	**LF**	**YAHSP**	**MAD**
Zenotravel	8.74	17.41	12.17	7.39	0.53	13.26	0.00	17.89
Satellite	5.41	15.39	8.40	3.07	0.81	9.26	0.00	19.57
Rover	2.52	20.00	4.25	2.49	0.00	3.88	0.00	0.00
Driverlog	0.00	2.84	4.85	5.94	0.00	6.85	0.00	0.00
Blocks	4.59	2.36	6.57	6.57	7.10	6.04	14.00	0.00
Rover-graph	15.48	17.44	14.77	12.05	10.46	6.90	0.00	0.00
VRP	5.33	18.39	1.76	3.92	0.15	3.34	0.00	10.73
Depots-robots	9.48	11.11	10.00	6.37	9.55	9.27	0.00	1.59
Total Mkspn	51.55	**104.95**	62.78	47.81	28.61	58.81	14.00	49.78
Total time	98.14	79.93	88.47	**113.29**	31.69	97.25	7.65	52.48

a direct impact on makespan. In the VRP, Rover-graph and Depots-robots domains, PMR-RRPT-PLAN-LB increases the makespan score more than the other planners; problems can be easily divided, creating balanced subtasks for each agent. In VRP, PMR-RRPT-PLAN outperforms the rest of planners, as it is a good example of domain, where multi-agent planning can improve through factorization. This can also be seen on Rover-graph with CMAP-T and LAMA. The main difference between them is factorization. CMAP-T is able to solve every problem while LAMA cannot scale enough to solve a task of 200 goals. Domains such as Zenotravel, Satellite and Rover, that have a low number of interactions, are also good for our LB configuration.

Table C.5 in the Appendix B shows the time score, where ADP-L and LAMA are the fastest ones. PMR is slower, because of the goal assignment phase, as our algorithm spends some time on identifying which agent solves best each goal. The fastest configurations usually assign all goals to all agents by default. This strategy works well when optimizing for time or coverage. However, we claim that the makespan score should be the main performance criteria if the goal is to generate plans in a real multi-agent environment, where a big number of agents is available to work.

7.9 Discussion on the experiments' results

After describing the results of six different sets of conducted experiments some specific conclusions and remarks can be extracted from them:

- The performance of PMR and its adaptability cannot be appreciated on easy MAP tasks. For instance, results on CoDMAP (Section 7.5) reflect that the centralized planner LAMA was able to solve more problems than PMR's BC, LB and CMAP-T; even its coverage results are very close to those of ADP-L and SIW. This indicates that MAP planners do not usually have a remarkable advantage over centralized approaches on easy MAP tasks without privacy concerns.

- Regarding interactions among agents, PMR works best on low and medium interaction domains. Those are generally identified as loosely-coupled domains by the planning community. Some of those domains are: Rovers, Satellite, Zenotravel, Depots, Hammer and VRP.

- PMR biases towards optimizing the makespan metric, independently of the MAP task received as input. Thus, the more the task can be factorized and work equally distributed among agents, the better the makespan obtained will be. Usually, those tasks are the loosely-coupled ones.

- PMR scales well on hard loosely-coupled planning tasks, and also in those that contain a topology that can be factored for the agents to work independently. This can be appreciated on the results from Rovers-graph or VRP.
- The goal assignment strategy Load-Balance works best for loosely-coupled domains, as it balances the amount of goals among the agents. In turn, this has a direct impact on improving the makespan. Best-cost biases towards improving the plan length and works best for tightly-coupled domains. As less agents are used during planning, it will reduce the number of conflicts to solve.
- RRPT-PLAN behaves very similar to LPG-ADAPT, which is important regarding that LPG-ADAPT is considered the state-of-the-art replanner.

  In addition, RRPT-PLAN covers the full range of reuse scenarios from the best case for reuse (when the input incorrect plan is very similar to the final correct one) and the worst case (when the input and final plans are very different).
- The plan-reuse phase of RRPT-PLAN is more useful when it is selected first. If we assume that the input plan is similar to the final plan, running plan-reuse first will automatically include most of the input plan actions into the final plan, boosting the performance of RRPT-PLAN as a result. However, in case the input plan is completely different from the final plan, we expect that the plan-reuse will fail fast and RRPT-PLAN will switch to the search phase. This issue was tested on the experiments with increasing number of goals in Section 7.4.

Also, some limitations have been identified:

- The main issue that faces PMR with tightly coupled domains is the number of interactions to solve. If the planning task presents many interactions, the plan-reuse phase could be potentially solved better by planning from scratch. Examples of such planning tasks are the ones defined in the IPC for the Driverlog or Sokoban. However, one must take into account that the distribution of problems generated by the IPC organizers for each domain, albeit randomly generated, usually focus on a specific subarea of the set of potential problems that can be generated. Therefore, it is easy to see that even in these two tightly-coupled domains, one can generate problems with lower interaction. For instance, one could have several rooms in

Sokoban, each with its own set of robots, or one could have different network subgraphs in Driverlog, each one with its own set of drivers and vehicles. So, the property of being tightly-coupled is connected to the planning task (domain and problem) and not only to the domain.

- There exists a bottleneck on the goal assignment process. When the MAP task contains a considerable amount of goals, the time spent on estimating the cost per goal-agent can be heavily increased if the search space of the problem is big. In real-life environments, this issue can be solved by including external information to boost the cost estimation process, as shown in the next Chapter.
- Given that our objective was to focus on loosely-coupled tasks, PMR cannot deal with joint actions (actions that require more than one agent to be executed, as moving a table by using two agents).

Part IV

Application in robotics

Chapter 8

Use case: PMR on Robotics environments

Real-world robotic scenarios, in which a set of robots need to solve a certain amount of tasks, usually requires the combination of path-planning and motion-planning techniques. In this Chapter we show a combined approach between Actuation Maps and Multi-Agent Planning to boost the path-planning computation. Our approach will be able of solving big size problems in terms of goals, environment and search space.

8.1 Introduction

Many real-world robotic scenarios require performing task planning to decide courses of actions to be executed by (possibly heterogeneous) robots. A classical centralized planning approach has to find a solution inside a search space that contains every possible combination of robots and goals. This leads to inefficient solutions that do not scale well. Multi-Agent Planning (MAP) provides a new way to solve this kind of tasks efficiently. Previous works on MAP have proposed to factorize the problem to decrease the planning effort i.e dividing the goals among the agents (robots). However, these techniques do not scale when the number of agents and goals grow. Also, in most real world scenarios with big maps, goals might not be reached by every robot so it has a computational cost associated.

This section presents a joint work with Tiago R. Pereira, Antonio M. Moreira and

Manuela M. Veloso [Luis et al., 2019; Pereira et al., 2018] where we propose a combination of robotics and planning techniques to alleviate and boost the computation of the goal assignment process. Actuation Map (AMs) was an approach already proposed by Pereira et al. [2015] to analyze the reachability limit of robots' motion and actuation in building floors to perform an efficient assignment of tasks afterwards. The joint work aims to combine AMs and MAP to efficiently solve path-planning tasks.

Given a map, AMs can determine the regions each agent can actuate on. Thus, specific information can be extracted to know which goals can be tackled by each agent, as well as cheaply estimating the cost of using each agent to achieve every goal. Experiments show that when information extracted from AMs is provided to a multi-agent planning algorithm, the goal assignment is significantly faster, speeding-up the planning process considerably. Experiments also show that this approach greatly outperforms classical centralized planning.

From a MAP point of view, the multi-robot problem we propose forces us to deal with two issues regarding the performance of the planning process: (1) the size of the search space grows with the number of waypoints and goals; and (2) some goals are not feasible for some robots. On one hand, real-world scenarios are big enough to make almost impossible for a planner to solve this problem in a reasonable amount of time by just assigning all goals to all agents (following a centralized planning approach). On the other hand, some Multi-Agent planners invoke a goal-allocation phase before starting to plan to decrease the effort of computing individual plans such as PMR or [Borrajo and Fernández, 2018]. During goal allocation, a relaxed plan is computed per goal and robot to either return an estimated cost or to identify unfeasibility. This process would be repeated multiple times, concretely $|Agents| \times |Goals|$, resulting in a large amount of time lost, especially to identify unfeasible goals i.e exploring most of the search space.

Here we consider Actuation Task as a robot performing an operation that results in some task being executed in the environment. Therefore, the actuation capabilities could be modeled not only as the operations where a robot changes its surrounding, but also as any perception-like operation e.g. the planning problem to solve would be mathematically equivalent if the overall goal was not to clean all reachable space, but instead to measure the temperature everywhere.

Our framework allows to solve planning problems where sensing operations need to be executed at specific waypoint locations e.g. mapping the Wi-Fi signal strength in buildings, taking measurements of temperature and humidity on a set of pre-defined locations such as a computer cluster and server sites, different kinds of inspection or even surveillance problems. In general, for each robot and domain a different actuation model can be considered e.g the actuation radius being smaller than the robot's footprint, the actuation range being exactly the same as the robot's footprint, or extending it further than the footprint. An example of the last case is a mobile manipulator, where an arm can be extended and actuate on regions beyond the robot's range in terms of its shape and footprint.

8.2 The coverage problem

Our approach can be easily applied to any robotic problem that involves at least the following elements:

- a map of the environment;
- a set of potential goals, such as cleaning actuation goals, to be executed by a set of agents over the environment;
- a way to model that scenario into a PDDL domain and problem.

The map can be modeled in different ways (e.g. a navigation graph, a grid of waypoints, a building floor etc.). In this work, the starting point is an image representing the world's floor plan. From that image, we extract a navigation graph over a grid of waypoints, which is used to generate the PDDL problem. However, from the floor plan, additional information can be extracted related to the environment, which is accomplished through the process of building the AMs. The potential of our approach relies on the ability to extract information from the map related to the tasks. Then, that information is transformed into a set of estimation costs that can speed up the planning process.

The set of potential goals can vary depending on the problem to solve. In this case we are focusing on the coverage problem and as a result it is enough for the robots to move through the environment. On alternative problems, the goals could be: looking for objects, opening doors or achieving some clients' orders through the environment.

The coverage problem planning task is to find a route for each robot so that all the feasible space is covered by the robots' actuators, while minimizing the execution time. Vacuum cleaning robots can be potential candidates for this problem. We assume that we have a team of heterogeneous robots with different sizes. While the smallest robot can reach more areas, a bigger robot cleans a wider area while traveling a smaller distance.

We have computed the estimated cost as the distance from the robots to each of the waypoints on the coverage problem.

For vacuum cleaning robots, the sensing is performed through the robot's actuator that cleans the floor. Cleaning a specific waypoint location can be seen as the robot accomplishing a specific cleaning task, and therefore the coverage problem would represent a robot moving through a map and executing multiple tasks, i.e., actuating on all available waypoints on the environment with the objective of cleaning all reachable regions of the environment. Some other alternatives tasks would be looking for objects, opening doors or achieve some clients' orders through the environment. Also, the actuation capabilities of each robot depends on the specific actuator each robot uses, e.g., its shape and size.

Some other examples of problems to solve could include heterogeneous robots executing surveillance tasks, cooperative mapping of the environment or search and rescue tasks. Also, other measurements for computing estimated costs could be related to the distance to a required object, the dangerousness or reliability of a path, or the features of a robot, such as maximum velocity or the existence of manipulation capabilities.

In order to transform this kind of problems into PDDL we have to model (1) a domain; (2) a problem; and compute (3) a set of estimated costs.

In this Section we are only considering heterogeneous teams of circular robots that actuate in a 2D environment, where the world is represented by a 2D image that can be down sampled to a 2D grid of waypoints. The AM gives information about the actuation capabilities of each robot, as a function of robot size and initial position [Pereira et al., 2015]. In the example with vacuum cleaning robots, the AM represents the regions of the world each robot can clean.

We assume that robots are circular and thus the only *Robot* feature is its size, with 2D grid positions being rotation-invariant. Other shapes will be later considered in our approach by extending the PDDL domain file to take into consideration robot orientation as well.

The domain has two types of objects: *robots*, which act as agents; and *waypoints*, which represent positions in the discretized world. We consider a coverage problem, where the goal is to have the robots actuating on waypoints. In this version of the coverage problem, robots actuate a waypoint if it is inside its actuation radius. Thus, they do not need to be exactly placed on the waypoint to actuate it.

Therefore, following the MAP task defined in 3.2.2, the set G is a list of waypoints to actuate on (positions that need to be covered). The PDDL domain we created has four predicates:

- `At(robot, waypoint)`: defines the robot position;
- `Connected(robot, waypoint, waypoint)`: establishes the connectivity between waypoints, specified for each robot, and given the robot heterogeneity, some connections might be traversable by some robots and not by others;
- `Actuated(waypoint)`: indicates which waypoints were already actuated; this predicate is used to specify goals;
- `Actuable(robot, waypoint, waypoint)`: shows which waypoints can be actuated by a robot when located on a different waypoint location.

Robots have to actuate every waypoint in G (as long as the goal is feasible). The waypoints, when connected, generate a navigation graph for a certain robot. The three actions that are defined in the domain are called `navigate` (Figure 8.1), `actuate-on` (Figure 8.2) and `actuate-other` (Figure 8.3). The action `actuate-on` is used to actuate the current position of the robot. The third action is employed to mark a waypoint as `actuated` if the waypoint is identified as *actuable* from the robot's current location i.e. the waypoint is located inside the robot's actuation radius on the real environment. Actions `navigate` and `actuate-/on/other` can be executed by an agent when it is placed on a waypoint. Both `actuate-on` and `actuate-other` have as effect the predicate `actuated`.

```
(:action navigate
  :parameters (?r - robot ?y - waypoint ?z - waypoint)
  :precondition (and (connected ?r ?y ?z) (at ?r ?y))
  :effect (and (not (at ?r ?y)) (at ?r ?z))
)
```

Figure 8.1: Action Navigate in PDDL

```
(:action actuate-on
    :parameters (?r - robot ?y - waypoint)
    :precondition (at ?r ?y)
    :effect (actuated ?y)
)
```

Figure 8.2: Action Actuate-on in PDDL

```
(:action actuate-other
    :parameters (?r - robot ?y - waypoint ?z - waypoint)
    :precondition (and (at ?r ?y) (actuable ?r ?y ?z))
    :effect (actuated ?z)
)
```

Figure 8.3: Action Actuate-other in PDDL

In order to generate a PDDL problem, the waypoints' grid resolution is defined in advance using a discretization step. After that, a navigation graph and a set of reachable waypoints are defined for each robot, taking into account their physical characteristics. All this information is generated on the preprocessing step, further explained in Section 8.6.

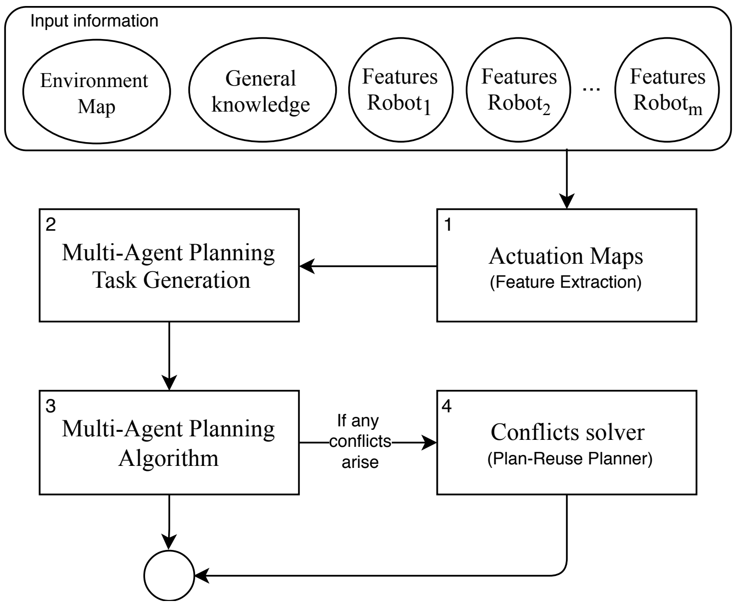


Figure 8.1: Complete architecture that combines Actuation Maps and Multi-Agent Planning

8.3 Architecture

This work combines Actuation Maps (AM) with MAP. The contributed architecture can be seen in Figure 8.1. It has been divided into four modules and receives as input the map of the environment, the general knowledge related to the task to solve and the features of the set of robots. The aim of each module is described as follows:

1. Actuation Maps module: it is in charge of generating the AMs for each given robot. It also extracts the map features that can potentially alleviate the planning process (e.g. estimation costs), and it generates the planning problem in PDDL. It is explained on Section 8.4.

2. Multi-Agent Planning Task Generation module: once the outputs from the prior module and the domain are received as input, the goal assignment process is launched. This module is in charge of dividing the goals among the agents by following some goal strategy. Then, a set of domain and problem is generated for each agent, which is known as factorization. We have used the MAPR factorization explained on Section 3.2.

3. Multi-Agent Planning Algorithm module: the individual planning process and the merging phase of PMR are run on this module.

4. Conflicts solver module: if any interactions need to be solved, this module employs a plan-reuse-planner to fix them. It is explained on Section 8.7.

The following section explains the essential information regarding AMs. The aim is to fully understand later the preprocessing step.

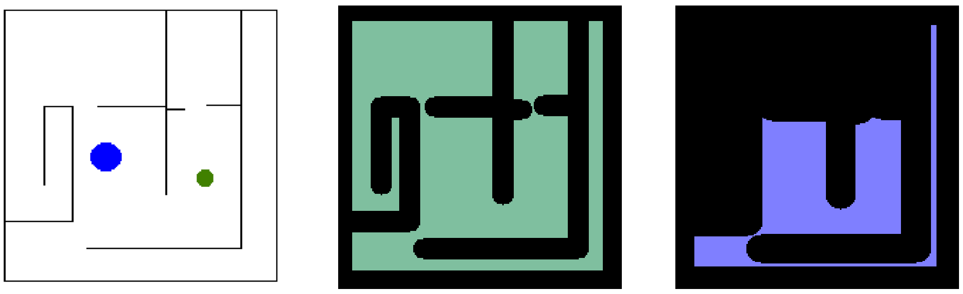
8.4 Actuation Maps

Our system receives as input the *Environment map* which represents a 2D environment (e.g. building floor plan) and *m Robot* models with the agents' features. There is a third input provided by the user that refers to the *General knowledge* of the environment (i.e: tasks to solve). These three inputs represent the input information described in Figure 8.1.

The formalization of Actuation Maps (AMs) for circular-robots was carried out by the coauthors of this contribution, T. Pereira, A. Moreira and M. Veloso. For the sake of simplicity and to respect authorship, throughout this section we are only focusing on the key concepts of AMs. The formalization of AMs for circular-robots is defined in [Pereira et al., 2015].

We assume there is an occupancy grid map, i.e., a gray-scale image representing the environment. In this image each pixel has a value with the probability of the corresponding world position being occupied by an obstacle. This occupancy grid map is first transformed into a binary image of free and obstacle pixels, using a fixed threshold. An example of the resulting black and white image is shown in Figure 8.2(a).

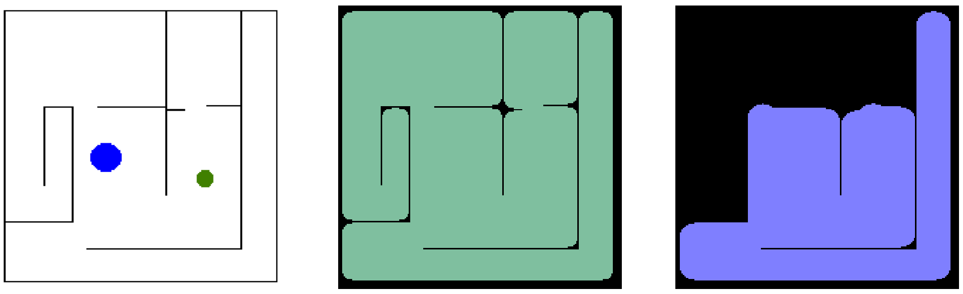
The Actuation Space represents what the robot can actuate from any point reachable from its initial position. In Figure 8.2 we show the Actuation Spaces after applying the partial morphological closing operation to the original map (explained in [Pereira et al., 2018]). The Actuation Spaces belong to two robots with different sizes.



(a) Map (b) Robot1 (c) Robot2

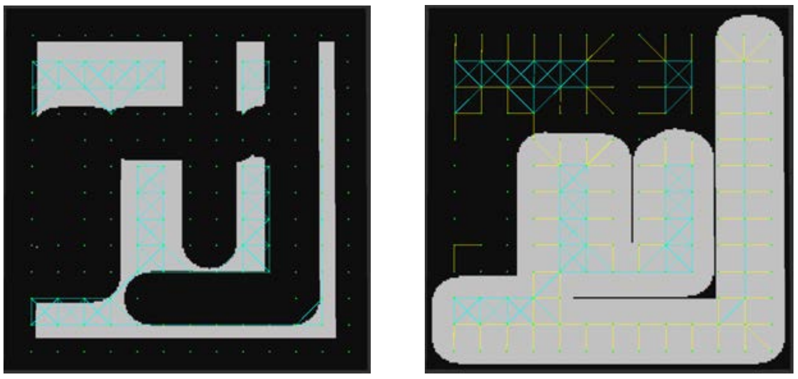
Figure 8.2: Simulated map and two heterogeneous robots with different sizes in (a); colored regions represent the navigable space for 2 robots with different sizes, depending on size and initial position of robots.

While the Actuation Space is the set that contains all the waypoints that can be actuated by a robot, its representation as an image, as shown in Figure 8.3, is the AM. We also use the term AM to refer to the overall technique to determine actuation capabilities of robots. In the figures below we show the AMs. The AM can be used as a



(a) Map with 2 Robots (b) Actuation Map 1 (c) Actuation Map 2

Figure 8.3: Colored regions in Figure (b) and Figure (c) represent actuation spaces for respective robots, i.e. the points in the environment that each robot can actuate, depending on their size and initial position shown in Figure (a); the actuation capability in this example is completely coincident with the entire robot footprint, i.e., the actuation range is equal to the robot radius.



(a) Navigability (b) Actuation

Figure 8.4: In Figure (a), the free configuration space for the bigger robot in Figure 8.2(a), with the discretization waypoints shown as green dots. The blue lines represent the connectivity between waypoints in the navigation graph of the robot. Using parameters δ and α it is possible to maintain the topology of the free configuration space by allowing points in the navigation graph that were originally unfeasible for the robot. In Figure (b), the actuation map of the same robot, and the respective actuation graph represented with yellow lines.

visual representation to show what the robot can actuate from any reachable point from its initial position. We use them to visually represent in figures the sets corresponding to each robot's actuation space.

As an example, we can consider again the vacuum cleaning robot case. The actuation space represents the regions the robot can clean; and the non-actuable regions are positions that the robot cannot clean. For circular vacuum cleaning robots, corners of the environment are non-actuable regions they cannot clean due to the robot's circular shape.

8.5 Discretization

For the planning problem, it is possible to consider each individual pixel as a waypoint. However, that approach results in a high density of points that would make the planning problem excessively complex. Moreover, there is some redundancy in having points that are too close to each other, as their difference is not significant in terms of the environment size and localization accuracy.

Therefore, we reduced the set of locations from all pixels to a smaller set of locations.

We considered again waypoints distributed into a grid, but now the grid-size is greater than one pixel. Then, we can find the connectivity between points to construct the navigation graph of each robot, shown in Figure 8.4(a). It is also possible to find which waypoints can be actuated from other waypoints using the distance between them, as shown in Figure 8.4(b), by considering the maximum actuation radius. However, the problem of such discretization is the change in the actuation space topology. Adjusting the position of waypoints could allow a better representation of the topology of the environment, but the multi-robot nature of the problem compromises that solution.

In order to deal with multiple robots with different reachable sets, for each agent, we independently adjust the waypoint position -temporarily- in a hidden manner invisible to the other agents. When discretizing each robot's configuration space, we might consider a waypoint as belonging to the free configuration space even if it is strictly outside it, as we assume an error margin to compensate for the discretization error. Nevertheless, we still maintain the original waypoint position in further steps, such as determining actuation feasibility of that waypoint, and for visualization purposes as well. When determining the navigation graph of each robot, an unreachable waypoint position might be moved to the closest point in the configuration space, if the adjustment is under a given margin δ. As stated previously, the adjustment is always temporary to the construction of the connectivity graph of each robot. After the navigation connectivity is tested, the waypoint position resets to its default grid position for the next steps, such as determining the actuation feasibility, and the navigation and actuation graphs of other robots.

Moreover, when determining the connectivity of waypoints for the navigation graph, only the eight grid neighbors are considered. A^* is then used to determine the real distance between waypoints (e.g., around obstacles), and connectivity is only considered if the real distance is at most a factor of $\alpha = 1.2$ the straight line distance between them.

Finally, all waypoints that belong to the robot actuation map should be connected to some waypoint of its navigable graph. If that is not the case after the previous steps, we connect the isolated waypoints to the closest navigable vertex in line of sight, even if their distance is greater than the maximum actuation distance, again to compensate for the discretization error. Therefore, while the planner may return an `actuate` action to cover waypoint A from the navigable waypoint B in the discretized world, a real robot would have to move closer from the waypoint B to waypoint A in order to actuate the latter.

The grid density is chosen manually in order to adjust the level of discretization. As for the α and δ parameters, they were tuned empirically such as the free space topology is still maintained even while using lower density discretization of the environment. By trial and error, we found empirically that $\alpha = 1.2$ works for all the tested scenarios.

As for the δ parameter, we set it to always start with a value of 3 pixels, then build the discretized model and verify if it is valid, i.e., if all the waypoints belonging to the actuation map become feasible for the respective robot in terms of the discretized representation. If not, we increment the parameter until a topologically consistent representation is found (number of feasible goals equals number of waypoints inside actuation map). Even though this fine-tuning methodology seems sensitive to the robot heterogeneity, the truth is that the final δ value depends on the size of the bigger robot, because the correct discretization of the configuration space is more sensitive to the δ parameter for bigger robots. Through experimentation, we found out that if a certain value of the δ parameter works well for the biggest robot, it always produces the correct discretization for smaller robots. Moreover, we also observed that $\delta = 4$ pixels worked well for all the different and very diverse maps we tested in our experiments with circular robots, only failing for the any-shape experiments where the configuration space discretization is more sensitive to the possible robot orientation. For the any-shape robot experiments, we found that $\delta = 6$ pixels was enough to obtain a good discretization for all the environments tested. The consistency of the δ parameter over different environment maps shows that these parameters can be map-independent to a certain extent, with most of the work being easily automated.

8.6 Preprocessing

The contributed preprocessing step is shown in Figure 8.5. This is the point where both techniques, AMs and MAP, are combined and complement each other. In this Section we describe the generation of goals, the detection of unfeasible regions and the computation of estimated costs. Section 3.2.4 described the process of generating the MAP task and how the task is factorized (divided) in subtasks.

When converting the original map and the Actuation Space to the PDDL description, it is possible to consider each individual pixel as a waypoint in a grid with the size of the whole image. However, that approach results in a high density of points that makes the planning problem excessively complex. There is also redundancy in having points that are too close to each other, as their difference is not significant in terms of the environment size and localization accuracy.

Therefore, we reduce the set of possible locations by downsampling the grid of waypoints. The downsampling rate s_r is set manually. If the original pixel resolution is used, the resulting grid of waypoints $\mathcal{G}'$ contains all pixels and is equivalent to $\mathcal{G}$.

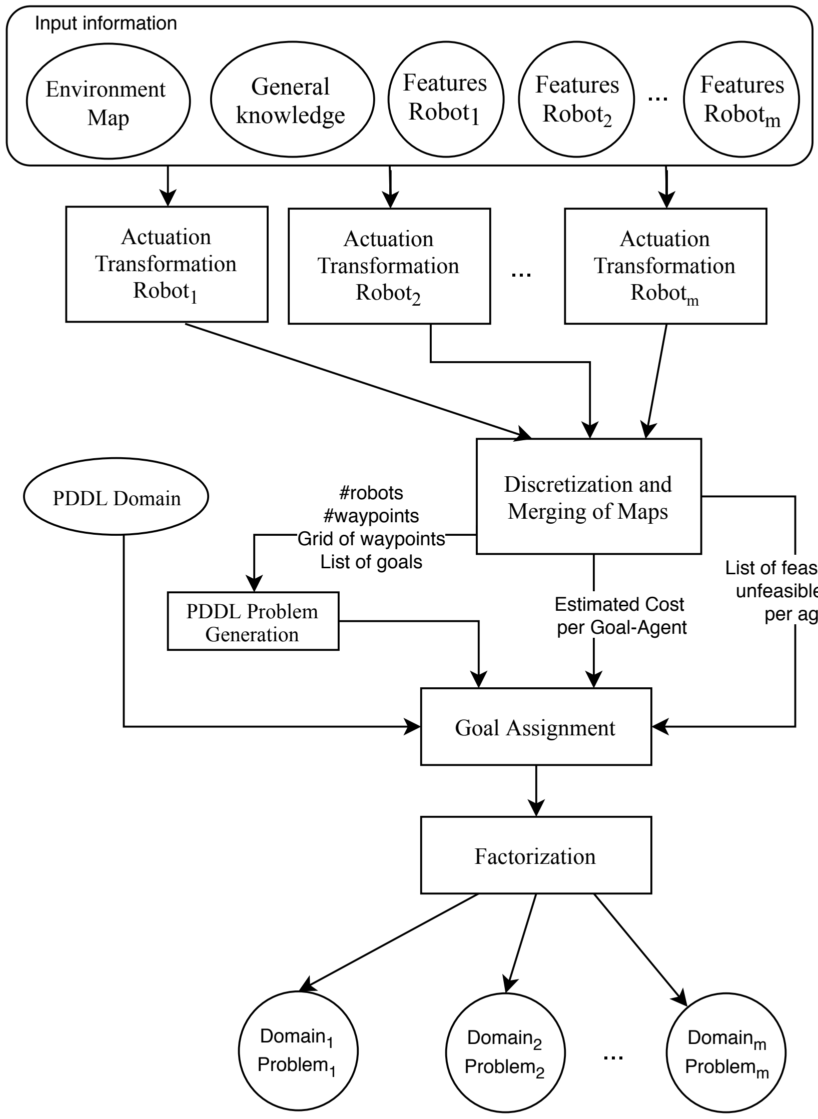


Figure 8.5: Preprocessing stage before the planning process starts. First, inputs are processed in order to generate the AM for each agent. Then, a discretization is applied to generate all the required information for planning such as the navigation graph for the PDDL problem, the list of estimated costs, etc. Once this information has been generated, the Goal-Assignment starts and the MAP problem is divided into subproblems with specific goals assigned to each individual problem.

Otherwise, the set $\mathcal{G}'$ represents the grid of waypoint positions after downsampling.

Using the Actuation Space it is possible to very easily find UG, the list of *unfeasible goals per agent*:

$$UG = \{g \in \mathcal{G}', \phi_i \in \Phi \mid g \notin \mathcal{A}_i(\mathbf{r}_i^0)\} \tag{8.1}$$

The positions in the actuation space ($\mathcal{A}_i(\mathbf{r}_i^0)$) are feasible actuation goals for agent ϕ_i.

The information from the UG list can speed-up goal assignment by avoiding computation related to unfeasible goals, but it does not provide any information about the cost for each robot to accomplish a feasible actuation goal.

For that purpose, we contribute the following extension. We build the navigable space $\mathcal{L}_i$ in an iterative procedure, from the starting position $\mathbf{r}_i^0$. In the first iteration we have $\mathcal{L}_i^0(\mathbf{r}_i^0) \leftarrow \{\mathbf{r}_i^0\}$, and then the following rule applies:

$$\begin{aligned} \mathcal{L}_i^j(\mathbf{r}_i^0) = \{\mathbf{p} \in \mathcal{G} \mid \exists \mathbf{q} \in \mathcal{L}_i^{j-1}(\mathbf{r}_i^0) : \mathbf{p} \text{ neighbor of } \mathbf{q} \\ \wedge \mathbf{p} \in \mathcal{C}_i^{free} \wedge \mathbf{p} \notin \mathcal{L}_i^a(\mathbf{r}_i^0) \quad \forall a < j\} \end{aligned} \tag{8.2}$$

When using this recursive rule to build the navigable space, we guarantee that any point in the set $\mathcal{L}_i^j(\mathbf{r}_i^0)$ is exactly at distance j from the initial position $\mathbf{r}_i^0$.

Furthermore, if we build the actuation space sets with the intermediate navigable sets $\mathcal{L}_i^j(\mathbf{r}_i^0)$,

$$\mathcal{A}_i^j(\mathbf{r}_i^0) = \mathcal{L}_i^j(\mathbf{r}_i^0) \oplus \mathcal{R}_i \tag{8.3}$$

then the intermediate actuation set $\mathcal{A}_i^j(\mathbf{r}_i^0)$ represents the points that can be actuated by the robot from positions whose distance to $\mathbf{r}_i^0$ is j. The actuation space defined in the previous section can also be alternatively defined as

$$\mathcal{A}_i(\mathbf{r}_i^0) = \{\mathbf{p} \in \mathcal{G} \mid \exists a : \mathbf{p} \in \mathcal{A}_i^a(\mathbf{r}_i^0)\} \tag{8.4}$$

The actuation cost is defined for $g \in \mathcal{A}_i(\mathbf{r}_i^0)$:

$$\mathcal{AM}_i(\mathbf{r}_i^0, g) = \min\{j \mid g \in \mathcal{A}_i^j(\mathbf{r}_i^0)\} + 1 \tag{8.5}$$

The actuation cost $\mathcal{AM}_i(\mathbf{r}_i^0, g)$ represents, for each $g \in \mathcal{A}_i(\mathbf{r}_i^0)$, the minimum number of actions needed for the robot to actuate the grid waypoint g if starting from the initial position $\mathbf{r}_i^0$, measured in the pixel-based grid $\mathcal{G}$. In Equation 8.5, the minimum j^* represents the minimum distance (i.e., minimum number of *navigate* actions) needed to travel from $\mathbf{r}_i^0$ to some point from where g can be actuated. The added one in Equation 8.5 accounts for the one *actuate* action needed to actuate g, after the j^* *navigate* actions needed to reach a place from where the robot can actuate g.

Thus, the cost function C presented previously in Section 3.2 is defined in Equation 8.6, where s_r is the downsampling rate. The division by s_r transforms the estimated cost of actions measured in the pixel-based grid $\mathcal{G}$, $\mathcal{AC}_i(\mathbf{r}_i^0, g)$, to the respective cost value in the downsampled grid of waypoints $\mathcal{G}'$. The *ceil* function rounds up the result of the division to the smallest integer value that is not less than $\mathcal{AC}_i(\mathbf{r}_i^0, g)/s_r$. The cost function

C is domain-dependent and works for the coverage problem. If a different problem is given as input, the cost function should be redefined.

$$c(\mathcal{AM}_i(\mathbf{r}_i^0, g)) = ceil\left(\mathcal{AM}_i(\mathbf{r}_i^0, g)/s_r\right) \tag{8.6}$$

Finally, the Estimated Cost per Goal-Agent list EC is defined in Equation 8.7.

$$EC = \{\langle g, \phi_i, c\rangle \mid g \in \mathcal{G}' \wedge \phi_i \in \Phi \wedge g \in \mathcal{A}_i(\mathbf{r}_i^0) \wedge c \in c(\mathcal{AM}_i(\mathbf{r}_i^0, g))\} \tag{8.7}$$

Once the discretization of maps has been performed, we have all the information needed to generate the MAP task M, which is formed by a domain (received as input) and a problem (generated through the discretization). The inputs to the Goal Assignment (GA) phase are (1) the PDDL domain; (2) the PDDL problem; (3) the list of estimated costs where c is computed as the number of steps for an agent to reach the goal position g from its initial position; and (4) the list of unfeasible goals $UG = \{g \in G, \phi_i \in \Phi \mid C(g, \phi_i) = \infty\}$. As long as EC is provided, UG is not used inside the MAP algorithm. The case when EC is not provided is later explained in this Section.

In Multi-Agent systems, in order to perform task allocation [Conitzer, 2010; Gerkey and Matarić, 2004] some strategy has to be determined or implemented, as the aim it is to divide the MAP task in subtasks to alleviate the planning process afterwards. In addition, a goal-assignment strategy (GAS) needs to be chosen to define the way goals are assigned to agents by the system. In our approach, we have chosen the Load-Balance (LB) strategy previously defined in Section 3.3. The LB assignment strategy is used when minimizing the maximum number of actions per agent (makespan). As a second option, we also chose the Best-Cost (BC) when minimizing the total number of actions over all robots (plan length).

As stated in [Borrajo, 2013], only when the information about estimated cost per pair robot-goal is not available for some reason, our MAP algorithm would perform Goal Assignment by computing a relaxed plan using the FF heuristic [Hoffmann and Nebel, 2001]. However, in that work, when a goal was unfeasible for every agent, it was assigned to all of them. In our approach, when the goal has been identified as unfeasible by all agents, the relaxed plan is not computed for that pair robot-goal and the goal is not included into the M task. This behavior is not common in classical deterministic Automated Planning, as planners expect that the problem does not contain any unfeasible goal. As our approach separates goal allocation from planning, we can easily deal with unfeasible goals. This small contribution gives us more flexibility when working for real environments, as it is better to obtain a plan that solves 95% of the goals than just failing during planning. To plan using soft-goals [Krulwich, 1992] or working on oversubscription planning [Smith, 2004; García-Olaya et al., 2011] would have been other ways to deal with unfeasibility, but they are out of the scope of this work. In summary, there are two contributions to the GA process: (1) the detection and deletion of unfeasible goals is a contribution that helps not only on skipping the computation of those relaxed plans but also avoids the planning process to fail; and (2) to use information from AMs, as the algorithm receives and processes the estimated costs from the AMs to skip the computation of the relaxed plans. The MAP algorithm to solve the MAP task is PMR.

On Figure 8.6 we show a solution example obtained from the MAP algorithm. It corresponds to the scenario called Corridor-High later on the experiments.

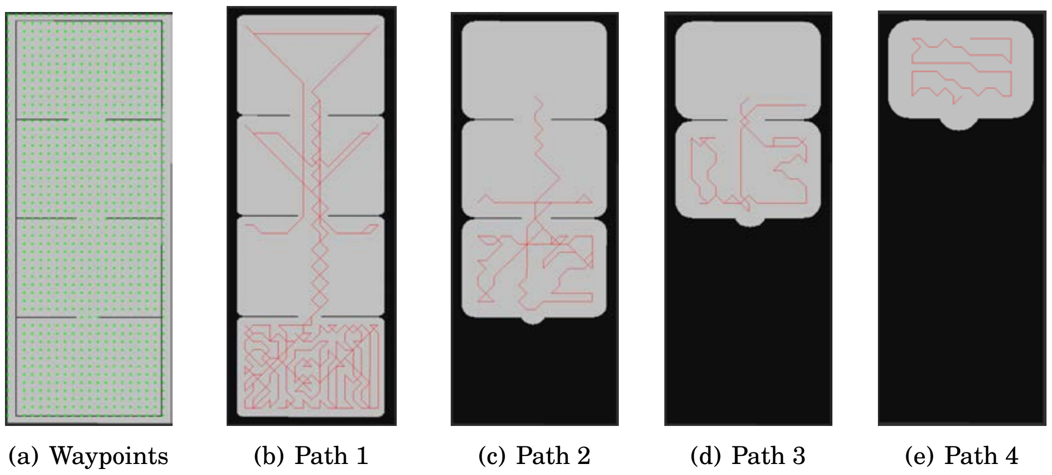


Figure 8.6: These figures represent the Corridor scenario used in the experiments. The waypoint discretization is shown in Figure (a). The resulting path for each robot is shown in Figures (b) to (e), after solving the planning problem using load balance as goal-strategy. Path 1 belongs to the smallest robot. Path 4, to the biggest one.

8.7 Dealing with Interactions

Real-world robotics environments might imply to deal with potential interactions among robots (at the very least) e.g. collisions, sharing a resource, or cooperation. On the previous description of the coverage problem we did not explicitly consider any kind of interactions. Our idea was (1) to test first the scalability of the MAP algorithm; and (2) to generate, as fast as possible, a valid solution. Robots might occasionally collide at some specific step of the solution plan. However, that collision could be easily resolved during execution by forcing one of the robots to wait until the other robot has left the conflict zone. Then, the stopped robot will continue executing the rest of its plan. On the other hand, there is a subarea of Automated Planning called Planning by Reuse that has been widely employed in areas such as Case-Based Planning [Borrajo et al., 2015], or replanning when plan execution fails [Fox et al., 2006]. Usually, planners that perform plan reuse receive three inputs: a domain, a problem and a plan to be fixed. Examples of this kind of planners are LPG-ADAPT [Fox et al., 2006] or ERRT-PLAN [Borrajo and Veloso, 2012]. More information about Plan Reuse is given in Chapter 4.

Therefore, an improvement of our approach is to detect and fix potential collisions right after the individual planning process using an off-the-shelf plan reuse planner, as PMR does. This new feature makes our architecture more robust when executing the solution plan in a real environment. Thus, we slightly changed our PDDL domain to track the collisions by adding a new predicate called `occupied`.

- `Occupied (waypoint)`: indicates that there is a robot on that waypoint.

That predicate is set as a new `precondition` of the `navigate` action described in Figure 8.3. This allows the agent to only traverse a connection if the destination waypoint is not occupied by a robot. The predicate `occupied` in combination with the parallelization algorithm avoids two situations: (1) Two robots cannot be on the same waypoint at the same plan step. (2) As we build a parallel plan, neither of them can swap positions during the same plan step. The reason is that in order to move *robot1* to y from z, y should be not occupied first. The same happens to z for *robot2*. Thus, both actions are mutually exclusive and the parallelization algorithm does not allow both actions to be performed together. One of the robots would need to move somewhere else first.

As the MAP algorithm starts with the individual planning phase, no collisions will be detected at that point (line 1). After concatenation, the solution plan is validated by VAL (line 3). The validator will detect, if any, mutex actions related to occupied positions as explained above. If so, the plan will be invalid. As a result, the M' task and the invalid plan are sent to the plan reuse planner (line 6). When the plan is fixed, the parallelization step is applied (line 7). Finally, the MAP algorithm runs VAL again (line 8). If the plan is valid, it is returned as the solution. The configuration of our algorithm is the same as the previous version - the Single-Agent Planner is LAMA-UNIT-COST. The plan reuse planner is LPG-ADAPT.

```
(:action navigate
   :parameters (?r - robot ?y - waypoint ?z - waypoint)
   :precondition
     (and (connected ?r ?y ?z) (at ?r ?y)
          (occupied ?y) (not (occupied ?z))
     )
   :effect
     (and (not (at ?r ?y)) (not (occupied ?y))
          (at ?r ?z) (occupied ?z)
     )
)
```

Figure 8.4: Action Navigate that now checks occupied positions

8.8 Extending the Approach to Any-Shape Robots

For the case of non-circular robot footprints (Figure 8.7), given that the robot model is not rotation invariant, we need to discretize orientation as well. We use a world representation that is composed of multiple layers, where each layer represents one orientation. Then, we determine individually for each orientation the corresponding actuation space. Again, for the sake of simplicity and to respect authorship, throughout this section we are only focusing on the key concepts of the extended approach for any-shape robots. The entire formalization can be found in [Pereira et al., 2016].

First, the algorithm needs images to model both the robot and its actuation capabilities. Both are parametrized by images that can be rotated and scaled to represent

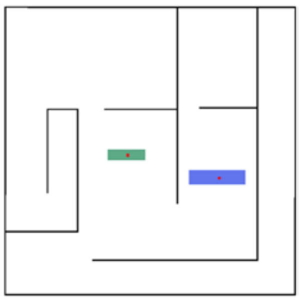
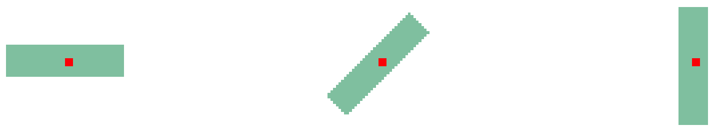


Figure 8.7: Environment and robot models used to test the extended approach to any-shape robots .



(a) Robot Model $\mathcal{R}$ for θ = 0ž

(b) Robot Model $\mathcal{R}$ for θ = 45ž

(c) Robot Model $\mathcal{R}$ for θ = 90ž

Figure 8.8: Example of an image representing the robot footprint, rotated for three different angles, and used as structuring element in the morphological operations applied to the respective orientation layers; robot center shown in red.

any robot. As input, it is also necessary to give the center of the robot and actuation in terms of their model images, and their relative position. Figure 8.8 shows an example of how the input image of the robot has been rotated to build a model of the robot for each possible orientation.

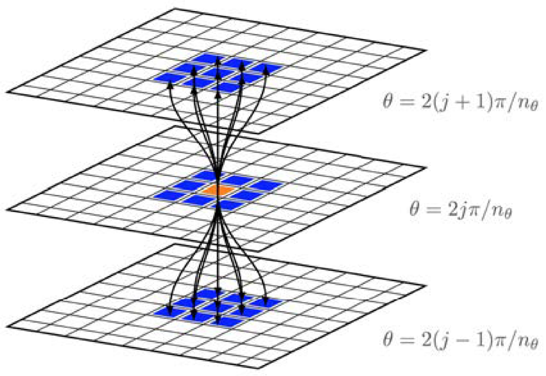
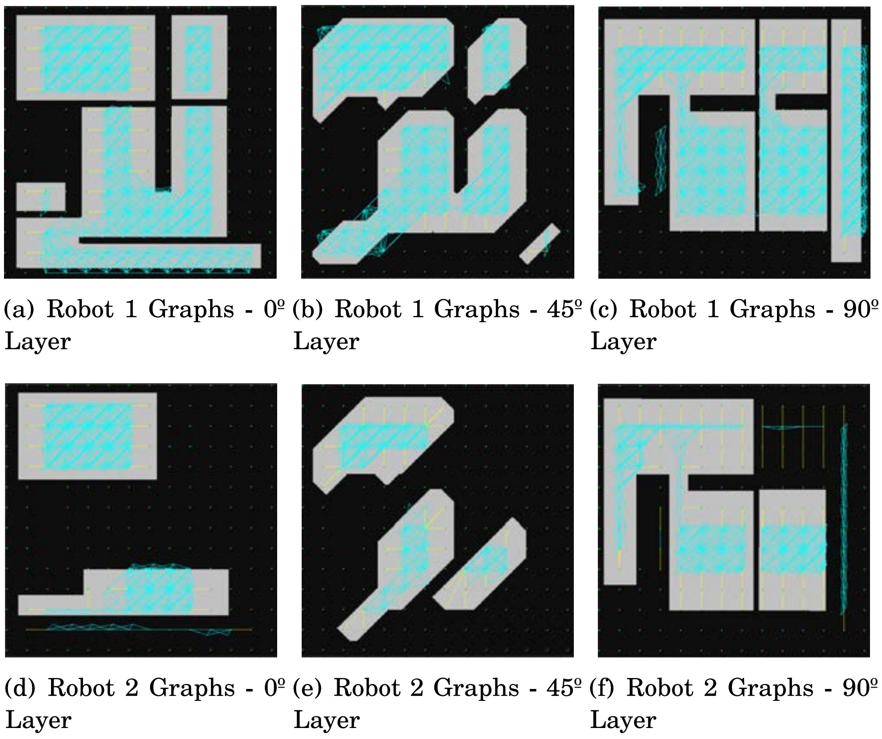

$\theta = 2(j+1)\pi/n_\theta$
$\theta = 2j\pi/n_\theta$
$\theta = 2(j-1)\pi/n_\theta$

Figure 8.9: Three adjacent layers of the discretized orientation, showing in blue the neighbor points of a central orange dot, representing the connectivity/motion model.

In order to model a robot that navigates through waypoints, we need to establish the type of connectivity between points in different layers, such as it is equivalent to the type of motion the robot actually has. As an example, using the connectivity graph from Figure 8.9, where one point is connected to all its neighbors in the same layer, and the respective positions in adjacent layers, is equivalent to considering an omnidirectional model of navigation.

Therefore, while on the rotation-invariant scenario the domain was discretized in a series of 2D waypoints, for the any-shape case there are two types of waypoints: the


(a) Robot 1 Graphs - 0º Layer (b) Robot 1 Graphs - 45º Layer (c) Robot 1 Graphs - 90º Layer

(d) Robot 2 Graphs - 0º Layer (e) Robot 2 Graphs - 45º Layer (f) Robot 2 Graphs - 90º Layer

Figure 8.10: The `connected` and `actuable` graphs shown in blue and yellow, respectively; as shown for each layer, the yellow actuation graph connects 3D waypoints to the original 2D green waypoints, and the blue connectivity graph connects 3D waypoints not only to neighbors in the same layer, but also in adjacent layers.

3D waypoints representing (x, y, θ) position, and the 2D waypoints representing (x, y) positions invariant to orientation.

The navigability graph now becomes a graph of 3D waypoints connected to each other, modeling the motion capabilities of robots in the world in terms of both rotation and translation, individually or combined, as exemplified for different orientation layers on Figure 8.10.

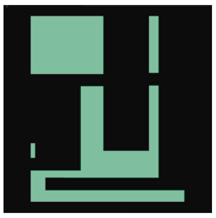
As a result, the actuation space gives the actuation capabilities for each orientation for a given robot shape and starting position. We show in Figure 8.11 the navigable and actuation spaces for different layers, given the robots and map shown in Figure 8.7.

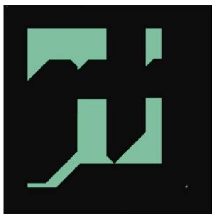
After determining the actuation space for each layer, we can obtain the overall actuation map in a rotation-invariant representation by projecting the multiple layers into one single 2D image.

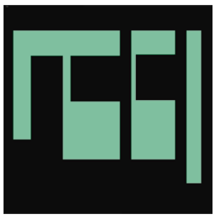
The actuation graph is now a graph of 3D waypoints connected to 2D waypoints, representing the actuation of a rotation-independent position in the projected 2D actuation

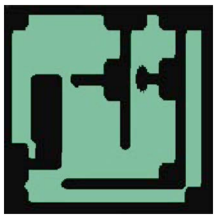
(a) Robot 1

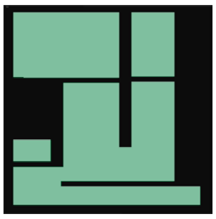
(b) Robot 2

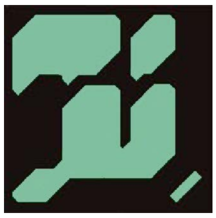

(c) Robot 1 - Navigable Space - 0º Layer

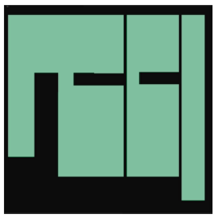

(d) Robot 1 - Navigable Space - 45º Layer

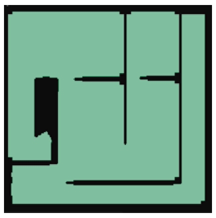

(e) Robot 1 - Navigable Space - 90º Layer

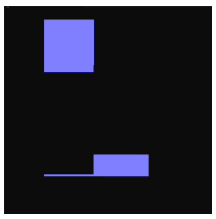

(f) Robot 1 - 2D Projected Navigability

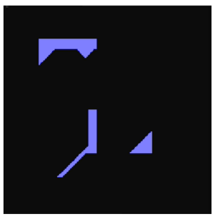

(g) Robot 1 - Actuation Space - 0º Layer

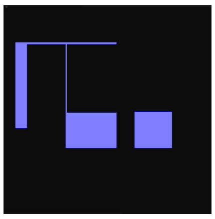

(h) Robot 1 - Actuation Space - 45º Layer

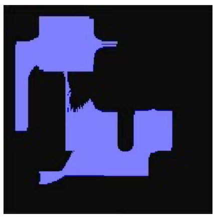

(i) Robot 1 - Actuation Space - 90º Layer

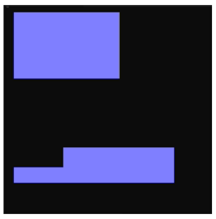

(j) Robot 1 - 2D Projected Actuation Map

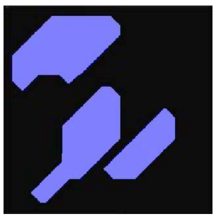

(k) Robot 2 - Navigable Space - 0º Layer

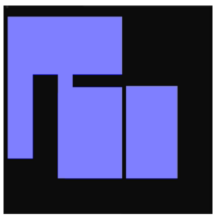

(l) Robot 2 - Navigable Space - 45º Layer


(m) Robot 2 - Navigable Space - 90º Layer


(n) Robot 2 - 2D Projected Navigability


(o) Robot 2 - Actuation Space - 0 Layer


(p) Robot 2 - Actuation Space - 45º Layer


(q) Robot 2 - Actuation Space - 90º Layer

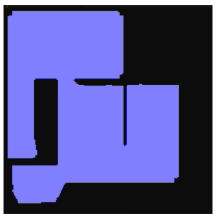
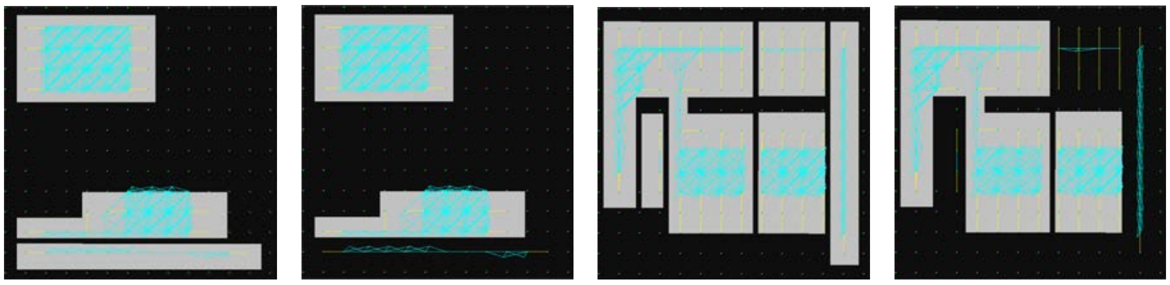

(r) Robot 2 - 2D Projected Actuation Map

Figure 8.11: Navigable and Actuation Space for 2 non-circular robots with different sizes, for the scenario shown in Figure 8.7.



(a) Robot 2 - Graphs on Free Configuration Space - 0º Layer (b) Robot 2 - Graphs on Navigable Space - 0º Layer (c) Robot 2 - Graphs on Free Configuration Space - 90º Layer (d) Robot 2 - Graphs on Navigable Space - 90º Layer

Figure 8.12: The discretized graphs constructed are independent of the initial robot positions, allowing to run the problem from different initial positions; the white regions (navigable space, dependent on initial position) are covered by the graphs, but some black regions (if grey in the respective configuration space, independent of initial position) are also covered by the constructed graphs.

map, from a 3D robot waypoint location, also shown in Figure 8.10. The predicates on the PDDL problem are represented as follows:

- `Connected (robot, 3Dwaypoint, 3Dwaypoint)`
- `Actuable (robot, 3Dwaypoint, 2Dwaypoint)`

For each 2D waypoint in the circular robot scenario, there are now n_θ 3D waypoints in the same (x, y) position, representing the different orientations a robot can have on the same 2D waypoint. As we show in Figure 8.12, the two graphs are constructed independently of the initial position, allowing very easily to change the starting location of any robot and solve a different instance of the same problem. Thus, there were no modifications in the modeling of the PDDL problem. The 3D to 2D representation is transparent to the planning process.

The `navigate` action moves through 3D waypoints, and the `actuate` action makes 2D waypoints have the `actuable` predicate. The list of goals to solve the problem is still given by a list of 2D waypoints that cover all the space. Thus, for the same map, the coverage problem is still the same in terms of goal waypoints and model. However, now we plan for robots to move through the environment and actuate goal positions from some planned orientation. The PDDL domain did not need any further modifications.

If we project the multiple layers of the graphs in a 2D image, we can analyze which waypoints are navigable in terms of the robot motion, and which ones are only feasible

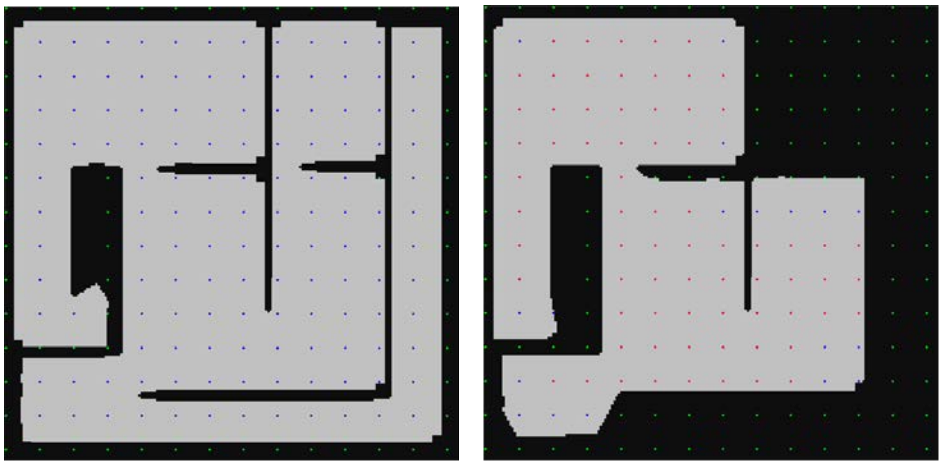

(a) Robot 1 (b) Robot 2

Figure 8.13: All goal waypoints are shown as spheres on top of the Actuation Map: green represent unfeasible waypoints, in red the ones covered by the `connected` graph, and in blue the ones only covered by the `actuable` graph; for the smaller robot 1, the two graphs are the same.

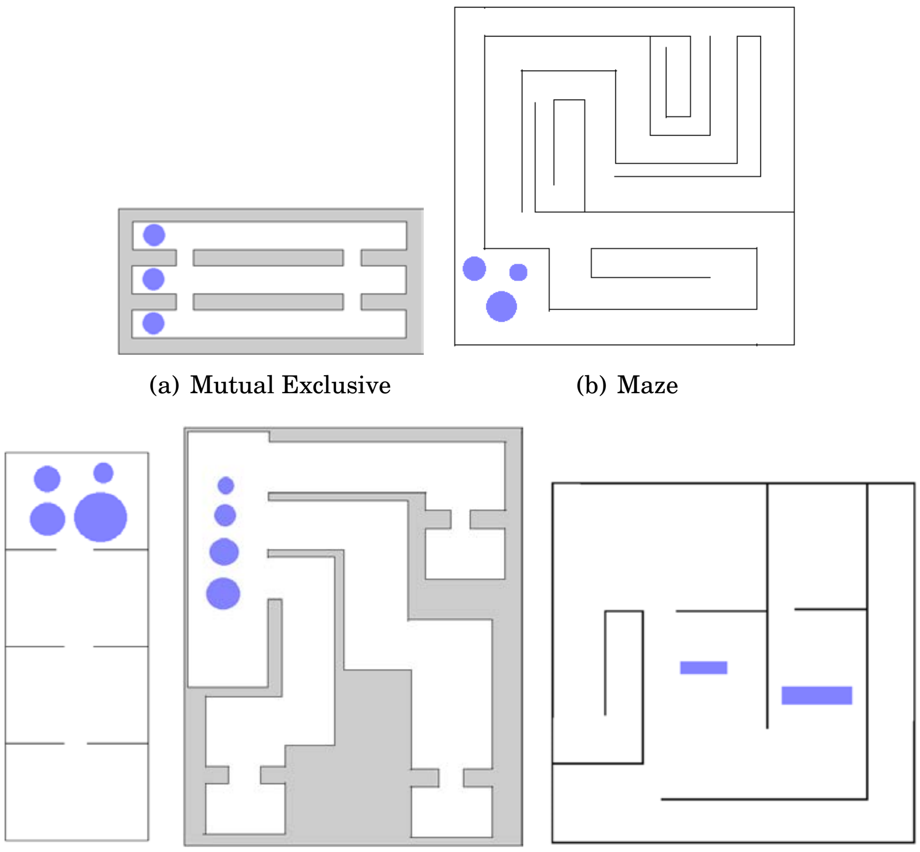
through an actuation action. As we show in Figure 8.13, some of the waypoints are not feasible by any of the robots, and all the feasible waypoints lie inside the Actuation Space (grey region of the images).

8.9 Experiments and results

In this Section we show the results of the experiments that were designed to test the impact of the preprocessing on two different versions of our algorithm, PMR. First, on the following Section we describe the five scenarios designed to run the experiments. Then, on Section 8.9.2 the experiments on the coverage problem are analyzed. These results were partially included on [Pereira et al., 2018]. Finally, on Section 8.9.3 we show the results on the coverage problem including collision detection.

8.9.1 Simulation Setup

Here we describe in detail the scenarios used for running the experiments. We designed five different scenarios, shown in Figure 8.14, each one with two levels of waypoint density (H, the higher, and L, the lower density). The scenarios are designed for circular robots except for the last one (called *Rooms*) that is designed for any-shape robots.



(a) Mutual Exclusive (b) Maze

(c) Corridor (d) Extremities (e) Rooms

Figure 8.14: Maps of the five scenarios used in the experiments. Grey regions represent out-of-reach regions which cannot contain goal waypoints. They are unfeasible for all the robots. Robots are represented with blue circles/rectangles positioned in the region of their starting position.

- Mutual Exclusive: three wide parallel horizontal halls, connected between them by two narrow vertical halls; 3 robots move within the horizontal sections, one in each, and their actuation reachabilities are mutually exclusive.
- Maze: maze-like scenario with narrow halls and passages with different sizes, resulting in bigger robots not reaching some parts of the maze, or needing to traverse bigger paths to arrive to the same locations as smaller robots.
- Corridor: four wide sections with openings of different sizes connecting them; the opening decreases from the top to the bottom, with all 4 robots being able to actuate in the top region, to only one being able to reach the bottom.
- Extremities: wide open section with three halls departing to different directions,

where all 4 robots actuate; at the end of each hall there is a room that can be accessed through an opening, with only one robot reaching the extremity connected with the smallest opening, to three reaching the one connected with the biggest opening.

- Rooms: simple floorplan environment with some room-like spaces connected through passages of different sizes as well, used to test the non-circular robot case where they can traverse the passages using only certain orientations.

Furthermore, in Table 8.1 we present the size of each map image, and the number of agents and feasible and unfeasible goals for each scenario. We present in Table 8.1 the grid size in terms of the downsampled grid of waypoints. The original image had a pixel size approximately 10 times bigger, with a pixel resolution corresponding to 10cm. Therefore, the maps we tested represent environments with a size always bigger than 300 square meters.

We have generated two problems per scenario, one of them with low density of waypoints (which we identify as L in tables) and the other one with a higher density of waypoints (H). We have also designed versions of *Maze*, *Extremities* and *Rooms* for 10 robots in order to test the behavior of the planners in crowded scenarios. *Rooms2r* is a similar version of *RoomsL* but for circular robots.

Table 8.1: Number of agents, feasible and unfeasible goals and respective grid size for each problem. Scenarios are designed for circular robots except for those marked with (*), where robots are any-shape

Scenario	**Agents**	**Feasible**	**Unfeasible**	**Grid Size**
CorridorH	4	819	118	49x19
CorridorL	4	384	92	33x13
ExtremeH	4	1993	1325	51x63
ExtremeL	4	896	589	34x42
MutExH	3	499	513	45x21
MutExL	3	223	242	30x14
MazeH	3	1389	154	38x38
MazeL	3	672	100	25x25
Rooms2r	2	192	52	13x13
Rooms10r	10	192	52	13x13
Extreme10r	10	1442	589	34x42
Maze10r	10	672	100	25x25
RoomsH*	2	835	182	28x28
RoomsL*	2	131	61	13x13

For the experiments on this Section, the actuation model is always considered to be equal to the robot footprint.The Actuation Map determination was developed in C++.

8.9.2 Experiments on the Coverage problem

In this Section we show some experiments that test the impact of the preprocessing in our approach, called MAP. In these experiments, MAP only contains the first two steps of PMR: individual planning and merging. As it was previously said, we have modeled five different scenarios that include up to four agents with different sizes, and thus different actuation capabilities. Planning results are shown using as metrics the time in seconds, the length of the resulting plan and the makespan. In non-temporal domains, we refer as makespan the length of the parallel plan (number of execution steps, where several actions can be executed at the same execution step). Given that we are dealing with MAP tasks that have no interactions, it is expected that agents can execute their actions in parallel whenever possible.

Four different configurations of our approach have been set up:

- MAP-LB-EC with estimated-cost information (EC). EC refers to the configuration that combines Actuation Maps and MAP.
- MAP-BC-EC with estimated-cost information (EC), also combining Actuation Maps and MAP.
- MAP-LB, same as before but without EC information.
- MAP-BC same as before but without EC information.

As it was mentioned in Section 3.2.4, in low-interaction domains, applying the LB strategy fosters the parallelization of actions, which minimizes the *makespan* metric. The BC strategy focuses on minimizing the *plan length* metric. We also run the problems without the preprocessing stage in order to evaluate our impact in terms of computation time and plan quality.

Furthermore, the following state-of-the-art planners have been chosen as a comparison baseline:

- LAMA [Richter and Westphal, 2010], centralized planer and winner of IPC 2011.
- YAHSP [Vidal, 2004], a greedy centralized planner.
- ADP [Crosby, 2015], a multi-agent planner that automatically detects agents.

- SIW [Muise et al., 2015], a multi-agent planner that factorizes the problem into subproblems solving one atomic goal at a time until all atomic goals are achieved jointly.

- CMAP [Borrajo and Fernández, 2015], a multi-agent planner that employs a centralized approach to solve the problem.

The three multi-agent planners that have been chosen participated on the 1st Competition of Distributed and Multi-agent Planners (CoDMAP[1]) and obtained good results on the final classification.

Neither of these five planner perform a goal allocation phase separated from the planning process. Thus, we had to test them using the equivalent PDDL problems that do not contain unfeasible goals. Also, in order to fairly compare the results of the makespan metric, we had to apply our parallelization algorithm to the resulting plans of ADP and SIW, as they only return the sequential plan.

We have generated two problems per scenario, one of them with less number of waypoints (which we identify as L in tables) and the other one with a high density of waypoints (H), except for the last two scenarios that only have one density level (*Maze* and *Rooms*), making it a total of eight problems. The *Rooms* scenario works for any-shape robots while the rest work for circular robots. Before discussing the results on the tables we need to clarify that a maximum of two hours was given to each planner to solve each scenario. YAHSP results do not appear in the tables because it could not solve any of the scenarios.

The maximum time spent on the preprocessing for any scenario was 170 milliseconds, for the Extremities problem with 4 robots. We included the preprocessing times (to generate the AMs) in the GA column of Table 8.2, and in the total time in Tables 8.3. Hardware used for running the planner was IntelXeon 3,4GHz QuadCore 32GB RAM. AMs were computed using a 2.5GHz DualCore 6GB RAM. Table 8.2 shows the remarkable impact that information from Actuation Maps (AMs) has in combination with the MAP algorithm. Goal assignment (GA) times in Table 8.2 are minimal (MAP-LB-EC) in comparison with the ones of MAP-LB, where it needs to compute the relaxed plans for every goal-agent pair. Even though the individual planning and parallelization time for MAP-LB-EC is similar to MAP-LB, the time gains in GA completely dominate the overall planning time. Before running any problem, MAP performs a MAP compilation of the original problem to generate each agent's individual problem after goals are assigned (*M'*

[1]http://agents.fel.cvut.cz/codmap/

task). Usually this transformation takes seconds and it is included in GA time. However, we observed that given the size and complexity of any-shape scenarios (*RoomsH* and *RoomsL*), the compilation time increases considerably and becomes more than half of the time spent on solving the task. This phenomenom is marked with + in column Total time from Table 8.2.

Table 8.2: Detailed time results in seconds for the MAP algorithm using the Load Balance strategy with and without estimated cost information. From left to right total time, goal assignment time, individual planning time and parallelization time. Symbol + indicates that the MAP compilation time is very high.

	Time (s)							
	MAP-LB-EC				MAP-LB			
Scenario	Total	GA	Planning	Parallel	Total	GA	Planning	Parallel
CorridorH	88.07	0.90	68.34	18.83	1748.38	1672.46	58.83	17.09
CorridorL	13.37	0.29	10.54	2.54	484.48	179.19	303.28	2.01
ExtremH	639.54	4.53	427.91	207.10	timeout			
ExtremL	86.78	1.22	64.11	21.45	5491.00	5377.80	84.64	28.56
MutExH	10.59	0.44	7.36	2.79	1276.18	1265.62	7,78	2.78
MutExL	2.10	0.11	1.64	0.35	103.40	100.72	2.31	0.37
MazeH	1200.37	0.69	944.21	255.47	timeout			
MazeL	1179.21	0.24	1152.71	26.26	882.93	603.78	240.3	38.85
Rooms2r	0.19	0.05	0.05	0.09	8.31	5.70	1.85	0.76
Rooms10r	4.70	0.19	4.17	0.34	40.54	36.34	3.84	0.36
Extreme10r	93.79	2.97	68.89	21.93	timeout			
Maze10r	36.37	0.81	28.74	6.82	2121.11	2070.49	39.6	11.02
RoomsH	6180.37$^+$	9.85	1022.19	1922.87	timeout			
RoomsL	448.63$^+$	2.31	2.54	95.05	timeout			

Regarding time results in Table 8.3, MAP-LB-EC is generally faster if all total times are summed up except in Maze. Also, the impact of combining information from AMs with MAP can be easily appreciated if columns from MAP-LB-EC and MAP-LB are compared. The same happens with BC configurations. Our two configurations MAP-LB-EC and MAP-BC-EC solved every problem. There is an exception in *RoomsH*, in which the parallel plan of both solutions could not be obtained in the remaining time before the 7200 seconds were reached. In general, the easiest scenario to be solved using planning is the Mutual Exclusive (*MutExH, MutExL*) because it is designed for each robot to traverse a mutual exclusive subset of waypoints. This is the reason why time results are very similar among all planners except for MAP-LB and MAP-BC where the planner needs to compute the relaxed plans for each pair robot-goal. However, CMAP had some trouble during

Table 8.3: Total time results in seconds. From left to right MAP with estimated-cost information in Load-balance (LB-EC); MAP without estimated cost information in LB; MAP with estimated cost information in Best-cost (BC-EC); MAP without estimated cost information in BC; ADP, SIW and CMAP are other multi-agent planners and LAMA is a centralized planner. Symbol * indicates that the planner solved the problem but parallelization over-passed the alloted time (7200s).

	Total Time (s)							
	MAP				Other approaches			
Scenario	LB-EC	LB	BC-EC	BC	SIW	ADP	CMAP	LAMA
CorridorH	**88.07**	1748.38	304.91	1791.85	129.71	484.48	1761.82	95.45
CorridorL	13.37	484.48	33.78	203.00	**10.98**	85.97	187.57	22.74
ExtremH	**639.54**	timeout	642.04	timeout	1923.32*	439.58*	timeout	timeout
ExtremL	86.78	5491.00	**82.51**	5547.00	156.71	402.15	91.24	72.07
MutExH	10.59	1276.18	10.55	1277.47	11.65	**6.15**	1277.89	6.93
MutExL	2.1	103.40	2.09	97.27	**0.81**	0.89	96.38	1.06
MazeH	1200.37	timeout	2718.96	timeout	**429.87**	meml.	2575.24	2005.42
MazeL	1179.21	882.93	161.03	1213.26	**37.92**	meml.	timeout	334.8
Rooms2r	**0.19**	8.31	2.64	8.42	0.77	2.24	7.32	1.54
Rooms10r	4.7	40.54	4.18	36.06	2.26	**2.04**	34.21	2.55
Extrm10r	93.79	timeout	98.44	timeout	258.72	169.68	timeout	**76.56**
Maze10r	36.37	2121.11	29.63	2110.11	61.95	**20.04**	2099.74	29.7
RoomsH	6180.37*	timeout	6120.28*	timeout	timeout	timeout	timeout	timeout
RoomsL	448.63	timeout	447.56	timeout	286.34	140.1	timeout	**132.02**

planning in the high density scenario. The circular robot version of *Rooms* (*Rooms2r*) is also very easy to solve, even though the number of goals is higher than the any-shape version (*RoomsL*). If times from *Rooms2r* and *RoomsL* are compared, the complexity of just changing from circular to any-shape robots can be empirically appreciated. ADP reached the memory limit in *Maze* when planning the solutions before the two hours limit. Even though ADP is a multi-agent planner, the effort of computing plans in big-size environments when all goals are assigned to all agents is very big. LAMA has the same issue as ADP because of its centralized approach (*Maze, Extremities, RoomsH*). From the set of planners that we chose to compare our approach, SIW is the one that obtains the best results.

Table 8.4 shows the results regarding the plans' length and Table 8.5 the results regarding makespan. We have used the words *timeout* to indicate that a planner consumed the alloted time and could not return a solution, *memlimit* to indicate that the planner's memory limit was reached before timeout and *parallel* to indicate that the

Table 8.4: Plan length: from left to right MAP with estimated-cost information in Load-balance (MAP-LB-EC); MAP without estimated cost information in LB; MAP with estimated-cost in Best-Cost (MAP-BC-EC); MAP without estimated cost information in BC; SIW, ADP, CMAP and LAMA. Symbol * indicates that the planner solved the problem but parallelization over-passed the alloted time (7200s).

	Plan Length							
	MAP				Other approaches			
Scenario	LB-EC	LB	BC-EC	BC	SIW	ADP	CMAP	LAMA
CorridorH	1511	1512	1653	1556	1543	3545	1541	**1471**
CorridorL	727	692	784	791	699	**627**	746	748
ExtremH	**3830**	timeout	**3830**	timeout	3580*	8687*	timeout	timeout
ExtremL	1715	1850	1715	1786	1627	2848	1659	**1546**
MutExH	**658**	**658**	**658**	**658**	758	773	**658**	**658**
MutExL	**301**	**301**	**301**	**301**	306	302	**301**	**301**
MazeH	3358	timeout	3004	timeout	2570	meml.	**1353**	2686
MazeL	1599	1387	1434	1441	**1236**	meml.	timeout	1345
Rooms2r	**142**	333	313	315	303	398	302	319
Rooms10r	292	344	287	286	**266**	318	268	268
Extrm10r	1604	timeout	1627	timeout	**1534**	2692	timeout	1612
Maze10r	1480	1852	1308	1261	**1202**	1452	1224	1253
RoomsH	1403*	timeout	1337*	timeout	timeout	timeout	timeout	timeout
RoomsL	370	timeout	**336**	timeout	**366**	627	timeout	370

planner solved the problem but the parallelization algorithm could not return a solution in the remaining time to reach 7200s (SIW, ADP in *ExtemeH*; MAP-LB-EC, MAP-BC-EC in *RoomsH*). The best configurations overall regarding plan length are MAP-BC-EC and SIW. Moreover, MAP-LB-EC configuration is generally the best for reducing makespan. Configurations MAP-LB or SIW also obtain good results in specific scenarios. This issue can be explained by the discretization errors from Equation 8.5, which are greater when the downsampling rate is bigger. When allocating goals, the estimation costs are the only guide for the MAP algorithm. The consequence of having slightly inaccurate cost estimates results in the allocation of some goals to different agents than the ones that the estimated costs from the relaxation of plans would suggest. However, this issue does not have a big impact on makespan and plan length results.

From the set of planners chosen to compare our approach, SIW obtains the best performance on time, plan length and makespan. SIW is able to solve most of the scenarios due to its serialization of goals. The importance of factorizing a MAP problem is a conclusion that can be extracted after observing Tables 8.4 and 8.5, as the planners

Table 8.5: Makespan: from left to right MAP with estimated-cost information in Load-balance (MAP-LB-EC); MAP without estimated cost information in LB; MAP with estimated-cost in Best-Cost (MAP-BC-EC); MAP without estimated cost information in BC; SIW, ADP, CMAP and LAMA.

	Makespan							
	MAP				Other approaches			
Scenario	LB-EC	LB	BC-EC	BC	SIW	ADP	CMAP	LAMA
CorridorH	609	**583**	1353	862	973	1699	717	698
CorridorL	298	**256**	657	601	265	1073	444	452
ExtremH	**905**	timeout	**905**	timeout	parallel	parallel	timeout	timeout
ExtremL	**376**	702	376	1091	450	1378	564	484
MutExH	**117**	**117**	**117**	**117**	149	155	**117**	**117**
MutExL	**58**	**58**	**58**	**58**	64	**58**	**58**	**58**
MazeH	1631	timeout	1631	timeout	1941	meml.	timeout	**1369**
MazeL	**696**	993	1035	1384	837	meml.	1288	1217
Rooms2r	264	262	264	271	214	342	**212**	237
Rooms10r	**47**	55	57	54	59	73	67	67
Extrm10r	258	timeout	258	timeout	**214**	816	timeout	552
Maze10r	246	367	213	246	**212**	288	221	243
RoomsH	parallel	timeout	parallel	timeout	timeout	timeout	timeout	timeout
RoomsL	240	timeout	242	timeout	180	510	timeout	**177**

that do not perform factorization (LAMA, ADP, CMAP, YAHSP) have to solve bigger and more complex tasks.

Regarding our configurations, MAP-BC-EC and MAP-LB-EC perform better in general than equivalent configurations without estimation costs. On the other hand, the lower the number of agents used to plan, the harder the planning task. Total time in BC is usually worse than in LB configurations on scenarios with higher density of waypoints and multiple robots to plan (*CorridorH, MazeH*).

8.9.3 Experiments Detecting Potential Collisions

In this Section we show the results obtained on the same scenarios as in the previous Section but using instead the PDDL domain that detects collisions described in Section 8.7. In Tables 8.6 and 8.7, we refer to MAP&R-LB-EC as running the Algorithm 1 using the LB strategy. MAP&R-BC-EC runs the BC strategy instead. We have also compared our approach against the same set of planners as in Section 8.9.2. The maximum time for each planner to solve each scenario is two hours. YAHSP results are not shown in the tables as it was not able to solve any problem.

The aim of this experiment is to analyze the impact of detecting and fixing collisions on makespan and time metrics. Plan length is not relevant on this experiment, as the difficulty lies on the planner's ability to manage several agents and collision avoidance at the same time. That is a feature that directly affects the makespan result.

Regarding time results in Table 8.6, it can be seen that the number of problems solved decreases considerably. Also, time results have increased in all planners. This is due to the collision avoidance effect. On one hand, centralized approaches can deal with it more easily, as the master agent has the whole control of the agents. However, it is still facing the same issue as in the previous experiments: the tasks are harder to solve and now the search space is bigger.

Table 8.6: Time in seconds from left to right MAP with estimated-cost information in Load-balance (MAP&R-LB-EC); MAP with estimated-cost in Best-Cost (MAP&R-BC-EC); SIW, ADP, CMAP and LAMA.

	Total Time (s)					
Scenario	MAP&R-LB-EC	MAP&R-BC-EC	SIW	ADP	CMAP	LAMA
CorridorH	timeout	timeout	263.82	timeout	timeout	293.11
CorridorL	73.57	81.51	50.64	193.54	202.16	53.24
ExtremH	timeout	timeout	timeout	timeout	timeout	timeout
ExtremL	timeout	timeout	752.5	timeout	timeout	timeout
MutExH	timeout	timeout	7.61	9.01	1328.77	70.08
MutExL	3.25	3.35	1.35	1.06	104.45	3.25
MazeH	timeout	timeout	986.17	memlimit	timeout	timeout
MazeL	timeout	timeout	67.34	memlimit	945.04	timeout
Rooms2r	4.78	4.71	1.5	5.85	8.47	2.24
Rooms10r	7.11	7.12	4.86	3.51	40.42	9.62
Extreme10r	timeout	timeout	timeout	timeout	timeout	timeout
Maze10r	406.75	timeout	466.03	memlimit	2196.02	560.40
RoomsH	timeout	timeout	timeout	timeout	timeout	timeout
RoomsL	3406.21	3092.80	timeout	timeout	timeout	timeout

Our approach is halfway between the centralized and the distributed approach. The first part of our algorithm is distributed while the plan-reuse phase is centralized. Thus, the success of our algorithm depends on the number of collisions and the difficulty of solving them. LPG-ADAPT focus first on reutilizing the greater possible number of the actions from the invalid plan. When a collision is detected, LPG-ADAPT will search for a valid action on the part of the search space that is closer to the invalid action and its current planning state. This causes LPG-ADAPT to iteratively explore the search space starting from a very concrete section. The exploration distance will be increased as long as the valid action is still not found. This approach works well if the collision requires a small change to be fixed i.e. it only affects to a couple of navigation steps - the solution can be found near the search space of the action and current state. But if the way to avoid the collision affects to a bigger part of the plan i.e. robots have to move back several waypoints and change path directions, LPG-ADAPT might get stuck on the search space, as it will try to search first on the space closer to the invalid action and the solution might be far away from there. Thus, *timeout* will be reached before a solution is found. Scenarios not solved by our approaches on Table 8.6 fail for that reason. On the other hand, Multi-Agent centralized approaches as SIW solve more problems. Next paragraphs contain a discussion on this aspect. We analyze why this particular situation is given with SIW even though MAP centralized approaches are generally worse in performance on big scenarios.

SIW only solves one atomic goal at a time (serialization), which means that goals are not assigned to agents in the first step of the algorithm. The process is interleaved with search. Thus, only one estimation is computed per iteration and current positions of the agents are updated after each goal is reached. Agents work individually but coordination (and thus, collisions) are checked after each iteration. The centralized approach followed by SIW is very efficient on the coverage problem. Collisions can be fully avoided because of solving first only one goal at a time and then updating robots' positions. Thus, the algorithm obtains good results in number of problems solved and time. However, when the size of the problem increases, as in the any-shape scenarios, SIW has more difficulties to solve the problem in time. This scenario penalizes SIW because the search space is huge in comparison with circular robot scenarios. Planning one goal at a time following a centralized approach now becomes a worse choice. SIW has to usually deal with the following situation when collisions are given: a set of goals has been reached and the next goal on the list cannot be achieved unless the previous part of the plan is partially modified. In any-shape scenarios this aspect takes more time to fix given the size of the search space.

Regarding our approach, even though we obtain estimation costs through AMs, they are not as effective in guidance as SIW's serialization. The estimation of costs is provided to our approach at the beginning. If agents have to modify their route due to collisions, their estimations and assignment of goals might not be as useful as in the beginning. It can even penalize the agent's performance. Also, if an agent needs to change its route several times, it could mean that the original assignment of goals is completely useless.

Table 8.7: Makespan: from left to right MAP with estimated-cost information in Load-balance (MAP&R-LB-EC); MAP with estimated-cost in Best-Cost (MAP&R-BC-EC); SIW, ADP, CMAP and LAMA.

	Makespan					
Scenario	MAP&R-LB-EC	MAP&R-BC-EC	SIW	ADP	CMAP	LAMA
CorridorH	timeout	timeout	1289	timeout	timeout	847
CorridorL	801	806	490	1378	452	424
ExtremH	timeout	timeout	timeout	timeout	timeout	timeout
ExtremL	timeout	timeout	704	timeout	timeout	timeout
MutExH	timeout	timeout	158	211	131	135
MutExL	87	87	80	80	101	95
MazeH	timeout	timeout	1061	memlimit	timeout	timeout
MazeL	timeout	1481	758	memlimit	1139	timeout
Rooms2r	293	293	189	465	228	250
Rooms10r	61	61	61	107	77	54
Extreme10r	timeout	timeout	timeout	timeout	timeout	timeout
Maze10r	293	timeout	432	memlimit	221	282
RoomsH	timeout	timeout	timeout	timeout	timeout	timeout
RoomsL	250	250	timeout	timeout	timeout	timeout

Also, plan reuse planners are not efficiently prepared to perform an extensive search. They would rather prefer to reutilize actions from previous plans, which in the coverage problem results in generating redundant actions around the planning task. We have tried different features of LPG-ADAPT (low memory, speed mode, increasing different fixed constants...) to check if its performance could be improved but neither of them helped.

Regarding makespan results from Table 8.7, MAP&R-LB-EC and SIW are the two planners that obtain the best results. Although the *RoomsL* scenario might seem easier to solve by just looking at Table 8.1. However, as it works for any-shape robots, the grid of waypoints is bigger and harder to navigate from the planning point of view. The search space is very big and thus centralized approaches are especially penalized. The reason of failing on the *Extremities* and *Maze* scenarios is due to the changes on the robots' paths caused by the collision avoidance or the topology of the scenario. Those scenarios contain narrower areas and large halls where only some robots can reach the end. Thus, robots might spend a lot of time looking for the correct path while at the same time avoiding the rest of the agents.

As a final conclusion, we would like to discuss the overall performance of our contribution. MAP-LB-EC and MAP-BC-EC clearly complement each other on the set of proposed scenarios. This is an advantage, as the algorithm can be easily adapted to different situations and environments. It is true that we lost some performance on collisions, but we have empirically shown that it is also related to the topology of the scenario and the coverage problem itself. The unexpected advantage of SIW in the experiments detecting collisions has also been addressed and analyzed. The serialization of goals and the nature of the coverage problem where interactions are given occasionally, makes any centralized (Multi-agent or Single-agent) planner to behave well. LAMA is closer to SIW and our approaches in that sense, and it is not even a MAP algorithm. We also want to put in value the scalability of our approach. Through the experiments we have shown that we can successfully deal with different topologies, number of agents, agent's orientation, huge planning tasks, unfeasible goals, independent goal-assignment and pre-processed estimation costs. State-of-the-art planners are not used to satisfy all these features at the same time.

8.10 Further improvements: Contract-Net and simplifying the model

In order to improve the latest experiments, with and without collisions, we had to pay attention to the behavior of the resulting plans.

On one hand, depending on the topology of the scenario, the agents' paths generated after planning could be optimized if the accumulated cost is taken into account during the Goal-Assignment process. Thus, we proposed to run again the set of problems using instead the *Contract-net* goal strategy explained in Section 3.3.3.

On the other hand, we were obtaining a considerable amount of *timeouts* when detecting collisions. As the environments have a considerable size and each of the robots starts at different random positions, we thought that maybe the domain could be simplified in some way. That is how we found that the planner was extra-instantiating the predicate *occupied* due to the current modelling of the domain. We were double-checking that both positions (current and destination) were not occupied. As robots plan individually on the first step, there was no reason to include `(occupied ?y)` in the preconditions of the `navigate` action from Figure 8.4. The only occupied position at that point is the current position of the robot. Besides, as the robot moves, the occupied position is updated on the effects.

Thus, we can eliminate that precondition `(occupied ?y)` and maintain the rest. We can also use this `navigate` version on the plan reuse phase as long as the initial positions

Table 8.8: Makespan: from left to right in MAP with estimated-cost information in Load-balance (MAP-LB-EC), Best-cost (MAP-BC-EC) and Contract-net (MAP-CN-EC). Also, results from SIW are included, as it was the best approach.

	Makespan - No Collisions			
	MAP			Other approach
Scenario	LB-EC	BC-EC	CN-EC	SIW
CorridorH	609	1353	**543**	973
CorridorL	298	657	**232**	265
ExtremH	**905**	**905**	1442	1712
ExtremL	**376**	**376**	620	450
MutExH	**117**	**117**	**117**	149
MutExL	**58**	**58**	**58**	64
MazeH	1631	1631	**1185**	1941
MazeL	696	1035	**584**	837
Rooms2r	264	264	228	**214**
Rooms10r	**47**	57	58	59
Extreme10r	258	258	485	**214**
Maze10r	246	213	383	**212**
RoomsH	parallel	parallel	parallel	parallel
RoomsL	240	242	274	**180**

of the robots are marked as `occupied`, which means that the scenario has been correctly modelled.

The following Tables show the makespan results obtained with *Contract-net* and the makespan results obtained after rerunning the collision experiments with the new domain. Table 8.8 shows how the new goal-strategy Contract-net works better in scenarios with narrow and multiple halls, as in the Maze. The real cost of having one goal waypoint in a certain hall and another one assigned to the same robot but on the neighbour hall is very high. This behavior is greatly reduced with Contract-net. The same phenomenon is given in the Corridor scenario, even though there are no halls. As a result, the robots obtain optimized paths to actuate every waypoint from each room. Finally, Table 8.9 shows a noticeable improve in our configurations regarding the number of problems solved in comparison with Table 8.7. The SIW planner is still better in some scenarios such as *Extreme* but the difference in performance is now small. Also, the BC configuration works better than LB or CN in scenarios where a higher number of planning agents are available (*Extreme10r, Maze10r*). This is due to the fact that neither LB or CN can easily avoid to plan with all the available agents, which ends up in having larger and more redundant plans.

Table 8.9: Makespan: from left to right MAP with estimated-cost information in Load- (MAP&R-LB-EC), Best-cost (MAP&R-BC-EC) and Contract-net (MAP&R-CN-EC). Also, results from SIW are included, as it was the best approach.

	Makespan - Collisions			
	MAP &R			Other approach
Scenario	LB-EC	BC-EC	CN-EC	SIW
CorridorH	**836**	1551	**836**	1289
CorridorL	**248**	762	249	490
ExtremH	3537	2027	3537	**1232**
ExtremL	1648	937	1643	**704**
MutExH	**99**	**99**	**99**	158
MutExL	**71**	**71**	**71**	80
MazeH	timeout	timeout	timeout	1061
MazeL	1293	1481	1287	**758**
Rooms2r	179	**57**	172	61
Rooms10r	371	277	371	**189**
Extreme10r	1658	**497**	1667	timeout
Maze10r	1415	**414**	1443	432
RoomsH	timeout	timeout	timeout	timeout
RoomsL	250	**248**	250	timeout

Part V

Conclusions

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

IN THIS CHAPTER we summarize the contributions of this Thesis, present the conclusions and describe future directions of our work.

9.1 Conclusions

In this thesis, we have successfully addressed the challenges presented in Chapter 1.

First, we contributed with PMR, an algorithm capable of solving a MAP task by merging individual plans and applying plan-reuse techniques. A key feature of PMR is that automatically adapts to the interaction level among agents and goals, varying its behavior from distributed to centralized. It generates individual plans and merges them in the merging phase (M) with a low computational effort if the domain has a low degree of interaction. Otherwise, it uses plan reuse (R) in domains with more interaction and resorts to centralized planning (C) in case of domains with strong interactions.

Moreover, PMR can easily be configured to target coverage, cost or makespan by just changing its goal allocation strategy. The aim was to focus on minimizing the makespan in big-size problems that have low interaction. Actually, the potential of PMR was neither appreciated on CoDMAP results nor on the experiments that changed the agentification. Thus, we moved into harder problems in terms of size. Therefore, the results of PMR-RRPT-PLAN have shown that it easily adapts to any type of MAP problem, independently of the problem's features (e.g. number of agents, goals, interactions). It specifically adapted with success to those that had little interaction and a topology that fostered the goal-assigment to divide indirectly the available space of actuation among the agents. PMR-RRPT-PLAN maintains good results on coverage and time and remarkable results on makespan, specially in combination with the *Load-balance* strategy.

Another advantage of PMR is that it only includes off-the-shelf planners on its three phases (M, C and R). Hence, we can trivially improve the performance of PMR by just changing the planners used by better ones once they are developed.

Regarding plan reuse, the second contribution presented was RRPT-PLAN, an algorithm that combines plan reuse, sampling and local search to solve a planning problem. RRPT-PLAN receives the domain, the problem and an input plan (usually invalid) from which it will try to reuse actions to include in the final plan. We have shown in the experiments that RRPT-PLAN adapts to diverse plan reuse scenarios, including the ones that are not usually considered by state-of-the-art plan reuse planners. This aspect was tested using the Hammers domain.

Third, we contributed with a joint work that showed how to combine information from Actuation Maps with Multi-Agent Planning to solve multi-robot path planning problems. The key was to skip the computation of estimated cost during planning, which was our main bottleneck to scale. We employed Actuation Maps in a preprocessing step to determine the feasibility of pairs robot-goal and to extract an estimated cost. That cost was used later to avoid the computation of relaxed plans during Goal-Assignment. The environment map was discretized into a grid of waypoints. The goals were distributed thanks to a goal-allocation algorithm and unfeasible goals identified and discarded from the planning task. Then, the planning task was factorized for each robot. Each robot generates its individual path, that results in a maximal space coverage in terms of actuation. We also contributed with and adapted version of PMR to fix agents' interactions after the individual planning phase. On the experiments we designed a total of six scenarios, five for circular robots and one for any-shape robots.

Our approach greatly reduces the GA time. We were able to also reduce the overall planning time when preprocessed information was provided to the MAP algorithm. The gains in performance depend greatly on the topology of the environment and the characteristics of each robot.

Finally, experiments from the third contribution also proved that when solving big size multi-agent problems using planning, it is essential to first factorize the problem into subtasks. Also, it is helpful when working on problems that explicitly involve agents' interactions. Experiments on collision avoidance showed the importance of task factorization and the topology of the scenario in order to successfully fix collisions.

9.2 Future Work

There are different research lines and ideas that could define future directions of our contributions.

Regarding PMR, we could work on different plan merging strategies to merge more efficiently the individual plans obtained from the agents. Also, we could work on new policies and techniques that improve the goal-assignment process.

Regarding RRPT-PLAN, we could allow probabilities to change their values over planning time to adapt the algorithm to different situations e.g. decrease reuse after all the input plan has been evaluated.

Regarding the combination of Actuation Maps and Multi-Agent Planning, we would like to extend the preprocessing technique to other domains and consider different -robot or agent- models. Our approach can be easily extended to path planning tasks or real-time strategy videogames. We gave some examples of the former such as surveillance tasks or search and rescue tasks. Videogames could be an interesting domain to explore when designing bots that play automatically. Our approach could improve the player/bot performance when extracting information from the map to decide which goals are more relevant to achieve first. On the other hand, we also would like to improve the performance of fixing interactions. Plan reuse works well when collisions only affect to a couple of actions. For biggest plan modifications plan-reuse is not enough. We would also like to study in the future the possibility of using plan-reuse in order to deal with dynamic environments during plan execution. We would need to create a technique to efficiently update the PDDL problem file according to the changes detected in the environment regarding obstacles.

APPENDIX A

PLANNING DOMAINS

PLANNING BENCHMARKS mainly consist on sets of domains and problems that represent different environments or situations. In order to help the reader, here we provide a brief explanation of the domains employed on the experiments of this Thesis. The origin of most of these domains were the classical International Planning Competition. They all had been adapted to multi-agent environments.

- **Blocks:** It comes from the classical Blocksworld. In the original version, a set of blocks should be piled up into a tower following a certain ordering by using a robotics hand. In the multi-agent environment there exist multiple hands available to pick up blocks.
- **Depots:** It can be described as a blocksworld version that includes transportation. The aim is to move the crates between depots and distributors. At the beginning, crates are placed on pallets the middle of both. In order to move the crates, distributors use trucks. A crate can be placed on a hoist, at a depot or on a truck.
- **Depots-robots:** The aim is to use robots to deliver specific pods to human workers. The environment simulates a warehouse. Robots can pick up pods and move through the environment. Pods are fixed in columns. Humans are situated in line in the lower row.
- **Driverlog:** The aim is to deliver a set of packages to different locations. In order to do that, there exist drivers which either drive the trucks between locations or walk

to reach a truck. Locations are mapped into a grid. If two locations are connected, it means that the driver can drive from one point to the other.

- **Elevators:** There is a set of passengers located in different floors of a building. They need to be transported to a different floor. Thus, there exist a set of fast and slow elevators that will carry out that mission dividing their resources between the different floors.

- **Hammers:** There are some rooms in which a set of robots have to hang up specific paintings. Rooms are connected by halls. A robot needs a hammer and a nail in order to hang up the painting successfully.

- **Logistics:** Another transportation task similar to Driverlog. There are several cities and several packages to be delivered. Also, there are sets of trucks and airplanes to deliver the packages. Drivers are implicitly included on each truck. Airplanes are allowed to travel between cities while trucks can only drive inside a city. Thus, cooperation is usually needed to achieve the goals.

- **Rover:** It it simulation of a future space environment e.g. Mars ground. A space robot (Rover) has to pick up samples and communicate the results to the base. They have different tools available to perform each task. Rovers work individually by moving through a grid of waypoints.

- **Rover-graph:** The goals to achieve are the same as in Rover. The main difference can be found in the size of the grid of waypoints and its connections. There are two big islands of waypoints only connected by one edge.

- **Satellite:** In an outer-space environment there are several satellites with different capabilities. The aim is to use different tools to perform different task e.g. take pictures, get samples. Some tools need to be powered and calibrated in advance. Satellites work individually.

- **Sokoban:** Simulation of the classical puzzle game where players need to push boxes to certain cells of the game environment in order to win. This environment is similar to a maze and neither walls nor blocks can be traversed. Thus, players can block themselves or reach a dead-end if they push boxes in the wrong direction.

- **Taxi:** This is a new domain designed for CoDMAP. It reflects an on-demand transportation problem between taxis and passengers inside a connected grid of roads.

Taxis can drive only one person per ride and can drop the person off into a location only if it is free-of-taxis.

- **VRP:** Emulates the classical vehicle routing problem. There is a grid of roads where each road (edge of the graph) has a certain cost assigned. Thus, the aim is to deliver a set of packages into a set of locations by minimizing the cost of traversing each of the roads. The best solution would be the one that delivers every package and sums the least cost.
- **Wireless:** This is a new domain designed for CoDMAP. It emulates a wireless sensor network where data messages that are initially allocated to different nodes of the network have to be sent to a base station. Node sensors can transmit the messages with their node neighbours (wireless radio reach each other). Each time a message is sent, the node sensor decreases its energy in one.
- **Woodworking:** The aim is to process raw wood to obtain pieces of wood with a certain color or shape. In order to do that, the domain provides different tools, which act as agents of the environment.
- **Zenotravel:** There are several planes with limited fuel whose aim is to transport different passengers to different airports. Also, some of the planes should be parked at a concrete airport after delivering every passenger.

APPENDIX B

APPENDIX B: PLANS OBTAINED IN THE HAMMERS DOMAIN

THIS APPENDIX contains the resulting plans obtained with PMR-LPG-ADAPT and PMR-RRPT-PLAN in Section 7.3. The aim is to show in detail the differences of LPG-ADAPT and RRPT-PLAN when applying plan-reuse and search.

Figure B.1: Resulting plan obtained by PMR-LPG-ADAPT and PMR-RRPT-PLAN for the scenario shown in Figure 7.1a

```
0: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM6)
0: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM5)
0: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM4)
0: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM3)
0: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM2)
0: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM1)
1: (PICK_UP_HAMMER ROBOT1 HAMMER6 ROOM6)
1: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM6)
1: (PICK_UP_HAMMER ROBOT2 HAMMER5 ROOM5)
1: (PICK_UP_NAIL ROBOT2 NAIL5 ROOM5)
1: (PICK_UP_HAMMER ROBOT3 HAMMER4 ROOM4)
1: (PICK_UP_NAIL ROBOT3 NAIL4 ROOM4)
1: (PICK_UP_HAMMER ROBOT4 HAMMER3 ROOM3)
1: (PICK_UP_NAIL ROBOT4 NAIL3 ROOM3)
1: (PICK_UP_HAMMER ROBOT5 HAMMER2 ROOM2)
1: (PICK_UP_NAIL ROBOT5 NAIL2 ROOM2)
1: (PICK_UP_HAMMER ROBOT6 HAMMER1 ROOM1)
```

```
 1: (PICK_UP_NAIL ROBOT6 NAIL1 ROOM1)
 2: (HANG_PAINTING ROBOT1 PAINTING6 HAMMER6 NAIL6 ROOM6)
 2: (HANG_PAINTING ROBOT2 PAINTING5 HAMMER5 NAIL5 ROOM5)
 2: (HANG_PAINTING ROBOT3 PAINTING4 HAMMER4 NAIL4 ROOM4)
 2: (HANG_PAINTING ROBOT4 PAINTING3 HAMMER3 NAIL3 ROOM3)
 2: (HANG_PAINTING ROBOT5 PAINTING2 HAMMER2 NAIL2 ROOM2)
 2: (HANG_PAINTING ROBOT6 PAINTING1 HAMMER1 NAIL1 ROOM1)
```

Figure B.2: Resulting plan obtained by PMR-LPG-ADAPT for the scenario shown in Figure 7.1b

```
 0: (PICK_UP_HAMMER ROBOT1 HAMMER1 COMMON_ROOM)
 1: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM6)
 2: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM6)
 3: (HANG_PAINTING ROBOT1 PAINTING6 HAMMER1 NAIL6 ROOM6)
 4: (MOVE_TO_ROOM ROBOT1 ROOM6 COMMON_ROOM)
 5: (DROP_HAMMER ROBOT1 HAMMER1 COMMON_ROOM)
 6: (PICK_UP_HAMMER ROBOT2 HAMMER1 COMMON_ROOM)
 7: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM5)
 8: (PICK_UP_NAIL ROBOT2 NAIL5 ROOM5)
 9: (HANG_PAINTING ROBOT2 PAINTING5 HAMMER1 NAIL5 ROOM5)
10: (MOVE_TO_ROOM ROBOT2 ROOM5 COMMON_ROOM)
11: (DROP_HAMMER ROBOT2 HAMMER1 COMMON_ROOM)
12: (PICK_UP_HAMMER ROBOT3 HAMMER1 COMMON_ROOM)
13: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM4)
14: (PICK_UP_NAIL ROBOT3 NAIL4 ROOM4)
15: (HANG_PAINTING ROBOT3 PAINTING4 HAMMER1 NAIL4 ROOM4)
16: (MOVE_TO_ROOM ROBOT3 ROOM4 COMMON_ROOM)
17: (DROP_HAMMER ROBOT3 HAMMER1 COMMON_ROOM)
18: (PICK_UP_HAMMER ROBOT4 HAMMER1 COMMON_ROOM)
19: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM3)
20: (PICK_UP_NAIL ROBOT4 NAIL3 ROOM3)
21: (HANG_PAINTING ROBOT4 PAINTING3 HAMMER1 NAIL3 ROOM3)
22: (MOVE_TO_ROOM ROBOT4 ROOM3 COMMON_ROOM)
23: (DROP_HAMMER ROBOT4 HAMMER1 COMMON_ROOM)
24: (PICK_UP_HAMMER ROBOT5 HAMMER1 COMMON_ROOM)
25: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM2)
26: (PICK_UP_NAIL ROBOT5 NAIL2 ROOM2)
27: (HANG_PAINTING ROBOT5 PAINTING2 HAMMER1 NAIL2 ROOM2)
28: (MOVE_TO_ROOM ROBOT5 ROOM2 COMMON_ROOM)
29: (DROP_HAMMER ROBOT5 HAMMER1 COMMON_ROOM)
30: (PICK_UP_HAMMER ROBOT6 HAMMER1 COMMON_ROOM)
31: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM1)
```

```
32: (PICK_UP_NAIL ROBOT6 NAIL1 ROOM1)
33: (HANG_PAINTING ROBOT6 PAINTING1 HAMMER1 NAIL1 ROOM1)
```

Figure B.3: Resulting plan obtained by PMR-RRPT-PLAN for the scenario shown in Figure 7.1b

```
 0: (PICK_UP_HAMMER ROBOT1 HAMMER1 COMMON_ROOM)
 1: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM6)
 2: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM6)
 3: (HANG_PAINTING ROBOT1 PAINTING6 HAMMER1 NAIL6 ROOM6)
 4: (MOVE_TO_ROOM ROBOT1 ROOM6 COMMON_ROOM)
 5: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM1)
 6: (PICK_UP_NAIL ROBOT1 NAIL1 ROOM1)
 7: (HANG_PAINTING ROBOT1 PAINTING1 HAMMER1 NAIL1 ROOM1)
 8: (MOVE_TO_ROOM ROBOT1 ROOM1 COMMON_ROOM)
 9: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM2)
10: (PICK_UP_NAIL ROBOT1 NAIL2 ROOM2)
11: (HANG_PAINTING ROBOT1 PAINTING2 HAMMER1 NAIL2 ROOM2)
12: (MOVE_TO_ROOM ROBOT1 ROOM2 COMMON_ROOM)
13: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM3)
14: (PICK_UP_NAIL ROBOT1 NAIL3 ROOM3)
15: (HANG_PAINTING ROBOT1 PAINTING3 HAMMER1 NAIL3 ROOM3)
16: (MOVE_TO_ROOM ROBOT1 ROOM3 COMMON_ROOM)
17: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
18: (PICK_UP_NAIL ROBOT1 NAIL4 ROOM4)
19: (HANG_PAINTING ROBOT1 PAINTING4 HAMMER1 NAIL4 ROOM4)
20: (MOVE_TO_ROOM ROBOT1 ROOM4 COMMON_ROOM)
21: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM5)
22: (PICK_UP_NAIL ROBOT1 NAIL5 ROOM5)
23: (HANG_PAINTING ROBOT1 PAINTING5 HAMMER1 NAIL5 ROOM5)
```

Figure B.4: Sequential plan obtained after merging in the scenario shown in Figure 7.1c

```
0: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM6)
1: (PICK_UP_HAMMER ROBOT1 HAMMER1 ROOM6)
2: (MOVE_TO_ROOM ROBOT1 ROOM6 COMMON_ROOM)
3: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
4: (DROP_HAMMER ROBOT1 HAMMER1 ROOM4)
5: (MOVE_TO_ROOM ROBOT1 ROOM4 COMMON_ROOM)
6: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM2)
7: (PICK_UP_NAIL ROBOT1 NAIL4 ROOM2)
8: (MOVE_TO_ROOM ROBOT1 ROOM2 COMMON_ROOM)
9: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
```

```
10: (PICK_UP_HAMMER ROBOT1 HAMMER1 ROOM4)
11: (HANG_PAINTING ROBOT1 PAINTING6 HAMMER1 NAIL4 ROOM4)
12: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM6)
13: (PICK_UP_HAMMER ROBOT2 HAMMER1 ROOM6)
14: (MOVE_TO_ROOM ROBOT2 ROOM6 COMMON_ROOM)
15: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM4)
16: (DROP_HAMMER ROBOT2 HAMMER1 ROOM4)
17: (MOVE_TO_ROOM ROBOT2 ROOM4 COMMON_ROOM)
18: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM2)
19: (PICK_UP_NAIL ROBOT2 NAIL4 ROOM2)
20: (MOVE_TO_ROOM ROBOT2 ROOM2 COMMON_ROOM)
21: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM4)
22: (PICK_UP_HAMMER ROBOT2 HAMMER1 ROOM4)
23: (HANG_PAINTING ROBOT2 PAINTING5 HAMMER1 NAIL4 ROOM4)
24: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM6)
25: (PICK_UP_HAMMER ROBOT3 HAMMER1 ROOM6)
26: (MOVE_TO_ROOM ROBOT3 ROOM6 COMMON_ROOM)
27: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM4)
28: (DROP_HAMMER ROBOT3 HAMMER1 ROOM4)
29: (MOVE_TO_ROOM ROBOT3 ROOM4 COMMON_ROOM)
30: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM2)
31: (PICK_UP_NAIL ROBOT3 NAIL4 ROOM2)
32: (MOVE_TO_ROOM ROBOT3 ROOM2 COMMON_ROOM)
33: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM4)
34: (PICK_UP_HAMMER ROBOT3 HAMMER1 ROOM4)
35: (HANG_PAINTING ROBOT3 PAINTING4 HAMMER1 NAIL4 ROOM4)
36: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM6)
37: (PICK_UP_HAMMER ROBOT4 HAMMER1 ROOM6)
38: (MOVE_TO_ROOM ROBOT4 ROOM6 COMMON_ROOM)
39: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM1)
40: (DROP_HAMMER ROBOT4 HAMMER1 ROOM1)
41: (MOVE_TO_ROOM ROBOT4 ROOM1 COMMON_ROOM)
42: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM2)
43: (PICK_UP_NAIL ROBOT4 NAIL4 ROOM2)
44: (MOVE_TO_ROOM ROBOT4 ROOM2 COMMON_ROOM)
45: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM1)
46: (PICK_UP_HAMMER ROBOT4 HAMMER1 ROOM1)
47: (HANG_PAINTING ROBOT4 PAINTING3 HAMMER1 NAIL4 ROOM1)
48: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM6)
49: (PICK_UP_HAMMER ROBOT5 HAMMER1 ROOM6)
50: (MOVE_TO_ROOM ROBOT5 ROOM6 COMMON_ROOM)
51: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM1)
```

```
52: (DROP_HAMMER ROBOT5 HAMMER1 ROOM1)
53: (MOVE_TO_ROOM ROBOT5 ROOM1 COMMON_ROOM)
54: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM2)
55: (PICK_UP_NAIL ROBOT5 NAIL4 ROOM2)
56: (MOVE_TO_ROOM ROBOT5 ROOM2 COMMON_ROOM)
57: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM1)
58: (PICK_UP_HAMMER ROBOT5 HAMMER1 ROOM1)
59: (HANG_PAINTING ROBOT5 PAINTING2 HAMMER1 NAIL4 ROOM1)
60: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM6)
61: (PICK_UP_HAMMER ROBOT6 HAMMER1 ROOM6)
62: (MOVE_TO_ROOM ROBOT6 ROOM6 COMMON_ROOM)
63: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM1)
64: (DROP_HAMMER ROBOT6 HAMMER1 ROOM1)
65: (MOVE_TO_ROOM ROBOT6 ROOM1 COMMON_ROOM)
66: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM2)
67: (PICK_UP_NAIL ROBOT6 NAIL4 ROOM2)
68: (MOVE_TO_ROOM ROBOT6 ROOM2 COMMON_ROOM)
69: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM1)
70: (PICK_UP_HAMMER ROBOT6 HAMMER1 ROOM1)
71: (HANG_PAINTING ROBOT6 PAINTING1 HAMMER1 NAIL4 ROOM1)
```

Figure B.5: Resulting plan obtained by PMR-LPG-ADAPT for the scenario shown in Figure 7.1c

```
  0: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM6)
  0: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM6)
  0: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM6)
  0: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM6)
  0: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM1)
  0: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM6)
  1: (PICK_UP_HAMMER ROBOT1 HAMMER1 ROOM6)
  2: (MOVE_TO_ROOM ROBOT1 ROOM6 COMMON_ROOM)
  3: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM2)
  4: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM2)
  5: (MOVE_TO_ROOM ROBOT1 ROOM2 COMMON_ROOM)
  6: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
  7: (DROP_NAIL ROBOT1 NAIL6 ROOM4)
  8: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM4)
  9: (DROP_NAIL ROBOT1 NAIL6 ROOM4)
 10: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM4)
 11: (DROP_NAIL ROBOT1 NAIL6 ROOM4)
 12: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM4)
 13: (DROP_NAIL ROBOT1 NAIL6 ROOM4)
```

```
14: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM4)
15: (DROP_NAIL ROBOT1 NAIL6 ROOM4)
16: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM4)
17: (DROP_NAIL ROBOT1 NAIL6 ROOM4)
18: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM4)
19: (DROP_NAIL ROBOT1 NAIL6 ROOM4)
20: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM4)
21: (HANG_PAINTING ROBOT1 PAINTING6 HAMMER1 NAIL6 ROOM4)
22: (MOVE_TO_ROOM ROBOT1 ROOM4 COMMON_ROOM)
23: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
24: (DROP_HAMMER ROBOT1 HAMMER1 ROOM4)
25: (MOVE_TO_ROOM ROBOT1 ROOM4 COMMON_ROOM)
26: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM2)
27: (PICK_UP_NAIL ROBOT1 NAIL4 ROOM2)
28: (MOVE_TO_ROOM ROBOT1 ROOM2 COMMON_ROOM)
29: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
30: (PICK_UP_HAMMER ROBOT1 HAMMER1 ROOM4)
 31: (PICK_UP_NAIL ROBOT1 NAIL4 ROOM4)
32: (DROP_NAIL ROBOT1 NAIL4 ROOM4)
33: (PICK_UP_NAIL ROBOT1 NAIL4 ROOM4)
34: (MOVE_TO_ROOM ROBOT1 ROOM4 COMMON_ROOM)
35: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM3)
36: (DROP_NAIL ROBOT1 NAIL4 ROOM3)
37: (MOVE_TO_ROOM ROBOT1 ROOM3 COMMON_ROOM)
38: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM6)
39: (DROP_HAMMER ROBOT1 HAMMER1 ROOM6)
40: (PICK_UP_HAMMER ROBOT2 HAMMER1 ROOM6)
41: (MOVE_TO_ROOM ROBOT2 ROOM6 COMMON_ROOM)
42: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM4)
43: (DROP_HAMMER ROBOT2 HAMMER1 ROOM4)
44: (MOVE_TO_ROOM ROBOT2 ROOM4 COMMON_ROOM)
45: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM3)
46: (PICK_UP_NAIL ROBOT2 NAIL4 ROOM3)
47: (MOVE_TO_ROOM ROBOT2 ROOM3 COMMON_ROOM)
48: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM4)
49: (PICK_UP_HAMMER ROBOT2 HAMMER1 ROOM4)
50: (MOVE_TO_ROOM ROBOT2 ROOM4 COMMON_ROOM)
51: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM6)
52: (DROP_HAMMER ROBOT2 HAMMER1 ROOM6)
53: (PICK_UP_HAMMER ROBOT3 HAMMER1 ROOM6)
53: (MOVE_TO_ROOM ROBOT2 ROOM6 COMMON_ROOM)
54: (MOVE_TO_ROOM ROBOT3 ROOM6 COMMON_ROOM)
```

```
54: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM4)
55: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM2)
55: (MOVE_TO_ROOM ROBOT2 ROOM4 COMMON_ROOM)
56: (PICK_UP_NAIL ROBOT3 NAIL5 ROOM2)
56: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM2)
57: (MOVE_TO_ROOM ROBOT3 ROOM2 COMMON_ROOM)
57: (DROP_NAIL ROBOT2 NAIL4 ROOM2)
58: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM4)
59: (HANG_PAINTING ROBOT3 PAINTING4 HAMMER1 NAIL5 ROOM4)
60: (MOVE_TO_ROOM ROBOT3 ROOM4 COMMON_ROOM)
61: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM4)
62: (DROP_HAMMER ROBOT3 HAMMER1 ROOM4)
63: (MOVE_TO_ROOM ROBOT3 ROOM4 COMMON_ROOM)
64: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM2)
65: (PICK_UP_NAIL ROBOT3 NAIL4 ROOM2)
66: (MOVE_TO_ROOM ROBOT3 ROOM2 COMMON_ROOM)
67: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM4)
68: (PICK_UP_HAMMER ROBOT3 HAMMER1 ROOM4)
68: (DROP_NAIL ROBOT3 NAIL4 ROOM4)
69: (PICK_UP_NAIL ROBOT3 NAIL4 ROOM4)
70: (MOVE_TO_ROOM ROBOT3 ROOM4 COMMON_ROOM)
71: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM5)
72: (PICK_UP_NAIL ROBOT3 NAIL3 ROOM5)
73: (MOVE_TO_ROOM ROBOT3 ROOM5 COMMON_ROOM)
74: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM1)
75: (HANG_PAINTING ROBOT3 PAINTING2 HAMMER1 NAIL3 ROOM1)
76: (MOVE_TO_ROOM ROBOT3 ROOM1 COMMON_ROOM)
77: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM6)
78: (DROP_HAMMER ROBOT3 HAMMER1 ROOM6)
79: (PICK_UP_HAMMER ROBOT4 HAMMER1 ROOM6)
79: (MOVE_TO_ROOM ROBOT3 ROOM6 COMMON_ROOM)
80: (MOVE_TO_ROOM ROBOT4 ROOM6 COMMON_ROOM)
80: (MOVE_TO_ROOM ROBOT3 COMMON_ROOM ROOM2)
81: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM5)
81: (DROP_NAIL ROBOT3 NAIL4 ROOM2)
82: (PICK_UP_NAIL ROBOT4 NAIL1 ROOM5)
82: (PICK_UP_NAIL ROBOT4 NAIL2 ROOM5)
83: (MOVE_TO_ROOM ROBOT4 ROOM5 COMMON_ROOM)
84: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM1)
85: (HANG_PAINTING ROBOT4 PAINTING3 HAMMER1 NAIL2 ROOM1)
85: (DROP_NAIL ROBOT4 NAIL1 ROOM1)
86: (PICK_UP_NAIL ROBOT6 NAIL1 ROOM1)
```

```
 86: (MOVE_TO_ROOM ROBOT4 ROOM1 COMMON_ROOM)
 87: (MOVE_TO_ROOM ROBOT6 ROOM1 COMMON_ROOM)
 88: (DROP_NAIL ROBOT6 NAIL1 COMMON_ROOM)
 89: (PICK_UP_NAIL ROBOT4 NAIL1 COMMON_ROOM)
 89: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM6)
 90: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM4)
 91: (HANG_PAINTING ROBOT4 PAINTING5 HAMMER1 NAIL1 ROOM4)
 92: (MOVE_TO_ROOM ROBOT4 ROOM4 COMMON_ROOM)
 93: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM1)
 94: (MOVE_TO_ROOM ROBOT4 ROOM1 COMMON_ROOM)
 95: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM1)
 96: (DROP_HAMMER ROBOT4 HAMMER1 ROOM1)
 97: (MOVE_TO_ROOM ROBOT4 ROOM1 COMMON_ROOM)
 98: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM2)
 99: (PICK_UP_NAIL ROBOT4 NAIL4 ROOM2)
100: (MOVE_TO_ROOM ROBOT4 ROOM2 COMMON_ROOM)
101: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM1)
102: (PICK_UP_HAMMER ROBOT4 HAMMER1 ROOM1)
103: (MOVE_TO_ROOM ROBOT4 ROOM1 COMMON_ROOM)
104: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM6)
105: (DROP_HAMMER ROBOT4 HAMMER1 ROOM6)
106: (PICK_UP_HAMMER ROBOT5 HAMMER1 ROOM6)
106: (MOVE_TO_ROOM ROBOT4 ROOM6 COMMON_ROOM)
107: (MOVE_TO_ROOM ROBOT5 ROOM6 COMMON_ROOM)
107: (MOVE_TO_ROOM ROBOT4 COMMON_ROOM ROOM2)
108: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM1)
108: (DROP_NAIL ROBOT4 NAIL4 ROOM2)
109: (DROP_HAMMER ROBOT5 HAMMER1 ROOM1)
110: (MOVE_TO_ROOM ROBOT5 ROOM1 COMMON_ROOM)
111: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM2)
112: (PICK_UP_NAIL ROBOT5 NAIL4 ROOM2)
113: (MOVE_TO_ROOM ROBOT5 ROOM2 COMMON_ROOM)
114: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM1)
115: (PICK_UP_HAMMER ROBOT5 HAMMER1 ROOM1)
116: (MOVE_TO_ROOM ROBOT5 ROOM1 COMMON_ROOM)
117: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM6)
118: (DROP_HAMMER ROBOT5 HAMMER1 ROOM6)
119: (PICK_UP_HAMMER ROBOT6 HAMMER1 ROOM6)
119: (MOVE_TO_ROOM ROBOT5 ROOM6 COMMON_ROOM)
120: (MOVE_TO_ROOM ROBOT6 ROOM6 COMMON_ROOM)
120: (MOVE_TO_ROOM ROBOT5 COMMON_ROOM ROOM2)
121: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM1)
```

```
121: (DROP_NAIL ROBOT5 NAIL4 ROOM2)
122: (DROP_HAMMER ROBOT6 HAMMER1 ROOM1)
123: (MOVE_TO_ROOM ROBOT6 ROOM1 COMMON_ROOM)
124: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM2)
125: (PICK_UP_NAIL ROBOT6 NAIL4 ROOM2)
126: (MOVE_TO_ROOM ROBOT6 ROOM2 COMMON_ROOM)
127: (MOVE_TO_ROOM ROBOT6 COMMON_ROOM ROOM1)
128: (PICK_UP_HAMMER ROBOT6 HAMMER1 ROOM1)
128: (DROP_NAIL ROBOT6 NAIL4 ROOM1)
129: (PICK_UP_NAIL ROBOT6 NAIL4 ROOM1)
130: (DROP_NAIL ROBOT6 NAIL4 ROOM1)
131: (PICK_UP_NAIL ROBOT6 NAIL4 ROOM1)
132: (HANG_PAINTING ROBOT6 PAINTING1 HAMMER1 NAIL4 ROOM1)
```

Figure B.6: Resulting plan obtained by PMR-RRPT-PLAN for the scenario shown in Figure 7.1c

```
 0: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM6)
  0: (MOVE_TO_ROOM ROBOT2 COMMON_ROOM ROOM6)
  1: (PICK_UP_HAMMER ROBOT1 HAMMER1 ROOM6)
  2: (MOVE_TO_ROOM ROBOT1 ROOM6 COMMON_ROOM)
  3: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
  4: (DROP_HAMMER ROBOT1 HAMMER1 ROOM4)
  5: (MOVE_TO_ROOM ROBOT1 ROOM4 COMMON_ROOM)
  6: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM2)
  7: (PICK_UP_NAIL ROBOT1 NAIL4 ROOM2)
  8: (MOVE_TO_ROOM ROBOT1 ROOM2 COMMON_ROOM)
  9: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
 10: (PICK_UP_HAMMER ROBOT1 HAMMER1 ROOM4)
 11: (HANG_PAINTING ROBOT1 PAINTING6 HAMMER1 NAIL4 ROOM4)
 12: (MOVE_TO_ROOM ROBOT1 ROOM4 COMMON_ROOM)
 13: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM2)
 14: (PICK_UP_NAIL ROBOT1 NAIL5 ROOM2)
 15: (MOVE_TO_ROOM ROBOT1 ROOM2 COMMON_ROOM)
 16: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM1)
 17: (HANG_PAINTING ROBOT1 PAINTING1 HAMMER1 NAIL5 ROOM1)
 18: (MOVE_TO_ROOM ROBOT1 ROOM1 COMMON_ROOM)
 19: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM2)
 20: (PICK_UP_NAIL ROBOT1 NAIL6 ROOM2)
 21: (MOVE_TO_ROOM ROBOT1 ROOM2 COMMON_ROOM)
 22: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM1)
 23: (HANG_PAINTING ROBOT1 PAINTING2 HAMMER1 NAIL6 ROOM1)
 24: (MOVE_TO_ROOM ROBOT1 ROOM1 COMMON_ROOM)
```

```
25: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM5)
26: (PICK_UP_NAIL ROBOT1 NAIL1 ROOM5)
27: (MOVE_TO_ROOM ROBOT1 ROOM5 COMMON_ROOM)
28: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM1)
29: (HANG_PAINTING ROBOT1 PAINTING3 HAMMER1 NAIL1 ROOM1)
30: (MOVE_TO_ROOM ROBOT1 ROOM1 COMMON_ROOM)
31: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM5)
32: (PICK_UP_NAIL ROBOT1 NAIL2 ROOM5)
33: (MOVE_TO_ROOM ROBOT1 ROOM5 COMMON_ROOM)
34: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
35: (HANG_PAINTING ROBOT1 PAINTING4 HAMMER1 NAIL2 ROOM4)
36: (MOVE_TO_ROOM ROBOT1 ROOM4 COMMON_ROOM)
37: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM5)
38: (PICK_UP_NAIL ROBOT1 NAIL3 ROOM5)
39: (MOVE_TO_ROOM ROBOT1 ROOM5 COMMON_ROOM)
40: (MOVE_TO_ROOM ROBOT1 COMMON_ROOM ROOM4)
41: (HANG_PAINTING ROBOT1 PAINTING5 HAMMER1 NAIL3 ROOM4)
```

APPENDIX C

EXTRA RESULTS ON MAP

THIS APPENDIX contains detailed results that were summarized in Chapter 7.

C.1 CodMAP problems extra results

The following tables show specific results on Time, Makespan and Cost obtained on the CoDMAP experiments from Section 7.5.

Table C.1: IPC scores in Time per configuration in CoDMAP domains.

	PMR-RRPT-PLAN							
	BC	**LB**	**ALL**	**CMAP-T**	**ADP-L**	**SIW**	**LF**	**YAHSP**
Driverlog	18.92	17.44	16.54	19.66	17.29	16.04	18.40	14.53
Zenotravel	19.31	18.00	15.74	19.84	18.80	18.75	18.64	10.71
Elevators	16.78	16.73	16.45	19.40	20.00	19.32	18.48	1.58
Logistics	20.00	19.65	19.07	20.00	20.00	20.00	20.00	20.00
Rovers	17.16	15.56	16.26	19.87	19.34	17.68	18.10	2.97
Satellites	18.54	17.70	17.04	20.00	19.31	17.96	17.18	7.46
Sokoban	8.82	6.56	11.32	12.13	15.16	15.52	13.14	5.76
Taxi	19.84	19.83	17.77	20.00	20.00	20.00	20.00	19.03
Blocks	20.00	15.25	14.98	20.00	18.42	16.56	18.19	14.88
Wireless	0.77	0.75	1.77	1.79	8.95	3.24	3.35	15.17
Depots	13.48	12.75	13.01	14.27	12.73	15.03	12.91	12.60
Woodworking	13.26	13.26	15.91	14.76	17.59	11.75	18.75	2.00
Total	186.87	173.48	175.86	201.72	**207.59**	191.84	197.14	126.69

Table C.2: IPC scores in Makespan per configuration in CoDMAP domains.

	PMR-RRPT-PLAN						
	BC	**LB**	**ALL**	**CMAP-T**	**ADP-L**	**LF**	**YAHSP**
Driverlog	11.82	12.65	8.86	10.99	13.84	12.43	16.00
Zenotravel	11.22	17.03	17.03	11.08	10.62	12.88	12.00
Elevators	16.25	16.41	16.41	17.96	16.30	17.44	2.00
Logistics	15.09	15.09	15.09	15.39	15.28	16.14	20.00
Rovers	10.56	9.86	9.75	17.51	11.06	16.39	4.00
Satellites	7.68	16.01	15.95	9.82	7.31	9.08	10.00
Sokoban	7.78	6.50	6.50	11.43	13.64	14.36	7.00
Taxi	13.37	13.37	13.37	13.37	13.43	13.32	20.00
Blocks	8.38	6.02	6.95	7.79	11.10	9.69	17.00
Wireless	1.89	1.89	1.89	4.52	7.21	4.55	18.00
Depots	10.13	10.76	11.00	9.49	7.19	10.00	14.00
Woodworking	13.43	13.43	13.43	10.80	19.67	13.43	0.90
Total	127.60	139.02	136.24	140.12	146.66	**149.72**	140.90

Table C.3: IPC scores in Cost per configuration in CoDMAP domains.

	PMR-RRPT-PLAN							
	BC	**LB**	**ALL**	**CMAP-T**	**ADP-L**	**SIW**	**LF**	**YAHSP**
Driverlog	16.90	15.57	14.28	17.27	18.30	14.40	17.92	14.64
Zenotravel	19.38	16.31	18.26	17.07	18.09	13.64	18.61	10.06
Elevators	15.95	15.87	15.11	14.25	15.23	14.59	15.74	2.00
Logistics	18.91	18.91	18.94	19.17	19.34	13.34	19.60	19.82
Rovers	18.64	16.87	18.10	19.43	19.11	16.94	18.41	3.75
Satellites	16.44	14.17	13.26	19.51	17.29	18.54	18.21	7.98
Sokoban	8.54	6.64	15.06	11.65	13.61	16.39	14.54	6.71
Taxi	18.94	18.94	18.94	18.94	18.67	16.03	18.62	16.11
Blocks	13.35	8.00	13.35	12.62	12.81	18.98	12.54	14.52
Wireless	1.93	1.93	4.44	4.44	9.00	6.39	4.47	17.44
Depots	15.07	14.54	14.54	14.62	9.05	14.66	12.97	13.25
Woodw.	17.57	17.57	17.57	15.20	17.85	15.18	17.57	2.00
Total	181.61	165.33	181.87	184.17	188.34	179.05	**189.22**	128.27

C.2 Extra results of PMR changing CoDMAP agentification

The following tables show specific results on Time and Cost when CoDMAP agentification is changed, which is explained on the experiments from Section 7.6.

Table C.4: Cost score in domains were plan reuse phase is executed inside PMR and agentification is changed (Depots-our, Logistics, Elevators-fast)

	PMR-RRPT-PLAN-BC	**PMR-RRPT-PLAN-LB**	**SIW**	**CMAP-T**	**ADP-L**
Depots	15.58	15.60	15.91	16.06	9.93
Logistics	19.28	19.28	13.60	19.54	19.72
Elevators	17.22	17.13	15.79	15.40	16.64
Depots-trucks	8.38	3.63	17.03	14.20	6.89
Logistics-air	18.65	17.59	13.05	18.99	19.73
Elevators-fast	8.53	11.01	12.23	16.40	17.36
Total	87.63	84.25	87.61	**100.59**	90.28

C.3 Hard MAP problems extra results

The following table shows specific results on Time obtained on the Hard problems experiments from Section 7.8

Table C.5: Time score score in hard and specific problems.

	PMR-RRPT-PLAN						
	BC	**LB**	**CMAP-T**	**ADP-L**	**SIW**	**LF**	**YAHSP**
Zenotravel	16.66	15.37	14.03	17.15	0.40	15.63	0.00
Satellite	16.61	15.58	15.53	19.86	0.53	18.72	0.00
Rover	10.34	10.25	10.22	19.40	0.00	19.87	0.00
Driverlog	0.00	2.05	4.37	5.22	0.00	7.56	0.00
Blocks	14.37	4.46	15.13	10.13	7.54	9.19	7.57
Rover-graph	8.33	8.33	9.60	20.00	11.73	6.23	0.00
VRP	20.00	12.16	10.79	13.57	1.56	11.03	0.00
Depots-robots	11.57	11.48	8.86	7.94	10.16	9.23	0.00
Total	97.89	79.68	88.53	**113.27**	31.92	97.47	7.57

References

V. Alcázar, M. M. Veloso, and D. Borrajo. Adapting a rapidly-exploring random tree for automated planning. In *SOCS*, 2011.

C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–655, 1995.

B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5 – 33, 2001. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(01)00108-4. URL http://www.sciencedirect.com/science/article/pii/S0004370201001084.

D. Borrajo. Plan sharing for multi-agent planning. In R. Nissim, D. L. Kovacs, and R. Brafman, editors, *Proceedings of ICAPS DMAP Workshop*, pages 57–65, 2013.

D. Borrajo and S. Fernández. MAPR and CMAP. In *Proceedings of CoDMAP-15*, 2015. URL http://agents.fel.cvut.cz/codmap/results/CoDMAP15-proceedings.pdf.

D. Borrajo and S. Fernández. Efficient approaches for multi-agent planning. *KAIS*, 14: 253–302, 2018.

D. Borrajo and M. M. Veloso. Probabilistically reusing plans in deterministic planning. In *Proceedings of ICAPS Workshop on HSDIP*, 2012.

D. Borrajo, A. Roubíčková, and I. Serina. Progress in case-based planning. *ACM Comput. Surv.*, 47(2):35:1–35:39, 2015. ISSN 0360-0300. doi: 10.1145/2674024. URL http://doi.acm.org/10.1145/2674024.

R. I. Brafman. A privacy preserving algorithm for multi-agent planning and search. In *IJCAI*, pages 1530–1536, 2015. ISBN 978-1-57735-738-4. URL http://dl.acm.org/citation.cfm?id=2832415.2832462.

R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, pages 28–35, 2008. URL http://www.aaai.org/Library/ICAPS/2008/icaps08-004.php.

R. I. Brafman and C. Domshlak. On the complexity of planning for agent teams and its implications for single agent planning. *Artificial Intelligence*, 198:52–71, 2013. doi: http://dx.doi.org/10.1016/j.artint.2012.08.005.

R. I. Brafman, C. Domshlak, Y. Engel, and M. Tennenholtz. Planning games. In *IJCAI*, pages 73–78, 2009. URL http://dl.acm.org/citation.cfm?id=1661445.1661458.

M. Brenner. Multiagent planning with partially ordered temporal plans. In *IJCAI*, pages 1513–1514, 2003. URL http://dl.acm.org/citation.cfm?id=1630659.1630916.

J. Britanik and M. Marefat. Hierarchical plan merging with application to process planning. In *Proceedings of IJCAI*, pages 1677–1684, 1995. URL http://dl.acm.org/citation.cfm?id=1643031.1643116.

T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

V. Conitzer. Comparing multiagent systems research in combinatorial auctions and voting. *AMAI*, 58(3-4):239–259, 2010. doi: 10.1007/s10472-010-9205-y.

S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971. doi: 10.1145/800157.805047. URL http://doi.acm.org/10.1145/800157.805047.

M. Crosby. ADP an agent decomposition planner. *Proceedings of CoDMAP-15*, page 4, 2015.

M. Crosby, M. Rovatsos, and R. P. A. Petrick. Automated agent decomposition for classical planning. In *ICAPS*, 2013. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/paper/view/6051.

T. De la Rosa, D. Borrajo, and A. Olaya. Replaying type sequences in forward heuristic planning. In *AAAI'06 Workshop on Learning for Search*, 2006.

S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *European Conference on Planning*, pages 135–147, 1991.

E. Ephrati and J. S. Rosenschein. Multi-agent planning as the process of merging distributed sub-plans. In *In Proceedings of the Workshop on Distributed Artificial Intelligence*, pages 115–129, 1993.

R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of IJCAI*, pages 608–620, 1971. URL http://dl.acm.org/citation.cfm?id=1622876.1622939.

D. Fišer, M. Štolba, and A. Komenda. Maplan. In *Competition of Distributed and Multiagent Planners*, pages 8–10, 2015.

D. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2-3):143–181, 1992.

M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR*, 20, 2003.

M. Fox, A. Gerevini, D. Long, and I. Serina. Plan stability: Replanning versus plan repair. In *ICAPS*, pages 212–221, 2006.

A. García-Olaya, T. de la Rosa, and D. Borrajo. Using the relaxed plan heuristic to select goals in oversubscription planning problems. In J. A. Lozano, J. A. Gámez, and J. A. Moreno, editors, *Advances in Artificial Intelligence*, pages 183–192, 2011. ISBN 978-3-642-25274-7.

H. Geffner. The model-based approach to autonomous behavior: A personal view. In *AAAI*, 2010.

A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *AIPS*, pages 112–121, 2000.

A. Gerevini and I. Serina. LPG: A planner based on local search for planning graphs with action costs. In *ICAPS*, pages 13–22, 2002. URL http://www.aaai.org/Library/AIPS/2002/aips02-002.php.

B. P. Gerkey and M. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, 2004.

P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *AIPS*, pages 140–149, 2000. ISBN 1-57735-111-8. URL http://dl.acm.org/citation.cfm?id=3090475.3090491.

M. Helmert. The fast downward planning system. *JAIR*, 26:191–246, 2006.

M. Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173:503–535, 2009. doi: 10.1016/j.artint.2008.10.013.

J. Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20(1):291–341, 2003. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622452.1622463.

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

R. Howey, D. Long, and M. Fox. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *ICTAI 2004*, pages 294–301, 2004.

A. Jonsson and M. Rovatsos. Scaling up multiagent planning: A best-response approach. In *ICAPS*, pages 114–121, 2011.

P. Jonsson and C. Bäckström. State-variable planning under structural restrictions: algorithms and complexity. *Artificial Intelligence*, 100(1):125 – 176, 1998. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(98)00003-4.

J. Jordán, A. Torreño, M. de Weerdt, and E. Onaindia. A better-response strategy for self-interested planning agents. *Applied Intelligence*, 48(4):1020–1040, 2018. doi: 10.1007/s10489-017-1046-5. URL https://doi.org/10.1007/s10489-017-1046-5.

S. Koenig, D. Furcy, and C. Bauer. Heuristic search-based replanning. In *AIPS*, pages 294–301, 2002. URL http://dl.acm.org/citation.cfm?id=3036884.3036923.

D. Kohan Marzagão, N. Rivera, C. Cooper, P. McBurney, and K. Steinhöfel. Multi-agent flag coordination games. In *AAMAS*, pages 1442–1450, 2017. URL http://dl.acm.org/citation.cfm?id=3091125.3091324.

J. R. Kok, M. T. Spaan, and N. Vlassis. Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems*, 50(2):99 – 114, 2005. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2004.08.003. URL http://www.sciencedirect.com/science/article/pii/S0921889004001290.

D. L. Kovacs. A multi-agent extension of PDDL 3.1. In *ICAPS*, pages 19–27, 2012.

J. Kozlak, J.-C. Créput, V. Hilaire, and A. Koukam. Multi-agent environment for dynamic transport planning and scheduling. In *Computational Science ICCS*, pages 638–645, 2004. doi: 10.1007/978-3-540-24688-6_83.

R. V. D. Krogt and M. D. Weerdt. Plan repair as an extension of planning. In *ICAPS*, pages 161–170, 2005.

B. Krulwich. Planning for soft goals. In *Proceedings of AIPS 1992*, pages 289 – 290, 1992. ISBN 978-0-08-049944-4. doi: 10.1016/B978-0-08-049944-4.50047-1. URL https://www.sciencedirect.com/science/article/pii/B9780080499444500471.

S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *I. J. Robotics Res.*, 20(5):378–400, 2001.

N. Lipovetzky and H. Geffner. Width-based algorithms for classical planning: New results. In *ECAI*, 2014.

N. Luis, T. Pereira, S. Fernández, A. Moreira, D. Borrajo, and M. Veloso. Using pre-computed knowledge for goal allocation in multi-agent planning. *Journal of Intelligent & Robotic Systems*, 2019. ISSN 1573-0409. doi: 10.1007/s10846-019-01022-0. URL https://doi.org/10.1007/s10846-019-01022-0.

A. D. Mali. Plan merging & plan reuse as satisfiability. In *ECP*, pages 84–96, 2000.

S. Maliah, G. Shani, and R. Stern. Privacy preserving landmark detection. *Frontiers in Artificial Intelligence and Applications*, 263:597–602, 2014. doi: 10.3233/978-1-61499-419-0-597.

S. Maliah, R. I. Brafman, and G. Shani. Increased privacy with reduced communication in multi-agent planning. In *ICAPS*, 2017.

L. Mudrova, B. Lacerda, and N. Hawes. Partial order temporal plan merging for mobile robot tasks. In *Proceedings of ECAI 2016*, volume 285, pages 1537 – 1545, 2016.

C. Muise, N. Lipovetzky, and M. Ramirez. MAP-LAPKT: Omnipotent multi-agent planning via compilation to classical planning. In *Competition of Distributed and Multiagent Planners*, 2015. URL http://www.haz.ca/papers/muise_CoDMAP15.pdf.

B. Nebel and J. Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, 76:427–454, 1995.

N. Nigam. The multiple unmanned air vehicle persistent surveillance problem: A review. *Machines*, 2(1):13–72, 2014. ISSN 2075-1702. doi: 10.3390/machines2010013.

R. Nissim and R. I. Brafman. Cost-optimal planning by self-interested agents. In *AAAI*, 2013.

R. Nissim and R. I. Brafman. Distributed heuristic forward search for multi-agent planning. *JAIR*, 51:293–332, 2014. doi: 10.1613/jair.4295. URL http://dx.doi.org/10.1613/jair.4295.

E. P. D. Pednault. Adl and the state-transition model of action. *Journal of Logic and Computation*, 4(5):467–512, 1994. doi: 10.1093/logcom/4.5.467. URL http://dx.doi.org/10.1093/logcom/4.5.467.

J. S. Penberthy and D. S. Weld. UCPOP: a sound, complete, partial order planner for ADL. pages 103–114. Morgan Kaufmann, 1992.

T. Pereira, M. Veloso, and A. Moreira. Multi-robot planning using robot-dependent reachability maps. In *Robot 2015: Second Iberian Robotics Conference*, pages 189–201. Springer, 2015.

T. Pereira, M. Veloso, and A. P. Moreira. Visibility maps for any-shape robots. In *IROS'16, the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 29, pages 428–459, 2016.

T. Pereira, N. Luis, A. Moreira, D. Borrajo, M. Veloso, and S. Fernandez. Heterogeneous multi-agent planning using actuation maps. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 219–224, 2018. doi: 10.1109/ICARSC.2018.8374186.

S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39:127–177, 2010.

J. Rintanen. Madagascar : Scalable planning with SAT. 2014.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

J. M. Santos, T. Krajník, and T. Duckett. Spatio-temporal exploration strategies for long-term autonomy of mobile robots. *Robotics and Autonomous Systems*, 88:116 – 126, 2017. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2016.11.016.

S. Shekhar and R. I. Brafman. Representing and planning with interacting actions and privacy. In *ICAPS*, 2018.

D. E. Smith. Choosing objectives in over-subscription planning. In *ICAPS*, 2004.

R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, Dec 1980. ISSN 0018-9340. doi: 10.1109/TC.1980.1675516.

J. M. Such, A. García-Fornes, A. Espinosa, and J. Bellver. Magentix2: A privacy-enhancing agent platform. *Engineering Applications of Artificial Intelligence*, 26(1):96 – 109, 2013. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2012.06.009.

A. Torreño, E. Onaindia, A. Komenda, and M. Štolba. Cooperative multi-agent planning: A survey. *ACM Comput. Surv.*, 50(6):84:1–84:32, 2017. ISSN 0360-0300. doi: 10.1145/3128584. URL http://doi.acm.org/10.1145/3128584.

A. Torreño, E. Onaindia, and Ó. Sapena. FMAP: Distributed cooperative multi-agent planning. *Applied Intelligence*, 41(2):606–626, 2014. doi: 10.1007/s10489-014-0540-2.

A. Torreño, Óscar Sapena, and E. Onaindia. FMAP: A platform for the development of distributed multi-agent planning systems. *Knowledge-Based Systems*, 145:166 – 168, 2018. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2018.01.013. URL http://www.sciencedirect.com/science/article/pii/S0950705118300212.

J. Tozicka, J. Jakubuv, and A. Komenda. Psm-based planners description for codmap 2015 competition. pages 29–32, 2015.

J. Tozicka, M. Štolba, and A. Komenda. The limits of strong privacy preserving multi-agent planning. In *ICAPS*, 2017.

R. van der Krogt and M. de Weerdt. Self-interested planning agents using plan repair. In *Workshop on Multiagent Planning and Scheduling, ICAPS*, pages 36–44, 2005.

M. Veloso, M. A. Pérez, and J. G. Carbonell. Nonlinear planning with parallel resource allocation. In *Proceedings of the DARPA Workshop on IPSC*, pages 207–212, 1990.

M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal. CoBots: Robust symbiotic autonomous mobile service robots. In *IJCAI*, 2015.

V. Vidal. The YAHSP planning system: Forward heuristic search with lookahead plans analysis. In *Proceedings of the 4th IPC*, pages 59–60, 2004.

M. Štolba and A. Komenda. The MADLA planner: Multi-agent planning by combination of distributed and local heuristic search. *Artificial Intelligence*, 252:175 – 210, 2017. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2017.08.007. URL http://www.sciencedirect.com/science/article/pii/S0004370217301042.

M. Štolba, A. Komenda, and D. L. Kovacs, editors. *Proceedings of CoDMAP*, 2015.

M. Štolba, A. Komenda, and D. L. Kovacs. Competition of Distributed and Multiagent Planners (codmap). In *AAAI*, 2016a.

M. Štolba, J. Tozicka, and A. Komenda. Secure multi-agent planning algorithms. In *ECAI*, 2016b.

P. R. Wurman, D. Raffaello, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *Proceedings of IAAI*, pages 1752–1759, 2007. ISBN 978-1-57735-323-2. URL http://dl.acm.org/citation.cfm?id=1620113.1620125.

R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *The International Journal of Robotics Research*, 25(1):73–101, 2006. doi: 10.1177/0278364906061160.