Cooperative Planning in Multi-Agent Systems

Alejandro Torreño Lerma

Departamento de Sistemas Informáticos y Computación

Universitat Politècnica de València

A thesis submitted for the degree of

Título de Doctor por la Universitat Politècnica de València

Under the supervision of:

Dr. Eva Onaindía de la Rivaherrera

Dr. Óscar Sapena Vercher

March 2016

PhD Thesis

Title:	Cooperative Planning in Multi-Agent Systems
Author:	Alejandro Torreño Lerma
Advisors:	Dr. Eva Onaindía de la Rivaherrera Dr. Óscar Sapena Vercher
Reviewers:	Prof. Pedro Meseguer González Prof. Guy Shani Prof. Mathijs de Weerdt
Examination Board:	
President	Prof. Vicente Juan Botti Navarro
Secretary	Prof. Dirk Sascha Ossowski
Member	Prof. Mathijs de Weerdt

Date of the defense: 12th May 2016

Abstract

Automated planning is a centralized process in which a single planning entity, or *agent*, synthesizes a course of action, or *plan*, that satisfies a desired set of goals from an initial situation. A Multi-Agent System (MAS) is a distributed system where a group of autonomous agents pursue their own goals in a reactive, proactive and social way.

Multi-Agent Planning (MAP) is a novel research field that emerges as the integration of automated planning in MAS. Agents are endowed with planning capabilities and their mission is to find a course of action that attains the goals of the MAP task. MAP generalizes the problem of automated planning in domains where several agents plan and act together by combining their knowledge, information and capabilities.

In *cooperative MAP*, agents are assumed to be collaborative and work together towards the joint construction of a competent plan that solves a set of common goals. There exist different methods to address this objective, which vary according to the typology and coordination needs of the MAP task to solve; that is, to which extent agents are able to make their own local plans without affecting the activities of the other agents.

The present PhD thesis focuses on the design, development and experimental evaluation of a general-purpose and domain-independent resolution framework that solves cooperative MAP tasks of different typology and complexity. More precisely, our model performs a multi-agent multi-heuristic search over a plan space. Agents make use of an embedded search engine based on forward-chaining Partial Order Planning to successively build refinement plans starting from an initial empty plan while they jointly explore a multi-agent search tree. All the reasoning

processes, algorithms and coordination protocols are fully distributed among the planning agents and guarantee the preservation of the agents' private information.

The multi-agent search is guided through the alternation of two state-based heuristic functions. These heuristic estimators use the *global* information on the MAP task instead of the local projections of the task of each agent. The experimental evaluation shows the effectiveness of our multi-heuristic search scheme, obtaining significant results in a wide variety of cooperative MAP tasks adapted from the benchmarks of the International Planning Competition.

Resumen

La planificación automática es un proceso centralizado en el que una única entidad de planificación, o *agente*, sintetiza un curso de acción, o *plan*, que satisface un conjunto deseado de objetivos a partir de una situación inicial. Un Sistema Multi-Agente (SMA) es un sistema distribuido en el que un grupo de agentes autónomos persiguen sus propias metas de forma reactiva, proactiva y social.

La Planificación Multi-Agente (PMA) es un nuevo campo de investigación que surge de la integración de planificación automática en SMA. Los agentes disponen de capacidades de planificación y su propósito consiste en generar un curso de acción que alcance los objetivos de la tarea de PMA. La PMA generaliza el problema de planificación automática en dominios en los que diversos agentes planifican y actúan conjuntamente mediante la combinación de sus conocimientos, información y capacidades.

En *PMA cooperativa*, se asume que los agentes son colaborativos y trabajan conjuntamente para la construcción de un plan competente que resuelva una serie de objetivos comunes. Existen distintos métodos para alcanzar este objetivo que varían de acuerdo a la tipología y las necesidades de coordinación de la tarea de PMA a resolver; esto es, hasta qué punto los agentes pueden generar sus propios planes locales sin afectar a las actividades de otros agentes.

La presente tesis doctoral se centra en el diseño, desarrollo y evaluación experimental de una herramienta independiente del dominio y de propósito general para la resolución de tareas de PMA cooperativa de distinta tipología y nivel de complejidad. Particularmente, nuestro modelo realiza una búsqueda multi-agente y multi-heurística sobre el espacio de planes. Los agentes hacen uso de un motor de búsqueda embebido basado en Planificación de Orden Parcial de encadenamiento progresivo para generar planes refinamiento de forma sucesiva mientras exploran

conjuntamente el árbol de búsqueda multiagente. Todos los procesos de razonamiento, algoritmos y protocolos de coordinación están totalmente distribuidos entre los agentes y garantizan la preservación de la información privada de los agentes.

La búsqueda multi-agente se guía mediante la alternancia de dos funciones heurísticas basadas en estados. Estos estimadores heurísticos utilizan la información *global* de la tarea de PMA en lugar de las proyecciones locales de la tarea de cada agente. La evaluación experimental muestra la efectividad de nuestro esquema de búsqueda multi-heurístico, que obtiene resultados significativos en una amplia variedad de tareas de PMA cooperativa adaptadas a partir de los bancos de pruebas de las Competición Internacional de Planificación.

Resum

La planificació automàtica és un procés centralitzat en el que una única entitat de planificació, o *agent*, sintetitza un curs d'acció, o *pla*, que satisfau un conjunt desitjat d'objectius a partir d'una situació inicial. Un Sistema Multi-Agent (SMA) és un sistema distribuït en el que un grup d'agents autònoms persegueixen les seues pròpies metes de forma reactiva, proactiva i social.

La Planificació Multi-Agent (PMA) és un nou camp d'investigació que sorgeix de la integració de planificació automàtica en SMA. Els agents estan dotats de capacitats de planificació i el seu propòsit consisteix en generar un curs d'acció que aconseguisca els objectius de la tasca de PMA. La PMA generalitza el problema de planificació automàtica en dominis en què diversos agents planifiquen i actúen conjuntament mitjançant la combinació dels seus coneixements, informació i capacitats.

En *PMA cooperativa*, s'assumeix que els agents són col·laboratius i treballen conjuntament per la construcció d'un pla competent que ressolga una sèrie d'objectius comuns. Existeixen diferents mètodes per assolir aquest objectiu que varien d'acord a la tipologia i les necessitats de coordinació de la tasca de PMA a ressoldre; és a dir, fins a quin punt els agents poden generar els seus propis plans locals sense afectar a les activitats d'altres agents.

La present tesi doctoral es centra en el disseny, desenvolupament i avaluació experimental d'una ferramenta independent del domini i de propòsit general per la resolució de tasques de PMA cooperativa de diferent tipologia i nivell de complexitat. Particularment, el nostre model realitza una cerca multi-agent i multi-heuristica sobre l'espai de plans. Els agents fan ús d'un motor de cerca embegut en base a Planificació d'Ordre Parcial d'encadenament progressiu per generar plans de refinament de forma successiva mentre exploren conjuntament l'arbre de cerca

multiagent. Tots els processos de raonament, algoritmes i protocols de coordinació estan totalment distribuïts entre els agents i garanteixen la preservació de la informació privada dels agents.

La cerca multi-agent es guia mitjançant l'aternança de dues funcions heurístiques basades en estats. Aquests estimadors heurístics utilitzen la informació *global* de la tasca de PMA en lloc de les projeccions locals de la tasca de cada agent. L'avaluació experimental mostra l'efectivitat del nostre esquema de cerca multi-heurístic, que obté resultats significatius en una ampla varietat de tasques de PMA cooperativa adaptades a partir dels bancs de proves de la Competició Internacional de Planificació.

A mi familia.

Acknowledgements

Tras muchos años de trabajo, llega el momento de cerrar esta tesis, y quiero aprovechar estas líneas para trasladar mi agradecimiento a todos los que directa o indirectamente han contribuido en su desarrollo.

En primer lugar quiero agradecer la labor de mis supervisores Eva Onaindía y Óscar Sapena. Esta tesis ha supuesto un gran esfuerzo de investigación y desarrollo, y he tenido la fortuna de contar con la supervisión de dos grandes profesionales en ambos campos, sin cuya ayuda habría resultado difícil llegar hasta aquí. Asimismo, quiero destacar la colaboración del resto de profesores del GRPS-AI, Eliseo, Laura, Toni e Inma, con cuya ayuda he podido contar siempre que lo he necesitado.

I want to thank the researchers of the multi-agent planning community, I hope we keep working together to expand this research field in the upcoming years. Particularly, I want to thank Prof. Daniel Borrajo, Dr. Raz Nissim and Dr. Guy Shani for their direct cooperation, which allowed me to compare my planners to some of the best MAP systems in the state of the art. I also want to thank Dr. Dániel László Kovács for his cooperation on the organization of the 2014 DMAP workshop.

Finally, I would like to acknowledge the researchers and staff members of the Agents Technology Center in the Czech Technical University in Prague for their help and supervision during my research stay last

year. Particularly, I want to thank Dr. Antonín Komenda and Michal Štolba for their direct involvement in the present research work. I hope we can keep collaborating in future research projects.

Quiero dedicar este trabajo a mis padres y a mis hermanos, Borja y Carlos, por su apoyo y paciencia durante estos años de investigación, especialmente durante la recta final de esta tesis.

A Lara, por su gran apoyo durante estos últimos dos años de trabajo, en especial durante mi estancia en Praga.

A los actuales miembros del laboratorio 208 del DSIC, Jesús, César, Alex y Debashis, un gran grupo con el que es un placer trabajar a diario. Quiero trasladar también mi agradecimiento al resto de investigadores que han pasado por el laboratorio durante estos años, entre ellos Juan Ángel, Alberto, Gustavo, Yolanda, Fabián, Javi o Stella. No quiero olvidarme de los miembros del resto de laboratorios del grupo, como Elena, Joanmi o Jaime entre otros.

A mi amigo Jaume, miembro honorífico del laboratorio 208, en especial por su ayuda durante los días previos a la defensa de esta tesis. A la futura doctora Bexy, con mis mejores deseos y esperando asistir en breve a la lectura de su tesis.

A Sonia, Clara, Lucía, Sergio y el resto de becarios "explotados" que han pasado por este nuestro departamento. Un saludo especial a Javi, por reñirme periódicamente por no presentar la tesis de una vez.

Por último, pero no por ello menos importante, quiero dedicar este trabajo a mis amigos Pablo Castejón, Sergio Esparcia, Sergio Pajares y Víctor Sánchez (y consortes), por tantas miserias compartidas dentro y fuera de Camden, NJ.

Contents

List of Figures

List of Tables

LIST OF TABLES

1

Introduction

Automated planning is the art of building control algorithms that synthesize a course of action to achieve a desired set of goals from an initial situation. Planning has been traditionally regarded as a centralized process in which a single entity is in charge of devising a plan that satisfies the problem goals (44).

A multi-agent system (MAS) is a distributed system where a group of autonomous entities known as *intelligent agents*, either human or software, pursue their own goals in a reactive, proactive and social way (57). MAS has been proposed as an appropriate modelling approach for domains such as electronic commerce (47), multi-robot systems (82), security applications (92), and so on.

In this context, Multi-Agent Planning (MAP) arises as a novel research field which pursues the integration of planning capabilities in intelligent agents so that a group of agents can develop a course of action that attains a set of goals. Therefore, MAP generalizes the problem of automated planning in domains where several planning entities, or agents, plan and act together by combining their knowledge, information and capabilities (76).

MAP can entail planning *by* multiple agents, i.e., distributed planning, or

planning *for* multiple agents, i.e., planning for multi-agent execution, thus giving rise to a great variety of tools and techniques. The approach traditionally adopted by the MAS research community assumes that agents are self-interested and that there is not a common goal to solve, thus focusing on coordinating the activities of multiple agents in a shared environment (27). The ultimate objective is to ensure that the agents' local objectives (private goals) will be achieved by their plans and so the emphasis is put on distributed execution, plan synchronization and collaborative activity at run-time planning (32, 60, 111).

The most common planning-oriented approach, known as *cooperative MAP*, assumes agents to be collaborative and focuses on extending planning into a distributed environment or, more particularly, on the joint construction of a competent plan that addresses a set of common goals. There exist different methods to address this objective, which vary according to the typology of the MAP task to solve. In particular, the adoption of one or another strategy depends on the coordination needs of the task; i.e., to which extent agents are able to make their own local plans without affecting what the other agents are planning to do.

In *loosely-coupled tasks* where agents are relatively independent, they carry out planning individually and coordinate either before or after the planning activity. Some approaches apply *pre-planning coordination* or *goal allocation*, which entails distributing the task goals among the participants on a pre-planning fashion, ensuring that the local plans generated by the agents afterwards can be effectively combined into a sound global solution (11). A wide range of approaches follow a *post-planning coordination* or *plan merging* scheme, i.e., they solve inconsistencies among local plans that have been constructed separately in order to come up with a coherent joint plan (28, 80).

In general, plan merging and goal allocation methods are rather inefficient when solving *tightly-coupled tasks* with a large amount of coordination points among

agents (80). In order to tackle this issue, a third group of MAP techniques intertwines the planning and coordination activities, resulting in a unified scheme that effectively attains complex tightly-coupled tasks. These *interleaved* methods, however, do not perform optimally in loosely-coupled tasks because their reasoning procedures rely strongly on a high degree of interdependency between the agents' actions.

The preservation of *privacy* arises as another relevant topic in cooperative MAP techniques that distribute the planning activity among agents. Despite working cooperatively in a MAP task, intelligent agents may become competitors in subsequent tasks, and therefore, it is desirable to minimize the exchanged information and share only the minimum required amount of data for the decentralized planning procedures to be successfully carried out. Privacy introduces additional challenges into the design and development of cooperative MAP methods and is only managed by the most recent approaches to cooperative MAP (11, 72).

Many of the state-of-the-art techniques resort to heuristic search in order to attain the cooperative MAP problem. Since information on the MAP task is usually distributed across agents, most methods guide the search for a solution through *local heuristic functions*; i.e., each agent estimates the quality of a plan according to its projection of the MAP task (the local information it possesses). In general, the accuracy of local heuristics is rather poor, which motivated the development of *global heuristic functions*. These estimators make use of the information of the MAP task as a whole to estimate the quality of the plans, and its development constitutes one of the current challenges of cooperative MAP, particularly in privacy-preserving settings (72, 107).

The present PhD thesis pursues the design and development of computational techniques that efficiently address the cooperative MAP problem. More precisely, our focus is on adapting heuristic-based search to a multi-agent context while

ensuring that the agents' critical information remains private. Our ultimate goal is to provide intelligent agents with cooperative planning capabilities, so that they can collaboratively develop a joint plan for a set of common objectives as a group.

The algorithms and techniques developed over the course of this research are subject to the following principles:

- **Decentralized techniques:** We understand cooperative MAP as a task simultaneously performed by a group of independent planning entities, such that the information on the task and the planning capabilities are distributed among agents. For this reason, all the techniques developed during the course of this research are decentralized and can be directly integrated in intelligent agents, thus providing them with cooperative MAP capabilities.

- **Interleaved search:** In our model, cooperative MAP tasks are attained through an integrated resolution procedure that interleaves the planning and coordination activities. Our objective is to come up with a general-purpose solution that effectively solves MAP tasks of any complexity, ranging from loosely-coupled to complex tightly-coupled tasks.

- **Privacy preservation:** Privacy is a key aspect of our cooperative MAP model, and therefore, all the techniques developed in this research are built ensuring that the agents' private information is effectively kept.

- **Global heuristic functions:** Our model attains the cooperative MAP problem through heuristic search. The heuristic functions we developed estimate the quality of the plans according to the global information on the MAP task while preserving agents' privacy.

According to the previous guidelines, the cooperative MAP resolution framework that constitutes the main contribution of this research follows a decentralized heuristic search scheme, interleaving planning and coordination to tackle both loosely-coupled and tightly-coupled tasks. Privacy is effectively preserved across all the procedures carried out by the framework, minimizing the amount of information shared throughout the joint tasks performed by the agents. Finally, in order to maximize the efficiency of the framework, the search is guided through a set of accurate global heuristic estimators.

1.1 Objectives

This section presents the precise objectives that guided the development of this PhD thesis, along with the associated tasks conducted throughout this research and the resulting contributions:

1. **Analysis of the state of the art:** This initial objective entails a thorough revision of the literature regarding the main topics of this research.

   Since most approaches to MAP reuse or adapt single-agent planning techniques, we reviewed the state of the art in automated planning, focusing on the main techniques, paradigms and specification languages, with an emphasis on *the Partial-Order Planning* (POP) paradigm and heuristic search techniques.

   The state of the art in cooperative MAP has also been reviewed. Given the wide variety of approaches to MAP, we analyzed topics such as information distribution, underlying planning paradigm, coordination of agents, privacy or heuristic strategy to properly characterize and classify the existing MAP methods.

2. **Formalization of the cooperative MAP task:** The second objective of our work implies establishing a theoretical formalization of the cooperative MAP task. The resulting formal definition tackles three basic aspects: the main components of a MAP task, such as actions, objects of the world or agents; the mechanisms to preserve agents' privacy; and the search scheme used to attain the MAP tasks.

3. **Design of a MAP specification language:** In order to solve cooperative MAP tasks, it is necessary to formally represent them. A definition language to model MAP tasks is thus one of the basic requirements of this research.

   Since the existing languages for single-agent planning do not support the particular requirements of the cooperative MAP tasks, such as information distribution and privacy, it was necessary to design a novel specification language with enough expressiveness to represent all the elements that compose a MAP task.

4. **Development of a cooperative MAP resolution framework:** The central objective of this work is the design of a resolution framework that efficiently attains cooperative MAP tasks. This objective was fulfilled through the continuous development of several iterations of the framework, each of them introducing several refinements over the previous ones.

   The first version of the framework, named MAP-POP, establishes the basics of our model: agents perform a coordinated exploration of a joint search tree, where plans are individually generated via the progressive refinement of the existing nodes of the tree. In order to build these *refinement plans*, each agent integrates a search engine based on POP. MAP-POP presents some limitations addressed in the following iterations of the framework, such as the lack of some theoretical properties (search completeness is not ensured), and

some performance issues caused by the use of POP-based heuristic functions to guide the search.

FMAP, the second iteration of our framework, replaces the embedded backward POP reasoning module by a forward-chaining POP. This change benefits the model by providing two direct improvements: search completeness is guaranteed since all the plans are now generated, and the forward search enables the usage of accurate state-based heuristic estimators. FMAP introduces h_{DTG}, a relaxation-based estimator that clearly boosts the overall performance of the system over MAP-POP.

The final evolution of our framework, MH-FMAP, introduces a multi-heuristic search scheme, thus enabling the simultaneous application of diverse heuristics to evaluate plans, rather than using a single estimator. The combination of the existing h_{DTG} heuristic and a novel global landmark-based estimator, h_{Land}, along with several optimizations in the communications among agents, provide great benefits regarding performance.

The different versions of the framework were systematically evaluated through an extensive benchmark that includes several MAP domains adapted from the testbeds of the International Planning Competition. Throughout this research, the performance of the system was compared to other state-of-the-art approaches to cooperative MAP to qualitatively measure the performance of the framework.

5. **Design of global heuristic functions for MAP:** In order to develop our resolution framework to its full potential, the heuristic guidance of the search is required to be as accurate as possible. For this reason, one of the basic objectives of this research entailed the study of single-agent estimators and its adaptation to a multi-agent context.

We developed two global, suboptimal and privacy-preserving multi-agent estimators, h_{DTG} and h_{Land}. This research objective involved the usage and multi-agent adaptation of several well-known single-agent planning techniques, such as *relaxed planning graphs*, *domain transition graphs* and *landmarks*.

As previously stated, in order to get the most of this research objective, the final iteration of the resolution framework, MH-FMAP, allows for the simultaneous usage of both global estimators, which effectively improves the overall performance of the system in many cooperative MAP domains.

1.2 Related research activities

This section lists the research activities performed during the development of this PhD thesis, namely the related scientific publications, research stays and research projects.

1.2.1 Related publications

The following subsections list all the scientific publications related to this research. We classify articles according to the type of publication they were included into: section 1.2.1.1 cites the articles appearing in journals listed in the Science Citation Index (SCI), while section 1.2.1.2 lists the papers published in the proceedings of relevant conferences included in the Computing Research and Education Association of Australasia (CORE) rankings.

Finally, section 1.2.1.3 lists other relevant scientific articles without an impact factor or not published in a ranked conference.

1.2.1.1 Publications in SCI journals

- A. Torreño, E. Onaindia and Ó. Sapena. **FMAP: distributed cooperative multi-agent planning**. *Applied Intelligence*. Volume 41(2), pages 606-626, 2014. Impact Factor (2012): 1,853.

- A. Torreño, E. Onaindia and Ó. Sapena. **A flexible coupling approach to multi-agent Planning under incomplete information**. *Knowledge and Information Systems*. Volume 38(1), pages 141-178, 2014. Impact Factor (2014): 1,782.

- Ó. Sapena, E. Onaindia and A. Torreño. **FLAP: applying least-commitment in forward-chaining planning**. *AI Communications*. Volume 28(1), pages 5-20, 2014. Impact Factor (2014): 0,547.

1.2.1.2 Publications in CORE conferences

- A. Torreño, E. Onaindia and Ó. Sapena. **Global heuristics for distributed cooperative multi-agent planning**. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling*. Pages 225-233, 2015. Conference ranking: CORE A*.

- A. Torreño, E. Onaindia and Ó. Sapena. **An approach to multi-agent planning with incomplete information**. In *Proceedings of the 20th European Conference on Artificial Intelligence*. Pages 762-767, 2012. Conference ranking: CORE A.

- Ó. Sapena, A. Torreño and E. Onaindia. **On the construction of joint plans through argumentation schemes**. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*. Pages 1195-1196, 2011. Conference ranking: CORE A*.

- Ó. Sapena, A. Torreño and E. Onaindia. **On the use of argumentation in multi-agent planning**. In *Proceedings of the 19th European Conference on Artificial Intelligence*. Pages 1001-1002, 2010. Conference ranking: CORE A.

- A. Torreño, E. Onaindia and Ó. Sapena. **Reaching a common agreement discourse universe on multi-agent planning**. In *Proceedings of the 5th International Conference on Hybrid Artificial Intelligence Systems*. Pages 185-192, 2010. Conference ranking: CORE C.

1.2.1.3 Other publications

- A. Torreño, Ó. Sapena and E. Onaindia. **MH-FMAP: alternating global heuristics in multi-agent planning**. In *Proceedings of the Competition of Distributed and Multi-Agent Planners*. Pages 25-28, 2015.

- A. Torreño, E. Onaindia and Ó. Sapena. **Integrating individual preferences in multi-agent planning**. In *Proceedings of the 2nd Workshop on Distributed and Multi-Agent Planning*. Pages 79-86, 2014.

- E. Onaindia, Ó. Sapena and A. Torreño. **Argumentation-based planning in multi-agent systems**. Book chapter, in *Negotiation and Argumentation in Multi-Agent Systems*. Bentham eBooks. Pages 361-398, 2014.

- A. Torreño, E. Onaindia and Ó. Sapena. **FMAP: a heuristic approach to cooperative multi-agent planning**. In *Proceedings of the 1st Workshop on Distributed and Multi-Agent Planning*. Pages 84-92, 2013.

- Ó. Sapena, E. Onaindia and A. Torreño. **Cooperative distributed planning through argumentation**. *International Journal of Artificial Intelligence*. Volume 4(10), pages 118-136, 2010.

1.2.2 Scientific research stays

The following research stay was completed during the research period associated to this PhD thesis:

- 26-05-2015 to 25-08-2015. *Czech Technical University in Prague, Czech Republic.* Research stay supervised by Professor Michal Pěchouček on the adaptation of the *Merge-and-Shrink* family of abstraction heuristics to cooperative MAP.

1.2.3 Research projects

This work has been performed in the context of several research projects that provided economical funding or technological support to its development:

- **"Agreement Technologies" Consolider-INGENIO 2010** under grant CSD2007-00022. (Main researcher: Carles Sierra, from 2007 to 2012). Agreement technologies is a term coined in the last years to refer to those technologies that allow computational entities to automatically solve conflicts. This research was initiated as a work package within this project that pursued the integration of group planning capabilities in intelligent agents.

- **"Magentix2: A Multi-agent Platform for Open Multi-agent Systems"** under grant TIN2008-04446 (Main Researcher: Ana Garcia-Fornes, from 2008 to 2011). Magentix2 is a multi-agent platform that aims to provide support for open systems where heterogeneous agents can enter and leave the system dynamically. The Magentix2 platform provided the infrastructure used by our framework to implement the communications among the planning agents.

- **"PlanInteraction: Multi-agent Interaction for Planning"** under grant TIN2011-27652-C03 (Main Researcher: Eva Onaindia, from 2012). This project aims to develop new agent techniques based on social dynamics for the design of a MAP platform composed of autonomous and, possibly heterogeneous, planning entities. The platform tackles aspects such as multi-agent execution, cooperative and non-cooperative MAP, plan merging and planning via argumentation. Our framework was integrated into the PlanInteraction platform as a means to provide PlanInteraction agents with cooperative MAP capabilities.

Additionally, this work has been supported by the Prometeo projects 2008/051 and II/2013/019 funded by the Valencian Government. Moreover, this research would not have been possible without a 4-year FPI-UPV research scholarship granted to the first author by the Universitat Politècnica de València.

1.3 Document structure

This PhD thesis is organized as a compendium of research articles that compile and synthesize the results of this research work. The remainder of this document is organized as follows:

- Chapter 2 analyzes the state of the art on automated planning. Whereas each of the articles in the compendium include some form of summary of the state of the art, one should note that cooperative MAP is an active and rapidly evolving research field. For this reason, a comprehensive and updated analysis of the current state of the art is required to fully understand the intricacies of this research area.

Since most approaches to cooperative MAP are built up from pre-existing single-agent planning techniques, section 2.1 is devoted to the single-agent approach to planning, discussing the most relevant paradigms, frameworks and task specification languages.

Section 2.2 focuses on cooperative MAP, characterizing the aspects that distinguish a MAP task from its single-agent counterparts as well as analyzing the problem of representing it. Afterwards, we provide an in-depth classification of the main MAP methods in the literature and present the most relevant MAS platforms used for the development of decentralized MAP frameworks.

- Chapter 3 compiles the four main impact articles related to this research work. The articles are chronologically arranged and provide a comprehensive view of the evolution of this research work and its results.

  Section 3.1 briefly summarizes the contents of each article, indicating the structure of the publication and the location of its main contributions, while sections 3.2 to 3.5 present the full text of the four research articles.

- Chapter 4 summarizes and thoroughly discusses the results obtained in this research, both the scientific contributions and the collected experimental results. Thus, this chapter serves as a reference guide to the scientific content provided by the articles of Chapter 3.

  Section 4.1 discusses the contributions of this research, describing in detail each individual contribution and referencing the sections of Chapter 3 where the related technical content can be found.

  Section 4.2 analyzes the experimental results obtained by each version of our MAP framework. Throughout this research, we compared the different

evolutions of our framework against other state-of-the-art methods via a variety of MAP domains. This results constituted the primary source to detect the strengths and flaws of our approach and define the refinements to undertake in the subsequent evolutions of the framework.

Finally, section 4.2 summarizes the results obtained by the most recent version of our MAP framework, MH-FMAP, in the 2015 Competition of Distributed and Multi-Agent Planners (CoDMAP), which are not disclosed in the articles of Chapter 3. We also provide some additional experiments that justify the performance differences of MH-FMAP in our setting and the CoDMAP.

- Chapter 5 presents our concluding remarks, putting the focus on the strengths and weaknesses of our approach to cooperative MAP, and disclosing our future lines of research.

2

State of the art in automated planning

Automated planning in Artificial Intelligence (AI) can be defined as the art of building control algorithms for dynamic systems. More precisely, a planning task is a search problem whose purpose is finding a set of actions that leads the system to an objective state from a given initial situation. The vast majority of approaches model planning as a single-agent procedure, in which a single entity carries out the entire search process, developing the complete course of action to solve the task at hand.

Recently, an interest in Multi-Agent Planning (MAP) has developed. MAP is a relatively novel research field that combines technologies, algorithms and techniques developed by the AI planning and multi-agent systems (MAS) communities. While planning has been traditionally regarded as a centralized process, MAP generalizes this concept by considering a set of heterogeneous entities, or *agents*, that work together to develop a course of action that satisfies the goals of the group. Therefore, MAP introduces a social approach to planning by which multiple in-

telligent entities work together to solve planning tasks that they are not able to solve by themselves, or to at least accomplish them better by cooperating (24).

MAP introduces many challenges that are not present in classical single-agent planning, such as the distribution of information among agents and the design of robust communication protocols among the planning entities. Additionally, many MAP models are designed to guarantee the privacy of the agents' sensitive information.

This chapter analyses the state of the art in single and multi-agent planning. Most approaches to MAP draw upon a wide variety of single-agent techniques. For this reason, the first part of this chapter discusses single-agent planning: first, we formalize a single-agent planning task; next, we analyse the representation of planning tasks and the most relevant task specification languages; and finally, we discuss the most relevant single-agent planning paradigms in the literature. Particularly, we put the focus on state-based heuristic search and partial-order planning, because these approaches constitute the fundamentals in which the present work is rooted.

The second part of the chapter reviews the state of the art in cooperative MAP, discussing the main aspects of this research field, analysing the representation of MAP tasks and classifying the most relevant cooperative MAP approaches.

2.1 Single-agent planning

Single-agent planning is a search process by which a single entity synthesizes a set of actions or *plan* to reach a set of objectives from an initial situation (122). In order to formally define a single-agent planning task, we focus on the classical planning modelling (44). Classical models introduce various assumptions to reduce the complexity of the single-agent planning problem:

- The world is represented through a finite set of situations or *states*.
- The world is *fully observable*; that is, the planning entity has complete knowledge of the environment.
- The world is *deterministic*; that is, the application of an action over a state leads deterministically to a single other state.
- The world is *static*; that is, the state of the world does not evolve until an action is applied.
- The planner handles an *explicit* and *immutable* goal state.
- Actions have *no duration*. Time and numeric reasoning is not considered.
- The planning activity is carried out *offline*, that is, the planner is not concerned with external changes that occur in the world.

Despite these simplifications, domain-independent single-agent planning is a complex problem. In particular, single-agent planning in its classical form is a PSPACE-complete problem (44).

Each state of the world is defined through a finite set of facts or *literals* that describe the properties of the world. A literal is an atom composed by a predicate symbol and a finite set of parameters referred to objects of the world.

Definition 2.1. *(**State**) A **state** S is a finite set of literals that represents a situation of the world.*

The states of the world evolve through the application of planning *actions*, which are defined as follows:

Definition 2.2. *(**Action**) A **planning action** is a tuple $\alpha = PRE(\alpha) \rightarrow \{ADD(\alpha), DEL(\alpha)\}$, where $PRE(\alpha)$ is a set of literals describing the preconditions of α, and $ADD(\alpha)$ and $DEL(\alpha)$ are two sets of literals that express the additive and delete effects of α, respectively.*

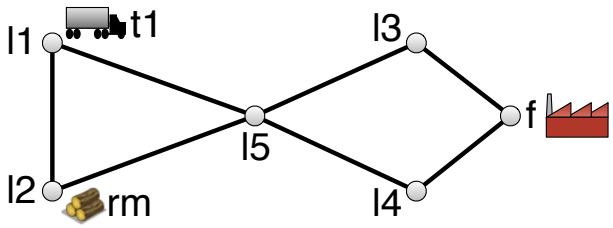

Figure 2.1: Single-agent planning task example

Given a state S, an action α can be executed if and only if all its preconditions hold in S, that is, $\forall p \in PRE(\alpha)$, $p \in S$. Executing an action α in a world state S leads to a new state $S' = (S - DEL(\alpha)) \cup ADD(\alpha)$; that is, the literals in $DEL(\alpha)$ are removed from S' and the literals in $ADD(\alpha)$ are added to S'.

Given the previous definitions, a single-agent planning task is formally defined as follows:

Definition 2.3. *(**Single-agent planning task**) A **single-agent planning task** is a tuple $T = \langle I, A, G \rangle$. I is a state that represents the initial situation of the world. A is a set of actions of the form $\alpha = PRE(\alpha) \rightarrow \{ADD(\alpha), DEL(\alpha)\}$ that can be applied by the planning agent to solve T. G is the goal state we desire to reach.*

Finally, we define a solution plan Π for a task T as follows:

Definition 2.4. *(**Solution plan**) A **solution plan** Π for a single-agent planning task T is a sequence of actions $\{\alpha_0, \ldots, \alpha_n\}$ whose application over I leads to a state S, where $G \subseteq S$.*

In order to motivate a single-agent planning task, let us present a brief application example:

Example 1. *Figure 2.1 shows a single-agent planning task whose goal is to deliver a package of raw materials rm into a factory f. To do so, a truck $t1$ must pick up the package rm, transport it through a network of roads connecting the locations $l1$-$l5$ and f, and deliver it to location f.*

In Example 1, the initial situation of the world is modelled through the literals $(pos\ t1\ l1)$ and $(pos\ rm\ l2)$, which describe the initial location of the objects involved in the task, as well as a set of immutable or *static* literals establishing the connections among the different locations, such as $(link\ l1\ l2)$.

The goal state is defined as $G = \{(pos\ rm\ l2)\}$. In order to complete the task, the planning entity can apply actions such as driving the truck between locations, $(drive\ t1\ l1\ l2)$, loading the package in the truck, $(load\ rm\ t1\ l2)$, and unloading the package in a specific location, $(unload\ rm\ t1\ f)$.

2.1.1 Representation in single-agent planning

One of the main challenges addressed by the AI planning community is the representation problem. The use of an expressive language to specify planning tasks is one of the key aspects of an efficient planning process.

Representing a single-agent planning entails modelling its elements (see Definition 2.3) through a formal language. The representation of a planning task faces multiple challenges; in particular, it is necessary to overcome the *frame problem* (73): in most cases, the number of aspects of the world that remain unchanged when applying an action α is much higher than the number of aspects actually modified by α.

One of the first modelling languages, *STRIPS* (STanford Research Institute Problem Solver) (36), has widely influenced most of the planning works since the 1970s. *STRIPS* proposes a compact and simple model to specify planning domains, effectively solving the frame problem (73), and supporting divide-and-conquer strategies (39).

Among the multiple extensions to *STRIPS* developed over the last years (see subsection 2.1.1.1), the Planning Domain Definition Language *PDDL* (43) has

become not only the most popular one, but also the *de facto* standard for the single-agent planning community.

Modelling a single-agent planning task, such as Example 1, through *PDDL* entails defining two separate blocks: a *domain* and a *problem*. The domain describes the general features of a particular domain, such as the types of objects, the predicates that describe situations of the world and the operators that can be applied by the planning entity to solve the task. The problem block models the specific details of the task, such as the actual objects in the world, the initial situation of the task and the goals that must be achieved in order to solve the planning task.

In order to model the task in Example 1 we first define the type hierarchy (note that *PDDL* also supports untyped domain descriptions):

```
(:types location package - object
        truck place - location)
```

The `location` and `package` types directly derive from `object`, the basic type for all the objects in a *PDDL* domain. Both `truck` and `place` are defined as subtypes of `location`.

Next, we define the predicates that will describe the situations of the world:

```
(:predicates
    (pos ?t - truck ?p - place)
    (at ?p - package ?l - location)
    (link ?p1 - place ?p2 - place))
```

The `pos` predicate models the position of a `truck`, while `at` indicates the placement of a `package`, and `link` establishes the connections among `places`. Literals of a task are obtained by *grounding* the predicates, that is, giving actual values to their parameters via the objects defined in the problem. For instance, a literal (*pos* $t1$ $l1$) can be inferred from `(pos ?t - truck ?l - location)` in Example 1.

The last section of the domain is devoted to the modelling of the planning operators. This particular task includes the operators `drive`, `load` and `unload`. For simplicity, we show only the description of the `drive` operator:

```
(:action drive
    :parameters   (?t - truck ?p1 - place ?p2 - place)
    :precondition (and (pos ?t ?p1) (link ?p1 ?p2))
    :effect       (and (not (pos ?t ?p1))(pos ?t ?p2))
)
```

The `drive` operator is used to move a `truck` between two different `places`. The `:precondition` section defines the facts that must hold for the action to be executed, and `:effect` describes the changes made to the state after the execution of the operator. In this case, the preconditions entail the `truck ?t` being placed in the initial `place ?p1` and a direct connection between `?p1` and `?p2`. As an effect of the operator execution, the `truck ?t` will be placed in `?p2` instead of `?p1`.

Similarly to the literals, the task actions are obtained by grounding the domain operators. For instance, $(drive\ t1\ l1\ l2)$ can be inferred from the `drive` operator.

Regarding the problem block of the task, we first define the objects that take part in the task (see Figure 2.1), according to their types:

```
(:objects
    t1 - truck
    rm - package
    l1 l2 l3 l4 l5 f - place)
```

Next, we describe the initial state of the world, including the initial location of the `truck t1` and the `package rm`, as well as the `links` among `places`:

```
(:init
    (pos t1 l1)
    (at rm l2)
    (link l1 l2)(link l2 l1)(link l1 l5)(link l5 l1)(link l2 l5)
    (link l5 l2)(link l5 l3)(link l3 l5)(link l5 l4)(link l4 l5)
    (link l3 f)(link f l3)(link l4 f)(link f l4))
```

Finally, we define the goal of the task; that is, delivering `rm` into f:

```
(:goal (at rm f))
```

2.1.1.1 Planning task specification languages

Despite being one of the most successful and influential works on single-agent planning representation, *STRIPS* (36) has some expressive limitations that make it difficult to describe some real problems (97). As a result, many extensions to *STRIPS* have been developed over the past years, enriching its expressiveness and simplifying the definition of planning domains.

This subsection performs a brief historical summary of the most relevant planning specification languages and their main features, putting the focus on the most successful extensions to *STRIPS*.

ADL. The *Action Description Language* (*ADL*) (89) is one of the earlier extensions to *STRIPS*. *ADL* uses an algebraic model to define the states of the world, which increases its expressiveness over *STRIPS*, allowing the designer to represent a larger number of situations. *ADL* also improves *STRIPS* by incorporating new features such as types, negated goals and preconditions, equality restrictions and conditional effects, among others.

PDDL. As previously mentioned, *PDDL* (43) is the most relevant extension to *STRIPS*. *PDDL* was developed for the 1998 International Planning Competition (IPC) (74), aiming to provide a common notation for modelling planning tasks and evaluating the planners' results. Ever since its introduction, *PDDL* has become the reference modelling language for the vast majority of planners. *PDDL* inherits the action modelling of *STRIPS*, introducing a wide set of features, such as conditional effects, hierarchical actions and domain axioms.

Extensions to *PDDL*. One of the key outcomes of the IPC (71) is the introduction of several extensions to *PDDL*. *PDDL2.1* (38), introduced in the 2002 IPC (IPC-3), adds time management and numeric capabilities to *PDDL*.

After *PDDL2.1*, *PDDL2.2* (33) and *PDDL3.0* (41) were presented in subsequent editions of the IPC. These extensions add a set of new features on top of *PDDL2.1*: *PDDL2.2* introduces derived actions and timed initial literals, while *PDDL3.0* emphasizes the importance of plan quality, introducing preferences, soft constraints and state trajectory constraints.

Finally, the latest *PDDL* version, *PDDL3.1* (64), enriches the language with *SAS+*-like (2) task representations. More precisely, *PDDL3.1* introduces object fluents, which are state variables that are neither binary (true/false) nor numeric (real-valued), but instead are mapped to a finite domain of objects, a flexible solution inspired by the *Functional Strips* formalism (39).

2.1.2 Single-agent planning paradigms

Over the last years, single-agent planning has experienced great advances, specifically in the development of domain-independent planning techniques. Most single-agent planning systems are defined as search procedures that can be classified according to the search space they explore and the direction of the search (44). This gives rise to a wide variety of single-agent planning paradigms.

The next subsections describe in detail the single-agent planning paradigms that have influenced this PhD thesis to some extent, namely state-based and partial-order planning. We also briefly describe other relevant search paradigms.

2.1.2.1 State-based planning

The simplest single-agent planning methods are state-based search algorithms (44). As formalized in section 2.1, state-based planners assume that the world can be

described through a finite number of *states* (see Definition 2.1), and define a plan as a sequence of actions whose application over the initial situation of the world leads to a certain state. Most state-based planners are categorized as forward-chaining search algorithms, since they develop solution plans departing from the initial state of the task.

In most cases, state-based search algorithms are guided by a node-selection *heuristic* function that ranks the open nodes of the search tree according to their desirability. One of the most effective mechanisms to devise heuristic functions in state-based planning focuses on the *delete-relaxation* of the planning task; that is, neglecting the $DEL(\alpha)$ effects of the actions $\alpha \in A$. Generally, a heuristic function estimates the quality of a plan Π by solving the relaxed planning task from the state that results from the application of the sequence of actions in Π.

Nowadays, state-based planning remains as the most popular search paradigm in single-agent planning. Many state-based planners are among the most efficient single-agent planners, thanks to the use of accurate heuristic functions that allow for a precise and efficient exploration of the state space.

The Heuristic Search Planner (HSP) (9) is one of the first state-based systems that resort to domain-independent heuristic search. More precisely, HSP uses a weighted A* search scheme by which a plan Π is evaluated through a function $f(\Pi) = g(\Pi) + w * h_{add}(\Pi)$. $g(\Pi)$ represents the cost of the current plan Π, while $h_{add}(\Pi)$ estimates the cost of reaching the goal state G from Π by means of an *additive* heuristic.

The additive delete-relaxation heuristic h_{add} evaluates a plan Π by independently calculating the cost of reaching a goal g, $cost(g|\Pi)$, for each $g \in G$. Then, $h_{add}(\Pi)$ is obtained by adding up the resulting costs; that is, $h_{add}(\Pi) = \sum_{g \in G} cost(g|\Pi)$.

The Fast Forward planning system (FF) (54) is one of the most influential approaches to state-based planning. FF introduces the Enforced Hill Climbing search algorithm, which searches exhaustively for nodes with a better heuristic value than the previous best node. Additionally, FF presents h_{FF}, a heuristic estimator based on the concept of Relaxed Planning Graph (RPG) (8). Given a plan Π, h_{FF} devises a plan that attains the relaxed planning task departing from the state associated to Π. h_{FF} constitutes one of the most relevant results in heuristic planning to date.

The previous planning systems established heuristic search as the most usual approach to tackle state-based planning. However, most planning systems today are sophisticated tools that combine heuristic search with additional techniques to increase the efficiency of the search.

Fast Downward (FD) (50) is a heuristic-based planner that uses a multi-valued representation for the planning tasks, instead of the more common propositional representation. FD makes use of *SAS+*-like (2) state variables to model the facts that conform states. For each state variable, FD infers its associated Domain Transition Graph (DTG), a structure that reflects the evolution of the value of a variable according to the actions of the task. The information of the DTGs is compiled into the Causal Graph, which displays the dependencies between the different state variables. FD applies a best-first multi-heuristic search, alternating in an orthogonal way h_{FF} and h_{CG}, a heuristic estimator calculated by means of the Causal Graph.

The LAMA planner (95) is one of the first works to apply *landmarks* in order to improve the accuracy of the heuristic search. A landmark is a fact that holds at some point in every solution of a planning task. LAMA is built upon FD and reuses its multi-heuristic search strategy to alternate a landmark-based estimator and the

well-known h_{FF} heuristic. LAMA is still one of the top-performing single-agent planning systems.

2.1.2.2 Partial-Order Planning

State-based planners apply a total-order approach; i.e., the actions in a plan are sequentially executed in the same order in which they are obtained. If a state-based planner chooses a wrong action, it has to introduce another action to undo the effects of the first one. As opposite to state-based models, the POCL (Partial Order Causal Link) Partial-Order Planning (POP) paradigm (3) introduces a more flexible approach: POP-based planners work over all the task goals simultaneously, maintaining partial-order relations between actions without compromising a precise order among them until the plan's own structure determines it. This mechanism, based on deferring decisions during the planning search, is known as the *least commitment* strategy (121).

Instead of performing a state-based search, POP models adopt a plan-based search approach. That is, a POP system builds a search tree in which each node represents a different *partial plan*, rather than managing the notion of planning state. POP is classified as a backward-chaining search approach since it begins the search by supporting the problem goals, and builds the solution plan backwards.

Definition 2.5. *(**Partial-order plan**) A **partial plan** is a tuple* $\Pi = \langle \Delta, OR, CL \rangle$. *$\Delta$ is the set of actions or steps in Π. OR is a set of ordering constraints ($\prec$) over the steps in Δ. CL is a set of causal links over Δ. A causal link is of the form $\alpha \xrightarrow{l} \beta$, where $\alpha \in \Delta$ and $\beta \in \Delta$. $\alpha \xrightarrow{l} \beta$ indicates that there is a literal l such that $l \in ADD(\alpha)$ and $l \in PRE(\beta)$. α is then said to support the precondition $l \in PRE(\beta)$ through the causal link $\alpha \xrightarrow{l} \beta$.*

The set $\Delta \in \Pi$ contains two fictitious steps, α_i and α_f. The effects of α_i model the initial state I, while the preconditions of α_f represent the set of goals G.

An *empty* partial plan is defined as $\Pi_0 = \langle \Delta_0, OR_0, CL_0 \rangle$, where OR_0 and CL_0 are empty sets, and Δ_0 contains only the fictitious steps α_i and α_f.

The introduction of new actions in a partial plan may trigger the appearance of *flaws*. On the one hand, the plan may contain *open conditions*, that is, preconditions that are not yet supported through a causal link. Initially, the open goals of Π_0 are the preconditions of α_f, that is, the set of goals G. On the other hand, the causal links of a partial order plan may become unsafe because of the appearance of *threats*.

Definition 2.6. *(**Threat**) A **threat** over a causal link $\alpha \xrightarrow{l} \beta$ is caused by an action γ that is not ordered w.r.t. α or β, and $l \in DEL(\gamma)$. That is, γ may potentially compromise the causal link $\alpha \xrightarrow{l} \beta$ since it may delete l before the execution of β, thus making the causal link unsafe (44).*

Threats are addressed through the introduction of either an ordering constraint $\gamma \prec \alpha$ (this is called *demotion* because the causal link is posted after the threatening action) or an ordering $\beta \prec \gamma$ (this is called *promotion* because the causal link is placed before the threatening action) (44).

A *solution plan* for a partial-order planning task is a threat-free partial plan in which the preconditions of all the actions are supported through causal links; that is, $\forall \beta \in \Delta,\ \forall l \in PRE(\beta),\ \exists \alpha \xrightarrow{l} \beta,\ \alpha \in \Delta$.

UCPOP (90) is one of the first and most significant approaches in implementing a sound and complete POP algorithm. The Versatile Heuristic POP (VHPOP) (126), loosely based on UCPOP, combines the POP paradigm with some of the advancements in state-based heuristic planning. More precisely, VHPOP adapts the h_{add} additive heuristic to a plan-space backward-chaining context.

Planning systems such as UCPOP and VHPOP put the POP paradigm at the focus of the planning research in the 1990s. However, POP faces important challenges related to its lack of scalability, which caused the single-agent planning

research community to put the emphasis on state-based approaches. The problem of defining a high-quality heuristic to guide the POP plan-space search remains unsolved. Currently, there is not a single POP heuristic that can make this process competitive against the latest state-based planning frameworks (77, 126).

Some recent works, however, have successfully reformulated the classical POP algorithm as a forward-chaining plan-space search (18). Plans in forward-chaining POP are built departing from the initial state, which makes it possible to compute a *frontier state* from each plan; i.e., the state that results from applying the set of actions in the plan. The use of frontier states is a critical turning point that allows these methods to apply effective state-based heuristic functions, which has given rise to remarkably efficient POP systems, such as OPTIC (6) and FLAP (100).

POP has thus experienced a revival thanks to the novel forward-chaining techniques. Moreover, these techniques are used in other fields, such as temporal planning and, in particular, distributed and multi-agent planning. This is due to the flexibility of POP, along with its ability to efficiently handle concurrency (12).

2.1.2.3 Other approaches to single-agent planning

Aside from state-space heuristic search and POP, other search strategies have been proposed to solve the single-agent classical planning problem. Among these proposals, we can highlight the following ones:

Planning graph. This technique, firstly introduced in the Graphplan system (8), compiles a planning graph, a compact structure that encodes all the possible plans up to a pre-established length. This structure is then used to guide the search process. Planning graph approaches take a middle ground between state-based and POP techniques, exploring the search space defined by the planning graph, rather than the state or plan space.

The planning graph structure has been of remarkable influence in state-based heuristic planning, since it is used by the popular h_{FF} estimator to solve the delete-relaxed problem (54).

Planning as satisfiability. This approach maps the planning task into another well-known problem for which there exist efficient solvers. More precisely, the task is translated to a propositional satisfiability (SAT) problem. The task is encoded as a propositional formula, and then addressed through a SAT solver, which solves it by determining whether the formula has a model.

The SATPLAN system (61) first devises a planning graph and then translates its information into a SAT problem, iteratively applying a general SAT solver to solve it. The solution of the SAT problem is then translated again into a plan.

Hierarchical Task Network (HTN). In HTN planning (35), the objective is not to attain a set of goals, but to perform a set of tasks. Tasks can be either primitive (directly solvable through an action) or non-primitive (decomposable into a set of smaller tasks). The input of the planner includes a set of actions and a set of *methods*, which are prescriptions that indicate how to decompose a particular task. HTN proceeds by progressively decomposing the non-primitive tasks until only primitive tasks remain.

Nowadays, HTN is one of the more widely-used planning methods for practical applications (44).

2.2 Cooperative Multi-Agent Planning

Multi-Agent Planning (MAP) extends automated planning by distributing the planning task among several entities which work together to devise a competent joint plan that meets the task goals. This generalization entails some key differences with respect to the more restrictive single-agent planning approach, such as the distribution of the task among agents or the coordination of the agents' activities.

Whereas some approaches to MAP, like Best-Response Planning (58), consider the problem of planning with competitive and self-interested planning agents, this PhD thesis focuses on the topic of *cooperative MAP*. Self-interested MAP involves using techniques other than the ones used in cooperative planning, such as game theory, auction systems or preference-based planning.

Despite agents being fully cooperative, the preservation of the agents' privacy has arisen as one of the fundamental topics in cooperative MAP, and thus, privacy is one of the central aspects of this work.

This section first describes a cooperative MAP task to illustrate the additional challenges MAP presents with respect to single-agent planning. Next, we tackle the problem of representing a cooperative MAP task and review the most relevant MAP specification languages. Following, we analyze the topic of privacy preservation in MAP. Afterwards, we describe the main features that characterize a cooperative MAP framework and classify the most recent approaches. Finally, we discuss the use of heuristic search in cooperative MAP and briefly review the existing MAS platforms that facilitate the design of distributed MAP systems.

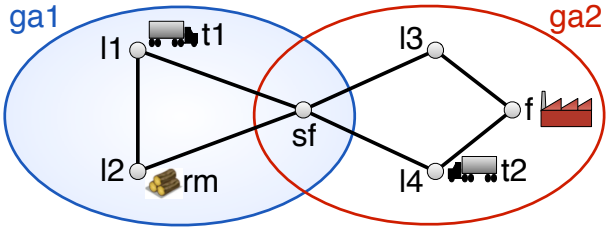

Figure 2.2: Example MAP task

2.2.1 Cooperative MAP task characterization

A cooperative MAP task involves several planning entities, or agents, working together in a shared environment. Agents must devise a joint plan to attain a set of global objectives or goals. In order to illustrate the features of a MAP task, let us introduce a brief application example loosely based on Example 1.

Example 2. *Consider the transportation task in Figure 2.2, which includes three different agents. There are two transport agencies, $ta1$ and $ta2$, each of them having a truck, $t1$ and $t2$, respectively. The two agencies work in two different geographical areas, $ga1$ and $ga2$, respectively. The third agent is a factory, f, which is placed in the area $ga2$.*

To manufacture products, factory f requires raw materials, rm, that are gathered from area $ga1$. In this task, agents are specialized: $ta1$ and $ta2$ have the same capabilities, but they act in different geographical areas; i.e., they are spatially distributed agents. Moreover, the factory agent f is functionally different from $ta1$ and $ta2$.

The goal of this task is for f to manufacture a final product fp. In order to carry out the task, $ta1$ will send its truck $t1$ to load the raw materials rm, located in $l2$, and then transport them to a storage facility, sf, that is placed in the intersection of both geographical areas. Then, $ta2$ will complete the delivery by using its truck $t2$ to transport rm from sf to f, which will in turn manufacture the final product fp. Therefore, this task involves three specialized agents that are spatially and functionally distributed and must cooperate to accomplish a common goal.

Example 2 emphasizes most of the key elements of a cooperative MAP task. First, the planning actions are distributed among the planning agents. The spatial and/or functional distribution of planning agents gives rise to *specialized agents* that have different knowledge and capabilities.

Agent specialization entails the need of *cooperation*, since agents are forced to interact in order to solve the task goals. The complexity of a MAP task is often described by means of its *coupling level* (14), that is, the number of interactions that arise among agents during the resolution of a MAP task. According to this parameter, cooperative MAP tasks can be classified as follows:

- *Loosely-coupled tasks*: these tasks require few to none interactions among agents. Consider a MAP version of the well-known *Satellite* domain: each satellite captures images in isolation, using its equipped instruments. Interactions in this domain are not required, since agents do not need to cooperate to fulfill the task goals (*positive interactions*) and their actions do not prevent other agents from attaining other goals (*negative interactions*).

- *Tightly-coupled tasks*: in these tasks, a large number of interactions among agents is required to obtain a solution plan. In Example 2, agent f cannot manufacture the final product fp unless $ta2$ delivers the raw materials rm to f, while agent $ta2$ requires $ta1$ to deliver rm in sf. Therefore, interactions among $ta1$, $ta2$ and f are necessary to solve this particular MAP task.

The coupling level is a key aspect of MAP tasks that determines the design of many approaches to MAP: some methods are more effective or even limited to loosely-coupled tasks, while others take a more general approach and are equally effective regardless of the coupling level of the task.

Another basic aspect of MAP tasks, which is not directly related to the coupling level, is the presence of *cooperative goals*; i.e., goals that cannot be solved

individually by any agent because their resolution involves the cooperation of specialized agents. The task in Example 2 has a cooperative goal, since none of the agents can achieve the manufacturing of the final product by itself. Instead, they must make use of their specialized capabilities and interact with each other to deliver the raw materials and manufacture the final product.

The distribution of the task information in MAP also stresses the issue of *privacy*, which is one of the basic aspects that should be considered in multi-agent applications (101). Since the three parties involved in Example 2 are specialized in different functional or geographical areas of the task, most of the information managed by factory f is not relevant for the transport agencies and vice-versa. The same occurs with the transport agencies $ta1$ and $ta2$. Additionally, agents may not be willing to share the sensitive information of their internal procedures with the others. For instance, $ta1$ and $ta2$ are cooperating in this particular delivery task, but they might be potential competitors since they work in the same business area. Therefore, agents in MAP want to minimize the information they share with each other, either for strategic reasons or simply because it is not relevant for the rest of the agents in order to address the planning task.

2.2.2 Representation in cooperative MAP

The representation of a cooperative MAP task involves modelling several elements that are not present in single-agent tasks. Therefore, single-agent planning specification languages are not expressive enough to fulfil the requirements of a MAP task, which stresses the need of specialized task specification languages for MAP.

Some MAP frameworks include a centralized entry point that distributes the MAP task among agents; however, this is not a feature shared by each and every MAP model. Particularly, fully-distributed approaches to MAP require a *factored* input, such that each agent receives its own, independent domain and problem.

Additionally, the representation of the individual tasks should include information regarding agents' privacy, so that the agent's model distinguishes among the information that can and cannot be shared with other planning entities.

Among the existing MAP languages, *MA-PDDL*[1] stands out as the first attempt to create a *de facto* standard specification language for cooperative MAP tasks. *MA-PDDL* extends and adapts *PDDL3.1* to a MAP context, and allows for factored (`:factored-privacy` requirement) and unfactored task representations (`:unfactored-privacy` requirement), which can be easily adopted by most of the existing MAP frameworks.

In order to model the task in Example 2 with *MA-PDDL*, we use the factored specification, so that each agent receives two independent files that encode its domain and problem specification. For the sake of simplicity, we show fragments of the *MA-PDDL* specification only for agents $ta1$ and f, since $ta1$ and $ta2$ are functionally equivalent.

The next two fragments of code define the type hierarchy of agents $ta1$ and f, respectively:

```
(:types area location package agency - object
        truck place - location)
```

The type hierarchy for $ta1$ includes an extra type with respect to Example 1, `agency`, which is used to define the transport agencies.

```
(:types location package product - object
        truck place - location
        factory - place)
```

Agent f is defined through the type `factory`, which is a subtype of `place` since f acts also as a `place` reachable by a `truck`.

[1] Please refer to `http://agents.fel.cvut.cz/codmap/MA-PDDL-BNF-20150221.pdf` for a complete BNF definition of the syntax of *MA-PDDL*.

The following code excerpts define the predicates used by agents $ta1$ and f, respectively:

```
(:predicates
    (at ?p - package ?l - location)
    (:private
        (link ?p1 - place ?p2 - place)
        (owner ?a - agency ?t - truck)
        (in-area ?p - place ?a - area)
        (pos ?t - truck ?l - location))
)
```

The domain specification for agent $ta1$ includes the public predicate `at`, which models the position of the `packages`. The predicates `link`, `owner`, `pos` and `in-area` are defined as private, which prevents `agency` $ta1$ from disclosing internal information, such as the `places` that compose the agent's working `area`, as well as the connections among them, the `trucks` that `agency` $ta1$ possesses and their locations.

```
(:predicates
    (manufactured ?p - product)
    (:private
        (at ?p - package ?l - location)
        (pending ?p - product))
)
```

Predicates for agent f also include `at`, so that the `factory` is notified about the arrival of `packages`. This predicate is defined as private since the `factory` does not need to inform the rest of agents about the position of the `packages`. The `manufactured` predicate is defined as public, so that the rest of agents know whether the task goal is fulfilled. On the other hand, `pending` is defined as private in order for the `factory f` to occlude its `pending` orders.

Agents in Example 2 are specialized and, among other differentiating elements, they have different planning operators. Three different operators, `load`, `unload`

and `drive`, are defined for transport `agencies` $ta1$ and $ta2$. The following fragment of code shows the encoding of the `drive` operator:

```
(:action drive
    :parameters   (?a - agency ?t - truck ?p1 - place ?p2 - place)
    :precondition (and (in-area ?p1 ?a)(in-area ?p2 ?a)(owner ?a ?t)
                  (pos ?t ?p1)(link ?p1 ?p2))
    :effect       (and (not (pos ?t ?p1))(pos ?t ?p2))
)
```

The `drive` operator of agent $ta1$ is used to drive a `truck` between two different `places`. The movements of $ta1$'s `trucks` are limited to its working area, `ga1`, through the `in-area` preconditions. $ta1$ can only `drive` a `truck` if it is the `owner` of the truck.

The `factory f` can only apply the `manufacture` operator:

```
(:action manufacture
    :parameters   (?f - factory ?rm - package ?fp - product)
    :precondition (and (at ?rm ?f)(pending ?fp))
    :effect       (and (not (pending ?fp))(manufactured ?fp))
)
```

This operator allows the `factory` f to `manufacture pending` products. The operator requires a `package` of raw materials to be delivered to the `factory` in order for it to be executed.

The problem block starts with the definition of the task objects for each agent.

```
(:objects
    ta1 - agency
    ga1 - area
    t1 - truck
    rm - package
    l1 l2 sf - place)
```

The transport `agency` $ta1$ has knowledge about its `truck t1` along with the `places` within its working `area ga1` and the `package rm`.

```
(:objects
    rm - package
    fp - product
    f - factory)
```

The objects known to agent f include the `package rm` and the `product fp`.

Next, we show the initial state specification for agents $ta1$ and f, respectively:

```
(:init
    (pos t1 l1)(at rm l2)(owner t1 ta1)
    (link l1 l2)(link l2 l1)(link l1 sf)(link sf l1)(link l2 sf)
    (link sf l2)(in-area l1 ga1)(in-area l2 ga1)(in-area sf ga1)
```

Agent $ta1$ knows the initial state of its working area `ga1`, which includes the position of its `truck t1` and the `package rm`, as well as the `links` among the `places` within `ga1`: `l1`, `l2` and `sf`.

```
(:init (pending fp))
```

The initial state of agent f includes a single literal which indicates that the production of `product fp` is still `pending`.

Finally, the `:goal` section, which is common to the three participating agents, includes a single global goal indicating that the `product fp` must be `manufactured` for the MAP task to be completed:

```
(:goal (manufactured fp))
```

2.2.2.1 MAP task specification languages

As reviewed in section 2.1.1.1, there is a large body of research on planning task specification languages. Since planning has traditionally been regarded as a centralized problem, the most popular definition languages, such as the different versions of *PDDL* (the Planning Domain Definition Language), are designed exclusively to model single-agent planning tasks.

MAP introduces a set of requirements that are not present in single-agent planning, such as privacy or specialized agents, which motivates the development of specification languages for multi-agent planning.

MA-STRIPS (14), which was designed as a minimalist multi-agent extension to *STRIPS* (36), is one of the most common MAP languages in the literature. *MA-STRIPS* enables the definition of a set of planning agents and provides each entity with the actions it can execute.

The MAP framework introduced in this PhD thesis presents several advanced features that motivated the definition of our own *PDDL*-based specification language (the language syntax is detailed in (115)), instead of using *MA-STRIPS*.

Since we model the world states through state variables instead of predicates, our MAP language is based on *PDDL3.1* (64), the latest version of *PDDL*. Unlike its predecessors, which model planning tasks through predicates, *PDDL3.1* incorporates state variables that map to a finite domain of objects of the task.

Due to the inherently distributed nature of the MAP model presented in this PhD thesis, we use *factored* task descriptions in our language; that is, each agent receives a separate domain and problem description, which define the typology of the agent and its local view of the MAP task, respectively. The domain description maintains the structure of a regular *PDDL3.1* domain, while the problem is extended with an additional `:shared-data` section, which specifies the information that an agent can share with the rest of participants.

As discussed in the previous section, the recently introduced *MA-PDDL* aims to be the *de facto* standard MAP specification language. *MA-PDDL* is derived from *PDDL3.1* and includes specific constructs to define the private information managed by each agent. MAP tasks can be defined in a factored or unfactored way, allowing a wide variety of planners to support *MA-PDDL*.

Regarding expressiveness, our MAP language and *MA-PDDL* are very similar. For this reason, our framework was recently updated to support both our specification language as well as the factored version of *MA-PDDL*.

2.2.3 Privacy in cooperative MAP

The MAP task of Example 2 includes two different agents, $ta1$ and $ta2$, that represent two transport agencies. Whereas both agents are working cooperatively in this particular task, they are likely not willing to reveal sensitive information on their internal procedures to a potential competitor.

Therefore, Example 2 emphasizes the role of *privacy* as one of the basic aspects that should be strengthen in distributed MAP. The topic of privacy was not raised until recently since early MAP approaches focused in other aspects, such as information distribution (80).

However, privacy is nowadays one of the main research topics in MAP. The current state-of-the-art literature reveals a growing effort on the analysis, formalization and usage of privacy as a means to improve the performance of MAP systems. Most of the recent MAP models are specifically designed to preserve agents' privacy (11, 72).

Privacy in MAP can be discussed from a theoretical or a practical perspective. There exist various theoretical privacy models which present different definitions and usage of private information. On the other hand, the implementation of privacy in MAP approaches also feature significant distinctiveness.

2.2.3.1 Theoretical privacy models

Privacy in MAP can be applied to both actions and literals (preconditions and effects). A private literal or action is ideally known only to an agent or a subset of agents. In Example 2, the literal `(pos t1 sf)` indicates that the truck `t1` is

in location `sf`, as modelled in section 2.2.2. Agent $ta2$ is aware of this situation, since the public location `sf` is known to both $ta1$ and $ta2$. However, $ta2$ will ignore the position of `t1` whenever the truck is placed at either `l1` or `l2`, because both locations are private to agent $ta1$. In other words, the literal `(pos t1 sf)` is defined as *public* in Example 2, while `(pos t1 l1)` and `(pos t1 l2)` are considered *private* to $ta1$.

We identify two different theoretical privacy models existing in the literature:

- **MA-STRIPS:** The MA-STRIPS model (14) is the currently most extended privacy scheme. Given an agent i, a literal is said to be private to i if it is required and affected only by the actions of i. An action α is thus said to be private to agent i if all its preconditions and effects are private to i. Otherwise, α is said to be public and is known to all the participants in the task.

  However, public actions may include some private preconditions and effects. We refer to the *public projection* of an action α as an abstraction α_p that contains only the public preconditions and effects of α. In this latter case, agent i will reveal the projection α_p to the rest of participants rather than α to enforce privacy.

- **Our model:** The MAP model introduced in this PhD thesis presents some key differences with respect to MA-STRIPS, since it defines a pairwise privacy model; that is, a literal can be either public, private to an agent, or known to a *subset of agents*.

  Given an action α of an agent i, the projection of α received by two agents j and k may then differ according to the pairwise privacy constraints between i and the other two agents.

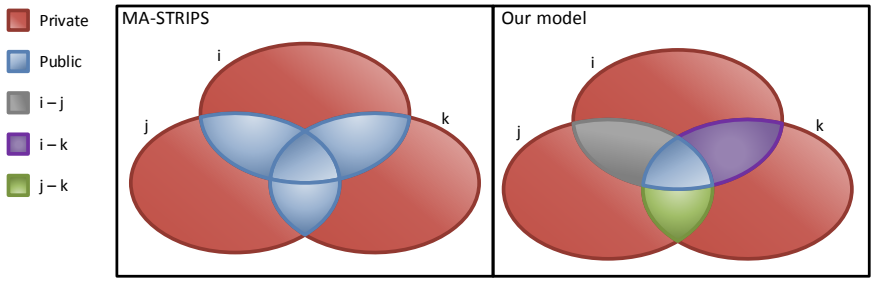

Figure 2.3: Privacy in MA-STRIPS and our model

Figure 2.3 illustrates the differences between the MA-STRIPS and FMAP privacy models. The Venn diagram represents the information known to three agents i, j and k. MA-STRIPS considers that the public information is known to all the agents that take part in the MAP task. On the contrary, FMAP only labels as public the information that is shared by all the agents in the task (that is, the central area of the Ventt diagram). The elements known to only two of the agents are private to the third participating agent.

2.2.3.2 Practical application of privacy

Besides the theoretical privacy models, recent studies also discuss the practical privacy guarantees of a MAP algorithm. The work in (13) analyzes the implementation of privacy in current MAP methods and concludes that most of the existing frameworks offer a *weak* form of privacy that allows an agent to infer or deduce private information that is not explicitly transmitted by other agents.

Authors in (13) and (103) suggest a four-level privacy classification to analyze the privacy models of current MAP approaches:

- **No privacy**: Most of the earlier approaches to MAP do not implement any degree of privacy and allow agents to share whole search states. In some

cases, private data is transmitted but unused, and thus, these methods can be easily extended to support a weak notion of privacy (79).

- **Weak privacy**: A MAP system is said to be weakly private if agents do not communicate the private values of a variable to other agents during search (13). Additionally, agents in a weakly private MAP framework only reveal the public projection of their own actions.

  A privacy setting is considered weak when agents may infer the existence and properties of private variables, values, and action preconditions, from information communicated during the resolution of the MAP task.

  For example, let us consider the private literal `(at rm t1)` in Example 2, which describes the location of the package `rm`. This literal is private because truck `t1` is property of agent $ta1$, and thus, its status is only known by such agent. Suppose that `rm` is located in `sf` in one state, but it disappears in the following state. This can lead the rest of agents to deduce that there is some private literal indicating the presence of an undisclosed resource used by $ta1$ to transport the package.

  - **Obfuscation**: The simplest way to achieve weak privacy is to encrypt the private values of a variable in the public projection of the associated actions. Since the private values of an agent do not appear in the preconditions or effects of other agents' actions, obfuscation does not affect the actions' applicability.

    This technique implies the substitution of private information by unique identifiers in the public projection of each action, effectively preventing the access of other agents to the private data (11).

- **Object cardinality privacy**: A MAP algorithm preserves object cardinality privacy if, given an agent i and a type t, no other agent can infer the value of the cardinality of type t of agent i from the set of messages sent by agent i.

  In other words, this level of privacy strongly preserves the number of objects of a given type t that an agent i possesses, thus representing a middle ground between the weak and strong privacy settings.

  Occluding the cardinality of private objects is motivated by real-world scenarios. Consider, for example, the logistics task in Example 2.2. One can assume that the transport agencies that take part in the MAP task, $ta1$ and $ta2$, know that packages are delivered using trucks. However, agents would likely like to hide sensitive information related to their capabilities, such as the number of trucks they have, or their transportation routes.

- **Strong privacy**: A MAP algorithm is said to be strongly private if no agent can deduce private data other than the information related to its own actions' description, the public projection of other agents' actions, and the public projection of the solution plan (13). More precisely, a variable or a value is said to be strongly private if the rest of agents cannot deduce its existence from the information they possess.

  Strong privacy implies considering additional factors beyond the MAP algorithm itself. The amount of information that can be acquired by other agents directly depends on the nature of the communication channel (synchronous, asynchronous, lossy), as well as the features and computational power of the agents.

As it will be discussed in section 2.2.4, the vast majority of the state-of-the-art MAP methods can be classified in the *no privacy* and *weak privacy* levels. Earlier

approaches to MAP do not consider privacy at all, while the most recent proposals implement some form of weak privacy, using mostly *obfuscation* to occlude the private information in the public projection of the actions and states.

The recent DPP planner (103) is the first MAP system to offer object cardinality privacy guarantees, while SECURE-MAFS (13), an upgraded version of MAFS (79), is the only method in the literature that supports strong privacy.

2.2.4 Cooperative MAP systems taxonomy

As described in section 2.2.1, cooperative MAP tasks involve some additional challenges with respect to the more compact single-agent planning task formulation, such as information distribution, specialized agents, coordination or privacy. The state of the art in cooperative MAP presents a wide variety of approaches that can be categorized according to the techniques they use to address these challenges.

In order to define a MAP taxonomy, we first enumerate the most relevant aspects according to which MAP approaches can be classified:

- **Centralized/distributed planning**: MAP is concerned with planning *by* multiple agents (distributed MAP) or planning *for* multiple agents by means of a single planning entity (centralized MAP). In general, centralized MAP approaches leverage the distributed structure of the MAP tasks to improve the efficiency of the single-agent planning process. In contrast, agents in distributed MAP carry out planning jointly. For this reason, distributed MAP involves research topics such as the coordination of agents, the design of privacy-preserving planning methods, and the development of distributed heuristic functions.

- **Coordination of agents**: Distributed MAP can be viewed as the problem of coordinating agents in a shared environment where information is

distributed (27). Coordination can be applied at different points of the distributed MAP process. Some approaches perform coordination before planning, distributing the task or allocating the goals among the agents. Other models follow a *plan merging* approach; that is, agents apply local planning and the individual plans are coordinated afterwards into a global solution plan that addresses the MAP task. Finally, other frameworks *interleave* the planning and coordination activities, so that agents jointly build a solution plan in a coordinated fashion.

- **Underlying planning paradigm**: Most MAP approaches reuse single-agent planning technology adapting it to a distributed context. Many MAP frameworks are underpinned by state-based planning, while others are based on approaches such as partial-order planning (POP), satisfiability or planning as a constraint satisfaction problem (CSP).

- **Privacy preservation**: Besides the need for information distribution, privacy is also one of the main motivations to adopt a MAP approach. Privacy entails coordinating agents without making sensitive information publicly available. Whereas this aspect was initially relegated in MAP (119), the most recent approaches tackle this issue through the development of robust privacy-preserving algorithms. As described in section 2.2.3, the privacy guarantees offered by a MAP algorithm can be classified in four levels, ranging from no privacy to strong privacy preservation.

- **Local/global heuristics**: Most of the state-of-the-art approaches to MAP use heuristic search to improve the overall performance of the planning process. However, heuristic functions are usually locally applied by each agent, which diminishes the accuracy of the estimates because of the limited view of the agents over the MAP task. One of the current challenges of MAP

focuses on the development of *global heuristics* to match the accuracy of single-agent estimators.

Table 2.1 displays the main features of the state-of-the-art cooperative MAP approaches discussed in this section. The MAP taxonomy is organized following the topics of information distribution and coordination: first, we revise the most relevant centralized approaches to MAP, and then, we classify the distributed techniques with special emphasis on the particular combination of planning and coordination of each planner.

Approach	Distribution	Coordination	Paradigm	Privacy	Heuristics
MA-STRIPS	Centralized	Interleaved	CSP+Planning	Weak	-
TFPOP	Centralized	Interleaved	Forward POP	No	-
ADP	Centralized	Agent decomposition	State-based	No	Local
DPP	Centralized	Interleaved	State-based	Object cardinality	Local
MAPR	Distributed	Goal allocation	State-based	Weak	Local
μ-SATPLAN	Distributed	Goal allocation	dis-SAT	No	-
Planning First	Distributed	Plan merging	dis-CSP	No	-
MAFS	Distributed	Interleaved	State-based	No	Local
MAD-A*	Distributed	Interleaved	State-based	No	Local
SECURE-MAFS	Distributed	Interleaved	State-based	Strong	Local
GPPP	Distributed	Interleaved	State-based	Weak	Global
MAP-POP	Distributed	Interleaved	POP	Weak	Global
FMAP	Distributed	Interleaved	Forward POP	Weak	Global
MH-FMAP	Distributed	Interleaved	Forward POP	Weak	Global

Table 2.1: Summary of the state-of-the-art MAP approaches

Centralized MAP. Several works in the literature apply a centralized approach to MAP, taking advantage of the distributed structure of a MAP task to maximize the performance of the planner. Centralized MAP features two basic operations: a planning process to solve the local planning tasks of the agents and a coordination

procedure that combines the local solutions into a global plan that achieves the goals of the MAP task.

Centralized MAP assumes a single planning entity with complete knowledge of the task, which is rather unrealistic if the agents involved in the task have sensitive private information that they are not willing to disclose (99). For instance, transport agencies in Example 2 wish to conceal the information regarding their internal processes and business strategies; therefore, a centralized setting is not an acceptable solution.

Authors in (14) introduce MA-STRIPS, an early centralized MAP approach that combines CSP and planning, aimed to efficiently solve loosely-coupled tasks. MAP tasks are encoded through the homonymous *MA-STRIPS* language (see section 2.2.2.1). MA-STRIPS classifies actions and literals as either public or internal, thus establishing the first MAP privacy approach in the literature (see section 2.2.3).

MA-STRIPS identifies the coordination points of the task and uses a CSP to determine the public actions that can be executed in these points. Each agent solves its internal (private) planning problem defined between coordination points, which finally results in a sound solution plan for the MAP task.

TFPOP (67) is a centralized MAP approach that plans for multiple agents using a forward-chaining POP search scheme. TFPOP combines the flexibility of backward-chaining POP and the performance of forward search.

The Agent Decomposition-based Planner (ADP) (22) aims to exploit the multi-agent structure inherent to some planning tasks to speed up centralized planning. First, an automated process that decomposes *STRIPS*-like problems into MAP tasks is run. Then, a state-based centralized plan synthesis procedure is applied. In each iteration, ADP determines a set of subgoals that can be attained from the current state by one of the agents. A search process, guided through the well-known h_{FF} heuristic (54), is carried out to find a plan that achieves those

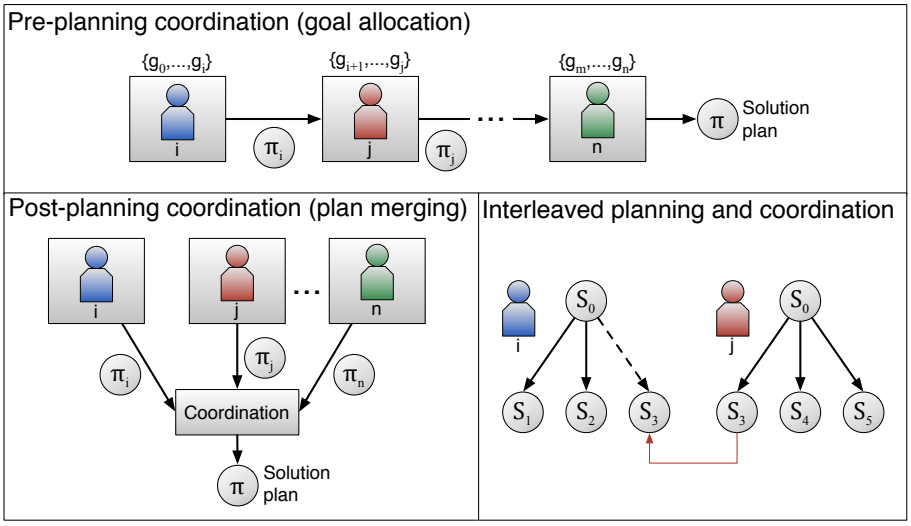

Figure 2.4: Planning and coordination schemes in distributed MAP

subgoals, which results in a new state. The process is repeated until a solution is found.

The DP-Projection Planner (DPP) (103), is a centralized MA-STRIPS planner that uses the Dependency-Preserving (DP) projection, a novel and accurate form of public projection. The DP projection is used by the single-agent planner Fast Downward to create a high-level plan. Then, the multi-agent solution plan is completed by extending the high-level plan with private agents' actions with the FF planner. DPP features object cardinality privacy guarantees, and, according to the results in (103), it is the current top-performing MA-STRIPS planner.

Distributed MAP. The principal aspect of distributed approaches is the combination of planning and coordination, which often determines the typology of tasks that can be solved by the planner (see Figure 2.4).

Various MAP models put the emphasis on coordinating agents before starting the planning activity. *Pre-planning coordination* divides the task among the agents, so that each agent is endowed with a portion of the MAP task. This is a remarkably efficient coordination scheme, but its usage may lead to some limitations in terms of applicability.

Multi-Agent Planning by plan Reuse (MAPR) (11) distributes the task goals among the agents before planning. Then, agents sequentially launch the state-based planner LAMA (95). Each agent takes the solution plan of the previous agent as an input; therefore, the global solution plan is built incrementally (see top picture in Figure 2.4). Agents weakly preserve privacy by obfuscating the private information they incorporate into the plan. While MAPR is one of the best-performing approaches to MAP, its applicability is limited to tasks without specialized agents or cooperative goals, since it is designed under the assumption that each stand-alone agent is capable of solving its allocated goals.

Another group of MAP techniques focuses on *plan merging*. In this approach, agents perform planning individually, while a subsequent coordination process is applied to come up with a global solution for the MAP task (see bottom left picture in Figure 2.4). Plan merging is suitable for solving loosely-coupled tasks in which agents are capable of achieving the problem goals by themselves (20). Therefore, plan merging is an appropriate post-planning coordination mechanism to tackle problems in which agents can solve the different problem goals independently and the majority of the environment resources are not shared.

However, plan merging presents several limitations. On the one hand, agents perform planning individually using their local projection of the MAP task. For this reason, MAP methods based on plan merging lose flexibility against other MAP proposals. On the other hand, individual planning combined with a post-

planning coordination strategy is not adequate to solve tightly-coupled problems, since merging may introduce exponentially many ordering constraints in problems which require a coordination effort (20).

Additionally, plan merging is not an effective method for attaining cooperative goals since this resolution scheme generally assumes that each agent is able to solve a subset of the task's goals by itself. However, some approaches use plan merging to coordinate local plans of specialized agents. In this case, the effort is placed on discovering the interaction points among agents through the public information that they share (80).

Planning First (80) is a MA-STRIPS planner built upon the combination of CSP and Planning introduced in (14). Planning First is designed to tackle loosely-coupled tasks with specialized agents in a fully-distributed fashion. Agents individually carry out planning through a state-based planner, and afterwards, the resulting local plans are coordinated by solving a distributed CSP (56).

μ-SATPLAN (28) extends the satisfiability-based planner SATPLAN (61). The MAP task goals in μ-SATPLAN are *a priori* distributed among the agents. Similarly to MAPR (11), μ-SATPLAN performs a sequential approach to plan coordination: agents perform individual planning in order, so that each participant takes the previous agent's solution as an input, thus progressively generating a coordinated plan. Authors confirm that μ-SATPLAN is limited to loosely-coupled tasks without cooperative goals, since it is assumed that each agent can solve its goals by itself (28).

The work in (20) introduces a distributed coordination framework based on POP that addresses the interactions that emerge between the agents' local plans. This framework, however, does not consider privacy. The proposal in (113) is based on the iterative revision of the agents' local plans. Agents in this model cooperate by mutually adapting their local plans, with a focus on improving their

common or individual benefit. This approach also ignores privacy and assumes agents to be fully cooperative. The method in (120) uses multi-agent plan repair to solve inconsistencies among the agents' local plans while maintaining privacy.

A third approach to MAP *interleaves* the planning and coordination activities, which results in a unified vision of cooperative MAP. Intertwining planning and coordination gives rise to general-purpose methods that are not constrained to loosely-coupled tasks, thus being able to efficiently tackle complex tasks with specialized agents and cooperative goals.

In general, MAP techniques that interleave planning and coordination are not as efficient as plan merging models at solving loosely-coupled tasks, since coordination is a costly process that usually has a negative impact in the computational execution time of the planning tasks (22). Nevertheless, the interleaved scheme offers an appropriate trade-off between efficiency and generality, solving a much wider range of MAP tasks.

One of the first domain-independent MAP models is the Generalized Partial Global Planning (GPGP) framework (68). Agents in GPGP have a partial view of the world and communicate their local plans to the rest of the agents, which in turn merge this information into their own partial global plan in order to progressively improve it.

The Multi-Agent Forward Search (MAFS) (79) is a distributed forward-chaining privacy-preserving MA-STRIPS system in which agents individually explore a state space. An agent in MAFS maintains an independent open list of states and expands the best one according to its local heuristic estimates. To optimize the search, the resulting states are only shared if they are relevant to other agents (see bottom right picture in Figure 2.4); that is, an agent i sends a state s to an agent j if j

has at least one action whose public preconditions hold in s, and the action that gave rise to s is public.

Authors in (79) also introduce an optimal variation of MAFS, the Multi-Agent Distributed A*, MAD-A*. In this case, an agent selects for its expansion the state that minimizes $f = g + h$, where h is estimated through an admissible heuristic. MAD-A* is proven to be cost-optimal if all the actions that achieve some goal condition are considered public.

Agents in both MAFS and MAD-A* share complete search states, but these approaches can be easily extended to support privacy, since agents do not take advantage of the states' private information. For this reason, a third version of MAFS, SECURE-MAFS (13), focuses on security, being the first work in the MAP literature to guarantee a form of *strong privacy*. Authors in (13) prove that agents in SECURE-MAFS cannot deduce any private information from the data they receive from other agents duringp planning.

The Greedy Privacy-Preserving Planner (GPPP) (72) builds upon MAFS and noticeably improves its search performance through the use of a global landmark-based heuristic function. In GPPP, agents effectively preserve privacy by masking the private information in the shared states through private state identifiers.

The three versions of the MAP framework introduced in this PhD thesis, MAP-POP, FMAP and MH-FMAP, apply a fully-distributed MAP scheme that interleaves planning and coordination, performing a general multi-agent search. In our model, the participants explore a distributed search tree in which nodes are partial-order plans whose actions are contributed by different agents. In MAP-POP, agents perform an incomplete POP search, while FMAP and MH-FMAP apply a sound and complete forward-chaining POP to progressively develop and coordinate a joint plan until its completion. Our MAP approaches weakly preserve

privacy by occluding the private information in the partial plans exchanged by the agents.

2.2.5 Heuristic search in cooperative MAP

Many of the aforementioned state-of-the-art cooperative MAP frameworks apply some form of heuristic search to guide the planning process. Since agents usually have a limited knowledge of the task, the quality of an agent's local estimates is rather poor in comparison to the global heuristics applied in single-agent planning.

A global heuristic in MAP is the application of a heuristic estimate to the MAP task carried out by several agents which have a different knowledge of the task and, possibly, privacy requirements.

The development of global estimators constitutes one of the current challenges in cooperative MAP (78). This is caused by the inherent features of MAP scenarios, which introduce additional requirements to the heuristic evaluation and make it an arduous task:

- The data of a MAP task is usually distributed across the agents; unlike single-agent planning, in MAP there does not exist an entity that centralizes the information of the task. Hence, a robust communication protocol among the agents is required to compute global heuristic estimates.

- Most MAP models deal with agents' privacy. The communication protocol must then guarantee that agents are able to calculate heuristic estimates without revealing sensitive private information.

In some works, the features of the planning model force the application of a local heuristic search scheme. For instance, in MAPR (11), goals are allocated to the agents, which then solve their subtasks iteratively, so that the solution of an

agent is communicated to the next participant. Thus, the heuristic functions used by MAPR, h_{FF} (54) and h_{Land} (95), are applied from a local standpoint.

Local search heuristics have also been used in MAP approaches whose planning model is suitable to accommodate global heuristic functions. Agents in MAFS and MAD-A* (79) generate and evaluate search states locally. An agent shares only the states that are relevant to other planning entities, along with their local heuristic values. When an agent receives a state from another participant, it performs a local evaluation of the state and then it chooses between or combines the heuristic value it has calculated and the value it received along with the state.

In (78), authors test MAD-A* with two optimal heuristic functions, LM-Cut (51) and merge-and-shrink (53), applied locally by each agent. Despite using only local estimators, MAD-A* is proven to be cost-optimal when using the aforementioned admissible heuristics.

Authors in (106) introduce a multi-agent design of the h_{FF} heuristic. This adaptation, based on the use of *distributed Relaxed Planning Graphs* (dis-RPGs) (127), yields the same results as the original single-agent design of h_{FF} (54). However, the construction and exploration of a dis-RPG entails many communications between agents, resulting in a computationally expensive approach.

The work in (107) presents the distributed design of several relaxation heuristics, namely h_{add}, h_{max} and a relaxed version of h_{FF}. In this work, authors replace the dis-RPG by an *exploration queue*, a more compact structure that significantly reduces the need of communications among agents. The distributed version of h_{FF}, however, does not yield the same results as the original single-agent version.

Finally, GPPP (72), introduces a distributed version of a privacy-preserving landmarks extraction algorithm for MAP. The heuristic value of a plan is calculated as the sum of the local heuristic estimates computed by each agent. GPPP improves

the performance of the MA-STRIPS-based planner MAFS thanks to the accurate estimates provided by this landmark-based heuristic (79).

This PhD thesis thoroughly explores the topic of distributed heuristics, introducing two different global estimators, h_{DTG} and h_{Land} (117). The first heuristic, h_{DTG}, estimates quality of a plan by building a relaxed plan between the *frontier state* (6) of such plan and the goal state. The relaxed plan is calculated by finding the shortest paths between values of the frontier state and the goals in the *Domain Transition Graph* (49) associated to each state variable of the MAP task.

Additionally, we introduce a fully-distributed landmark-based heuristic, h_{Land}, that estimates the quality of a plan by taking account of the landmarks that are not satisfied in such plan.

Both heuristics are combined through a novel multi-heuristic approach that alternates the heuristic estimators in an orthogonal fashion, dramatically improving the performance of the search procedure.

2.2.6 MAS platforms

In order to develop fully-distributed MAP systems, it is necessary to provide the planning agents with basic operation and communication capabilities, so that they can interact with each other and work cooperatively. Over the last years, some relevant works in frameworks for the design of MAP systems have been published.

The work in (123) presents a complete MAP architecture for large-scale problem solving, which organizes agents into planning cells committed to a particular planning process. The TAEMS domain independent coordination framework (68) provides agents with planning capabilities, and applies the GPGP method to coordinate them. The domain-independent multi-agent system infrastructure RETSINA (110) introduced a specific planning component (86). Once integrated

into the agents' internal structure, this component provides them with planning capabilities.

Aside from the frameworks exclusively designed to support MAP, there is a growing body of work on general-purpose middlewares for the design of Multi-Agent Systems (MAS). One of the basic requirements to build a MAS is to define a standard communication language. The Foundation for Intelligent Physical Agents (FIPA) provides a collection of standards to promote the interaction of heterogeneous agents and the services that they can represent. In particular, the FIPA Agent Communication Language (ACL) (81) has become the *de facto* standard for communicating agents in most MAS platforms.

The Java Agent DEvelopment Framework (JADE) (5) is a Java-based MAS platform that provides the sets of services, conventions and knowledge required by agents to interact with each other, including an asynchronous message passing mechanism and a yellow-pages service, among other features that facilitate the development of distributed frameworks, such as a MAP system.

The MAP framework introduced in this PhD Thesis is built upon Magentix2 (37), a general-purpose platform for open MAS. Magentix2 provides a variety of services and tools for the optimized management of open MAS, such as a communication infrastructure that allows agents to interact via the FIPA ACL language.

3

Selected papers

This chapter compiles the most relevant research papers published during the development of this PhD thesis. The articles are chronologically listed and provide a thorough description of the scientific contributions that conform this work.

This chapter is organized as follows: section 3.1 describes the selected articles and briefly summarizes their contents. The subsequent sections include the full text of the research articles adapted to the format of this investigation.

3.1 Summary of the selected papers

The results obtained during the development of the present PhD thesis have been systematically communicated through the publication of a wide range of scientific papers. This chapter focuses on the impact articles that synthesize the main body of this work, offering a clear and comprehensive summary of the obtained results.

The next sections arrange these articles according to their date of publication, which gives a clear idea of the evolution of the research and the main milestones reached during the development of this PhD thesis.

An approach to multi-agent planning with incomplete information. Section 3.2 presents the full text of our first impact paper, included in the proceedings of the 2012 20th European Conference on Artificial Intelligence[1] (ECAI).

This article offers an overview of our initial approach to cooperative MAP, the so-called MAP-POP. MAP-POP is a general-purpose approach to MAP capable of tackling complex tightly-coupled tasks in which agents have different abilities and an incomplete knowledge of the MAP task. Agents in MAP-POP incorporate a backward-chaining Partial Order Planning (POP) system that allows them to individually generate refinement plans.

Sections 3.2.2 and 3.2.3 provide our first formalization of a cooperative MAP task and a summary of the algorithms behind the MAP-POP framework, respectively. Section 3.2.3.3 analyzes the soundness and lack of completeness of MAP-POP, one of the main limitations of our initial approach.

The experimental results in section 3.2.4 compare the performance of MAP-POP against one of the most representative MAP frameworks at that point, Planning First (80). The benchmark used in this paper includes three well-known planning domains (*Satellite*, *Rovers* and *Logistics*) from the International Planning Competition[2] (IPC) suites, adapted to a MAP context. The results show that MAP-POP scales up much better than the distributed CSP approach of Planning First, obtaining a much greater coverage (number of solved problems) in the three tested domains.

[1] `http://www.eccai.org/ecai.shtml`
[2] `http://ipc.icaps-conference.org/`

A flexible coupling approach to multi-agent planning under incomplete information. An extension of MAP-POP, originally published in the Knowledge and Information Systems[1] journal in 2014, is presented in section 3.3.

This article provides insight into some aspects of MAP-POP not discussed in the previous paper, such as our MAP task definition language. Additionally, the formalization of the MAP task is revised and updated, and the main algorithms behind MAP-POP are discussed in depth. Finally, this paper contributes with an early formal definition of the agents' private information, an aspect that has been typically ignored in many MAP frameworks.

The article first presents a thorough analysis of the state of the art in MAP in section 3.3.2, as well as an extended formalization of the cooperative MAP task in section 3.3.5, including the formal definition of the components of a single and multi-agent POP.

Section 3.3.6 presents in detail our MAP task definition language, including a complete BNF description of the constructs that model the particular requirements of MAP tasks, such as privacy. Furthermore, we offer several modelling examples. The different stages of the MAP-POP framework, discussed in sections 3.3.7, 3.3.8 and 3.3.9, are thoroughly explained and illustrated through a comprehensive example of application.

The experimental results, presented in section 3.3.10, assess the performance of MAP-POP when solving single-agent and multi-agent tasks. Additionally, we analyze the scalability of MAP-POP when tackling tasks with an increasing number of agents. For these tests, we used two customized MAP domains: the first domain features complex tightly-coupled tasks, while the second one gives rise to simpler loosely-coupled MAP tasks.

[1] `http://link.springer.com/journal/10115`

FMAP: distributed cooperative multi-agent planning. The article in section 3.4, originally published in the Applied Intelligence[1] journal in 2014, introduces FMAP, a completely renewed MAP framework that inherits the architecture, input language and refinement planning model of the agents in MAP-POP but implements a forward multi-agent search. This allows us to use state-based heuristic functions instead of the traditional POP-based estimators of MAP-POP. The introduction of this new search scheme implies re-modelling some aspects of the former MAP-POP framework so as to integrate the state-based heuristics.

After analyzing the related work and formalizing the MAP task notion in sections 3.4.2 and 3.4.3, respectively, section 3.4.4 discusses the algorithms of the FMAP framework. First, the new search procedure, based on forward chaining POP, is introduced in section 3.4.4.1. Then, section 3.4.4.2 analyzes completeness and soundness: unlike MAP-POP, FMAP is proven to be a complete method. Next, h_{DTG}, the new global heuristic function that governs FMAP's search, is thoroughly analyzed in section 3.4.4.3. Finally, the limitations of the FMAP framework are critically discussed in section 3.4.4.4.

This paper offers a more comprehensive experimentation than the previous two papers, using a much larger benchmark: 10 different domains from the IPC were adapted to a MAP context and used to assess the performance of FMAP. Section 3.4.5 classifies the domains in the benchmark according to their features and the type of tasks they give rise to. Then, two different comparisons are carried out: FMAP is compared against MAP-POP and MAPR, a powerful state-of-the-art approach to MAP. Both tests prove that FMAP is a reliable and general approach to MAP that not only outperforms MAPR in terms of coverage, but also proves to be superior to MAP-POP in all the measured magnitudes.

[1] `http://link.springer.com/journal/10489`

Global heuristics for distributed cooperative multi-agent planning. The article presented in section 3.5, published in the proceedings of the 2015 25th International Conference on Automated Planning and Scheduling[1] (ICAPS), introduces MH-FMAP, our final approach to MAP, which extends the FMAP algorithm by incorporating a multi-heuristic search. This article discusses the design of two different global heuristic functions for MAP, as well as their integration in the MH-FMAP framework.

After summarizing the related work in section 3.5.2, we formalize the definition of a MAP task in section 3.5.3. The formalization of a MAP task is revised and updated in this article, providng a more compact definition than the previous papers. Section 3.5.4 is a remainder of the original FMAP search algorithm, which is revised in this paper to accommodate the multi-heuristic search.

Section 3.5.5 is devoted to discuss the global heuristic functions that constitute the core of this paper: h_{DTG}, which was firstly introduced in section 3.4.4.3, and h_{Land}, a landmark-based global MAP heuristic. After presenting the two heuristic estimators, we introduce MH-FMAP, a search algorithm that extends FMAP by combining h_{DTG} and h_{Land}.

For the experimentation, we reused the benchmark in section 3.4.5 and focused on two different tests: first, we tested MH-FMAP against FMAP, guiding the search through either h_{DTG} or h_{Land}. Then, we compared the performance of MH-FMAP and GPPP, the only other MAP system that guides search through a global landmark-based heuristic function. Both tests prove the efficiency of our multi-heuristic approach, which not only outperforms its single-heuristic counterparts, but also the GPPP framework.

[1] `http://www.icaps-conference.org/`

3.2 An approach to multi-agent planning with incomplete information

Paper data

Authors:	A. Torreño, E. Onaindia, Ó. Sapena
Conference:	20th European Conference on Artificial Intelligence
Pages:	762-767
Ranking:	CORE A
Year:	2012

Abstract *Multi-agent planning (MAP) approaches have been typically conceived for independent or loosely-coupled problems to enhance the benefits of distributed planning between autonomous agents as solving this type of problems require less coordination between the agents' sub-plans. However, when it comes to tightly-coupled agents' tasks, MAP has been relegated in favour of centralized approaches and little work has been done in this direction. In this paper, we present a general-purpose MAP capable to efficiently handle planning problems with any level of coupling between agents. We propose a cooperative refinement planning approach, built upon the partial-order planning paradigm, that allows agents to work with incomplete information and to have incomplete views of the world, i.e. being ignorant of other agents' information, as well as maintaining their own private information. We show various experiments to compare the performance of our system with a distributed CSP-based MAP approach over a suite of problems.*

3.2.1 Introduction

Multi-agent planning (MAP) refers to any planning or plan execution activity that involves several agents. In general terms, MAP is about the collective effort of

multiple planning agents to combine their knowledge, information, and capabilities so as to develop solutions to problems that each could not have solved as well (if at all) alone (31). There exists a great variety of tools and techniques for MAP. Agent-oriented MAP approaches put the emphasis on distributed execution, plan synchronization and collaborative activity at run-time planning to ensure that the agent's local objectives will be met (27, 111). Another research line in MAP focuses on coordination of already completed plans that agents have constructed to achieve their individual goals, as for example plan merging (19, 21, 113). In contrast, the cooperative distributed planning (CDP) approach puts the emphasis on planning and how it can be extended into a distributed environment, on building a competent plan carried out by multiple agents (27). In CDP, agents typically exchange information about their plans, which they iteratively refine and revise until they fit together well.

Following the cooperative approach, differences among MAP models lie in the integration of the planning and coordination stages (24, 31). Some recent works on fully cooperative MAP have emerged lately. The work in (67) considers agents as having sequential threads of execution and interaction only occurs when distributing sub-plans to individual agents for plan execution. This approach follows a single-agent planning and distributed coordination. A centralized algorithm for MAP can be found in (14), where multiple agents do planning over a centralized plan interleaving planning and coordination. In a distributed version of this latter work, authors use a distributed CSP solver to handle coordination (80).

The aforementioned approaches are conceived for loosely-coupled problems (LCP), where agents have little interaction between each other, as these processes are likely to be inefficient in tightly-coupled problems (TCP) (80). This way, the coupling level of a cooperative multi-agent system is formally defined as a set of parameters to limit the combinatorial blow-up of planning complexity (14). On

the other hand, these MAP models do not consider systems composed of multiple entities distributed functionally or spatially but rather agents endowed with the same capabilities and acting under complete information. When capabilities are distributed across the agents' domains, agents have *necessarily* to interact to solve the MAP problem while being unaware of the other agents' abilities or information about the world, i.e. working under incomplete information.

In this paper, we present a general-purpose MAP model able to work with inherently distributed entities and suitable for both LCP and TCP domains. Similarly to (58), we use an iterative planning refinement procedure that uses single-agent planning technology. Particularly, our model builds upon a partial-order planning (POP) paradigm, which also allow us to represent a collection of acting entities as a single agent. POP is a very suitable approach for centralized MAP with a small number of coordination points between agents (67), and the application of a multi-agent POP refinement framework also reveals as a very appropriate mechanism to address tightly-coupled problems.

This paper is organized as follows: The next section presents the specification of a MAP task. Section 3.2.3 describes the refinement planning algorithm carried out by the agents, respectively. Following, we show the results of the tests we have performed, and finally, we conclude and outline the future lines of research.

3.2.2 Multi-agent planning task

In our approach, the planning formalism of an agent is based on a STRIPS-like model of classical planning under partial observability. The model allows agents to represent their partial view of the world through the adoption of the open world assumption. States are represented in terms of state variables. $\mathcal{O}$ is a finite set of objects that model the elements of the planning domain; $\mathcal{V}$ is a finite set of *state variables* each with an associated finite domain, $\mathcal{D}_v$, of mutually exclusive

values. Values in $\mathcal{D}_v$ denote objects of the planning domain, i.e., $\forall v \in \mathcal{V}$, $\mathcal{D}_v \subseteq \mathcal{O}$. A state is a set of *positive fluents* of the form $\langle v, d\rangle$, and *negative fluents* of the form $\langle v, \neg d\rangle$, meaning that the variable takes on the value d or $\neg d$, respectively. A *formula* (v, d) evaluates to true if the fluent $\langle v, d\rangle$ is present in the state and it evaluates to false otherwise. More specifically, (v, d) evaluates to false if the fluent $\langle v, \neg d\rangle$ is in the state, or if no fluent relating the variable, v, and the value, d, is present in the state, in which case we say the current value of v is unknown. We will generally refer to as *fluents* both positive and negative fluents.

Actions are given as tuples $a = \langle \mathsf{pre}(a), \mathsf{eff}(a)\rangle$, where $\mathsf{pre}(a)$ denotes the formulas that must hold in a state S for a to be applicable, and $\mathsf{eff}(a)$ represents the new fluents in the resulting state S'. Effects of the form $(v = d)$ add a fluent $\langle v, d\rangle$ in the resulting state as well as a set of fluents $\{\langle v, \neg d_j\rangle\}, \forall d_j \neq d, d_j \in \mathcal{D}_v$, reflecting that (v, d_j) evaluates to false in the resulting state. Effects of the form $(v \neq d)$ add a fluent $\langle v, \neg d\rangle$ to the resulting state, which implies the current value of v is unknown unless there is a fluent $\langle v, d'\rangle$ in S', $d \neq d'$.

We define a MAP task as a tuple $\mathcal{T} = \langle \mathcal{AG}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}\rangle$ where:

- $\mathcal{AG} = \{1, \ldots, n\}$ is a finite non-empty set of planning agents.
- $\mathcal{V} = \{\mathcal{V}_i\}_{i=1}^{n}$, where $\mathcal{V}_i$ is the set of state variables managed by agent i. Variables can be shared by two or more different agents.
- $\mathcal{A} = \{\mathcal{A}_i\}_{i=1}^{n}$, where $\mathcal{A}_i$ is the set of actions that agent i can perform. Given two different agents i, j, $\mathcal{A}_i$ and A_j can share some common actions or be two disjoint sets.
- $\mathcal{I} = \{\mathcal{I}_i\}_{i=1}^{n}$, where $\mathcal{I}_i$ is the set of fluents known by agent i that represents the *initial state* of the agent. If two agents share a variable v then they also share all of the fluents regarding v.

- $\mathcal{G} = \{\mathcal{G}_i\}_{i=1}^{n}$, where $\mathcal{G}_i$ is a set of formulas known to agent i that must hold in the final state and denote the top-level goals of $\mathcal{T}$.

As defined above, state variables may not be known to all agents. Given a state variable $v \in \mathcal{V}_i$ and $v \notin \mathcal{V}_j$, $\forall j \neq i$, v is said to be *private* to agent i. Additionally, agents can have different visions of the domain of a state variable; that is, not every value in a variable domain has to be visible to all agents. Given an agent i, we denote its view of the domain of a variable v as $\mathcal{D}_{v_i} \subseteq \mathcal{D}_v$. Thus, the domain of a state variable v can be defined as $\mathcal{D}_v = \{\mathcal{D}_{v_i}\}_{i=1}^{n}$. Agents' incomplete views on the state variables and their domains directly affect the visibility of the fluents.

- An agent i has *full visibility* of a fluent $\langle v, d\rangle$ or $\langle v, \neg d\rangle$ if $v \in \mathcal{V}_i$ and $d \in \mathcal{D}_{v_i}$.
- An agent i has *partial visibility* of a fluent $\langle v, d\rangle$ or $\langle v, \neg d\rangle$ if $v \in \mathcal{V}_i$ but $d \notin \mathcal{D}_{v_i}$. Given a state S, where $\langle v, d\rangle \in S$, agent i will see instead a fluent $\langle v, \bot\rangle$, where $\bot$ is the undefined value.
- An agent i has *no visibility* of a fluent $\langle v, d\rangle$ or $\langle v, \neg d\rangle$ if $v \notin \mathcal{V}_i$.

Our MAP model can be viewed as a POP-based, multi-agent refinement planning framework, a general method based on the refinement of the set of all possible partial-order plans (59). An agent proposes a plan Π that typically enforces some top-level goals of the planning task; then, the rest of agents collaborate on the refinement of this base plan Π by proposing refinement steps that solve some *open goals* in $\mathsf{openGoals}(\Pi)$. This way, agents cooperatively solve the MAP task by consecutively refining an initially empty plan Π.

A *refinement step* Π_i devised by an agent i over a base plan Π^g, where $g \in \mathsf{openGoals}(\Pi^g)$, is a triple $\Pi_i = \langle \Delta, OR, CL\rangle$, where $\Delta \in \mathcal{A}_i$ is a set of actions and OR and CL are sets of *orderings* and *causal links* over Δ, respectively. Π_i is a plan free of *threats* (126) that solves g as well as all the new open goals that arise

from this resolution and can only be achieved by agent i, $\langle v, d\rangle$ or $\langle v, \neg d\rangle$, where $(v \in \mathcal{V}_i) \wedge (v \notin \mathcal{V}_j, \forall j \neq i)$. That is, when solving an open goal of a base plan, an agent i will also achieve the new arising open goals concerning fluents that are only visible to i, so are not visible to the rest of agents, leaving the rest of goals unsolved. Let $g \in \mathsf{openGoals}(\Pi^g)$ be a formula of the form (v, d) or $(v, \neg d)$; an agent i computes a refinement step over Π^g iff $v \in \mathcal{V}_i$.

Plans that agents build are concurrent multi-agent (MA) plans as two different actions in Π can now be executed concurrently by two different agents. Some MAP models adopt a simple form of concurrency: two actions can happen simultaneously if none of them changes the value of a state variable that the other relies on or affects, too (15). We impose the additional concurrency constraint that the preconditions of two actions have to be mutually consistent (**?**). This definition of concurrency is straightforwardly extended to a joint action $a = \langle a_1, \ldots, a_n\rangle$. Agents address concurrency inconsistencies through the detection of threats over the causal links of their actions. This way, concurrency issues between two different actions may not arise until their preconditions are supported through causal links.

A *refinement plan* Π devised by an agent i over a base plan Π^g is a concurrent MA plan that results from the composition of Π^g and a refinement step Π_i proposed by agent i. This refinement plan, which could eventually become a base plan, is defined as $\Pi = \Pi^g \circ \Pi_i$, where $\circ$ represents the composition operation. A composite plan Π is a concurrent MA plan if for every pair of unequal actions a_i and a_j, $i \neq j$, $\forall p_i \in \mathsf{pre}(a_i), p_i \notin \mathsf{openGoals}(\Pi)$, $\forall p_j \in \mathsf{pre}(a_j), p_j \notin \mathsf{openGoals}(\Pi)$, a_i and a_j are concurrently consistent.

In our model, each agent implements a POP planner to compute refinement plans over a base plan Π. If an agent is not capable to come up with a concurrent MA plan, then the agent refrains from suggesting such a refinement. If no agent elicits a consistent refinement plan for a base plan, the plan node is pruned.

3.2.3 Refinement planning

The cooperative refinement planning algorithm starts with a preliminary information exchange by which agents communicate shareable information. After this initial stage, agents execute the multi-agent refinement planning algorithm, which comprises two interleaved stages. First, agents individually elicit refinement plans over a centralized base plan through their embedded POP. Later, agents jointly select the most promising refinement as the next base plan.

Algorithm 1: Dis-RPG construction for an agent i

Build initial RPG_i
repeat
  $\forall j \neq i$, i sends j shareable fluents $SF_{i \to j} \in RPG_i$ of the form $\langle v, d \rangle$ or $\langle v, \neg d \rangle$, where $v \in \mathcal{V}_i \cap \mathcal{V}_j$ and $d \in \mathcal{D}_{v_i} \cap \mathcal{D}_{v_j}$
  $\forall j \neq i$, i receives from j shareable fluents $SF_{j \to i} \in RPG_j$ of the form $\langle v, d \rangle$ or $\langle v, \neg d \rangle$, where $v \in \mathcal{V}_i \cap \mathcal{V}_j$ and $d \in \mathcal{D}_{v_i} \cap \mathcal{D}_{v_j}$
  $RF \leftarrow \emptyset$
  $\forall j \neq i, RF_i \leftarrow RF_i \cup SF_{j \to i}$
  for all *received fluents* $f \in RF_i$ **do**
    if $f \notin RPG_i$ **then**
      Insert f in RPG_i
      $cost_{RPG_i}(f) \leftarrow cost(f)$
    if $(f \in RPG_i) \wedge (cost_{RPG_i}(f) > cost(f))$ **then**
      $cost_{RPG_i}(f) \leftarrow cost(f)$
  Expand RPG_i
until $RF_i = \emptyset$

3.2.3.1 Information exchange

Agents receive the information on the MAP task through a set of definition files. These files are encoded in a MAP language that extends *PDDL3.1* (64), including

a `:shared-data` section to configure the agent's vision of the planning task and which fluents it shares and with whom.

Prior to executing the refinement procedure, agents share information by building a distributed Relaxed Planning Graph (dis-RPG), based on the approach of (127). Agents exchange the fluents defined as shareable in the `:shared-data` section of the MAP definition files. Fluents are labeled with the list of agents that can achieve them, giving each agent a view of the possible interactions that can arise at planning time with other agents. Additionally, the dis-RPG provides an estimate of the best cost to achieve each fluent, a helpful information to design heuristics to guide the problem-solving process.

Algorithm 1 summarizes the construction of the dis-RPG. Agents compute an initial RPG and expand it by following the procedure in (54). The RPG contains a set of fluent and action levels that are interleaved. The first fluent level contains the fluents that are part of the initial state, and the first action level includes all the actions whose preconditions appear in the first fluent level. The effects of these actions are placed in the second fluent level, and this way the graph is expanded until no new fluents are found.

Once all the agents have computed their initial RPGs, the iterative dis-RPG composition begins. As depicted in Algorithm 1, agents start each iteration by exchanging the the fluents shareable with other agents. An agent i will send agent j the set of fluents $SF_{i\to j}$ that are visible to agent j, i.e., the new fluents of the form $\langle v, d\rangle$ or $\langle v, \neg d\rangle$, where $v \in \mathcal{V}_i \cap \mathcal{V}_j$ and $d \in \mathcal{D}_{v_i} \cap \mathcal{D}_{v_j}$. Likewise, agent i will receive from all agents $j \neq i$ the shareable fluents they have generated.

Agent i updates then its RPG_i with the set of new fluents it has received, RF_i. If a fluent f is not yet in RPG_i, it is stored according to $cost(f)$. If f is already in RPG_i, its cost is updated if $cost_{RPG_i}(f) > cost(f)$. Hence, agents only store the best estimated cost to reach each fluent. After updating RPG_i, agent i expands

it by checking whether the new inserted fluents trigger new actions in RPG_i or not. The fluents produced as effects of these new actions will be shared in the next iteration.

The process finishes when there are no new fluents in the system. Following, agents start the refinement planning process to build a solution plan jointly.

3.2.3.2 Multi-agent refinement planning

The refinement planning process is based on a democratic leadership by which a baton is scheduled among the agents following a round-robin strategy. Agents carry out two interleaved stages: the individual construction of refinement plans through a POP, and a coordination process by which agents jointly search the refinement space.

Algorithm 2 describes the refinement planning process. Each agent i computes a finite set of refinement plans for Π^g, $Refinements_i(\Pi^g)$, through its embedded POP planner. The internal POP system follows an A* search algorithm guided by a state-of-the-art POP heuristic function (126). The resulting refinement plans are exchanged by the agents in the system for their evaluation (send and receive operations in Algorithm 2).

Agent i has a local, partial vision of each refinement plan, $view_i(\Pi)$, according to its visibility over the planning task $\mathcal{T}$. Thus, when receiving a refinement plan Π, agent i will only view the open goals $(v, d) \in \mathsf{openGoals}(\Pi) \mid v \in \mathcal{V}_i$. With respect to the fluents, agent i will only view those fluents for which it has *full visibility*. If i has *partial visibility* of a fluent $\langle v, d\rangle$ or $\langle v, \neg d\rangle$, it will see instead a fluent $\langle v, \perp\rangle$, where $\perp$ stands for the undefined value. This notion of partial view directly affects the evaluation of the refinements.

The evaluation of refinement plans is carried out through a utility function $\mathcal{F}$ (currently, we use the same heuristic function that guides the agents' internal POP

for this purpose) that allows agents to estimate the quality of the plans. Since agents do not have complete information on the MAP task or the refinement plans, they evaluate plans according to its own view of each refinement plan Π, i.e., agent i evaluates a refinement plan Π according to $\mathcal{F}(view_i(\Pi))$ (see Algorithm 2).

Algorithm 2: Refinement planning process for an agent i

$\Pi \leftarrow \Pi_0$
$R = \emptyset$
repeat
  Select open goal $g \in \mathsf{openGoals}(\Pi)$
  Refine base plan Π^g individually
  $\forall j \neq i$, send $Refinements_i(\Pi^g)$ to agent j
  $\forall j \neq i$, receive $Refinements_j(\Pi^g)$
  $Refinements(\Pi^g) \leftarrow Refinements_i(\Pi^g)$
  $\forall j \neq i$, $Refinements(\Pi^g) \leftarrow Refinements(\Pi^g) \cup$
  $Refinements_j(\Pi^g)$
  for all *plans* $\Pi \in Refinements(\Pi^g)$ **do**
    Evaluate Π according to $\mathcal{F}(view_i(\Pi))$
  $R \leftarrow R \cup Refinements(\Pi^g)$ Select best-valued plan $\Pi_{best} \in R$
  $\Pi \leftarrow \Pi_{best}$ **if** $\mathsf{openGoals}(\Pi) = \emptyset$ **then**
    return Π
until $R = \emptyset$

Once evaluated, the new refinement plans are stored in the set of refinements R. Next, each agent votes for the best-valued candidate $\Pi_{best} \in R$. In case of a draw, the baton agent will choose the next base plan among the most voted alternatives.

Once a refinement plan is selected, agents adopt it as the new base plan Π. If $\mathsf{openGoals}(\Pi) = \emptyset$, a solution plan is returned. As some open goals might not be visible to some agents, every agent i must confirm that Π is a solution plan according to $view_i(\Pi)$, i.e., Π is a solution iff $\forall i \in \mathcal{AG}, \mathsf{openGoals}(view_i(\Pi)) = \emptyset$.

If the plan has still pending goals, the baton agent selects the next open goal $g \in \mathsf{openGoals}(\Pi)$ to be solved, and a new iteration of the refinement planning process starts.

The planning algorithm carried out by the agents can be regarded as a joint exploration of the refinement space. Nodes in the search tree represent refinement plans and each iteration of the algorithm expands a different node.

3.2.3.3 Soundness and completeness

The algorithm we have presented can be regarded as a multi-agent extension of the POP algorithm. A partial-order plan is sound if it is a threat-free plan. In our algorithm, we address inconsistencies among the concurrent MA plans by detecting and solving threats. Thus, in order to prove that our algorithm is sound, we should ensure that all the threats among the causal links of a concurrent MA plan are correctly detected and solved.

Under complete information, threats on a MA concurrent plan will be correctly detected by any agent, as all the fluents in the plan are fully visible. In our incomplete information model, we should study how visibility over fluents affects the detection of threats.

Let Π be a MA concurrent plan and let $\langle v, d_1 \rangle$ be a fluent in a causal link $cl \in CL(\Pi)$. Suppose that an agent i builds a refinement Π' over Π that adds a new action a_t to the plan which is not ordered with respect to cl and has an effect $(v = d_2)$. This effect causes a threat over cl as it conflicts with $\langle v, d_1 \rangle$. For Π' to be sound, agent i should be able to detect such a threat whatever visibility it has over the fluent $\langle v, d_1 \rangle$:

- If i has *full visibility* over $\langle v, d_1 \rangle$, the inconsistency between cl and a_t will be correctly detected.

- If i has *no visibility* over $\langle v, d_1 \rangle$, then $v \notin \mathcal{V}_i$. In this case, agent i does not have an action a_t with an effect involving variable v, i.e., such a threat can never occur.

- If i has *partial visibility* over $\langle v, d_1 \rangle$, agent i will see instead a fluent $\langle v, \perp \rangle$. Since $\perp \neq d_2$, the threat will be detected and solved.

Therefore, all the threats over MA concurrent plans are always detected and resolved, which proves that our MAP algorithm is sound.

As for completeness, we cannot ensure that our MAP algorithm is complete. According to the notion of refinement plan we have used in this work, the number of refinement plans that an agent can produce over a base plan may not be finite. Hence, we are implicitly pruning the refinement search space. Nevertheless, agents rely on an A* POP search process to build the refinement plans, which in most cases returns good refinement plans that guide the MAP algorithm towards a solution plan. The empirical results shown in the next section confirm our claim.

3.2.4 Experimental results

We designed and executed a set of tests to compare the performance and scalability of our MAP-POP approach with another state-of-the-art MAP system. Comparing the performance of multi-agent planning systems is not an easy task due to two main reasons. First, most MAP approaches are not general-purpose but domain-dependent systems specifically designed to address a particular problem, most typically traffic control or real-time planning applications. Second, unlike single-agent planners that have been promoted and populated through the celebration of the International Planning Competitions[1] (IPC) and, therefore, have been made

[1] http://ipc.icaps-conference.org/

publicly available, it is difficult to find a multi-agent planner able to run the benchmark domains and planning problem suites created for the IPCs.

Problem	#Ag	%CL	%DA	MAP-POP				Planning First			
				#A	#TS	#Par	Time	#A	#TS	#Par	Time
IPCSat1	1	1,2	54	9	8	1	0,23	10	9	1	0,14
IPCSat4	2	29,3	2082	21	11	2	18,80				†
IPCSat10	5	23,7	1786	29	20	3	90,30				†
IPCSat16	10	18,3	7196	51	24	5	73,70				†
IPCSat17	12	14,3	8324	46	16	4	53,90				†
IndSat1	2	5,2	40	9	4	2	0,83	9	4	2	0,16
IndSat2	4	1,4	274	14	3	4	2,20	14	4	4	0,31
IndSat3	7	0,3	1820	32	4	7	6,50	32	4	7	4,10
IndSat4	8	0,3	2082	28	3	8	8,70	28	4	8	11,10
IndSat5	14	0,1	11020	63	4	14	32,50				†
IPCRov1	1	1,2	81	10	7	1	0,34	11	7	1	0,36
IPCRov2	1	2,3	45	8	4	1	0,39	9	5	1	0,32
IPCRov7	3	77,4	157	18	6	3	8,58				†
IPCRov14	4	58,7	797	35	21	2	81,87				†
IPCRov15	4	85	536	42	16	4	42,01				†
IndRov1	2	45,5	160	24	11	2	3,61	22	7	2	2,75
IndRov2	3	45,5	239	36	11	3	5,50	33	7	3	12,14
IndRov3	4	45,5	318	48	11	4	9,19	44	7	4	120,72
IndRov4	5	45,5	397	70	11	5	14,14	55	7	5	674,00
IndRov5	6	45,5	476	72	11	6	20,69	66	7	6	2594,52
IPCLog2	3	20	52	27	9	3	18,19				†
IPCLog4	4	12,3	116	37	13	4	33,77				†
IPCLog5	4	14	116	31	11	4	40,19				†
IPCLog7	5	9,8	206	46	15	5	96,48				†
IPCLog9	5	11,7	206	45	17	5	239,58				†
IndLog1	3	44,4	20	6	6	2	1,58	9	8	3	0,58
IndLog2	3	55,5	20	10	9	3	2,25	10	9	3	0,61
IndLog3	4	65	42	13	10	4	3,23	9	8	4	66,19
IndLog4	4	70	42	14	6	4	3,77	14	6	4	284,09
IndLog5	6	54,1	98	21	6	6	13,58				†

Table 3.1: Performance comparison between MAP-POP and Planning First

Despite these drawbacks, we could assess the performance of MAP-POP and compare the results with those obtained in the Planning First approach presented in

(80)[1]. Planning First is a MAP system that also makes use of single-agent planning technology. More precisely, it builds upon a single-agent state-based planner (17), and handles agent coordination by solving a distributed CSP.

Planning First defines public actions as the actions of an agent whose descriptions contain atoms affected by and/or affecting the actions of another agent. Based on this concept, it defines the notion of coupling level as the average rate of public actions of an agent. A high value of coupling level results in many agent coordination points, thus giving rise to tightly-coupled problems. The approach followed by Planning First is especially effective when dealing with loosely-coupled problems (LCP) (80), but its performance decreases when tackling tightly-coupled problems (TCP).

The tests presented here involve three of the benchmark domains used on the IPCs: *satellite*, *rovers* and *logistics*, which are the domains used in the results presented in (80) as well. These domains give rise to problems of different coupling levels. The *satellite* problems are LCP as the different agents (the satellites) are not likely to interact with each other; they move, calibrate their instruments and take images by themselves. *Rovers* problems tend to present a medium coupling level: rover agents are independent but they have access to certain shared resources in their environment, namely the rock and soil samples they collect and analyze. The *logistics* problems fall into the TCP category since agents (trucks and planes) have to cooperate to transport the different packages to the target locations and problems present several coordination points (locations) at which agents can interact.

We adapted the STRIPS problem files used in the IPCs to both our MAP language and the *MA-STRIPS* language used by Planning First (14). Problems

[1] We want to especially thank Raz Nissim for providing us with the source code of his Planning First system for testing and comparison purposes.

from the IPCs turned out to be complex instances for Planning First because agents have necessarily to interact to each other and cooperate to find a solution plan for these problems and Planning First works better when plans for each agent can be computed (mostly) independently. For this reason, we encoded an additional set of problems limiting cooperation and interactions among agents as much as possible. Particularly, in these additional problems, agents can solve goals independently, i.e., an agent is able to solve a goal or set of goals by itself without need of interacting with the rest of agents (we will refer to these problems as independent problems in the remainder).

Table 3.1 shows the results when comparing the quality of the solution plans obtained with MAP-POP and Planning First and the execution times[1]. The quality of the solution plans is assessed through three parameters: a) the number of actions of the plan; b) the duration of the plan, i.e. the number of time units or *time steps* required to execute the plan; and c) the number of agents that take part in the solution plan). This latter parameter gives an idea of how the effort on solving the problem has been distributed among the agents.

Problems labeled with *IPC* are directly taken from the IPC benchmarks, while problems labeled with *Ind* are the extra set of *independent* problems we created to assess Planning First performance (for each domain, we show the results of 5 out of the 20 IPC problems we tested as well as 5 independent problems). The next three columns in the table show the difficulty of the planning problems: $\#Ag$ indicates the number of agents involved in the problem; $\%CL$ estimates the coupling level of the problem as the average rate of instantiated public actions of agents (taking into consideration the notion of public and private action defined in (80)), and $\#DA$ refers to the total number of instanced actions. The results

[1] All the tests were performed on a single machine with a 2.83 GHz Intel Core 2 Quad CPU and 8 GB RAM.

for each planner include the number of actions ($\#A$) and time steps ($\#TS$) of the solution plan, respectively. $\#Par$ indicates the number of agents that take part in the solution plan, and *Time* shows the total execution time. A dagger (†) indicates that Planning First was not able to solve the problem.

For the most loosely-coupled problems, the *satellite* domain, MAP-POP exhibited an excellent performance as our results confirmed that it was able to solve 18 out of 20 IPC problems. For the five IPC problems for the *satellite* domain shown in Table 3.1, we can see that our approach deals very efficiently with complex problems up to 12 agents. It is also noticeable that at least one third of the participating agents take part in the solution plans, which has a positive impact on the plan duration, as actions are carried out in parallel by different agents. Although the IPC *satellite* problems do not present a high coupling level (less than 30% of public actions in the worst case), Planning First only solves the first IPC problem, as these problems require cooperation among agents and it is more necessary for larger instances. As for the additional problems we encoded (IndSat1, ..., IndSat5), we can see that Planning First is not able to solve the largest one, IndSat5. Planning First is faster than MAP-POP when dealing with small problems, but its performance decreases when the size of the problem increases. For instance, while the first three problems are solved faster by Planning First, it is slower than MAP-POP when solving IndSat4, and it does not find a solution to the most complex instance, IndSat5. MAP-POP proves also to be more effective at parallelizing actions in this domain as it obtains plans of equal or shorter duration than Planning First.

With respect to the *rovers* domain, our results confirmed that MAP-POP solves 15 out of the 20 IPC problems for this domain. For the five IPC *rovers* problems shown in Table 3.1, we can see the workload in this domain is better distributed than in the *satellite* domain as most of the agents participate in the solution plan,

which considerably reduces the duration of the plan. For instance, the solution plan for problem IPCRov7 contains 18 actions and is solved in just 6 time steps. Planning First solves only the two smallest IPC problems. For the *independent* problems we modeled, Planning First obtains better-quality but more costly solutions than MAP-POP. The differences in execution time are far more noticeable than in the *satellite* domain. This is due to to the more tightly-coupled nature of the problems of this domain (45.5% coupling level for the *independent* problems), which affects negatively the performance of Planning First.

Finally, the *logistics* domain has proven to be the most complex one for both multi-agent approaches. Agents in this domain are trucks and airplanes that must cooperate in most of the cases to transport packages. Hence, solutions for these problems are more costly than in the *rovers* and *satellite* domains, as they require agent coordination, an important feature to determine the efficiency of a MAP approach. Our results confirmed that MAP-POP loses performance in this domain, being able to solve only 9 out of 20 IPC problems. However, it distributes the workload effectively since all of the agents participate in all the solution plans obtained. Planning First shows also a poorer performance in this domain as it is not able to solve any of the IPC problems. These results are in line with the conclusions exposed in (80), which reveals the difficulty of a CSP-based approach to deal efficiently with problems that exhibit a high level of inter-agent interaction. As for the *independent* problems, some of the solutions obtained by MAP-POP have better quality in terms of actions and duration than the solutions of Planning First. In addition, Planning First is still remarkably slower than MAP-POP, being unable to solve the IndLog5 problem, even though we defined rather small instances (notice the differences in execution time for the instance IndLog4). Again, Planning First only performs better than MAP-POP in the smaller problems.

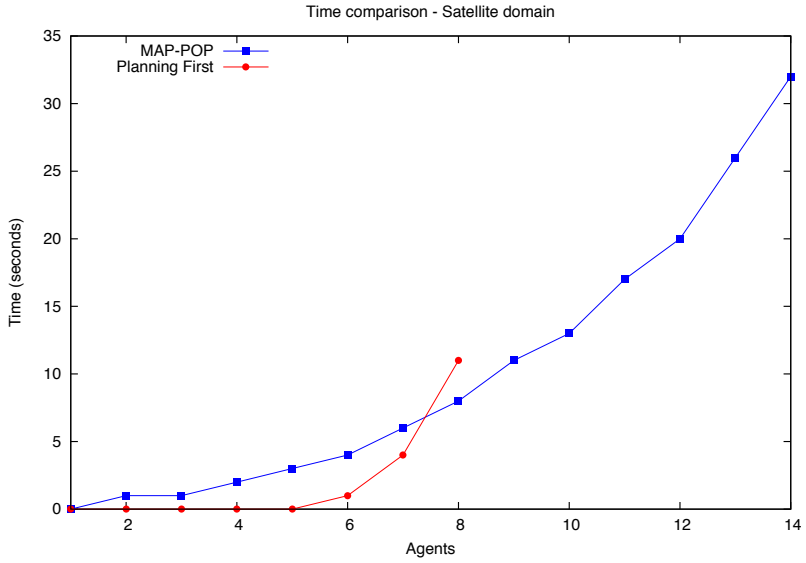

Figure 3.1: Scalability results for the *satellite* domain

The second test compares the scalability of both MAP frameworks, i.e. to which extent their efficiency is affected by the number of agents. In order to do so, we have run fourteen different tests for both the *satellite* and the *rovers* domains. Each test increases the number of agents in the task by one, from one agent to fourteen. The problems are modeled so that each of the participant agents has to achieve one of the problem's goals by itself.

Figure 3.1 shows the scalability results for the *satellite* domain. As it can be observed, Planning First show a better performance when solving small problems (up to seven agents). However, its performance decreases quickly as we execute larger problems. MAP-POP is faster at solving the 8-agent *satellite* problem, and Planning First is unable to find a solution for the 9-agent problem upwards. MAP-POP, however, finds a solution for the 14 problem instances, and execution times

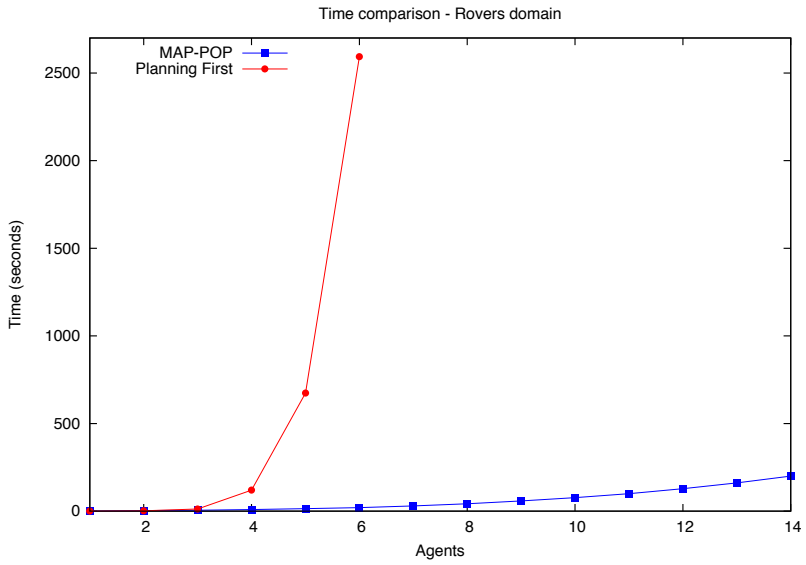

Figure 3.2: Scalability results for the *rovers* domain

suffer only a slight increase between problems.

The differences in performance of both systems are more noticeable in the more tightly-coupled *rovers* domain. The results of this test are depicted in Figure 3.2. In this case, Planning First requires more than 40 minutes to solve the 6-agent *rovers* problem, while MAP-POP takes only 20 seconds. Again, MAP-POP solves all the problems without losing performance in the larger instances.

In conclusion, MAP-POP proves to be a more robust approach than Planning First as it can tackle larger and more complex planning problems. Moreover, while Planning First is designed for solving LCP, MAP-POP is a general-purpose method that tackles problems of different coupling levels. Although MAP-POP behaves better in LCP problems, it can also solve complex TCP problems. Scalability results show that Planning First performs better when dealing with simple problems

that involve few agents. However, MAP-POP scales up far better, being able to solve much larger planning problems.

3.2.5 Conclusions and future work

This paper presents a general-purpose MAP model suitable to cope with a wide variety of MA planning domains under incomplete information. The ability to define incomplete views of the world for the agents allows us to deal with more real problems, from inherently distributed domains -functionally or spatially- to problems that handle global and centralized sources of information. Currently, we are testing our planner on large-size logistics applications in which agents are geographically distributed and are completely unaware of the other agent's information except for the coordination points within their working areas.

The MAP resolution process is a POP-based refinement planning approach that iteratively combines planning and coordination while maintaining for each agent only the information that is visible to the planning entity. This POP approach centered around the gradual construction of a joint solution plan for the MAP task highly benefits the resolution of cooperative distributed planning problems.

We have compared our MAP approach against Planning First, a system that handles agent coordination through a distributed CSP. Results show that MAP-POP efficiently solves loosely-coupled problems but it also shows competitive when solving problems that have a higher coupling level and when computing plans that require the cooperation among agents. Hence, we can conclude that MAP-POP is an efficient, domain-independent and general-purpose framework to solve MAP problems.

3.3 A flexible coupling approach to multi-agent planning under incomplete information

Paper data

Authors:	A. Torreño, E. Onaindia, Ó. Sapena
Journal:	Knowledge and Information Systems
Volume (number):	38(1)
Pages:	141-178
Impact factor (2014):	1.782 (first third)
Year:	2014

Abstract *Multi-agent planning (MAP) approaches are typically oriented at solving loosely-coupled problems, being ineffective to deal with more complex, strongly-related problems. In most cases, agents work under complete information, building complete knowledge bases. The present article introduces a general-purpose MAP framework designed to tackle problems of any coupling levels under incomplete information. Agents in our MAP model are partially unaware of the information managed by the rest of agents and share only the critical information that affects other agents, thus maintaining a distributed vision of the task.*

Agents solve MAP tasks through the adoption of an iterative refinement planning procedure that uses single-agent planning technology. In particular, agents will devise refinements through the Partial-Order Planning paradigm, a flexible framework to build refinement plans leaving unsolved details that will be gradually completed by means of new refinements. Our proposal is supported with the implementation of a fully-operative MAP system and we show various experiments when running our system over different types of MAP problems, from the most strongly-related to the most loosely-coupled.

3.3.1 Introduction

Planning is the art of building control algorithms that synthesize a course of action to achieve a desired set of goals from an initial situation. Traditionally, planning has been regarded as a centralized process in which a single entity is in charge of devising a plan that satisfies the problem goals.

Multi-Agent Planning (MAP) generalizes the problem of planning in domains where several agents plan and act together. MAP introduces a social approach to planning (76), focusing on the collective effort of multiple planning entities to accomplish tasks by combining their knowledge, information and capabilities. This is required when agents are unable to solve their tasks by themselves, or at least can accomplish them better (more quickly, completely, precisely, or certainly) when working with others (31).

MAP is concerned with planning *by* multiple agents, i.e., distributed planning, and planning *for* multiple agents, i.e., planning for multi-agent execution, thus giving rise to a great variety of tools and techniques. The approach traditionally adopted by the Multi-Agent Systems (MAS) research community assumes that, in general, agents are self-interested and that there is not a common goal to solve, thus focusing on coordinating the activities of multiple agents in a shared environment (27). In agent-oriented approaches, the ultimate objective is to ensure that the agents' local objectives (private goals) will be achieved by their plans and so the emphasis is put on distributed execution, plan synchronization and collaborative activity at run-time planning (32, 60, 111). All in all, these techniques use planning as a means to controlling and coordinating agents rather than building a competent and joint plan, and so they are very appropriate for the design of real-time systems (75).

In planning-oriented approaches dealing with contexts in which agents are assumed to be cooperative, the objective is to study how planning can be extended into a distributed environment or, more particularly, on the construction of a competent plan by several planning entities. There exist different approaches to address this objective, varying according to the typology of the planning problem to solve. In particular, the adoption of one or another strategy depends on the coordination needs of the problem, i.e., to which extent agents are able to make their own plans without affecting what the other agents are planning to do. Thus, when agents are assumed to be relatively independent, they carry out their planning activities individually and exchange information about their local plans, which they iteratively refine and revise until they fit together in order to ensure that the resulting plan will jointly execute in a coherent and efficient manner (27). This has been the predominant approach in cooperative MAP, existing a large body of research on post-planning coordination, i.e., solving inconsistencies among local plans that have been constructed separately. The well-known Partial Global Planning (PGP) framework (32) is one of the first techniques that allows agents to communicate and merge their local plans. Ever since, many works on plan merging methods for building a joint plan given the local plans of each participating agent have arisen (see section 3.3.2 for a detailed description).

The application of MAP to loosely-coupled multi-agent tasks, in which agents have little interaction to each other, is still an active area of research. Some recent works in this line, where agents are engaged in some cooperative behaviour, have emerged lately. These works follow a common approach that consists of coordinating the local solutions developed by the agents. For instance, the work in (67) considers that agents have sequential threads of execution and interactions only occur when distributing sub-plans to individual agents for plan execution. This approximation follows a single-agent planning and distributed coordination.

The work in (14) applies individual planning and coordinates the local solutions through the resolution of a Constraint Satisfaction Problem (CSP). In an extension of this latter work, authors use a distributed CSP to solve inconsistencies among agents' plans (80).

Most of the aforementioned approaches turn out to be inefficient at the time of solving strongly-related problems in which the number of coordination points among agents is large (80). To deal with these problems, other MAP models use a unified approach in which planning and coordination of activities are integrated rather than being treated as independent processes (4, 58). However, these approaches do not achieve high performance in loosely-coupled problems because the reasoning procedures rely very strongly on a high degree of interdependency between the agents' actions.

The problem of building a competent joint plan among several planning entities has been generally dismissed by the MAS community, more concerned with the development of coordination mechanisms for agents, and ignored by the planning community, which has traditionally resorted to efficient single-agent algorithms to solve planning problems. MAP is not only about a divide-and-conquer strategy to tackle large planning problems, it is also about the development of techniques for planning entities that are geographically or spatially distributed. While one might expect the number of coordination points in inherently distributed problems not to be very large, another issue that comes up is the distribution of information among agents. In frameworks like those presented in (4, 15) agents communicate all the available information and build complete knowledge bases, i.e., agents have complete information on the MAP task. However, in large-size problems with heterogeneous agents, building complete knowledge bases is not viable. Besides efficiency issues, agents may be unable to manage the information handled by other agents as they may have different knowledge and abilities.

3.3 A flexible coupling approach to multi-agent planning under incomplete information

In this paper, we present a novel approach to cooperative MAP that allows to efficiently solve problems with any level of interaction among agents. Unlike other techniques, our MAP system is capable of solving from the most loosely-coupled problems to the most strongly-related problems. The key point to address this aspect is to use a refinement planning approach (59) that allows agents to interleave planning and coordination, or more specifically, to coordinate their plans during planning. We also allow heterogeneous agents to work under incomplete information, sharing only the critical information that affects other agents and maintaining a distributed vision of the MAP task. This issue, which has been ignored in almost all of the MAP approaches, is of key importance to efficiently handle inherently distributed problems. Last but not least, our MAP approach is entirely based on the use of single-agent planning technology adapted to a multi-agent context. More precisely, agents follow the Partial-Order Planning paradigm (77, 126).

As well as introducing the MAP architecture and a theoretical model for multi-agent planning, our proposal is supported with the implementation of a fully-operative MAP system. The empirical evaluation of the system demonstrates this novel approach to be effective when dealing with both strongly-related problems and loosely-coupled problems in which agents manage incomplete information.

This paper is organized as follows: section 3.3.2 summarizes some background on the main topics related to this work and reviews the most recent literature on MAP; section 3.3.3 introduces the example MAP scenario we will use to illustrate the different aspects of our framework; section 3.3.4 outlines our MAP architecture; section 3.3.5 presents the theoretical planning model upon which our system is based; section 3.3.6 outlines the planning language used to model MAP tasks; section 3.3.7 provides an overview of the MAP algorithm followed by the agents; section 3.3.8 describes the first stage of our MAP algorithm, the initial

information exchange; section 3.3.9 outlines second stage of the MAP algorithm, the refinement planning and coordination protocol; section 3.3.10 presents the experimental results, and finally, section 3.3.11 concludes and summarizes our future lines of research.

3.3.2 Background

Our MAP model builds upon several single-agent planning techniques. This section provides a review on the principal single-agent planning concepts used in our MAP approach as well as the most relevant and recent approaches to cooperative MAP. We also outline the most relevant works on MAP architectures and frameworks and we conclude by summarizing the main contributions and novelties of our approach.

3.3.2.1 Single-agent planning

Single-agent planning is regarded as a search process by which a single entity synthesizes a set of actions (plan) to reach a set of objectives from an initial situation (122). Over the last years, single-agent planning has experienced great advances, specifically in the construction of domain-independent heuristics. Nowadays, it is possible to find a great variety of planning systems. The most recent planners combine different techniques in order to increase the algorithms efficiency: landmarks (95), domain transition graphs (50), forward-chaining partial-order planning (18), probes (69) or divide-and-conquer strategies (30), among others.

The work in (8) introduced the concept of Relaxed Planning Graph, which has proven to be one of the most effective constructs to devise heuristics in state-space planning (54). This technique has been integrated in many single-agent planning frameworks and has also been extended to a distributed context (127).

While state-space planners such as Fast Forward (54) are still a relevant research topic, plan-space planning has been replaced by other more efficient techniques. However, plan-space planning has recently seen a revival since its flexibility makes it specially suitable for distributed environments.

Among plan-space search algorithms, the Partial-Order Planning (POP) approach (90, 126) is particularly relevant. POP performs a plan-based, backward search process, refining partial plans through the addition of actions, causal links and ordering constraints. POP is based on the *least commitment strategy* (121), which defers planning decisions during the search process and introduces partial-order relations among actions rather than enforcing a concrete order among them. The particular nature of the POP paradigm (absence of states, backward search) makes it difficult to devise competitive heuristics to guide the search process. Although some recent works reformulate the basic algorithm to improve its performance (18), POP has been discontinued by the planning community in favor of other approaches. Nevertheless, it is still used in temporal planning and MAP environments as it is a flexible paradigm to handle concurrency (12).

3.3.2.2 Cooperative Multi-Agent Planning

MAP extends the single-agent planning problem by distributing the planning task among several entities which work together to devise a competent joint plan that meets the problem goals. This generalization entails some differences to the more restrictive single-agent planning approach. MAP can be viewed as the problem of coordinating agents in a shared environment where information is distributed (27). This definition emphasizes two aspects of MAP that are not present in single-agent planning: the coordination of the planning activities and the distribution of the information among agents.

In general, solving a cooperative MAP task involves the following stages (31): 1) global goal refinement, 2) task allocation, 3) coordination before planning, 4) individual planning, 5) coordination after planning, and 6) plan execution. Some of the previous stages can be avoided or combined. For instance, some works do not distribute the goals explicitly (avoiding stage 2) (4, 15), while others apply only coordination after planning (avoiding stage 3) (21, 120).

MAP problems can be classified according to their coupling level, a measure of the number of interactions or coordination points among agents that will arise during the task resolution (14). In loosely-coupled problems, each problem goal problem is likely to be solved by a single agent, while goals in strongly-related problems tend to require the cooperation of several agents. The number of coordination points in a MAP problem determines which approaches are more suitable to solve it efficiently.

A wide range of MAP approaches put the emphasis on coordination after individual planning (coordination is performed at stage 5 of the MAP scheme described above). This way, these frameworks perform the planning and coordination stages independently and separately, combining or merging solutions into a global joint plan (24, 31, 60, 113).

Different coordination techniques have been proposed for merging and gathering several individual plans into a single joint plan. The Partial Global Planning framework (32) and its extension, the Generalized Partial Global Planning approach (26), allow agents to communicate their local plans to the rest of agents and then they merge this information into their own partial global plan in order to improve it. This iterative process goes on until the agents' local plans fit together. The work in (113) proposes a post-planning coordination approach based on the iterative revision of the agents' local plans. Agents in this model cooperate by mutually adapting their individual plans, with a focus on maximizing their common

or individual profit. (80) introduces a cooperative MAP approach for loosely-coupled systems in which agents carry out planning individually through a state-based planner (17, 54). The resulting local plans are then coordinated by solving a distributed Constraint Satisfaction Problem. The approach in (120) solves inconsistencies among the local plans devised by self-interested agents through plan repair. Other proposals deal with insincere agents by combining planning, coordination, and execution (34) or consider the communication needs that arise when plans are being executed (112).

The aforementioned plan merging methods follow a common approach: agents build plans individually while a subsequent independent process is used to coordinate these plans. This approach is suitable for solving loosely-coupled problems efficiently as the agents' local solutions in these problems present few interdependencies with each other. Thus, plan merging through post-planning coordination is an appropriate method to tackle problems in which agents can solve the different problem goals independently and the majority of the environment resources are not shared.

However, plan merging methods present several limitations. On the one hand, goals must be a priori allocated to each agent or at least implicitly distributed among the planning entities, as agents perform their planning activity in an isolated manner. Because of this, methods based on plan merging lose flexibility against other MAP proposals. On the other hand, the previous merging approaches have proven to be inefficient when solving strongly-related problems in which most of the resources are shared and most of the goals require cooperation among agents (80). The individual planning combined with a post-planning coordination strategy is not adequate to solve these strongly-related problems, since merging may introduce exponentially many ordering constraints in problems which require a coordination effort.

Another research trend on cooperative MAP stresses the importance of combining and integrating planning and coordination activities, i.e., apply coordination during planning. Hence, this trend can be seen as an extension of single-agent planning to MAP, providing a unified vision of MAP. Proposals in this line focus on the cooperative incremental construction of a joint plan, allowing agents to perform their planning activity over a centralized plan representation. This is a more suitable approach than the plan merging techniques for tackling strongly-related MAP problems with a large number of coordination points, as agents work over a centralized plan representation and planning and coordination of activities are carried out in an integrated way.

The proposal in (27) applies the continual planning approach, which interleaves planning and execution and coordinates agents by synchronizing them at execution time (15). The approach in (58) introduces the best-response planning algorithm, which iteratively improves the quality of the agents' plans through single-agent planning technology. Finally, the works in (4, 84) solve inconsistencies among agents' plans through a coordination protocol based on iterated dialogues. Agents discuss and argument about the different plan proposals until the agents' viewpoints are aligned and an agreement is reached.

The integrated planning and coordination approach followed by the aforementioned MAP models copes with a wider range of MAP problems than the plan merging method, which can only deal with simpler, loosely-coupled problems. In addition, the continual revision and coordination of the agents' plans provides better results in terms of plan quality. However, integrating planning and coordination entails higher communication costs for loosely-coupled problems than using plan merging, as coordination has to be performed throughout the planning process, thus introducing an overhead. Hence, the simpler plan merging approach is far more effective for small-size and non-complex planning tasks.

Research on cooperative MAP, traditionally carried out by the planning community, has generally overlooked the management of incomplete information, an active research topic, though, within the MAS community. Planning with incomplete information has several different meanings: that certain facts of the initial state are not known, that operators can have random or nondeterministic effects, or that the plans built contain sensing operations and are branching (48). In our case, we interpret incomplete information as agents not knowing the initial state completely and being total or partially unaware of the information managed by other agents.

The issue of incomplete information has been treated from two different perspectives: the probabilistic way, with the development of formal models such as Dec-POMDPs (Decentralized Partial Observable Markov Decision Processes) for coordination among multiple agents in contexts with partial observability (66, 124); and the epistemological way, which assumes that agents have beliefs about the state of the world and beliefs over the other agents' knowledge (29, 65). This latter approach has been widely used in games of incomplete information (45). Both perspectives define agents as having an imprecise or uncertain view of the world and of the other agents' information but, to the best of our knowledge, there are not proposals to deal with ignorance, i.e., local views of agents that reflect agent's unawareness over the information of the rest of agents. This introduces a complexity factor in the planning process as agents can only plan on the basis of their information, being ignorant on the planning decisions of other agents. It is important to note, though, that the information unknown to one agent does not have a direct impact on the agents' choices because its actions are not involved with the unknown piece of information. However, this absence of information may have an indirect impact in the overall planning process and quality of the plan.

3.3.2.3 Architectures and frameworks for MAP

The design of architectures and frameworks constitutes another active research field in MAP. Over the last years, some relevant works in MAP frameworks have been published. The work in (123) presents a complete MAP architecture for large-scale problem solving, which organizes agents into planning cells committed to a particular planning process. The TAEMS domain-independent coordination framework (68) provides agents with planning capabilities, and applies the GPGP approach to coordinate them.

Other MAP architectures are based on general-purpose MAS platforms, rather than being designed from the ground up. MAS platforms, such as Magentix2 (1, 37) or JADE (5), provide the sets of services, conventions and knowledge required by agents to interact with each other. For instance, the domain-independent multi-agent system infrastructure RETSINA (110) introduced a planning component (86). Once integrated into the agents' internal architecture, this component provides them with planning capabilities.

Similarly, our MAP approach builds upon the Magentix2 MAS platform, which provides the communication services required by the agents. From this base, we introduce the additional components to provide the agents with planning capabilities and allow them to tackle MAP tasks.

3.3.2.4 Contributions of our model

Our novel approach to cooperative MAP can be classified into the research trend that integrates planning and coordination. The MAP system achieves two main objectives: 1) it solves complex strongly-related problems as well as loosely-coupled problems without losing generality; and 2) it allows heterogeneous agents to work under incomplete information, sharing only the critical information that affects

other agents and being partially unaware of the other agents' information on the MAP task.

Our MAP approach focuses on a novel method that combines single-agent planning technologies and a refinement-based methodology. More precisely, we combine a distributed refinement planning procedure (59) and an individual Partial-Order Planning (POP) (77, 126). Agents incrementally build local refinements to a certain base plan through their local POPs, and coordinate these partial solutions through the refinement planning process. Empirical evaluation proves this method to perform effectively for both strongly-related and loosely-coupled problems.

Another key feature of our method is the ability to work under incomplete information. Unlike many MAP proposals, agents in our approach do not require to build complete knowledge bases, but they can be partially unaware of the information on the initial state and the knowledge and abilities of the rest of agents. Our *PDDL3.1*-based MAP specification language (64) defines this partial visibility of the agents, allowing to specify which information can be shared with other agents for cooperation purposes. Agents exchange the shareable information with other agents through the construction of a distributed Relaxed Planning Graph (127) and perform planning while being partially unaware of the other agents' knowledge. This way, our proposal stresses the importance of privacy in a MAP context, as agents share only the essential information that affects other agents and are partially unaware of the information held by the rest of planning entities.

3.3.3 Motivating example

This section introduces the example MAP scenario we use in the following to illustrate the concepts presented throughout this paper. The example of application, depicted in Figure 3.3, describes a transportation and storage scenario in which two agents (Ag1 and Ag2) take the role of transport agencies and a third agent

(Ag3) manages a storage facility. Transport agents deliver packages through a network of cities. In turn, the warehouse agent is in charge of storing and delivering packages to the trucks. Packages can be either raw materials or final products. Agents in the MAP task are entrusted with two different goals: deliver the final product `p1` to city `cA` and the raw material `p3` to city `cE`.

This scenario includes bidirectional links among cities that allow transport agents to move trucks from one city to another. Transport agents Ag1 and Ag2 can perform three different actions: they can load and unload packages in the trucks and they can move the trucks between cities in their working areas. Ag1 and Ag2 can only move trucks within the cities included in their working areas, depicted in Figure 3.3 as two different circles. This way, transport agents have to interact and cooperate in order to deliver packages to a different working area.

A possible plan to solve the scenario depicted in Figure 3.3 involves Ag1 loading the raw material `p3` in the truck `t1`. Then Ag1 would handle `t1` to Ag2 in `cB` or `cD`, both included in the working areas of Ag1 and Ag2, and Ag2 would take care of transporting the product to `cE`. This leads to a key aspect of our model: in order to promote cooperation, Ag1 should share with Ag2 the information on the position of `t1` once it reaches `cB` or `cD`. As we will discuss in the following section, agents will share the information that is relevant for other agents in order to successfully cooperate.

The warehouse agent Ag3 is in charge of interacting with the trucks to store raw materials and deliver final products. The warehouse has a table in which packages can be stacked and unstacked. Packages are swapped in the city in which the warehouse is placed, the exchange city. As seen in Figure 3.3, `cF` is the exchange city used by Ag2 and Ag3 to swap packages.

Ag2 and Ag3 will also share information on the packages they leave in the exchange city, which will be necessary for them to interact. For example, to

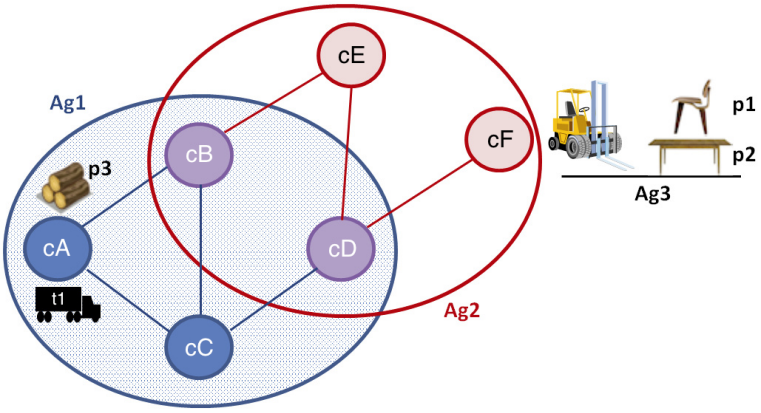

Figure 3.3: Transportation and storage scenario

accomplish the first goal of the task (transporting the final product `p1` to `cA`), Ag3 will deliver `p1` to the exchange city `cF`, informing Ag2 about the position of the package. Then, Ag2 will load `p1` in the truck `t1` and will drive `t1` to `cB` or `cD`. Finally, Ag1 will perform the final transportation, delivering p1 to city `cA`.

3.3.4 Multi-agent planning architecture

The architecture of our MAP system is depicted in Figure 3.4. The MAP architecture basically consists of a set of agents endowed with planning capabilities and an underlying communication infrastructure that allows them to interact with each other.

All the agents share the same internal structure, and the internal planning algorithm followed by each agent is a POP procedure, so they all develop the same rationale. However, since agents handle different information and knowledge, that is, *incomplete information* on the MAP task and different planning abilities, our MAP system features heterogeneous agents. In the example of application presented in section 3.3.3, two agents play the role of transport agencies and a third

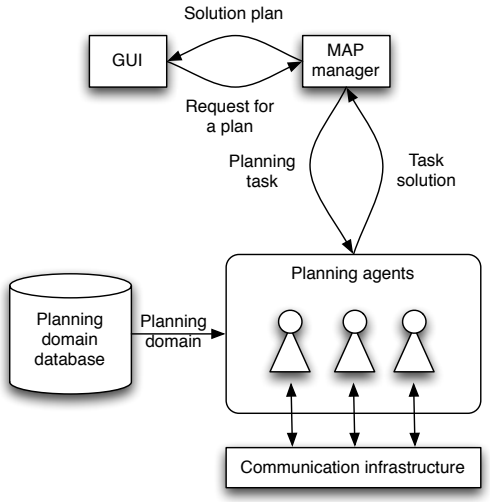

Figure 3.4: MAP system architecture

agent manages a storage facility. The first two agents will likely perform similar actions like driving vehicles from one location to another, which will be different from the planning abilities of the third agent devoted to stack and arrange packages in a warehouse. Additionally, agents will have a different view of the planning task accordingly to their abilities and initial knowledge; thus, the first two agents will have information about the trucks and roads connecting the different locations, and the third agent will manage the information about the packages and the hoists in the warehouse.

Together with the planning agents, the MAP architecture provides a set of components that allow the user to interact with the platform. The main components of the MAP architecture are:

- Graphical User Interface (GUI): This component allows the user to interact with the MAP system. The user requests the resolution of a MAP task

by providing, for each agent involved in the task, two input files encoded through our MAP specification language, the *domain* and *problem* file (see section 3.3.6). The first file defines the typology and the planning capabilities of the agent, while the second file defines the concrete aspects of the task it has to solve. Once a solution is found, it is displayed to the user through the GUI.

- MAP manager: This component interacts with the GUI by collecting the user's request for a plan and assigning the MAP task to a subset of agents that are available, i.e., they are not solving any particular planning task at the moment. Agents are fully reconfigurable and can be reused when they become available again by assigning a new MAP task to them.

- Pool of planning agents: The architecture includes a pool of planning agents which all share the same internal structure shown in Figure 3.5. Agents are configurable through the domain and problem files provided by the user, which define the agents' knowledge and abilities. Once a subset of the agents in the pool receive a planning task, they start working together to find a solution plan.

- Communication infrastructure: Agents interact with each other through a communication infrastructure, which allows them to exchange messages by following the FIPA communication protocols (63). The developed MAP system uses the Magentix2 MAS platform (37) as its communication infrastructure.

The internal structure of the planning agents includes several modules to accomplish the requirements of our refinement planning approach. Through these modules, agents make plan refinements over a base plan, select the best alternative

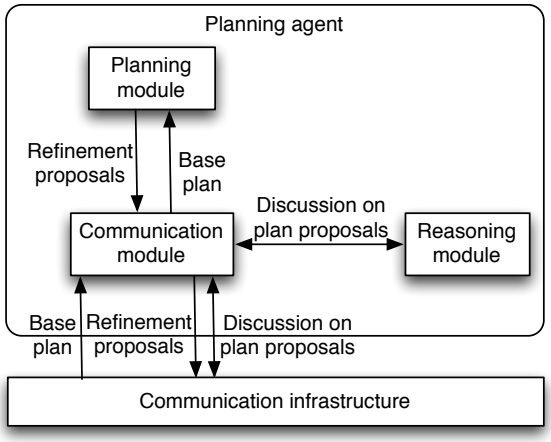

Figure 3.5: Internal structure of a planning agent

from a set of refinement plans and communicate with each other (see Figure 3.5). Although agents have the same internal structure, they have different planning abilities and visibility over the MAP task as defined in the domain and problem file provided by the user. The internal modules of a planning agent are:

- Communication module: Through this module, each planning agent interacts with the rest of agents via the communication infrastructure. The communication module receives messages from the rest of agents and transmits the received information to the rest of internal modules of the planning agent. When the agent wants to communicate with other agents, this module is in charge of sending the messages through the communication infrastructure (the Magentix2 MAS platform). Hence, this module acts as an interface between the planning agent and the rest of agents in the MAP task.

- Planning module: This module is in charge of performing the actual planning search. It includes an embedded Partial-Order Planner which has been

modified to be able to start the planning process from an incomplete plan and return valid refinements instead of complete solution plans. The planning module receives the current base plan from the communication module and returns a set of valid refinements over the base plan.

- Reasoning module: Agents coordination consists in evaluating the refinement plans and choosing the most promising one as the next base plan (see section 3.3.7). The reasoning module of each agent receives the refinement proposals of the agents and evaluates them according to the view of the MAP task of the respective agent. Hence, this module provides agents with facilities to perform the coordination process, allowing agents to reason about the different proposals and vote for the next base plan.

In conclusion, the internal design of planning agents provides them with the basic capabilities required to solve MAP tasks. Agents use their internal components to interact with each other through the communication infrastructure, reason about plans and proceed with the next plan refinement.

3.3.5 Planning model

This section presents the MAP model upon which our planning architecture is based. It also describes the procedure followed by the agents for building and exchanging plans among them.

The following subsections describe and formalize the main components of a MAP task and outline the Partial-Order Planning concepts used in the MAP algorithm (see section 3.3.7). In order to illustrate the formal definitions introduced in this section, we provide simple examples based on the transportation MAP task presented in section 3.3.3. Also, for the sake of clarification of some definitions, we point out the reader to the figures of plans showed in section 3.3.9.

3.3.5.1 Formalization of a MAP task

Definition 3.1. *(**MAP task**) A **MAP task** is a tuple* $\mathcal{T} = \langle \mathcal{AG}, \mathcal{O}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$. $\mathcal{AG} = \{1, \ldots, n\}$ *is a finite non-empty set of planning agents.* $\mathcal{O}$ *is a finite set of objects that model the elements of the planning domain over which the planning actions can act.* $\mathcal{V}$ *is a finite set of state variables that model the states of the world. Each state variable* $v \in \mathcal{V}$ *is mapped to a finite domain of mutually exclusive values* $\mathcal{D}_v$*. Each value in a state variables's domain corresponds to an object of the planning domain, i.e.* $\forall v \in \mathcal{V}$, $\mathcal{D}_v \subseteq \mathcal{O}$*. When a value is assigned to a state variable, the pair variable-value acts as a ground atom in propositional planning.* $\mathcal{A}$ *is the set of deterministic actions of the agents.* $\mathcal{I}$ *is the set of values assigned to the state variables in* $\mathcal{V}$ *and represents the initial state of the MAP task* $\mathcal{T}$*.* $\mathcal{G}$ *is the set of goals of the MAP task that agents have to achieve;* $\mathcal{G}$ *represents the values that the state variables are expected to take in the final state.*

Information that agents have on the states of the world (problem states) is modeled through a set of ground atoms or fluents. This includes the initial state, $\mathcal{I}$, and the goal state, $\mathcal{G}$. As opposite to STRIPS-like models (36), which apply negation by failure (only positive fluents are represented, the absence of a fluent implies its negation), we allow to explicitly represent both true and false information. Thus, our model adopts the open world assumption, considering that the information which is not explicitly stored in the internal model of agents is unknown to them. Again, this also refers to the information in the initial state, $\mathcal{I}$, and the goals, $\mathcal{G}$.

Definition 3.2. *(**Fluent**) A ground atom or **fluent** of the problem is a tuple of the form* $\langle v, d \rangle$*, where* $v \in \mathcal{V}$ *and* $d \in \mathcal{D}_v$*. A negative fluent is of the form* $\langle v, \neg d \rangle$*. A positive fluent* $\langle v, d \rangle$ *indicates that the variable* v *takes the value* d*, while a negative fluent* $\langle v, \neg d \rangle$ *indicates that the variable* v *does not take the value* d*.*

As stated in Definition 3.2, a fluent relates a variable with one of the values in its domain. For instance, let (`at t1`) be a variable that refers to the position of

a truck object `t1` in the example introduced in section 3.3.3. Possible values for this variable are the cities `cA`, `cB`, `cC`, `cD`, `cE` and `cF`. Then, a positive fluent ⟨`at t1`, `cA`⟩ indicates that `t1` is in `cA` while a negative fluent ⟨`at t1`, ¬`cA`⟩ indicates that `t1` is not in `cA`.

In our model, agents are heterogeneous as they may have different knowledge and planning capabilities. In addition, they may have *incomplete information* on the MAP task as this can be distributed across agents. In this case, agents must cooperate with each other to solve the MAP task. Even though information is distributed across agents, there must be a subset of state variables shareable between agents in order to exchange the values of such variables and successfully communicate between each other. To denote the actions, goals, etc. of an agent $i \in \mathcal{AG}$ we will use the superscript notation x^i for any such aspect x.

From the set of variables, $\mathcal{V}$, of the MAP task, we distinguish $\mathcal{V}^i$ as the set of variables managed by agent i, which includes the private variables, only known to agent i, and the public variables, shared with other agents. Thus, $\mathcal{V} = \{\mathcal{V}^i\}_{i=1}^n$. $D_v^i \subseteq D_v$ is the set of values of a variable $v \in \mathcal{V}^i$ that are visible to agent i. The information of the initial state of the MAP task, $\mathcal{I}$, is modeled through a set of positive and negative fluents. This information is distributed among agents under the assumption that agents' partial knowledge about $\mathcal{I}$ is consistent, i.e. there is not contradictory information among agents. Hence, $\mathcal{I}$ can be defined as $\mathcal{I} = \bigcup_{\forall i \in \mathcal{AG}} \mathcal{I}^i$. It is possible to define MAP tasks in which all the agents have a complete view of the initial state $\mathcal{I}$, i.e. $\forall i \in \mathcal{AG},\ \mathcal{I}^i = \mathcal{I}$.

For example, Ag1 and Ag2 are two transport agents in the MAP scenario of section 3.3.3. Initially, Ag1 knows that the truck `t1` is in city `cA` so the fluent ⟨`at t1`, `cA`⟩ is part of $\mathcal{I}^{Ag1}$. On the contrary, Ag2 does not know where `t1` is initially located, but it knows that the truck is not in city `cB`. Hence, the fluent ⟨`at t1`,

$\neg$cB$\rangle$ belongs to $\mathcal{I}^{Ag2}$, the initial state of Ag2.

Each agent $i \in \mathcal{AG}$ is associated with a set, $\mathcal{A}^i$, of possible actions such that the set of actions of a planning task is defined as $\mathcal{A} = \bigcup_{\forall i \in \mathcal{AG}} \mathcal{A}^i$. An action α is said to be public if it is shared by two or more agents, i.e. $\alpha \in A^i \wedge \alpha \in A^j$, $i \neq j$. $\alpha \in \mathcal{A}^i$ is private to agent i iff $\alpha \notin \mathcal{A}^j, \forall j \neq i$. An action $\alpha \in \mathcal{A}^i$ denotes that agent i has the capability expressed in α. If α forms part of the final plan then agent i is also responsible of executing α.

Definition 3.3. *(**Planning rule or action**) A **planning rule or action** $\alpha \in \mathcal{A}$ is a tuple $\langle PRE(\alpha),\ EFF(\alpha)\rangle$. $PRE(\alpha) = \{p_1, \ldots, p_n\}$ is a set of fluents that represents the preconditions of α, while $EFF(\alpha) = \{e_1, \ldots, e_m\}$ is a set of operations of the form $(v = d)$ or $(v \neq d)$, $v \in \mathcal{V}$, $d \in \mathcal{D}_v$, that represent the consequences of executing α.*

An action α may belong to different agents, i.e. $\alpha \in \mathcal{A}^i$ and $\alpha \in \mathcal{A}^j$, $i \neq j$. Executing an action α in a world state S gives rise to a new world state S' generated as the result of applying $EFF(\alpha)$ over S. Particularly:

- An operation $(v = d) \in EFF(\alpha)$ implies the addition of a fluent $\langle v, d\rangle$ and a set of fluents $\langle v, \neg d'\rangle$, $\forall d' \in \mathcal{D}_v \mid d' \neq d$, to the world state S'. If $\langle v, d'\rangle \in S$ or $\langle v, \neg d\rangle \in S$, $d' \neq d$, the operation $(v = d)$ also implies that the fluents $\langle v, d'\rangle$ or $\langle v, \neg d\rangle$ will not be present in S'. For example, suppose that agent Ag1 knows that the truck `t1` can be placed at the cities `cA`, `cB`, `cC` and `cD`, i.e., $\mathcal{D}^{Ag1}_{at\,t1} = \{$`cA`, `cB`, `cC`, `cD`$\}$. If Ag1 knows a positive fluent $\langle$`at t1`, `cA`$\rangle$, it also knows the negative fluents $\langle$`at t1`, $\neg$`cB`$\rangle$, $\langle$`at t1`, $\neg$`cC`$\rangle$ and $\langle$`at t1`, $\neg$`cD`$\rangle$.

- An operation $(v \neq d) \in EFF(\alpha)$ implies the addition of a fluent $\langle v, \neg d\rangle$ to the world state S'. If $\langle v, d\rangle \in S$, the operation $(v \neq d)$ also entails that the fluent $\langle v, d\rangle$ will not be present in S'. Note that the only existence of

a fluent $\langle v, \neg d\rangle$ in a state indicates that the value of the variable v is not known in such a state and, consequently, the rest of values in $\mathcal{D}_v$, except for d, are unknown values. For example, if the fluent $\langle$`at t1`, $\neg$`cB`$\rangle$ is in the world state of agent Ag2, then the agent only knows that the truck `t1` is not in city `cB` but the agent is not aware of the actual position of the truck. Thus, whether `t1` is in `cA`, `cC` or `cD` is unknown to Ag2.

The set of preconditions of an action α, $PRE(\alpha)$, defines the fluents that must hold in a world state S for that α is applicable in this state. A positive precondition of the form $\langle v, d\rangle$ indicates that the fluent $\langle v, d\rangle$ must hold in S, while a negative precondition $\langle v, \neg d\rangle$ indicates that the fluent $\langle v, \neg d\rangle$ must hold in S. Note that the existence of a positive fluent $\langle v, d\rangle$ also implies the existence of a negative fluent $\langle v, \neg d'\rangle$ for the rest of values in the variable's domain, i.e. $(\exists\langle v, d\rangle \in S) \Rightarrow (\forall d' \in \mathcal{D}_v,\, d' \neq d,\, \exists\langle v, \neg d'\rangle \in S)$.

Additionally, agents use a utility function $\mathcal{F}$ to evaluate the quality of the plan proposals. For each agent i, $\mathcal{F}$ assigns a cost, $cost(view^i(\Pi)) \in \mathbb{R}_0^+$, to each plan proposal Π according to the view that agent i has of that plan, $view^i(\Pi)$. Finally, the private goals of an agent i, $\mathcal{PG}^i$, are fluents that agent i is interested in attaining. Private goals are encoded as soft constraints (42), as it is not mandatory that agents achieve them.

3.3.5.2 Concepts on Partial-Order Planning

Our MAP model can be regarded as a multi-agent refinement planning framework, a general method based on the refinement of the set of all possible plans (59). An agent proposes a plan Π that typically enforces some of the goals that have not yet been supported (see definition 3.5); then, the rest of agents collaborate on the refinement of Π by solving some of these pending goals in Π. This way, agents cooperatively solve the MAP task by consecutively refining an initially empty plan.

In this context, Partial-Order Planning (POP) (3) arises as a suitable approach to address refinement planning, since it is focused on solving the pending goals progressively. Consequently, agents in our MAP approach plan concurrent actions through the adoption of the POP paradigm. In the following, we provide some basic definitions concerning single-agent POP and its adaptation to a MAP context.

Single-agent Partial-Order Planning.

Definition 3.4. *(**Partial plan**) A **partial plan** is a tuple* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$. $\Delta \subseteq \mathcal{A}$ *is the set of actions in* Π. $\mathcal{OR}$ *is a set of ordering constraints* ($\prec$) *on* Δ. $\mathcal{CL}$ *is a set of causal links over* Δ. *A causal link is of the form* $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ *or* $\alpha \overset{\langle v,\neg d \rangle}{\rightarrow} \beta$, *where* $\alpha \in \mathcal{A}$ *and* $\beta \in \mathcal{A}$ *are actions in* Δ. $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ *indicates that there is an operation* $(v = d)$ *such that* $v \in \mathcal{V}$, $d \in \mathcal{D}_v$, $(v = d) \in EFF(\alpha)$ *and a fluent* $\langle v, d \rangle \in PRE(\beta)$. $\alpha \overset{\langle v,\neg d \rangle}{\rightarrow} \beta$ *indicates that there is a fluent* $\langle v, \neg d \rangle$ *such that* $v \in \mathcal{V}$, $d \in \mathcal{D}_v$, $\langle v, \neg d \rangle \in PRE(\beta)$ *supported by an operation* $(v \neq d) \in EFF(\alpha)$ *or an operation* $(v = d') \in EFF(\alpha)$, $d' \in \mathcal{D}_v$, $d' \neq d$.

This definition of partial plan represents the mapping of a plan into a directed acyclic graph, where Δ represents the nodes of the graph (actions) and $\mathcal{OR}$ and $\mathcal{CL}$ are sets of directed edges representing the precedences and causal links among these actions, respectively.

An *empty* partial plan is defined as $\Pi_0 = \langle \Delta_0, \mathcal{OR}_0, \mathcal{CL}_0 \rangle$, where Δ_0 contains α_0 and α_f, the initial and final action of the plan, respectively. α_0 and α_f are fictitious actions that do not belong to the action set of any particular agent. $\mathcal{OR}_0$ contains the constraint $\alpha_0 \prec \alpha_f$ and $\mathcal{CL}_0$ is an empty set. This way, a plan Π for any given MAP task $\mathcal{T}$ will always contain the two fictitious actions such that $PRE(\alpha_0) = \emptyset$ and $EFF(\alpha_0) = \mathcal{I}$, $PRE(\alpha_f) = \mathcal{G}$, and $EFF(\alpha_f) = \emptyset$; i.e. α_0 represents the initial situation of the MAP task $\mathcal{T}$, and α_f represents the global goals of $\mathcal{T}$.

Assuming that $\mathcal{G} \neq \emptyset$, an empty plan is said to be incomplete if the preconditions of α_f are not yet supported through a causal link. The POP process is aimed at introducing causal links to support these preconditions, also called *open goals*.

Definition 3.5. *(**Open goal**) An **open goal** in a partial plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ is a fluent og of the form $\langle v, d \rangle$ or $\langle v, \neg d \rangle$, such that $v \in \mathcal{V}$, $d \in \mathcal{D}_v$, $og \in PRE(\beta)$, $\beta \in \Delta$, and $\nexists \alpha \in \Delta / \alpha \overset{og}{\rightarrow} \beta \in \mathcal{CL}$, i.e., an open goal og is a precondition of ab action β in the plan Π that is not yet supported by a causal link $\alpha \overset{og}{\rightarrow} \beta \in \mathcal{CL}$. $openGoals(\Pi)$ denotes the set of open goals in Π. A plan is **incomplete** if it has open goals. Otherwise, we say it is a **complete** plan.*

As the POP search progresses, the causal links in a partial plan may become unsafe as a result of the introduction of a new action which is not ordered with respect to the causal link. These conflicts are called *threats*.

Definition 3.6. *(**Threat**) A **threat** in a partial plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ represents a conflict between an action of the plan and a causal link. An action γ causes a threat over a causal link $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ if $((v = d') \in EFF(\gamma) \vee (v \neq d) \in EFF(\gamma))$, where $v \in \mathcal{V}$, $d \in \mathcal{D}_v$, $d' \in \mathcal{D}_v$ and $d \neq d'$, and there is neither an ordering constraint $\gamma \prec \alpha$ nor $\beta \prec \gamma$. The action γ will cause a threat over a causal link of the form $\alpha \overset{\langle v,\neg d \rangle}{\rightarrow} \beta$ if $(v = d) \in EFF(\gamma)$, where $v \in \mathcal{V}$, $d \in \mathcal{D}_v$, and there is neither an ordering $\gamma \prec \alpha$ nor $\beta \prec \gamma$. $Threats(\Pi)$ denotes the set of threats in Π.*

A threat $t \in Threats(\Pi)$ can be solved by *promoting* or *demoting* the threatening action γ with respect to the causal link $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ or $\alpha \overset{\langle v,\neg d \rangle}{\rightarrow} \beta$, i.e. introducing an ordering constraint $\gamma \prec \alpha$ or $\beta \prec \gamma$. Threats and open goals are referred to as the *flaws* of a partial-order plan. The POP process is guided by solving the pending flaws of an initially empty partial plan.

Figure 3.6 in section 3.3.9 depicts a refinement plan for the example introduced in section 3.3.3. This refinement plan includes a causal link `Init` $\overset{\langle \mathsf{at\,t1,cA} \rangle}{\rightarrow}$ `load t1 p3 cA`. Suppose that a new action `drive t1 cA cB`, that causes the truck `t1`

to move from `cA` to `cB`, is added to the refinement plan and that this new action is not ordered with respect to `load t1 p3 cA`. In this case, (`at t1` = `cB`) $\in$ EFF(`drive t1 cA cB`). This effect causes a threat over the previous causal link, as it introduces a fluent $\langle$`at t1, cB`$\rangle$ that affects the value of the variable (`at t1`). This threat can be solved by introducing an ordering constraint `load t1 p3 cA` $\prec$ `drive t1 cA cB`, i.e., demoting the threatening action `drive t1 cA cB` with respect to the causal link.

Multi-agent Partial-Order Planning. Agents in our MAP model cooperate on the refinement of a base plan Π by proposing *refinement steps* that solve some open goals in Π. This way, agents cooperatively solve the MAP task by consecutively refining Π, the initially empty base plan.

Definition 3.7. ***(Refinement step)*** *A refinement step Π^i devised by an agent i over a base plan Π_g, where $g \in openGoals(\Pi_g)$, is a triple $\Pi^i = \langle \Delta^i, OR^i, CL^i \rangle$, where $\Delta^i \subseteq \mathcal{A}$ is a set of actions and OR^i and CL^i are sets of orderings and causal links over Δ^i, respectively. Π^i is a threat-free partial plan that solves g as well as all the new open goals of the form $\langle v, d \rangle$ or $\langle v, \neg d \rangle$ that arise from this resolution and can only be achieved by agent i, where $(v \in \mathcal{V}^i) \wedge (v \notin \mathcal{V}^j, \forall j \neq i)$. That is, when solving a goal of a base plan, agents only accomplish the new open goals concerning their private fluents, leaving public goals unresolved. In other words, the refinement method only iterates over the public fluents. Let $g \in openGoals(\Pi_g)$ be a fluent of the form $\langle v, d \rangle$ or $\langle v, \neg d \rangle$; an agent i proposes a refinement step over Π_g iff $v \in \mathcal{V}^i$.*

In our MAP approach partial plans are multi-agent concurrent plans as two or more actions can be concurrently executed by different agents. Some MAP models adopt a simple form of concurrency: two concurrent actions are mutually consistent if none of them changes the value of a state variable that the other relies on or affects, too (15). We impose the additional concurrency constraint that the preconditions of two actions have to be consistent (12) for these two actions to be

mutually consistent. This definition of concurrency is straightforwardly extended to a joint action $\alpha = \langle \alpha_1, \ldots, \alpha_n \rangle$.

Definition 3.8. *(**Mutually consistent actions**) Two concurrent actions $\alpha \in \mathcal{A}^i$ and $\beta \in \mathcal{A}^j$ are **mutually consistent** if none of the following conditions holds:*

- $\exists (v = d) \in EFF(\alpha)$ *and* $\exists (\langle v, d' \rangle \in PRE(\beta) \vee \langle v, \neg d \rangle \in PRE(\beta))$, *where* $v \in \mathcal{V}^i \cap \mathcal{V}^j$, $d \in \mathcal{D}_v^i \cap \mathcal{D}_v^j$, $d' \in \mathcal{D}_v^j$ *and* $d \neq d'$, *or vice versa.*
- $\exists (v = d) \in EFF(\alpha)$ *and* $\exists ((v = d') \in EFF(\beta) \vee (v \neq d) \in EFF(\beta))$, *where* $v \in \mathcal{V}^i \cap \mathcal{V}^j$, $d \in \mathcal{D}_v^i \cap \mathcal{D}_v^j$, $d' \in \mathcal{D}_v^j$ *and* $d \neq d'$, *or vice versa.*
- $\exists \langle v, d \rangle \in PRE(\alpha)$ *and* $\exists (\langle v, d' \rangle \in PRE(\beta) \vee \langle v, \neg d \rangle \in PRE(\beta))$, *where* $v \in \mathcal{V}^i \cap \mathcal{V}^j$, $d \in \mathcal{D}_v^i \cap \mathcal{D}_v^j$, $d' \in \mathcal{D}_v^j$ *and* $d \neq d'$, *or vice versa.*

Going back to our example in section 3.3.3, two concurrent actions `drive t1 cA cB`, planned by agent Ag1, and `drive t1 cA cC`, planned by agent Ag2, are mutually inconsistent as (`at t1 = cB`) $\in EFF$(`drive t1 cA cB`) and $\langle$`at t1, cC`$\rangle \in PRE$(`drive t1 cC cB`) (the first condition in Definition 3.8 holds). Concurrent actions `drive t1 cA cB` and `drive t1 cA cC` are also mutually inconsistent as (`at t1 = cB`) $\in EFF$(`drive t1 cA cB`) and (`at t1 = cC`) $\in EFF$(`drive t1 cA cC`) (second condition holds). Finally, concurrent actions `drive t1 cA cB` and `drive t1 cC cB` are mutually inconsistent as $\langle$`at t1, cA`$\rangle \in PRE$(`drive t1 cA cB`) and $\langle$`at t1, cC`$\rangle \in PRE$(`drive t1 cC cB`) (third condition holds).

As agents address concurrency inconsistencies through the detection of threats over the causal links of their plans, concurrency is ensured among private actions since a refinement step put forward by an agent is always a threat-free plan. However, concurrency issues between two public actions introduced by different agents do not arise until their preconditions are fully supported through causal links. This way, it is not possible to ensure that two concurrent actions are mutually consistent until their preconditions are fully supported. Thus, our notion

of multi-agent concurrent plan distinguishes between public and private actions when dealing with concurrency.

Definition 3.9. *(**Multi-agent concurrent plan**) A partial plan* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ *is a **multi-agent concurrent plan** if for every pair of unequal, concurrent, public actions* α *and* β, $\alpha \neq \beta$, $\forall p_\alpha \in PRE(\alpha), p_\alpha \notin openGoals(\Pi)$, $\forall p_\beta \in PRE(\beta), p_\beta \notin openGoals(\Pi)$, α *and* β *are mutually consistent.*

Definition 3.10. *(**Refinement plan**) A refinement plan* Π *devised by an agent* i *over a base plan* Π_g *is a concurrent multi-agent plan that results from the composition of* Π_g *and a refinement step* Π^i *proposed by agent* i*.* Π *is defined as* $\Pi = \Pi_g \circ \Pi^i$*, where* $\circ$ *represents the composition operation.*

Thus, an agent i can build a refinement plan Π upon a base plan Π_g by composing Π_g and a refinement step Π^i that solves at least $g \in openGoals(\Pi_g)$, i.e. $\Pi = \Pi_g \circ \Pi^i$. As previously mentioned, refinement steps are always threat-free and their actions are mutually consistent. Hence, if a refinement step brings about a concurrency inconsistency or a threat on the composite plan, the proposer agent is responsible for addressing such a flaw. If an agent is not able to come up with a consistent refinement plan, then it refrains from suggesting it. In case no refinements for a base plan can be found, the base plan is said to be a dead-end plan.

Definition 3.11. *(**Dead-end plan**) A plan* Π *is called a **dead-end plan** if* $\exists g \in openGoals(\Pi)$ *and there is no refinement step* Π^i *such that* $g \notin openGoals(\Pi \circ \Pi^i)$*; that is, no refinement step solves the open goal* g*.*

Definition 3.12. *(**Solution plan**) A multi-agent concurrent plan* Π *is a **solution plan** for a planning task* $\mathcal{T}$ *if* $openGoals(\Pi) = \emptyset$ *(*Π *is a complete plan),* $Threats(\Pi) = \emptyset$*, and every pair of actions* $\alpha, \beta \in \Pi$ *are mutually consistent.*

That is, a solution plan is a complete multi-agent concurrent plan. Note that we require Π to be a complete plan so it cannot have pending open goals. Consequently, the preconditions of the fictitious final action α_f will also hold thus

guaranteeing that Π solves the MAP task $\mathcal{T}$. For instance, Figure 3.8 in section 3.3.9 shows a solution plan for the MAP task presented in section 3.3.3. The different shapes of the actions indicate which agent has proposed them. The solution plan in Figure 3.8 is a complete, concurrent plan to which all the agents in the MAP task have contributed.

3.3.6 Planning language for MAP tasks

In our MAP system, we define the agents' planning tasks through several specification files. These files encode the information of the agent on the MAP task, namely the variables, $\mathcal{V}^i$; the objects associated to the variables, $\mathcal{O}^i$; the planning actions, $\mathcal{A}^i$; and the initial state of the agent, $\mathcal{I}^i$. All this information is written in a planning definition language.

Traditionally, planning has been regarded as a single-agent problem, where only one centralized planning entity is required. MAP presents new requirements and challenges that are not present in classical, centralized planning. Planning agents in our MAP model can withhold their private information, and decide which information to share with the rest of agents. In addition, planning agents can have private individual objectives besides the goals of the planning task. Therefore, the planning language must provide support to allow us to define shareable information and private goals.

Planning definition languages have experienced a remarkable evolution over the last years, continuously increasing their expressivity through the addition of new features. Our MAP language is based on *PDDL3.1* (64), the most recent version of *PDDL* (43), which was introduced in the context of the 2008 International Planning Competition. Unlike its predecessors, that model a planning domain through logical predicates, *PDDL3.1* also incorporates state variables by adding fluents that map a tuple of objects to an object of the planning task. We have

extended the *PDDL3.1* language with some new structures to model the multi-agent features of a planning task.

In single-agent PDDL language, the user writes two files, one containing the *domain* of the task and another one containing the data of the *problem* to be solved. The domain file describes the planning actions, the types of objects and the state variables of the task, while the problem file details the current objects of the task, the initial state (the initial values of the state variables) and the task goals. These files have a similar structure to their *PDDL* counterparts, and reflects the additional information required by MAP tasks.

In our MAP system, each agent has a domain and a problem file that model, respectively, the typology of the planning agent and its particular vision of the MAP task. The domain and problem files also include the information that is shared among agents. The `shared-data` structure allows the problem designer to define which fluents will be shared by each agent and with whom. Through this structure, the designer can define the *incomplete information* of the agent. This way, the domain knowledge of the agents can be modeled (or specified) from a complete unawareness to a full visibility of the domain. Additionally, since agents in MAP may have both global and local goals, this information is modeled through the structures `global-goal` and `private-goal`. Finally, we have included an additional `multi-functions` structure in order to simplify the specification of fluents in the initial state of an agent.

The following subsections analyze the structures that cover the requirements of MAP domains, i.e. modeling the data shared among agents, and the definition of local and global goals. The last subsection provides an example that describes the encoding of the MAP task introduced in section 3.3.3 with our language.

3.3.6.1 Shared data

The `shared-data` structure, located on the agent's problem file, determines which fluents are shareable and with which agent or agents they will be shared. As shown in section 3.3.8, this structure directly affects the initial information exchange that agents perform before planning, and it also defines the partial view of the planning task of each agent.

As agents only exchange fluents, in the `:shared-data` structure the problem designer specifies the fluents that the agent can share and with which agents. The `shared-data` structure has the following BNF syntax:

```
<shared-data-def>  ::= (:shared-data <share-def>+)
<share-def>        ::= (<atom-formula-def>+ [- <agent-def>?])
<agent-def>        ::= <agent> | (either <agent> <agent>+)
<agent>            ::= <name>
<atom-formula-def> ::= (<predicate> <typed-list(element)>)
<atom-formula-def> ::= (= <object-fluent> <object>)
<predicate>        ::= <name>
<object-fluent>    ::= (<name> <object>*)
<object>           ::= <name>
<element>          ::= <variable> | <constant>
<variable>         ::= ?<name>
<constant>         ::= <name>
<typed-list(x)>    ::= x*
```

As the BNF syntax shows, it is possible to define fluents or predicates within the `:shared-data` section and associate them to one, some or all the agents in the system (if `agent` is not specified, the predicates or fluents are shared with all the agents).

3.3.6.2 Private and global goals

A particularity of the MAP approach when compared to traditional planning is the fact that agents have private and global goals. To reflect this information in the model, the `private-goal` and `global-goal` structures have been included into

the problem files. Similarly to the `goal` section in *PDDL3.1*, goals can be modeled through predicates or fluents. The `private-goal` and `global-goal` structures use the following BNF syntax:

```
<private-goal-def>  ::= (:private-goal <predicate-def>)
<global-goal-def>   ::= (:global-goal <predicate-def>)
<predicate-def>     ::= <atom-formula-def>
<predicate-def>     ::= (and <atom-form-def> <atom-form-def>+)
<predicate-def>     ::= (or <atom-form-def> <atom-form-def>+)
<atom-form-def>     ::= (<predicate> <typed-list(element)>)
<atom-form-def>     ::= (= <object-fluent-def> <object>)
<predicate>         ::= <name>
<object-fluent-def> ::= (<name> <object>*)
<object>            ::= <name>
<element>           ::= <variable> | <constant>
<variable>          ::= ?<name>
<constant>          ::= <name>
<typed-list(x)>     ::= x*
```

As shown in the BNF syntax description, both global and local goals are described as conjunctions or disjunctions of fluents and predicates, or rather as a single fluent or predicate.

3.3.6.3 Encoding example

This subsection describes the encoding of the MAP task presented in section 3.3.3 with our MAP language. This MAP task describes a transportation and storage scenario in which two agents (Ag1 and Ag2) take the role of transport agencies and an agent (Ag3) manages a storage facility. Transport agents deliver packages through a network of cities, while the warehouse agent stores and loads the packages in trucks. In the following, we provide a description of the domain and problem files of the agents for this task, stressing the specification of shareable information.

Planning agents receive two different description files, namely the domain and problem file. The domain file contains a general description of the capabilities of the agent, including the actions that the agent can perform and the predicates and functions it can manage. All agents of the same type share the same domain file, e.g. transport agents Ag1 and Ag2 in this example receive the same domain file. The problem file models the concrete problem assigned to each agent, including a description of the objects of the task, the initial situation and the global goals of the task as well as private goals of the agent. Each agent receives its particular problem file.

Domain files. The domain file for transport agents specifies bidirectional links among cities, which allow trucks to move from one city to another. Trucks can only travel within the cities included in their working areas, depicted in Figure 3.3 with two circles. This way, transport agents have to interact and cooperate in order to deliver packages to a different area. The domain file for transport agents is modeled as follows:

```
(define (domain Transport)
 (:requirements :typing :equality :fluents)
 (:types truck package agent city - object
         raw-material final-product - package)
 (:predicates (empty ?c - city))
 (:functions (at ?t - truck) - city
             (pos ?p - package) - (either city truck)
 )
 (:multi-functions (link ?c - city) - city
                   (area) - city
 )
 (:action load
  :parameters (?t - truck ?p - package ?c - city)
  :precondition (and (member (area) ?c)(= (at ?t) ?c)(= (pos ?p) ?c))
  :effect (and (assign (pos ?p) ?t)(empty ?c))
 )
 (:action unload
```

```
  :parameters (?t - truck ?p - package ?c - city)
  :precondition (and (empty ?c)(member (area) ?c)
                (= (at ?t) ?c)(= (pos ?p) ?t))
  :effect (and (assign (pos ?p) ?c)(not (empty ?c)))
 )
 (:action drive
  :parameters (?t - truck ?c1 ?c2 - city)
  :precondition (and (member (area) ?c1)(member (area) ?c2)
                     (member (link ?c1) ?c2)(= (at ?t) ?c1))
  :effect (assign (at ?t) ?c2)
 )
)
```

The domain file shown above is structured similarly as a regular *PDDL3.1* file. The main sections of the file are highlighted in bold. The `:requirements` section indicates the *PDDL* features that have been used to encode the domain information. `:types` describes the object-type hierarchy of this particular domain. As it can be seen, the planning domain of transport agents includes four different types of objects, namely `truck`, `agent`, `city` and `package`. A `package` can be either a `raw-material` or a `final-product`.

Structures `:predicates`, `:functions` and `:multi-functions` define the state variables used in the transport domain. During the planning process, these variables will be instantiated to objects defined in the transport agents' problems, thus giving rise to the fluents that will be used throughput the planning process. For instance, let us consider the function `(at ?t - truck) - city`, where `(at ?t)` is a state variable and `city` is the type of its domain values. Given a `truck` object `t1` and a `city` object `c1`, the previous function will result in a fluent of the form `(= (at t1) c1)`, which indicates that `t1` is located at `c1`.

The domain file of transport agents include the following predicates, functions and multi-functions: `empty` is a predicate to indicate whether a `city` is empty or already contains a `package` (a `city` can only have one `package` simultaneously);

function `at` returns the `city` in which a certain `truck` is placed; function `pos` describes the position of a `package`, either a `truck` or a `city`; multi-function `link` returns the outcoming connections (roads) from a certain `city`; and `area` describes the working area of an agent in terms of the cities it can drive a `truck` to.

The last portion of a *PDDL3.1* domain file defines the abilities of the agent, i.e., the actions it can perform. Actions are described through its `parameters` (objects that take part in the action), `preconditions` (predicates and functions that must hold for the action to be applicable) and `effects` (predicates and functions that describe the consequences of applying the action). As in the case of predicates, functions and multi-functions, actions are described through state variables. In particular, preconditions encode queries on fluents that check whether a variable takes on a particular value, and effects encode assignment operations on fluents to make a state variable take on a value.

As described in the domain file, transport agents can perform three different actions: `load` and `unload` a `package` into/from a `truck`, and `drive` a `truck` from a `city` to another one of the `agent`'s `area`.

The domain file for warehouse agents is similar to the classical *blocksworld* domain, in which packages can be stacked and unstacked on/from the table or other packages. In this case, only one pile of packages can be stacked on the table, and there are two types of packages, raw materials and final products. The transportation and storage scenario depicted in Figure 3.3 includes two final products (packages `p1` and `p2`) and a raw material (package `p3`). The warehouse agent delivers final products to the city in which the warehouse is placed (the exchange city, `cF` in Figure 3.3), and acquires raw materials that are unloaded from the trucks in the exchange city. Following, we show a sketch of the warehouse domain file encoding:

```
(define (domain Warehouse)
 (:requirements :typing :equality :fluents)
 (:types package agent city table hoist - object
         raw-material final-product - package)
 (:predicates (empty ?c - city)
              (clear ?p - (either package table hoist))
              (exchange-city ?c - city)
 )
 (:functions (pos ?p - package) - (either city package table hoist))
 (:action acquire
  :parameters (?p - raw-material ?c - city ?h - hoist)
  :precondition (and (= (pos ?p) ?c)(clear ?h)(exchange-city ?c))
  :effect (and (assign (pos ?p) ?h)(not (clear ?h))(empty ?c))
 )
 ...
)
```

As the transport agents, warehouse agents manage `city`, `hoist` and `package` objects. Additionally, warehouse agents consider `table` and `hoist` objects. The `hoist` is used to `deliver` and `acquire` packages, while the `table` is used to `stack` and `unstack` packages within the warehouse.

Warehouse agents perform the four actions indicated above: they can `stack` and `unstack` a `package` on top/from a clear `table` or `package`; and can also `acquire` and `deliver` a `package` from/to the `exchange-city` by using a `hoist`. The sketch of the domain file illustrates the encoding of the `acquire` action.

Problem files. Each agent receives its own problem file that models the particular objects managed by the agent, the initial situation known to the agent and the global and private goals that the agent must achieve. Moreover, the problem files include the definition of the shareable fluents and with which agents they can be shared.

We now explain the problem file of transport agent Ag1 (this problem will be later used to illustrate the construction of the dis-RPG). Problem files describe the

initial state of the task by including both the positive and negative information known by the agent. This way, the information not represented in the problem file is unknown to the agent. Ag1's problem file is encoded as follows:

```
(define (problem Ag1)
(:domain Transport)
(:objects
     Ag1 Ag2 Ag3 - agent
     t1 - truck
     cA cB cC cD cE cF - city
     p3 - raw-material
     p1 p2 - final-product)
(:shared-data
     (empty ?c - city) - (either Ag2 Ag3)
     ((at ?t - truck) - city)
     ((pos ?p - package) - (either city truck)) - Ag2
     ((pos ?p - package) - city) - Ag3
)
(:init
     (empty cB) (empty cC) (empty cD) (not (empty cA))
     (= (at t1) cA) (not (= (at t1) cB)) (not (= (at t1) cC))
     (not (= (at t1) cD)) (= (pos p3) cA) (not (= (pos p3) cB))
     (not (= (pos p3) cC)) (not (= (pos p3) cD))
     (= (link cA) {cB cC}) (not (= (link cA) {cA cD}))
     (= (link cB) {cA cC}) (not (= (link cB) {cB cD}))
     (= (link cC) {cA cB cD}) (not (= (link cC) {cC}))
     (= (link cD) {cC}) (not (= (link cD) {cA cB cD}))
     (= (area) {cA cB cC cD}) (not (= (area) {cE cF}))
)
(:global-goal (and (= (pos p1) cA)(= (pos p3) cE)))
)
```

Sections of the problem file are also highlighted in bold. A problem file starts with a description of the `:objects` that the agent manages. As shown in the code, agents are represented as objects. Ag1 knows that there is a `truck t1` in the task, and it has knowledge of six different cities, although it only manages the four cities included in its working `area` (see Figure 3.3). The agent also knows that there are three packages in the MAP task, the `final-products p1` and `p2`

and the `raw-material p3`.

The `:shared-data` section is a key aspect of our MAP language, as it defines the information shareable by the agents and directly affects their knowledge of the task. The predicates and functions defined in this structure are the patterns of the fluents that the agent regards as shareable with other agents. For instance, Ag1's `:shared-data` section includes the following pattern: `(empty ?c - city) - (either Ag2 Ag3)`. This pattern indicates that Ag1 will share the fluents that match the pattern `(empty ?c - city)` with both Ag2 and Ag3. Given that Ag1 knows the cities `cA`, `cB`, `cC`, `cD`, `cE` and `cF`, fluents as `(= (empty cA) true)` or `(= (empty cD) false)` match the pattern, and Ag1 shares this information with Ag2 and Ag3.

The `:init` section describes the initial state of Ag1, i.e., the initial situation of the world known to Ag1. It is defined with predicates like `(empty cB)`, functions like `(= (at t1) cA)`) and multi-functions like `(= (link cA) {cB cC})`, that hold in the initial situation. The initial state includes both positive and negative information. For instance, the function `(not (= (at t1) cC))` indicates that Ag1 knows that `truck t1` is not initially placed at `city cC`. The information not included in the initial state is considered unknown to Ag1.

While the initial state contains predicates, functions and multi-functions, internally the system treats all of them as fluents. For instance, a predicate `(empty cB)` is internally converted into a fluent `(= (empty cB) true)`, while functions like `(= (at t1) cA)` are already in the form of fluents. Multi-functions are used to easily define multiple functions through a simplified notation. The conversion into fluents is straightforward: given a multi-function `(= (link cA) cB cC)`, we generate the fluents `(= (link cA cB) true)` and `(= (link cA cC) true)`.

Finally, the `:global-goal` structure shows the global objective of the MAP task. In this case, the goal is to transport the `raw-material p3` to `city cF`, and

to deliver a `final-product` to `city cA`. Notice that, in this example, there is not a `:private-goal` section.

3.3.7 MAP algorithm overview

This section summarizes the main stages of the MAP algorithm followed by the agents to devise, exchange and select refinement plans to come up with a solution for the MAP task. Agents follow a procedure that integrates planning and coordination, allowing agents to solve both *strongly-related* and *loosely-coupled* problems without losing generality. Agents perform an individual Partial-Order Planning (POP) search to build refinements over the current base plan, while one of the agents leads the process of gathering the new refinement plans and selecting the next base plan.

Algorithm 3: Overview of the MAP algorithm

```
Initial information exchange
repeat
    Individual refinement process
    Coordination process
until a solution plan is found or the search space is completely explored
```

Algorithm 3 shows the main steps of the MAP algorithm. The stages of the algorithm are outlined as follows:

- **Initial information exchange:** The algorithm starts with an initial communication stage by which agents exchange the shareable information on the planning domain in order to generate the data structures that will be used in the subsequent planning process. Agents take advantage of the exchanged information to build a distributed Relaxed Planning Graph, which provides them with their partial view on the MAP task.

- **Resolution process:** Once agents have exchanged the shareable information and the distributed Relaxed Planning Graph is computed, they start the iterative resolution process by which they explore the search space until they find a solution for the MAP task. As shown in Algorithm 3, this process comprises two different interleaved stages, an individual planning process by which agents devise refinements over a centralized base plan and a coordination process to exchange the new refinement plans and to select the next base plan:
  - **Individual refinement process:** Agents individually refine the current base plan of the MAP system. Each planning agent is provided with an internal POP system. The classical POP algorithm has been adapted to a MAP context in order to obtain valid refinement plans over an incomplete base plan (see section 3.3.9.1).
  - **Coordination process:** Agents communicate and exchange the new refinement plans over the current base plan. Later, they jointly evaluate these refinement plans and select the most promising one as the next base plan.

The following sections detail the two main stages of the MAP algorithm. Section 3.3.8 describes the initial information exchanging stage performed by the agents, while section 3.3.9 details the resolution process, including both the coordination process and the individual construction of the refinement plans.

3.3.8 Initial information exchange

Prior to the resolution process itself, agents perform a preliminary stage to share public planning information effectively. This initial stage builds a distributed Relaxed Planning Graph (dis-RPG), whose construction is inspired by the approach

in (127). Unlike the proposal in (127), which stops the graph construction once all the problem goals appear in the graph, our procedure builds a complete dis-RPG by maintaining the *incomplete information* of the agents, so they only exchange the information defined as shareable in the input files (see section 3.3.6). This section describes in detail the dis-RPG building process and subsection 3.3.8.1 provides a trace based on the MAP task presented in section 3.3.3 that illustrates this process.

Algorithm 4: Dis-RPG construction for an agent i

Build initial RPG^i
repeat
  $\forall j \neq i$, i sends j shareable fluents $SF^{i \to j} \in RPG^i$ of the form $\langle v, d \rangle$ or $\langle v, \neg d \rangle$, where $v \in \mathcal{V}^i \cap \mathcal{V}^j$ and $d \in \mathcal{D}_v^i \cap \mathcal{D}_v^j$
  $\forall j \neq i$, i receives from j shareable fluents $SF^{j \to i} \in RPG^j$ of the form $\langle v, d \rangle$ or $\langle v, \neg d \rangle$, where $v \in \mathcal{V}^i \cap \mathcal{V}^j$ and $d \in \mathcal{D}_v^i \cap \mathcal{D}_v^j$
  $RF^i \leftarrow \emptyset$
  $\forall j \neq i, RF^i \leftarrow RF^i \cup SF^{j \to i}$
  for all *received fluents* $f \in RF^i$ **do**
    if $f \notin RPG^i$ **then**
      Insert f in RPG^i
      $level_{RPG^i}(f) \leftarrow level(f)$
    if $(f \in RPG^i) \wedge (level_{RPG^i}(f) > level(f))$ **then**
      $level_{RPG^i}(f) \leftarrow level(f)$
  Expand RPG^i
until $RF^i = \emptyset$

The dis-RPG provides the agents with valuable planning information that will be used throughout the refinement planning process:

- Agents exchange the fluents defined as shareable in the `:shared-data` section of the MAP domain definition files (see subsection 3.3.6.1). Fluents are

labeled with the list of agents that can achieve them, giving each agent a view of the possible interactions that can arise at planning time with other agents.

- An estimate of the best cost to achieve each fluent is computed. This information is used to design heuristics to guide the refinement planning process.

Following Algorithm 4, the first step of the dis-RPG construction consists in computing the initial RPG for each agent i, RPG^i, taking only into account the fluents and actions initially known to the agent. Agents compute this initial planning graph by following the procedure presented in (54). The RPG consists of a set of alternating fluent and action levels. The first fluent level contains the fluents that are part of the initial state, and the first action level includes all the actions whose preconditions appear in the first fluent level. Fluents that are part of the effects of these actions (and have not been included in the first fluent level) are placed in the second fluent level, and actions whose preconditions are included in the two prior fluent levels of the graph (and are not in the first action level) are stored in the second action level. By following this procedure, the RPG is expanded until no new fluents are found. This way, the level of the graph in which an action or fluent appears gives an estimate of the cost of achieving such an action or fluent.

Once all agents have computed their initial RPGs, the iterative composition of the dis-RPG begins. As depicted in Algorithm 4, after computing the initial RPG^i, agent i executes the first iteration of the algorithm and exchanges the fluents and actions of its RPG^i with the rest of agents. Agents only exchange the fluents defined as shareable in the `:shared-data` structure of the input files (see subsection 3.3.6.1). Agent i sends agent j the set of fluents $SF^{i \to j}$ that are visible to agent j, i.e., the fluents in RPG^i of the form $\langle v, d \rangle$ or $\langle v, \neg d \rangle$, where $v \in \mathcal{V}^i \cap \mathcal{V}^j$

F0	
[2](empty cB) T	[2](link cE cB) T
[2](empty cD) T	[2](link cE cD) T
[2](empty cE) T	[2](link cF cD) T
[2](empty cF) T	[2](area cB) T
[2](link cB cE) T	[2](area cD) T
[2](link cD cE) T	[2](area cE) T
[2](link cD cF) T	[2](area cF) T

Table 3.2: Initial RPG built by agent Ag2

F0		A0	F1	A1
[1](at t1) cA	[1](pos p3) cA	load t1 p3 cA	[1](pos p3) t1	unload t1 p3 cB
[1](empty cA) F	[1](link cC cA) T	drive t1 cA cB	[1](empty cA) T	unload t1 p3 cC
[1](empty cB) T	[1](link cC cB) T	drive t1 cA cC	[1](at t1) cB	unload t1 p3 cA
[1](empty cC) T	[1](link cC cD) T		[1](at t1) cC	drive t1 cB cA
[1](empty cD) T	[1](link cD cC) T			drive t1 cB cC
[1](link cA cB) T	[1](area cA) T			drive t1 cC cA
[1](link cA cC) T	[1](area cB) T			drive t1 cC cB
[1](link cB cA) T	[1](area cC) T			drive t1 cC cD
[1](link cB cC) T	[1](area cD) T			

F2	A2	F3	A3
[1](empty cB) F	load t1 p3 cB	[1](empty cD) F	load t1 p3 cD
[1](empty cC) F	load t1 p3 cC	[1](pos p3) cD	
[1](at t1) cD	unload t1 p3 cD		
[1](pos p3) cB	drive t1 cD cC		
[1](pos p3) cC			

Table 3.3: Initial RPG built by agent Ag1

and $d \in \mathcal{D}_v^i \cap \mathcal{D}_v^j$. Likewise, agent i will receive from the rest of agents $j \neq i$ the shareable fluents of their RPG^j that are visible to agent i.

Agent i updates its RPG^i accordingly with the new fluents received from the rest of agents. We will refer to these fluents as RF^i (see Algorithm 4). If a fluent $f \in RF^i$ is not in RPG^i then it is stored according to $level(f)$. If f is already in RPG^i, its level in the graph is updated if $level_{RPG^i}(f) > level(f)$. Hence, agents only store the best estimated level to reach each fluent, placing each fluent at the lowest possible level of the graph. After updating RPG^i, agent i expands it by checking wether the new inserted fluents trigger new actions in RPG^i or not.

The fluents produced as effects of these new actions will be shared in the next information exchange iteration. The RPG expansion procedure also updates the existing actions by placing them at a lower action level if their preconditions have been updated.

Since agents only exchange those fluents defined as shareable, the dis-RPG process gives each agent a different view of the planning task, so no agent handles a complete representation of the dis-RPG. In contrast, each agent i maintains its own internal RPG^i, whose information depends on the fluents other agents have shared with it, which makes each agent have its own, partial view of the planning task. Thus, agents design plans under *incomplete information*, as they are partly aware of the information on the planning task.

The dis-RPG process finishes when agents do not receive more fluents from the others. Following, agents start the resolution process to jointly devise a solution plan.

3.3.8.1 dis-RPG example

In order to illustrate the dis-RPG stage of the MAP algorithm, this section provides an example trace based on the transportation and storage MAP task introduced in section 3.3.3. The planning agents receive the input files presented in subsection 3.3.6.3 and start the MAP algorithm by building the dis-RPG.

In the first stage of Algorithm 4, each agent individually generates an initial RPG, according to its problem file. To illustrate this stage of the process, we focus on the initial RPGs built by the transport agents Ag1 and Ag2.

Table 3.2 shows the initial RPG calculated by agent Ag2. The numbers in brackets indicate the agents that can generate the fluent, while the values T and F stand for *true* and *false*, respectively. Ag2 does not know the position of the packages and the truck because they are initially located out of its working area

(see Figure 3.3 in section 3.3.3). Therefore, its initial RPG only includes F0, the first level of fluents, which stores the fluents on the initial state of Ag2. The initial RPG of Ag2 does not contain any action level because there are no applicable actions, that is, their preconditions do not hold in F0.

Agent Ag1 does know the position of the package `p3` and the truck `t1`, and consequently, it can compute a much larger initial RPG (see Table 3.3). Notice that the level A0 includes the actions whose preconditions are satisfied in F0, while F1 stores the fluents that are part of the effects of the actions in A0 and are not in F0. For instance, the action `drive t1 cA cB`, at level A0, has the following preconditions: `(= (area) cA)`, `(= (area) cB)`, `(= (link cA cB) true)` and `(= (at t1) cA)`. As Table 3.3 shows, these fluents are at F0, which triggers the action `drive t1 cA cB` at A0.

Once agents have built their initial RPGs, they start the iterative dis-RPG building process by exchanging the shareable fluents in their RPGs.

In subsection 3.3.6.3, we show the `:shared-data` section of Ag1, which shares with Ag2 fluents that match the following patterns: `(empty ?c - city)`, `((at ?t - truck) - city)` and `((at ?t - truck) - city)`. The fluents shared by Ag1 and Ag2 are marked in red in Table 3.3. Ag2 also sends its shareable fluents to the rest of agents and stores the fluents received from other agents.

Agents expand their RPGs by checking if the fluents they have received trigger new actions in the graph. The process carries on until no new fluents appear in the dis-RPG. As each agent has a different `:shared-data` section, the information will vary from one RPG to another, giving each agent a different and incomplete view of the dis-RPG and the MAP task itself.

Table 3.4 shows the final dis-RPG of the transportation scenario as seen by agent Ag2. As it can be observed, the dis-RPG provides both an estimate of the

F0		A0	F1	A1
[1, 2](empty cB) T	[2](link cE cB) T	load t1 p3 cA	[1](empty cA) T	unload t1 p3 cB
[1, 2](empty cD) T	[2](link cE cD) T	drive t1 cA cB	[1, 2](at t1) cB	unload t1 p3 cC
[2](empty cE) T	[2](link cF cD) T	drive t1 cA cC	[1](at t1) cC	unload t1 p3 cA
[2](empty cF) T	[2](area cB) T		[1, 2](pos p3) t1	drive t1 cB cA
[2](link cB cE) T	[2](area cD) T			drive t1 cB cC
[2](link cD cE) T	[2](area cE) T			drive t1 cC cA
[2](link cD cF) T	[2](area cF) T			drive t1 cC cB
[1](empty cA) F	[1](at t1) cA			drive t1 cC cD
[1](pos p3) cA	[2, 3](empty cF) T			

F2	A2	F3	A3
[1, 2](empty cB) F	load t1 p3 cB	[1, 2](empty cD) F	load t1 p3 cD
[1](empty cC) F	load t1 p3 cC	[2](at t1) cF	
[1, 2](at t1) cD	unload t1 p3 cD	[1, 2](pos p3) cD	
[2](at t1) cE	drive t1 cD cC	[2](pos p3) cE	
[1, 2](pos p3) cB		[2](empty cE) F	
[1](pos p3) cC		[2, 3](pos p2) cF	
[2, 3](pos p1) cF			
[1, 2](empty cF) F			

F4	A4	F5	A5
[2]**(pos p3) cF**	unload t1 p1 cB	[1]**(pos p1) cA**	load t1 p1 cB
[1, 2](pos p1) t1	unload t1 p1 cC	[1, 2](pos p1) cB	load t1 p1 cD
[1, 2](pos p2) t1	unload t1 p1 cD	[1](pos p1) cC	load t1 p2 cB
	unload t1 p1 cA	[1, 2](pos p1) cD	load t1 p2 cD
	unload t1 p2 cB	[2](pos p1) cE	load t1 p1 cC
	unload t1 p2 cC	[1](pos p2) cA	load t1 p1 cA
	unload t1 p2 cD	[1, 2](pos p2) cB	load t1 p2 cC
	unload t1 p2 cA	[1](pos p2) cC	load t1 p2 cA
		[1, 2](pos p2) cD	
		[2](pos p2) cE	

Table 3.4: Final dis-RPG as viewed by agent Ag2

cost of achieving each fluent (this cost corresponds to the level at which the fluent appears), and the set of agents that achieve that fluent in the RPG.

3.3.9 Resolution process

After the information exchange, agents initiate the resolution process (see Algorithm 5), which comprises two interleaved stages: the individual refinement stage and the coordination stage. The first stage involves agents building individual

refinements over a centralized base plan by using a POP. In the second stage, agents follow a coordination process to gradually build a joint solution plan for the MAP task, exchanging and evaluating the refinements generated individually and selecting the most promising one in order to reach a solution.

Algorithm 5: Resolution process for an agent i

$\Pi \leftarrow \Pi_0$
$R = \emptyset$
repeat
  Select open goal $g \in openGoals(\Pi)$
  $Refinements^i(\Pi_g) \leftarrow$ Refine base plan Π_g individually
  $\forall j \neq i$, send $Refinements^i(\Pi_g)$ to agent j
  $\forall j \neq i$, receive $Refinements^j(\Pi_g)$
  $Refinements(\Pi_g) \leftarrow Refinements^i(\Pi_g)$
  $\forall j \neq i$, $Refinements(\Pi_g) \leftarrow Refinements(\Pi_g) \cup Refinements^j(\Pi_g)$
  Evaluate $Refinements(\Pi_g)$
  $R \leftarrow R \cup Refinements(\Pi_g)$
  Vote for the best plan $\Pi^i \in R$
  $\Pi \leftarrow \Pi^i$
  if $openGoals(\Pi) = \emptyset$ **then**
    return Π
until $R = \emptyset$

3.3.9.1 Individual refinement stage

A planning agent i executes its individual POP process in order to refine the current base plan Π. As shown in Algorithm 5, agent i refines Π by solving a particular open goal $g \in openGoals(\Pi)$, thus obtaining a set of valid refinement plans over Π_g, $Refinements^i(\Pi_g)$.

Our definition of refinement plan (see subsection 3.3.5.2) states that a refinement plan Π^i of an agent i over a base plan Π solves one of its open goals

$g \in openGoals(\Pi)$, as well as all the private open goals g^i of the form $\langle v, d\rangle$ or $\langle v, \neg d\rangle$ that arise from this resolution, where $v \in \mathcal{V}^i \wedge d \in \mathcal{D}_v^i \wedge ((\forall j \neq i, v \notin \mathcal{V}^j) \vee (\forall j \neq i, d \notin \mathcal{D}_v^j)) \wedge (g^i \notin openGoals(\Pi))$.

We have designed a customized version of the classical POP algorithm compliant with the requirements introduced by the MAP approach. Our POP system is able to start the search process from any given base plan, rather than starting with an empty plan as in a traditional POP process. In addition, the POP is aimed at building refinement plans, rather than complete solution plans.

3.3.9.2 Coordination process

The coordination process is based on a democratic leadership in which a leadership baton is scheduled among the agents (initially, the baton is randomly assigned to one of the participating agents). The resolution process interleaves the coordination process with the individual refinement stage. A coordination iteration is led by the agent which has the baton (baton agent). Once the coordination iteration is completed, the baton is handed over to the following agent.

Algorithm 5 depicts the main steps of the coordination process. After the individual refinement stage, agents exchange the refinement plans they have elicited over the current base plan Π. Following, agents receive the refinement plans of the other agents and evaluate them according to their view of the planning task. Agents apply a voting process to adopt the most promising plan as the next base plan Π, and check if the selected plan is a solution. Otherwise, agents choose a new open goal of the plan $g \in openGoals(\Pi)$ and each agent i starts a new individual refinement stage to compute the refinements over Π, $Refinements^i(\Pi_g)$.

In the first step of the coordination process, the individual refinement plans are exchanged between agents for their evaluation. An agent i sends the refinement plans it has devised over the current base plan Π by solving $g \in openGoals(\Pi)$,

$Refinements^i(\Pi_g)$, to the rest of agents in the task. In turn, agent i receives the refinements devised by each agent j in the task, $Refinements^j\ (\Pi_g)$, where $j \neq i$. Note that agents have a local, partial view of the plans, so given a refinement plan Π, an agent i will only view the open goals $og \in openGoals(\Pi)$ of the form $\langle v, d\rangle$ or $\langle v, \neg d\rangle$ such that $v \in \mathcal{V}^i$ and $d \in \mathcal{D}^i_v$. The view agent i has on each refinement plan Π, $view^i(\Pi)$, ensures agents' privacy and directly affects the evaluation of the refinements.

The evaluation of the refinement plans is carried out through a utility function $\mathcal{F}$, by which agents estimate the quality of the plans. Since an agent i evaluates a plan accordingly to its view, $\mathcal{F}(view^i(\Pi))$, the results of the evaluation may be different from the other agents'. Therefore, agents will have different perspectives on the quality of the refinement plans.

Once the refinement plans are evaluated, agents vote for the most promising candidate in R, which stores all the refinement plans that have not yet been selected as a base plan (see Algorithm 5). Each agent i votes for the best refinement plan in R according to the utility function $\mathcal{F}$. In case of a draw, the baton agent will choose the next base plan among the most voted alternatives. If $R = \emptyset$, the refinement planning process ends with no solution found.

Once a refinement plan is selected, agents adopt it as the new base plan Π. If $openGoals(\Pi) = \emptyset$, a solution plan is returned. As some open goals might not be visible to some agents, all agents must confirm that Π is a solution plan according to their view of Π. Finally, the baton agent selects the next open goal $g \in openGoals(\Pi)$ to be solved, and a new iteration of the refinement and coordination process starts.

The resolution process carried out by the agents can be regarded as a joint exploration of the refinement space. Nodes in the search tree represent refinement plans and each iteration of the process expands a different node.

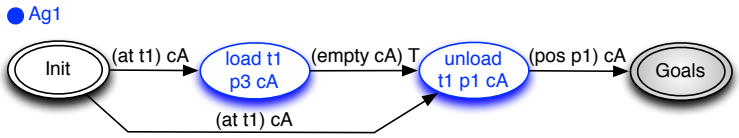

Figure 3.6: Refinement plan Π_{00}

3.3.9.3 Resolution example

This subsection illustrates the resolution process by showing a partial trace that follows the trace example described in subsection 3.3.8.1. After completing the initial information exchange and building the dis-RPG, agents proceed with the resolution process in order to solve the MAP task.

The plan construction starts with an initial empty plan, Π_0, which contains only the two fictitious steps that represent the initial state and the goals of the MAP task. The first open goal selected by Ag1 (which takes the role of baton agent in this first iteration) for its resolution is `(= (pos p1) cA)`, as it is the most costly one according to the dis-RPG. The goals of the task are highlighted in bold in Table 3.4. This dis-RPG shows that `(= (pos p1) cA)` has a cost of 5, `(= (pos p3) cF)` has a cost of 4, and the only agent that can achieve `(= (pos p1) cA)` is Ag1. Hence, Ag1 proposes a set of refinements over Π_0, $\Pi_{00}, \ldots, \Pi_{09}$, while Ag2 and Ag3 refrain from making proposals. The proposed refinements are evaluated through the utility function $\mathcal{F}$, and the best-valued one, Π_{00}, is selected as the new base plan.

Figure 3.6 depicts the refinement plan Π_{00}. Since all the causal links in Π_{00} involve shareable fluents, all the agents have a complete view of this refinement plan. However, agents Ag1 and Ag3 have different views of the refinement Π_{06} (see Figure 3.7). In order to guarantee privacy, several causal links (black arrows) of Π_{06} have been occluded to Ag3, which only sees ordering constraints instead (grey

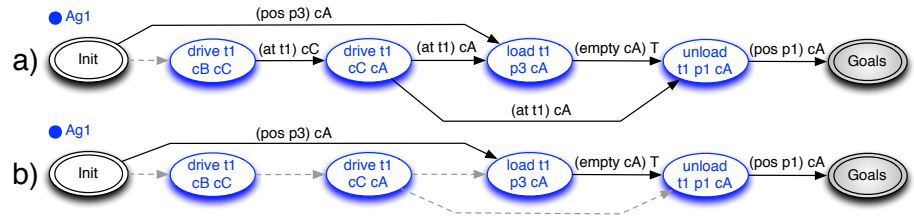

Figure 3.7: Refinement plan Π_{06} as observed by: a) Ag1 b) Ag3

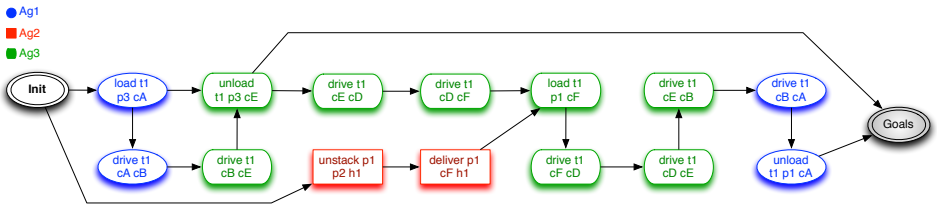

Figure 3.8: Solution plan for the MAP task

arrows). According to the problem definition files, the fluents involved in these causal links are private to the transport agent Ag1 because they are not shareable data, and therefore, Ag1 does not communicate them to Ag3.

Once the refinement plan Π_{00} is chosen as the new base plan, Ag1 passes on the baton to Ag2 and a new iteration of the resolution process starts. The MAP process will carry on until a solution plan is found. Since some open goals are not visible to some agents, all participating agents must confirm that the plan has no pending open goals. Figure 3.8 depicts the solution plan for the MAP task at hand, showing in different shapes the actions to be executed by each agent. As it can be observed, the solution of the MAP task is a joint plan to which all the participant agents have contributed.

3.3.10 Experimental results

Several tests have been performed to evaluate the performance of our MAP system. The tests compare the MAP model with a single-agent approach to analyze its advantages and shortcomings against a centralized planning model. We have used two different planning domains for our experiments. Next subsections present the MAP domains and analyze the results of the different tests.

3.3.10.1 Multi-agent planning domains

The two planning domains used to test the MAP system have been taken from real-life problems or adapted from well-known case studies. The two domains were designed such that we could test the performance and the quality of the solutions obtained with a wide range of problems. We tested our MAP system with different levels of complexity: from *loosely-coupled* problems, in which interactions among agents are rather low, to *strongly-related* problems, that require a strong coordination effort to be solved. Additionally, we created both a multi-agent and a single-agent version for each problem.

In section 3.3.3, we described a transportation and storage domain, in which agents take the roles of transport agencies and storage facilities, which work together to transport raw materials and final products to different cities. This domain gives rise to *strongly-related* problems as interactions between agents are required in order to accomplish the different objectives. Agents in the *transportation* domain have different abilities, so they should cooperate with each other in order to achieve the different goals.

We defined an additional planning domain, the *picture* domain. This domain gives rise to simpler, *loosely-coupled* problems as agents can work independently in order to solve the objectives, and hence cooperation and interactions among agents are not mandatory to find a solution. Planning agents in the picture domain

(workers) are not specialized, they all share the same abilities and so they all can perform the same actions. In addition, agents in this domain do not keep private information for themselves but all the problem information is shared among the agents. Next subsection describes the *picture* domain.

Picture domain. This domain, adapted from the case study in (87), presents a situation in which several workers have to cooperate to hang a set of pictures on walls. To do so, they have to acquire different tools that are scattered over several locations. Agents move through the locations to get the tools and hang the pictures. The domain defines a set of bidirectional links that connect the locations.

Figure 3.9 depicts an example of this planning domain. In contrast to the transportation domain, agents in the picture domain share the same capabilities: agents can `pickUp` and `putDown` a `tool` in the `location` where the `agent` and the `tool` are placed; an agent can also `pass` the `tool` it is carrying on to another `agent` at the same `location`; agents can `walk` from one `location` to another through the link that connects both locations; finally, an agent can `hang` a `picture` on a certain `location` with the `tool` it is carrying.

This domain gives rise to *loosely-coupled* problems because an individual agent is likely to solve the problem goals by itself in most cases. Moreover, agents share the same abilities and have access to all the locations, so they are able to work independently and cooperation is not a requirement to complete the task. Cooperation is however useful to improve the quality of the solutions and to solve conflicts on the use of the tools, as they are limited resources shared by all the participating agents.

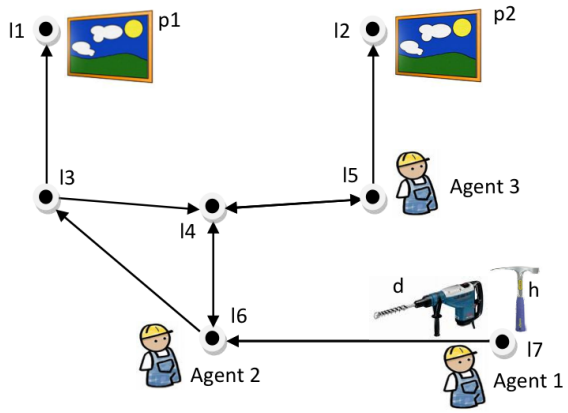

Figure 3.9: Picture domain example

3.3.10.2 Tests and results

The following subsections show the experimental results. We carried out two different tests[1]. The first one compares the quality of the solution plans obtained by a single agent and by a set of planning agents working together on the problem. To do so, we defined a set of MAP problems and the single-agent equivalent version. Finally, we measured the robustness and scalability of the MAP system by executing a planning problem several times, increasing each time the number of planning agents in the system.

Multi-Agent vs. Single-Agent Planning. This first set of tests compares the quality of the solution plans of our MAP approach versus the centralized single-agent framework. The testbed includes twenty planning problems (ten problems per domain) of increasing difficulty.

As stated in subsection 3.3.10.1, the *transportation* problems present a high

[1] All the tests were performed on a single machine with a 2.83 GHz Intel Core 2 Quad CPU and 8 GB RAM.

coupling level because agents are required to interact between each other to solve most of the problem goals. In contrast, agents in the *picture* problems can solve the goals independently in most cases so the coupling level in these problems is rather low. Another key difference between both domains is that planning agents in the *picture* domain (workers) are also the entities that execute the plans, whereas agents in the *transportation* domain are merely planning entities. This way, given two parallel actions in a plan of the picture domain, each one is associated to a different agent (worker) whereas two parallel actions in a plan of the transportation domain can be associated to two different trucks of the same transport agent, which is the planning entity. In other words, concurrency is associated to the agents in the *picture* domain and to the resources managed by the agents (trucks, hoists, etc.) in the *transportation* domain.

Table 3.5 shows the obtained results. $\#Ag$ indicates the number of agents that perform the planning problem in the MAP tests. $\#Actions$ and $\#TS$ refer to the number of actions and *time steps* of the solution plan, respectively (notice that we do not count the plans' fictitious actions). Finally, *Parallelism* indicates the maximum number of parallel branches in the MAP solution plans.

Time steps are the number of time units necessary to execute the plan, i.e., the duration of the plan. For instance, Figure 3.10 depicts the solution plan for the Picture2 MAP problem. Although the plan is composed of twelve planning actions (without taking into account the two fictitious actions), it can be executed in only eight time steps, since most of its actions can be executed in two parallel branches. Then, the duration of the plan in Figure 3.10 is 8 time units.

Discussion on the results. In the *transportation* domain, the MAP approach obtains the same results than the single-agent approach w.r.t. the number of actions and time steps. The single-agent approach performs rather well in this

particular domain, obtaining good-quality solutions, if not optimal, for almost all the tested problems. Notice that the single-agent approach features a single planning entity that has a full visibility on the planning problem. Despite the fact that the participating agents on the MAP tests have an incomplete view of the problem, the results show that MAP agents cooperate effectively, obtaining plans of the same quality as the single-agent approach, both in terms of number of actions and plan duration (time steps).

In the *transportation* domain, planning agents have a set of resources at their disposal (truck and hoists) to execute the actions of the plan. Since partial-order planners allow for parallelism, both the MAP and single-agent plans contain parallel actions. Actions in this domain are executed by the trucks and hoists instead of the planning agents themselves. Hence, the number of parallel branches and the duration of the solution plans of this domain is only conditioned by the number of available resources (trucks and hoists). For this reason, both approaches give rise to plans with the same number of actions and time steps. On the basis of these results, we can affirm that the quality of the MAP plans is not diminished by the limited view and *incomplete information* of the agents and the existence of private information among agents.

The results of the *picture* domain present more differences between both approaches. The single-agent approach obtains sequential plans because the single planning agent is also the only execution entity. MAP, however, takes advantage of having several planning/execution agents cooperating. MAP enforces cooperation as agents can work together to reach an objective. For instance, Figure 3.10 shows that an agent can pick up a tool and pass it on to another agent. This cooperation improves the solution because it prevents the agent from going for the tool and retrace its steps, thus reducing the number of actions of the plan. Agents also cooperate by proposing different parts of the plan that can be executed concurrently,

which reduces the duration of the plans with respect to the centralized approach. Table 3.5 shows that all the MAP solution plans for the *picture* domain include at least two parallel branches of actions, meaning that at least two agents work concurrently, which improves the quality of the solutions as shown in Table 3.5.

Problem	Multi-Agent Planning				Single-Agent Planning	
	#Ag	#Actions	#TS	Parallelism	#Actions	#TS
Transportation1	2	14	11	2	14	11
Transportation2	2	11	9	2	11	9
Transportation3	3	9	5	2	9	5
Transportation4	3	11	6	2	11	6
Transportation5	4	13	6	3	13	6
Transportation6	4	11	5	3	11	5
Transportation7	5	10	8	2	10	8
Transportation8	5	15	9	3	15	9
Transportation9	6	11	5	3	11	5
Transportation10	6	17	10	3	17	10
Picture1	2	11	6	2	14	14
Picture2	2	12	8	2	11	11
Picture3	3	6	2	3	8	8
Picture4	3	11	7	2	11	11
Picture5	4	8	2	4	11	11
Picture6	4	10	6	2	10	10
Picture7	5	8	5	2	8	8
Picture8	5	10	2	5	14	14
Picture9	6	9	5	2	9	9
Picture10	6	12	2	6	17	17

Table 3.5: Single-Agent vs. Multi-Agent Planning comparison

In conclusion, while being a more costly approach (see next subsection for scalability tests), MAP obtains equal or better solution plans in terms of both number of actions and duration of the plans than the single-agent model. We have

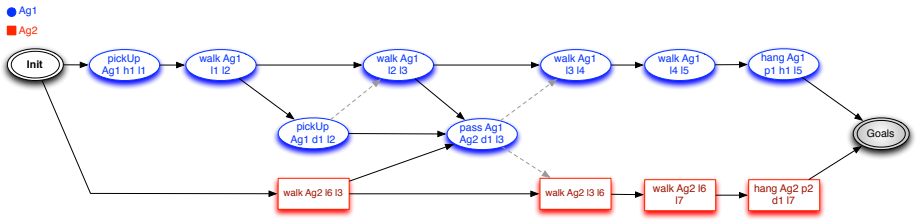

Figure 3.10: Solution plan for the Picture2 MAP problem

shown that MAP promotes cooperation among agents thus improving the quality of the solution. In addition, MAP agents manage their *incomplete information* on the MAP task efficiently as the quality of the solution plans is not affected, being at least on par with the single-agent approach. Moreover, results show that our approach obtains good-quality solution plans for problems with different coupling and complexity levels, from *loosely-coupled* to *strongly-related* problems.

Scalability analysis. In this subsection we evaluate the scalability of our MAP framework, i.e., how the number of agents in the MAP system affects its efficiency. To do so, eight different test problems were generated for both the *transportation* and the *picture* domains. Each test increases the number of agents by one, keeping the rest of the planning problem's parameters unchanged.

All the *transportation* tests include ten different cities, one truck, one empty table in the warehouse and one package of raw material. All the problems include a single warehouse agent, and each of them adds an extra transport agent, up to eight transport agents. The problem goal for all the test problems is to deliver the raw material to the warehouse, which must place it on the empty table. The optimal solution plan for all the problems includes ten actions and involves the participation of at least one transport agent and the warehouse agent.

As for the *picture* domain, all the test problems include two different tools and

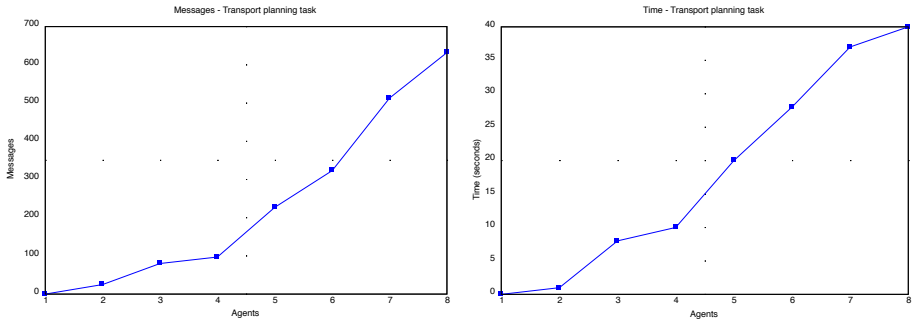

Figure 3.11: Scalability results for the *transportation* domain

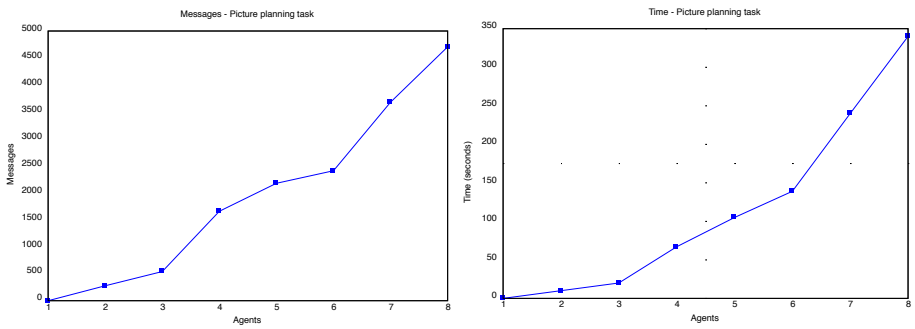

Figure 3.12: Scalability results for the *picture* domain

twelve different locations. The goal for all the problems is to hang two different pictures. The optimal solution plan for these problems has eight actions and involves the participation of two different agents. Each agent picks up one tool and hangs one picture.

Figures 3.11 and 3.12 depict the results for each domain. As it can be observed, the number of messages experiences a notable increase with each new agent included in the MAP process. So does the execution time, which is conditioned by the number of messages exchanged among agents.

Discussion on the results. These results are caused by the growing number of refinement plans proposed by the agents. Refinement plans are communicated to all the agents in the MAP system, reason why the addition of a planning agent represents such an overhead as each new agent proposes and communicates a number of extra refinement plans. In addition, the refinement plans proposed by each new planning agent increase the complexity of the search tree as they may also be adopted as base plans at some point.

Notice that the number of messages is much larger in the case of the *picture* problems, even though we have defined similar size and complexity problems for the two planning domains. This is due to the *loosely-coupled* nature of the *picture* problems because agents in this domain share the same abilities and every agent can make a plan proposal over any base plan.

As opposite to the *picture* domain, agents in the *transportation* domain are specialized, which makes them unable to make plan refinements over every base plan. Transport agents are limited by their working areas, while warehouse agents cannot take part in the transportation of the packages. This fact limits the number of exchanged messages, which also benefits the execution time. This way, our system proves to be more stable when solving *strongly-related* problems like the *transportation* tests since the addition of a new agent causes a lower increase in the number of messages, which directly affects the execution time.

In conclusion, the number of agents in the MAP system is a parameter that has a significant influence on its efficiency because the number of messages among agents constitutes one of the bottlenecks of the system. This issue is more noticeable when dealing with *loosely-coupled* problems, as agents can devise plan proposals over almost any base plan, whereas our MAP system shows a more robust behavior when solving *strongly-related* problems. Therefore, our immediate challenge is to reduce the number of messages between agents. This way, we will

improve the scalability of the system and we will be able to test more complex planning problems.

3.3.11 Conclusions

This article presents a MAP model that allows agents to plan under *incomplete information*. Our approach is suitable to solve a wide range of MAP problems, from *strongly-related* problems with a high degree of interaction among agents to simpler *loosely-coupled* problems, which present limited interactions among agents. Our model allows for heterogeneous agents with different information, capabilities and private goals to cooperatively build a joint plan while handling an incomplete view of the MAP task. Agents keep their private data and share only the relevant information for their interactions with other agents, thus being unaware of part of the information managed by the rest of agents.

Shareable information is defined through our MAP language, extended from *PDDL3.1*. The information exchange is carried out through the construction of a distributed Relaxed Planning Graph, by which agents share the public fluents and estimate the best cost to achieve them.

The MAP resolution process is based on a refinement planning procedure whereby agents propose successive refinements to an initially empty base plan until a consistent joint plan is obtained. This procedure, that iteratively combines planning and coordination, uses single-agent planning technology to build the refinement plans. More precisely, we adapt the POP paradigm to a MAP context, which allows agents to build refinement plans leaving details unresolved that will be gradually completed by other agents until a solution plan is found.

Conclusions drawn from the experiments show that MAP agents obtain solution plans of equal or better quality than a single-agent approach for both *loosely-coupled* and *strongly-related* problems. Despite agents do not have a complete view

of the MAP task and keep private information, the quality of the MAP solution plans is not affected, neither in terms of number of actions nor plan duration. Hence, we can affirm that our model tackles large MAP tasks in which information is distributed among a number of planning entities at least as effectively as a single-agent planning approach working under complete information.

Moreover, our MAP approach enforces cooperation among agents since they work together to solve goals more efficiently. MAP improves plan concurrency as agents can solve different goals in parallel, which reduces the duration and the number of actions of the solution plans.

3.4 FMAP: distributed cooperative multi-agent planning

Paper data

Authors:	A. Torreño, E. Onaindia, Ó. Sapena
Journal:	Applied Intelligence
Volume (number):	41(2)
Pages:	606-626
Impact factor (2012):	1.853 (first third)
Year:	2014

Abstract *This paper proposes FMAP (Forward Multi-Agent Planning), a fully-distributed multi-agent planning method that integrates planning and coordination. Although FMAP is particularly aimed at solving problems that require cooperation among agents, the flexibility of the domain-independent planning model allows FMAP to tackle any type of multi-agent planning tasks. In FMAP, agents jointly explore the plan space by building up refinement plans through a complete and flexible forward-chaining partial-order planner. Search is guided by h_{DTG}, a novel heuristic function based on the concepts of Domain Transition Graph and frontier state, and optimized to evaluate plans in distributed environments. Agents in FMAP apply an advanced privacy model that allows them to effectively keep private information while communicating only the data of the refinement plans that is relevant to each of the participating agents. Experimental results show that FMAP is a general-purpose approach that efficiently solves tightly-coupled domains with specialized agents and cooperative goals as well as loosely-coupled problems. In particular, the empirical evaluation shows that FMAP outperforms current MAP systems at solving complex planning tasks adapted from the International Planning Competition benchmarks.*

3.4.1 Introduction

Multi-agent planning (MAP) introduces a social approach to planning by which multiple intelligent entities work together to solve planning tasks that they are not able to attain by themselves, or at least they can accomplish them better by cooperating (24). MAP puts the focus on the collective effort of multiple agents to accomplish tasks by combining their knowledge and capabilities.

The complexity of solving a MAP task directly depends on its typology. In order to illustrate the features of a MAP task, let us introduce a brief application example.

Example 3. *Consider the transportation task in Figure 3.13, which involves three different agents: two transport agencies, $ta1$ and $ta2$, each having a truck ($t1$ and $t2$, respectively), that work in two different geographical areas, $ga1$ and $ga2$, respectively; and a factory, f, which is placed in the area $ga2$. To manufacture products, the factory f requires raw materials that are gathered from area $ga1$. In this task, $ta1$ and $ta2$ share the same abilities but act in different areas; i.e. they are spatially distributed agents. Additionally, the factory agent f is functionally different to $ta1$ and $ta2$. The goal of this task is for f to manufacture a set of final products. In order to carry out the task, $ta1$ will send its truck $t1$ to load the raw materials, rm, located in $l2$, and transport them to a storage facility, sf, placed in the intersection of both geographical areas. Then, $ta2$ will complete the delivery by using its truck $t2$ to transport the materials from sf to f, which will in turn manufacture the final products. This task involves thereby three specialized agents that are spatially and functionally distributed and should cooperate to accomplish a common goal.*

Example 3 emphasizes most of the key elements of a MAP task. First, the spatial and/or functional distribution of planning agents gives rise to *specialized agents* that have different knowledge and capabilities. In turn, this information distribution stresses the issue of *privacy*, which is one of the basic aspects that should be considered in multi-agent applications (101).

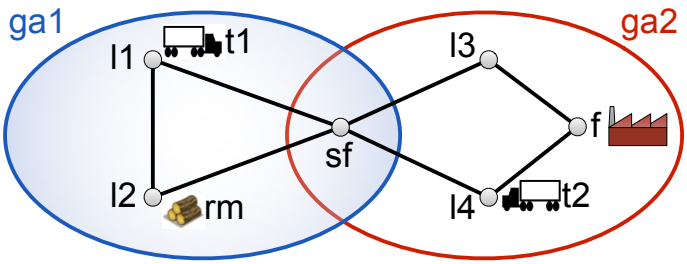

Figure 3.13: Example transportation task

As the three parties involved in Example 3 are agents specialized in different functional or geographical areas of the task, most of the information managed by the factory f is not relevant for the transport agencies and vice-versa, and the same can be applied to the transport agencies $ta1$ and $ta2$. Additionally, agents might not be willing to share the sensitive information on their internal procedures with the others. For instance, $ta1$ and $ta2$ are cooperating in this particular delivery task but they might be potential competitors as they work on the same business area. Then, agents in a MAP context want to minimize the information they share with each other, either for strategic reasons or simply because it is not relevant for the rest of the agents to address the planning task.

In addition to the need for a computational or information distribution, privacy is also one of the reasons to adopt a multi-agent approach. This aspect, however, has been traditionally relegated in MAP, particularly by the planning community (119). While some approaches define a basic notion of privacy (11, 80), others allow agents to share detailed parts of their plans, or do not consider private information at all (67).

The complexity of a MAP task is often described by means of its *coupling level* (14), measured as the number of interactions that arise among agents during the resolution of a MAP task. According to this parameter, MAP tasks can be clas-

sified into *loosely-coupled tasks*, that present few interactions among agents, and *tightly-coupled tasks*, which involve many interactions among agents. The coupling level, however, does not take into consideration one key aspect of MAP tasks: the presence of *cooperative goals*; that is, goals that cannot be solved individually by any agent as they require the cooperation of specialized agents. Example 3 illustrates a tightly-coupled task with one of such goals, as none of the agents can achieve the manufacturing of the final products by itself. Instead, they must make use of their specialized capabilities and interact with each other to deliver the raw materials and manufacture the final products.

In this paper we present FMAP (Forward MAP), a domain-independent MAP system designed to cope with a great variety of planning tasks of different complexity and coupling level. FMAP is a fully distributed method that interleaves planning and coordination by following a cooperative refinement planning approach. This search scheme allows us to efficiently coordinate agents' actions in any type of planning task (either loosely-coupled or tightly-coupled) as well as to handle cooperative goals.

FMAP relies on a theoretical model which defines a more sophisticated notion of privacy than most of the existing MAP systems. Instead of using a single set of private data, FMAP allows agents to declare the information they will share with each other. For instance, the transport agency $ta2$ in Example 1 will share with factory f information which is likely to be different from the one shared with agent $ta1$. Our system enhances privacy by minimizing the information that agents need to disclose. FMAP is a complete and reliable planning system which shows to be very competitive when compared to other state-of-the-art MAP systems. The experimental results will show that FMAP is particularly effective for solving tightly-coupled MAP problems with cooperative goals.

This article is organized as follows: the next section presents some related work on multi-agent planning, with an emphasis on issues like the coupling level of planning tasks, privacy or cooperative goals. Section 3.4.3 formalizes the notion of a MAP task; section 3.4.4 describes the main components of FMAP, the search procedure and the DTG-based heuristic function; finally, section 3.4.5 provides a thorough experimental evaluation of FMAP and section 3.4.6 concludes the paper.

3.4.2 Related work

In the literature, we can find two main approaches to solve MAP tasks like the one described in Example 3. *Centralized* MAP implies using an intermediary agent that has a complete knowledge of the task. The *distributed* approach allows agents to perform planning by themselves, interacting with each other and coordinating their local solutions, if necessary. The adoption of a centralized approach is aimed at improving the planner performance by taking advantage of the inherent structure of the MAP tasks (22, 67). Centralized approaches assume a single planning entity which has a complete knowledge of the task, which is rather unrealistic if the parties involved in the task have sensitive private information that they are not willing to disclose (99). In Example 3, the three agents involved in the task want to protect the information regarding their internal processes and business strategies, so a centralized setting is not an acceptable solution.

We then focus on fully distributed MAP, that is, the problem of coordinating agents in a shared environment where information is distributed. The distributed MAP setting involves two main tasks: the *planning* of local solutions and the *coordination* of the agents' plans into a global solution. Coordination can be performed at one or various stages of the distributed resolution of a MAP task. Some techniques are used for problems in which agents build local plans for the individual goals they have been assigned. MAP is about coordinating the local

plans of agents so as to mutually benefit by avoiding duplicating effort. In this case, the goal is not to build a joint plan among entities that are functionally or spatially distributed but to apply *plan merging* to coordinate local plans of multiple agents that are capable to achieve the problem goals by themselves (20).

There is a large body of work on plan-merging techniques. The work in (20) introduces a distributed coordination framework based on partial-order planning that addresses the interactions emerging between the agents' local plans. This framework, however, does not consider privacy in agents. The proposal in (113) is based on the iterative revision of the agents' local plans. Agents in this model cooperate by mutually adapting their local plans, with a focus on improving their common or individual profit. This approach also ignores privacy and agents are assumed to be fully cooperative. The approach in (120) uses multi-agent plan repair to solve inconsistencies among the agents local plans while keeping privacy. μ-SATPLAN (28) extends a satisfiability-based planner to coordinate the agents' local plans by studying positive and negative interactions among them.

Plan-merging techniques are not particularly well suited to cope with tightly-coupled tasks as they may introduce exponentially many ordering constraints in problems which require a big coordination effort (20). In general, plan merging is not an effective method for attaining cooperative goals, as this resolution scheme generally assumes that each agent is able to solve a subset of the task's goals by itself. However, some approaches use plan merging to coordinate local plans of specialized agents. In this case, the effort is put on discovering the interaction points among agents through the public information they share. For instance, Planning First (80) introduces a cooperative MAP approach for loosely-coupled tasks, in which specialized agents carry out planning individually through a state-based planner. The resulting local plans are then coordinated by solving a distributed

Constraint Satisfaction Problem (CSP) (56). This combination of CSP and planning to solve MAP tasks was originally introduced by the MA-STRIPS framework (14).

Another major research trend in MAP interleaves planning and coordination, providing a more unified vision of cooperative MAP. One of the first approaches to domain-independent MAP is the Generalized Partial Global Planning (GPGP) framework (68). Agents in GPGP have a partial view of the world and communicate their local plans to the rest of agents, which in turn merge this information into their own partial global plan in order to improve it. Approaches to continual planning, interleaving planning and execution in a world under continual change, assume there is uncertainty in the world state and thereby agents do not have a complete view of the world (15). Particularly, in (15), agents have a limited knowledge of the environment and limited capabilities but the authors do not explicitly deal with a functional distribution among agents or cooperative goals. TFPOP is a fully centralized approach that combines temporal and forward-chaining partial-order planning to solve loosely-coupled MAP tasks (67). The Best-Response Planning algorithm departs from an initial joint plan built through the Planning First MAP system (80), and iteratively improves the quality of this initial plan by applying cost optimal planning (58). Agents can only access the public information of the other agents' plans, thus preserving privacy, and they optimize their plans with the aim to converge to a Nash equilibrium regarding their preferences. MAP-POP is a fully distributed method that effectively keeps agents privacy (114, 115). Agents in MAP-POP perform an incomplete partial-order planning search to progressively develop and coordinate a joint plan until its completion.

Finally, MAPR is a recent planner that performs goal allocation to each agent (11). Agents iteratively solve the assigned goals by extending the plan of the previous agent. In this approach, agents work under limited knowledge of the

environment by obfuscating the private information in their plans. MAPR is particularly effective for loosely-coupled problems, but it cannot deal with tasks that feature specialized agents and cooperative goals as it assumes that each goal is achieved by a single agent. Section 3.4.5 will show a comparative performance evaluation between MAPR and our proposed approach FMAP.

3.4.3 MAP task formalization

Agents in FMAP work under a limited knowledge of the planning task by assuming that information not represented in an agent's model is unknown to the agent. The states of the world are modeled through a finite set of *state variables*, $\mathcal{V}$, each of them associated to a finite domain, $\mathcal{D}_v$, of mutually exclusive values that refer to the objects in the world. Assigning a value d to a variable $v \in \mathcal{V}$ generates a *fluent*. A *positive fluent* is a tuple $\langle v, d \rangle$, which indicates that the variable v takes the value d. A *negative fluent* is of the form $\langle v, \neg d \rangle$, indicating that v does not take the value d. A *state* S is a set of positive and negative fluents.

An *action* is a tuple $\alpha = \langle PRE(\alpha),\ EFF(\alpha) \rangle$, where $PRE(\alpha)$ is a finite set of fluents that represents the preconditions of α, and $EFF(\alpha)$ is a finite set of positive and negative *variable assignments* that model the effects of α. Executing an action α in a world state S leads to a new world state S' as a result of applying $EFF(\alpha)$ over S. An effect of the form $(v = d)$ assigns the value d to the variable v, i.e. it adds the fluent $\langle v, d \rangle$ to S', as well as a set of fluents $\langle v, \neg d' \rangle$ for each other value d' in the variable domain, in order to have a consistent state representation. Additionally, any fluent in S of the form $\langle v, \neg d \rangle$ or $\langle v, d'' \rangle$, $d'' \neq d$, is removed in state S'. This latter modification removes any fluent that contradicts $\langle v, d \rangle$. On the other hand, an assignment $(v \neq d)$ adds the fluent $\langle v, \neg d \rangle$ to S' and removes $\langle v, d \rangle$ from S', if such a fluent exists in S.

For instance, let us suppose that the transportation task in Example 3 includes a variable *pos-rm* that describes the position of the raw materials rm, which can be any of the locations in the task. Let S be a state that includes a fluent $\langle \textit{pos-rm}, l2 \rangle$, which indicates that rm is placed in its initial location (see Figure 3.13). Agent $ta1$ performs an action to load rm into its truck $t1$, which includes an effect of the form ($\textit{pos-rm} = t1$). The application of this action results in a new world state S' that will include a fluent $\langle \textit{pos-rm}, t1 \rangle$ and fluents of the form $\langle \textit{pos-rm}, \neg l \rangle$ for each other location $l \neq t1$; the fluent $\langle \textit{pos-rm}, l2 \rangle$ will no longer be in S'.

Definition 3.13. *A **MAP task** is defined as a tuple $\Im_{MAP} = \langle \mathcal{AG}, \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$. $\mathcal{AG} = \{1, \ldots, n\}$ is a finite non-empty set of agents. $\mathcal{V} = \bigcup_{i \in \mathcal{AG}} \mathcal{V}^i$, where $\mathcal{V}^i$ is the set of state variables known to an agent i. $\mathcal{I} = \bigcup_{i \in \mathcal{AG}} \mathcal{I}^i$ is a set of fluents that defines the initial state of $\Im_{MAP}$. As specialized agents are allowed, they may only know a subset of $\mathcal{I}$. Given two agents i and j, $\mathcal{I}^i \cap \mathcal{I}^j$ may be $\emptyset$ or not; in any case, the initial states of the agents never contradict each other. $\mathcal{G}$ is the set of goals of $\Im_{MAP}$, i.e., the values of the state variables that agents have to achieve to accomplish $\Im_{MAP}$. Finally, $\mathcal{A} = \bigcup_{i \in \mathcal{AG}} \mathcal{A}^i$ is the set of planning actions of the agents. $\mathcal{A}^i$ and $\mathcal{A}^j$ of two specialized agents i and j will typically be two disjoint sets as the agents have their own different capabilities; otherwise, $\mathcal{A}^i$ and $\mathcal{A}^j$ may overlap. $\mathcal{A}$ includes two fictitious actions α_i and α_f that do not belong to the action set of any particular agent: α_i represents the initial state of $\Im_{MAP}$, i.e., $PRE(\alpha_i) = \emptyset$ and $EFF(\alpha_i) = \mathcal{I}$, while α_f represents the global goals of $\Im_{MAP}$, i.e., $PRE(\alpha_f) = \mathcal{G}$, and $EFF(\alpha_f) = \emptyset$.*

As discussed in Example 3, our model considers specialized agents that can be functionally and/or spatially distributed. This specialization defines the local *view* that each agent has of the MAP task. Local views are a typical characteristic of multi-agent systems and other distributed systems. For instance, distributed CSPs use local views, such that agents only receive information on the constraints in which they are involved (56, 125). Next, we define the information of an agent i on a planning task $\Im_{MAP}$.

The *view* of an agent i on a MAP task $\mathcal{T}_{MAP}$ is defined as $\mathcal{T}^i_{MAP} = \langle \mathcal{V}^i, \mathcal{A}^i, \mathcal{I}^i, \mathcal{G} \rangle$. $\mathcal{V}^i$ is the set of state variables known to agent i; $\mathcal{A}^i \subseteq \mathcal{A}$ is the set of its capabilities (planning actions); $\mathcal{I}^i$ is the subset of fluents of the initial state $\mathcal{I}$ that are visible to agent i; and $\mathcal{G}$ is the set of global goals of $\mathcal{T}_{MAP}$. Since agents in FMAP are fully cooperative, they are all aware of the global goals of the task. Obviously, because of specialization, a particular agent may not understand the goals as specified in $\mathcal{G}$; defining $\mathcal{G}$ as global goals implies that all agents contribute to the achievement of $\mathcal{G}$, either directly (achieving a $g \in \mathcal{G}$) or indirectly (introducing actions whose effects help other agents achieve g).

The state variables of an agent i are determined by the view it has on the initial state, I^i, the planning actions it can perform, A^i, and set of goals of $\mathcal{T}_{MAP}$. This also affects the domain D_v of a variable v. We define $\mathcal{D}^i_v \subseteq \mathcal{D}_v$ as the set of values of the variable v that are known to agent i.

Consider again the *pos-rm* variable in Example 3. The domain of *pos-rm* contains all the locations in the transportation task, including the factory f, the storage facility sf and the trucks; that is, $\mathcal{D}_{pos\text{-}rm} = \{l1, l2, l3, l4, f, sf, t1, t2\}$. However, agents $ta1$ and $ta2$ have local knowledge about the domain of *pos-rm* as some of the values of such variable refer to objects of $\mathcal{T}_{MAP}$ that are unknown to them. Hence, $ta1$ will manage $\mathcal{D}^{ta1}_{pos\text{-}rm} = \{l1, l2, sf, t1\}$, while $ta2$ will manage $\mathcal{D}^{ta2}_{pos\text{-}rm} = \{l3, l4, sf, f, t2\}$.

Agents in FMAP interact with each other by sharing information on their state variables. For each pair of agents i and j, the public information they share is defined as $\mathcal{V}^{ij} = \mathcal{V}^{ji} = \mathcal{V}^i \cap \mathcal{V}^j$. Additionally, some of the values in the domain of a variable can also be public to both agents. The set of values of a variable v that are public to a pair of agents i and j is defined as $\mathcal{D}^{ij}_v = \mathcal{D}^i_v \cap \mathcal{D}^j_v$.

Back to Example 3, the *pos-rm* variable is public to agents $ta1$ and $ta2$. The values that are public to both agents are defined as the intersection of the known

values to each of them, $\mathcal{D}_{pos\text{-}rm}^{ta1\ ta2} = \{sf\}$. This way, the only public location of rm to agents $ta1$ and $ta2$ is the storage facility sf, which is precisely the intersection between both geographical areas. Hence, if agent $ta1$ places rm in sf, it will inform $ta2$ accordingly, and vice versa. This allows agents $ta1$ and $ta2$ to work together while minimizing the information they share with each other.

Our MAP model is a multi-agent refinement planning framework, a general method based on the refinement of the set of all possible plans. The internal reasoning of agents in FMAP is configured as a Partial-Order Planning (POP) search procedure. Other local search strategies are applicable, as long as agents build *partial-order plans*. The following concepts and definitions are standard terms from the POP paradigm (44), which have been adapted to state variables. Additionally, definitions also account for the multi-agent nature of the planning task and the local views of the task by the agents.

Definition 3.14. *A **partial-order plan** or partial plan is a tuple* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$. $\Delta = \{\alpha | \alpha \in \mathcal{A}\}$ *is the set of actions in* Π. $\mathcal{OR}$ *is a finite set of ordering constraints* ($\prec$) *on* Δ. $\mathcal{CL}$ *is a finite set of causal links of the form* $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ *or* $\alpha \overset{\langle v,\neg d \rangle}{\rightarrow} \beta$, *where* α *and* β *are actions in* Δ. *A causal link* $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ *enforces precondition* $\langle v, d \rangle \in PRE(\beta)$ *through an effect* $(v = d) \in EFF(\alpha)$ *(44). Similarly, a causal link* $\alpha \overset{\langle v,\neg d \rangle}{\rightarrow} \beta$ *enforces* $\langle v, \neg d \rangle \in PRE(\beta)$ *through an effect* $(v \neq d) \in EFF(\alpha)$ *or* $(v = d') \in EFF(\alpha)$, $d' \neq d$.

An *empty* partial plan is defined as $\Pi_0 = \langle \Delta_0, \mathcal{OR}_0, \mathcal{CL}_0 \rangle$, where $\mathcal{OR}_0$ and $\mathcal{CL}_0$ are empty sets, and Δ_0 contains only the fictitious initial action α_i. A partial plan Π for a task $\mathcal{T}_{MAP}$ will always contain α_i.

The introduction of new actions in a partial plan may trigger the appearance of *flaws*. There are two types of flaws in a partial plan: preconditions that are not yet solved (or supported) through a causal link, and threats. A *threat* over a causal link $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ is caused by an action γ that is not ordered w.r.t. α or β and might

potentially modify the value of v (44) ($(v \neq d) \in EFF(\gamma)$ or $(v = d') \in EFF(\gamma)$, $d' \neq d$), making the causal link unsafe. Threats are addressed by introducing either an ordering constraint $\gamma \prec \alpha$ (this is called *demotion* because the causal link is posted after the threatening action) or an ordering $\beta \prec \gamma$ (this is called *promotion* because the causal link is placed before the threatening action) (44).

A *flaw-free* plan is a threat-free partial plan in which the preconditions of all the actions are supported through causal links.

Planning agents in FMAP cooperate to solve MAP tasks by progressively refining an initially empty plan Π until a solution is reached. The definition of *refinement plan* is closely related to the internal forward-chaining partial-order planning search performed by the agents. Refinement planning is a technique widely used by many planners, specifically in anytime planning, where a first initial solution is progressively refined until the deliberation time expires (98). We define a refinement plan as follows:

Definition 3.15. *A **refinement plan** $\Pi_r = \langle \Delta_r, \mathcal{OR}_r, \mathcal{CL}_r \rangle$ over a partial plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$, is a flaw-free partial plan which extends Π, i.e., $\Delta \subset \Delta_r$, $\mathcal{OR} \subset \mathcal{OR}_r$ and $\mathcal{CL} \subset \mathcal{CL}_r$. Π_r introduces a new action $\alpha \in \Delta_r$ in Π, resulting in $\Delta_r = \Delta \cup \alpha$. All the preconditions in $PRE(\alpha)$ are linked to existing actions in Π through causal links; i.e., all preconditions are supported and so it holds $\forall p \in PRE(\alpha)$, $\exists\ \beta \xrightarrow{p} \alpha \in \mathcal{CL}_r$, where $\beta \in \Delta$.*

Refinement plans in FMAP include actions that can be executed in parallel by different agents. Some MAP models consider that two parallel or non-sequential actions are mutually consistent if none of them modifies the value of a state variable that the other relies on or affects (15). We also consider that the preconditions of two mutually consistent actions have to be consistent (12). Hence, two non-sequential actions $\alpha \in \mathcal{A}^i$ and $\beta \in \mathcal{A}^j$ are *mutually consistent* if none of the following conditions holds:

- $\exists(v = d) \in EFF(\alpha)$ and $\exists(\langle v, d' \rangle \in PRE(\beta) \vee \langle v, \neg d \rangle \in PRE(\beta))$, where $v \in \mathcal{V}^{ij}$, $d \in \mathcal{D}_v^{ij}$, $d' \in \mathcal{D}_v^j$ and $d \neq d'$, or vice versa; that is, the effects of α and preconditions of β (or vice versa) do not contradict under the specified conditions.

- $\exists(v = d) \in EFF(\alpha)$ and $\exists((v = d') \in EFF(\beta) \vee (v \neq d) \in EFF(\beta))$, where $v \in \mathcal{V}^{ij}$, $d \in \mathcal{D}_v^{ij}$, $d' \in \mathcal{D}_v^j$ and $d \neq d'$, or vice versa; that is, the effects of α and effects of β (or vice versa) do not contradict under the specified conditions.

- $\exists\langle v, d \rangle \in PRE(\alpha)$ and $\exists(\langle v, d' \rangle \in PRE(\beta) \vee \langle v, \neg d \rangle \in PRE(\beta))$, where $v \in \mathcal{V}^{ij}$, $d \in \mathcal{D}_v^{ij}$, $d' \in \mathcal{D}_v^j$ and $d \neq d'$, or vice versa; that is, the preconditions of α and preconditions of β (or vice versa) do not contradict under the specified conditions.

Agents address parallelism by the resolution of threats over the causal links of the plan. Thus, consistency between any two non-sequential actions introduced by different agents is always guaranteed as refinement plans are flaw-free plans.

Finally, a *solution plan* for $\mathcal{T}_{MAP}$ is a refinement plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ that addresses all the global goals $\mathcal{G}$ of $\mathcal{T}_{MAP}$. A solution plan includes the fictitious final action α_f and ensures that all its preconditions (note that $PRE(\alpha_f) = \mathcal{G}$) are satisfied; that is, $\forall g \in PRE(\alpha_f)$, $\exists\ \beta \xrightarrow{g} \alpha_f \in \mathcal{CL}$, $\beta \in \Delta$, which is the necessary condition to guarantee that Π solves T_{MAP}.

3.4.3.1 Privacy in partial plans

Every time an agent i refines a partial plan by introducing a new action $\alpha \in \mathcal{A}^i$, it communicates the resulting refinement plan to the rest of the agents in $\mathcal{T}_{MAP}$. As stated above, the information that is public to a pair of agents is defined according to the common state variables and domain values. In order to preserve privacy,

agent i will only communicate agent j the fluents in action α whose variables are common to both agents. The information of a refinement plan Π that agent j receives from agent i configures its *view* of such plan, $view^j(\Pi)$. More particularly, given two agents i and j and a fluent $\langle v, d\rangle$, where $v \in \mathcal{V}^i$ and $d \in \mathcal{D}_v^i$ (equivalently for a negative fluent $\langle v, \neg d\rangle$), we distinguish the three following cases:

- **Public fluent**: if $v \in \mathcal{V}^{ij}$ and $d \in \mathcal{D}_v^{ij}$, the fluent $\langle v, d\rangle$ is public to both i and j, and thus, agent i will send agent j all the causal links, preconditions and effects regarding $\langle v, d\rangle$.

- **Private fluent to agent** i: if $v \notin \mathcal{V}^{ij}$, the fluent $\langle v, d\rangle$ is private to agent i w.r.t. agent j, and thus, agent i will occlude the preconditions and effects regarding $\langle v, d\rangle$ to agent j. Causal links of the form $\alpha \overset{\langle v,d\rangle}{\rightarrow} \beta$ will be sent to agent j as simply ordering constraints $\alpha \prec \beta$.

- **Partially private fluent to agent** i: if $v \in \mathcal{V}^{ij}$ but $d \notin \mathcal{D}_v^{ij}$, the fluent $\langle v, d\rangle$ is partially private to agent i w.r.t. agent j. Instead of $\langle v, d\rangle$, agent i will send agent j a fluent $\langle v, \bot\rangle$, where $\bot$ is the *undefined value*. Hence, preconditions of the form $\langle v, d\rangle$ will be sent as $\langle v, \bot\rangle$, effects of the form $(v = d)$ will be replaced by $(v = \bot)$ and causal links $\alpha \overset{\langle v,d\rangle}{\rightarrow} \beta$ will adopt the form $\alpha \overset{\langle v,\bot\rangle}{\rightarrow} \beta$.

If an agent j receives a fluent $\langle v, \bot\rangle$, $\bot$ is interpreted as follows: $\forall d \in \mathcal{D}_v^j,\ \langle v, \neg d\rangle$. That is, $\bot$ indicates that v is not assigned any of the values known to agent j (D_v^j). This mechanism is used to inform an agent that a resource is no longer available in its influence area. For instance, suppose that agent $ta2$ in Example 3 acquires the raw material rm from sf by loading it into its truck $t2$. Agent $ta2$ communicates $ta1$ that rm is no longer in sf, but agent $ta1$ does not know about the truck $t2$. To solve this issue, $ta2$ sends $ta1$ the fluent $\langle pos\text{-}rm, \bot\rangle$, meaning that

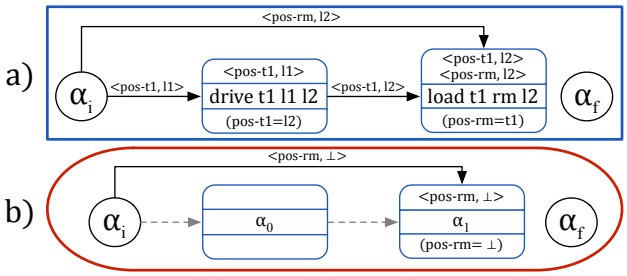

Figure 3.14: A refinement plan Π_r as viewed by: a) agent $ta1$ b) agent $ta2$

the resource rm is no longer available in the geographical area of agent $ta1$. Consequently, $ta1$ is now aware that rm is not located in any of its accessible positions $D^{ta1}_{pos\text{-}rm} = \{l1, l2, sf, t1\}$.

Figure 3.14 shows the view that transport agents $ta1$ and $ta2$ in Example 3 have of a simple refinement plan Π_r. In this plan, agent $ta1$ drives the truck $t1$ from $l1$ to $l2$ and loads rm into $t1$. As shown in Figure 3.14a), $view^{ta1}(\Pi_r)$ contains all the information of both actions in the plan as agent $ta1$ has introduced them. Agent $ta2$, however, does not know the truck $t1$, and hence, the variable $pos\text{-}t1$, which models the position of $t1$, is private to $ta1$ w.r.t. $ta2$. This way, all the preconditions and effects related to the fluents $\langle pos\text{-}t1, l1\rangle$ and $\langle pos\text{-}t1, l2\rangle$ are occluded in $view^{ta2}(\Pi_r)$ (see Figure 3.14b)). Additionally, the causal links regarding these two fluents are replaced by ordering constraints in $view^{ta2}(\Pi_r)$. On the other hand, the variable $pos\text{-}rm$ is public to both agents, but the *load* action refers to the locations $t1$ and $l2$, that are not in $\mathcal{D}^{ta2}_{pos\text{-}rm}$. Therefore, fluents $\langle pos\text{-}rm, l2\rangle$ and $\langle pos\text{-}rm, t1\rangle$ are partially private to agent $ta1$ w.r.t. $ta2$. This way, the precondition $\langle pos\text{-}rm, l2\rangle$ and the effect $(pos\text{-}rm = t1)$ of the *load* action are replaced by $\langle pos\text{-}rm, \bot\rangle$ and $(pos\text{-}rm = \bot)$ in $view^{ta2}(\Pi_r)$, respectively. The fluent $\langle pos\text{-}rm, l2\rangle$ is also replaced by $\langle pos\text{-}rm, \bot\rangle$ in the causal link $\alpha_i \overset{\langle pos\text{-}rm, l2\rangle}{\rightarrow}$ *load* $t1$ rm $l2$.

3.4.3.2 MAP definition language

There is a large body of work on languages to specify planning tasks. Since planning has been traditionally regarded as a centralized problem, the most popular definition languages, such as the different versions of *PDDL* (the Planning Domain Definition Language[1]), are designed to model single-agent planning tasks. MAP introduces a set of requirements that are not present in single-agent planning, such as privacy or specialized agents, which motivate the development of specification languages for multi-agent planning.

There are many different approaches to MAP as was described in section 3.4.2. *MA-STRIPS* (14), which was designed as a minimalistic extension to *STRIPS* (36), is one of the most common MAP languages. It allows to define a set of agents and associate the planning actions they can execute. FMAP presents several advanced features that motivated the definition of our own *PDDL*-based specification language (language syntax is detailed in (115)), rather than using *MA-STRIPS*.

As the world states in FMAP are modeled through state variables instead of predicates, our MAP language is based on *PDDL3.1* (64), the last version of *PDDL*. Unlike its predecessors, that model planning tasks through predicates, *PDDL3.1* incorporates state variables that map to a finite domain of objects of the task.

In a single-agent language, the user specifies the *domain* of the task (planning operators, types of objects, and functions) and the *problem* to be solved (objects of the task, initial state and goals). In FMAP, we define a domain and a problem file for each agent, which model, respectively, the typology of the agent and its local view of the MAP task. The domain files keep the structure of a regular *PDDL3.1* domain file. The problem files, however, are extended with an additional

[1] `http://en.wikipedia.org/wiki/Planning_Domain_Definition_Language`

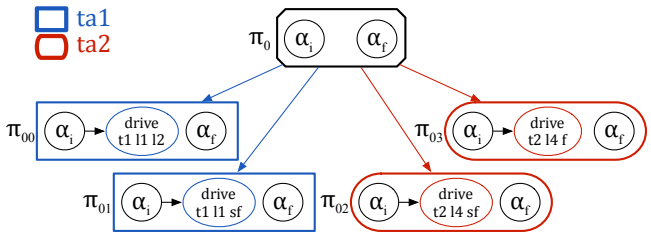

Figure 3.15: FMAP multi-agent search tree example

`:shared-data` section, which specifies the information that an agent can share with each other participating agent in the task.

3.4.4 FMAP refinement planning procedure

FMAP is based on a cooperative refinement planning procedure in which agents jointly explore a multi-agent plan-space search tree. A multi-agent search tree is one in which the partial plans of the nodes are built with the contributions of one or more agents.

Figure 3.15 shows the first level of the multi-agent search tree that would be generated for the transportation task of Example 3. At this level, agents $ta1$ and $ta2$ propose each two refinement plans, specifically plans to move their trucks within their geographical areas. In each of these refinement plans, the agent adds one action and the corresponding orderings and causal links. Agent f does not contribute here with any refinement plan because the initial empty plan Π_0 does not comprise the necessary supporting information for f to insert any of its actions. In a subsequent iteration (expansion of the next tree node), agents can in turn create new refinement plans. For instance, if node Π_{00} in Figure 3.15 is selected next for expansion, the three agents in the problem ($ta1$, $ta2$ or f) will try to create refinement plans over Π_{00} by adding one of their actions and supporting it through the necessary causal links and orderings.

Agents keep a copy of the multi-agent search tree, storing the local view they have of each of the plans in the tree nodes. Given a node Π in the multi-agent search tree, an agent i maintains $view^i(\Pi)$ in its copy of the tree.

FMAP applies a multi-agent A* search that iteratively explores the multi-agent tree. One iteration of FMAP involves: 1) agents select one of the unexplored leaf nodes of the tree for expansion; 2) agents expand the selected plan by generating all the refinement plans over this node; and 3) agents evaluate the resulting successor nodes and communicate the results to the rest of agents. Instead of using a broadcast control framework, FMAP uses a democratic leadership, in which a coordinator role is scheduled among the agents. One of the agents adopts the role of coordinator at each iteration, thus leading the procedure in one iteration (initially, the coordinator role is randomly assigned to one of the participating agents). More specifically, a FMAP iteration is as follows:

- **Base plan selection**: Among all the open nodes (unexplored leaf nodes) of the multi-agent search tree, the coordinator agent selects the most promising plan, Π_b, as the base plan to refine in the current iteration. Π_b is selected according to the evaluation of the open nodes (details on the node evaluation and selection are in section 3.4.4.3). In the initial iteration, the base plan is the empty plan Π_0.

- **Refinement plan generation**: Agents expand Π_b and generate its successor nodes. A successor node is a refinement plan over Π_b that an agent generates individually through its embedded forward-chaining partial-order planner (see subsection 3.4.4.1).

- **Refinement plan evaluation**: Every agent i evaluates its refinement plans Π_r by applying a classical A* evaluation function ($f(\Pi_r) = g(view^i(\Pi_r)) + h(view^i(\Pi_r))$. $g(view^i(\Pi_r))$) stands for the number of actions of Π_r. Since

agents view all the actions of the plans (but not necessarily all its preconditions and effects), $g(view^i(\Pi_r))$ is equivalent to $g(\Pi_r)$. $h(view^i(\Pi_r))$ applies our DTG-based heuristic (see subsection 3.4.4.3) to estimate the cost of reaching a solution plan from Π_r.

- **Refinement plan communication**: Agents communicate their refinement plans to each other agent. The information that an agent i communicates of its plan Π_r to the rest of agents depends on the level of privacy specified with each of them. Along with the refinement plan Π_r, agent i communicates the result of the evaluation of Π_r, $f(\Pi_r)$.

Once the iteration is completed, the leadership is handed to another agent, which adopts the coordinator role, and a new iteration starts. The next coordinator agent selects the new base plan Π_b as the open node Π that minimizes $f(\Pi)$, and agents proceed to expand it. This iterative process carries on until Π_b becomes a solution plan that supports the final action α_f, or when all the open nodes have been visited, in which case, agents will have explored the complete search space without finding a solution for the MAP task $\mathcal{T}_{MAP}$.

A refinement plan Π is evaluated only by the agent that generates it. The agent communicates Π along with $f(\Pi)$ to the rest of agents. Therefore, the decision on the next base plan is not affected by the agent that plays the coordinator role, as all the agents manage the same $f(\Pi)$ value for every open node Π.

Back to the example depicted in Figure 3.15, agent $ta1$ evaluates its refinement plans, Π_{00} and Π_{01}, and communicates them to agents $ta2$ and f, along with $f(\Pi_{00})$ and $f(\Pi_{01})$; likewise, $ta2$ with $ta1$ and f. In this first level of the tree, agents $ta1$ and $ta2$ have a complete view of the refinement plans they have generated as these plans only contain an action introduced by themselves. However, when $ta1$ and $ta2$ communicate their plans to each other, they will only send

the fluents according to the level of privacy defined between them, as described in subsection 3.4.3.1. This way, $ta1$ will send $view^{ta2}(\Pi_{00})$ and $view^{ta2}(\Pi_{01})$ to agent $ta2$, and $view^{f}(\Pi_{00})$ and $view^{f}(\Pi_{01})$ to agent f.

The next subsections analyze the key elements of FMAP, that is, the search algorithm that agents use for the generation of the refinement plans and the heuristic function they use for plan evaluation. We also include a subsection that addresses the completeness and correctness of the algorithm as well as a subsection that describes the limitations of FMAP.

3.4.4.1 Forward-Chaining Partial-Order Planning

Agents in FMAP use an embedded flexible forward-chaining POP system, which will be referred to as FLEX in the following, to generate the refinement plans. As other approaches, FLEX explores the potential of forward search to support partial-order planning. OPTIC (6), for instance, combines partial-order structures with information on the *frontier state* of the plan. Informally speaking, the frontier state of the partial plan of a tree node is the resulting state after executing the actions in such a plan. Given a refinement plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$, we define its *frontier state* $FS(\Pi)$ as the set of fluents $\langle v, d \rangle$ achieved by actions $\alpha \in \Delta \mid \langle v, d \rangle \in EFF(\alpha)$, such that any action $\alpha' \in \Delta$ that modifies the value of the variable v $(\langle v, d' \rangle \in EFF(\alpha') \mid d \neq d')$ is not reachable from α by following the orderings and causal links in Π.

The only actions that OPTIC add to a plan are those whose preconditions hold in the frontier state. This behaviour forces OPTIC to some early commitments although this does not sacrifice completeness, as search can backtrack. Also, TFPOP (67) applies a centralized forward-chaining POP for multiple agents, keeping a sequential execution thread per agent.

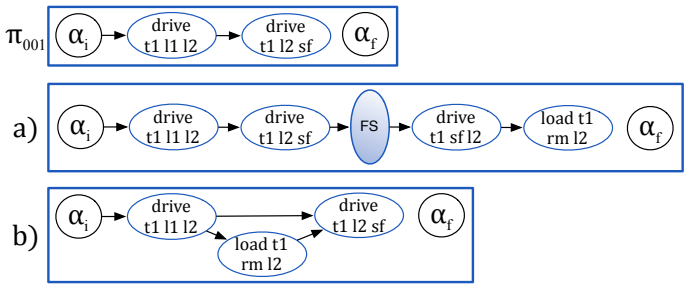

Figure 3.16: Loading rm in plan Π_{001}: a) inserting actions from a frontier state b) with FLEX

The aforementioned approaches only permit introducing actions that are applicable in the frontier state of the plan. In contrast, FLEX allows inserting actions at any position of the plan without assuming that any action in the plan has already been executed. This is a more flexible approach that is also more compliant with the least-commitment principle that typically guides backward-chaining POP. Figure 3.16 shows the advantages of our flexible search strategy. Consider the refinement plan Π_{001}, which is the result of a refinement of agent $ta1$ on plan Π_{00} (see Figure 3.15) after including the action $(drive\ t1\ l1\ sf)$. This is not the best course of action for taking the raw material rm to the factory f as $ta1$ should load rm into $t1$ before moving to sf. The frontier state $FS(\Pi_{001})$ reflects the state of the world after executing the plan Π_{001}, in which the truck $t1$ would be at sf. Planners like OPTIC would only introduce actions that are applicable in the frontier state $FS(\Pi_{001})$. In this example, OPTIC would insert first the action $(drive\ t1\ sf\ l2)$ to move the truck $t1$ back to $l2$ in order to apply then $(load\ t1\ rm\ l2)$ (see Figure 3.16a). FLEX, however, is able to introduce actions at any position in the plan, so the $load$ action can be directly placed between both $drive$ actions, thus minimizing the length of the plan (see Figure 3.16b).

Algorithm 6 summarizes the FLEX procedure invoked by an agent i to generate

refinement plans, and Figure 3.17 shows how agent $ta1$ in Example 3 uses the FLEX algorithm to refine plan Π_{00} on Figure 3.15. The first operation of an agent i that executes FLEX is to check whether the fictitious final action α_f is supportable in Π_b, that is, if a solution plan can be obtained from Π_b. If so, the agent will generate a set of solution plans that cover all the possible ways to support the preconditions of α_f through causal links.

Algorithm 6: FLEX search algorithm for an agent i

```
RP^i ← ∅
if potentiallySupportable(α_f, view^i(Π_b)) then
└return solutionPlans
CandidateActions ← ∅
for all α ∈ A^i do
    if potentiallySupportable(α, view^i(Π_b)) then
        CandidateActions ← CandidateActions ∪ α
for all α ∈ CandidateActions do
    Plans ← {view^i(Π_b)}
    repeat
        Select and extract Π_s ∈ Plans
        Flaws(Π_s) ← unsupportedPrecs(α, Π_s) ∪ Threats(Π_s)
        if Flaws(Π_s) = ∅ then
            RP^i ← RP^i ∪ Π_s
        else
            Select and extract Φ ∈ Flaws(Π_s)
            Plans ← Plans ∪ solveFlaw(Π_s, Φ)
    until Plans = ∅
return RP^i
```

In case a solution plan is not found, agent i analyzes all its planning actions $\mathcal{A}^i$ and estimates if they are supportable in Π_b. Given an action $\alpha \in \mathcal{A}^i$, the function $potentiallySupportable(\alpha, \Pi_b)$ checks if $\forall \langle v, d \rangle \in PRE(\alpha)$, $\exists \beta \in \Delta(\Pi_b) \mid (v =$

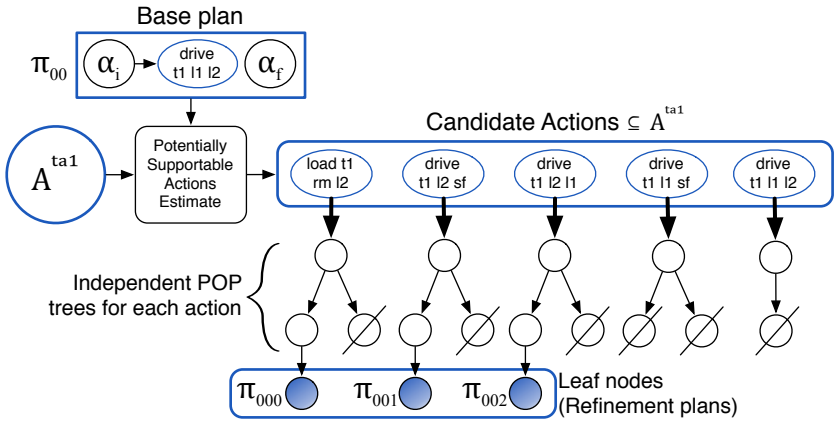

Figure 3.17: FLEX algorithm as applied by agent $ta1$ over plan Π_{00}

$d) \in EFF(\beta)$, i.e., the agent estimates that α is supportable if for every precondition of α there is a matching effect among the actions of Π_b.

In Figure 3.17, we can see an example of potentially supportable actions. Agent $ta1$ evaluates all the actions in $\mathcal{A}^{ta}$ and finds five candidate actions. In α_i, the initial state of Π_{00}, the truck $t1$ is at location $l1$. Consequently, $ta1$ considers $(drive\ t1\ l1\ sf)$ and $(drive\ t1\ l1\ l2)$ as potential candidate actions for its refinements. Notice that action $(drive\ t1\ l1\ l2)$ is already included in plan Π_{00}. Actions $(drive\ t1\ l2\ sf)$, $(drive\ t1\ l2\ l1)$ and $(load\ t1\ rm\ l2)$ are also classified as candidates, as they are applicable after the action $(drive\ t1\ l1\ l2)$ which is already in plan Π_{00}.

It is possible to introduce an action multiple times in a plan; for instance, a truck may need to travel back and forth between two different locations several times. For this reason, $ta1$ considers again $(drive\ t1\ l1\ l2)$ as a candidate action when refining Π_{00}, even if this action is already included in Π_{00}. By estimating potentially supportable actions in any position of the plan, FLEX follows the least commitment principle and does not leave out any potential refinement plan.

The *potentiallySupportable* procedure is an estimate because it does not actually check the possible flaws that arise when supporting an action. Hence, an agent analyzes the alternatives to support each candidate action α by generating a POP search tree for that particular action (*repeat* loop in Algorithm 6). All the leaf nodes of the tree (stored in the *Plans* list in Algorithm 6) are explored, thus covering all the possible ways to introduce α in Π_b.

As in backward-chaining POP, FLEX introduces the action α in Π_b by supporting its preconditions through causal links and solving the threats that arise during the search. The set of flaw-free plans obtained from this search are stored in RP^i as valid refinement plans of agent i over Π_b. This procedure is carried out for each candidate action. Completeness is guaranteed as all the possible refinement plans over a given base plan are generated by the agents involved in $\mathfrak{T}_{MAP}$.

Figure 3.17 shows that, for every candidate action, $ta1$ performs an independent POP search aimed at supporting the action. Actions ($load$ $t1$ rm $l2$), ($drive$ $t1$ $l2$ sf), and ($drive$ $t1$ $l2$ $l1$) lead to three different refinement plans over Π_{00}: $\{\Pi_{000}, \Pi_{001}, \Pi_{002}\}$. These plans will then be inserted in $ta1$'s copy of the multi-agent search tree. Agent $ta1$ will also send the information of these plans to agents $ta2$ and f according to the level of privacy defined with respect to them. $ta2$ and f also store the received plans in their copies of the tree.

Candidate action ($drive$ $t1$ $l1$ sf) does not produce valid refinement plans because it causes an unsolvable threat. This is because truck $t1$ cannot simultaneously move to two different locations from $l1$, which causes a conflict between the existing action ($drive$ $t1$ $l1$ $l2$) $\in \Delta(\Pi_{00})$ and ($drive$ $t1$ $l1$ sf). Similarly, action ($drive$ $t1$ $l1$ $l2$) does not yield any valid refinements. The resulting plan would have two actions ($drive$ $t1$ $l1$ $l2$) in parallel, both linked to α_i, which causes an unsolvable threat because $t1$ cannot perform two identical $drive$ actions in parallel.

3.4.4.2 Completeness and Soundness

As explained in the previous section, agents refine the base plan concurrently by analyzing all possible ways to support their actions in the base plan. Since this operation is done by every agent and for all their actions, we can conclude FMAP is a complete procedure that explores the whole search space.

As for soundness, a partial-order plan is sound if it is a flaw-free plan. The FLEX algorithm addresses inconsistencies among actions in a partial plan by detecting and solving threats.

When an agent i introduces an action α in a base plan Π, FLEX studies the threats that α causes in the causal links of Π and the threats that the actions of Π may cause in the causal links that support the preconditions of α. In both cases, i is able to detect all threats whatever its view of the plan is, $view^i(\Pi)$. That is, FMAP soundness is guaranteed regardless the level of privacy defined between agents.

Regarding the threats caused by the effects of a new action, privacy may prevent the agent from viewing some of the causal links of the plan. Suppose that agent i introduces an action α_t with an effect $(v = d')$ in plan Π. Additionally, there is a causal link in Π of the form $cl = \alpha_0 \overset{\langle v,d \rangle}{\rightarrow} \alpha_1$ introduced by an agent j; as cl is not ordered with respect to α_t, this situation generates a threat. According to $view^i(\Pi)$, agent i may find one of the following situations:

- If $\langle v, d \rangle$ is *public* to i and j, then cl is in $view^i(\Pi)$, and thus, the threat between cl and α_t will be correctly detected and solved by promoting or demoting α_t.

- If $\langle v, d \rangle$ is *private* to j w.r.t. i, then α_t cannot contain an effect $(v = d')$ because $v \notin \mathcal{V}^i$. Therefore, the threat described above can never occur in Π.

- If $\langle v, d\rangle$ is *partially private* to j w.r.t. i, then $cl = \alpha_0 \overset{\langle v,d\rangle}{\rightarrow} \alpha_1$ will be seen as $cl = \alpha_0 \overset{\langle v,\perp\rangle}{\rightarrow} \alpha_1$ in $view^i(\Pi)$. Since $\perp \neq d$, agent i will be able to detect and address the threat between α_t and cl.

Consequently, an agent can always detect the arising threats when it adds a new action, α_t, in the plan. Now, we should study whether the potential threats caused by actions in Π on the causal links that support the action α_t are correctly detected by agent i. Suppose that there is a causal link $cl' = \beta \overset{\langle v',e\rangle}{\rightarrow} \alpha_t$, and an action γ with an effect $(v' = e')$ which is not ordered with respect to α_t. Again, agent i may find itself in three different scenarios according to its view of $(v' = e')$:

- If $(v' = e')$ is *public* to i and j, the threat between cl' and γ will be correctly detected by i.

- If $(v' = e')$ is *private* to j w.r.t. i, then none of the variables in $PRE(\alpha_t)$ are related to v' because $v' \notin \mathcal{V}^i$. Thus, this threat will never arise in Π.

- If $(v' = e')$ is *partially private* to j w.r.t. i, $(v' = e')$ will be seen as $(v' = \perp)$ in $view^i(\Pi)$. Since $\perp \neq e$, the threat between γ and cl' will be correctly detected by agent i.

Notice that privacy does not prevent agents from detecting and solving threats nor affects the complexity of the process. If the fluent is public or partially private, the agent which is refining the plan will be able to detect the threat because it either sees the value of the variable or sees $\perp$, and both contradict the value of the variable in the causal link. If the fluent is private then there is no such threat. This proves that FMAP is sound.

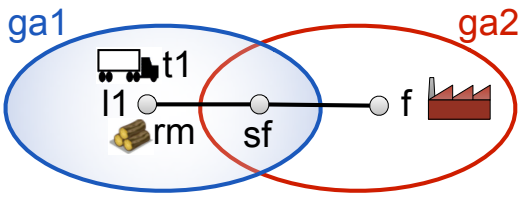

Figure 3.18: Reduced transport example task

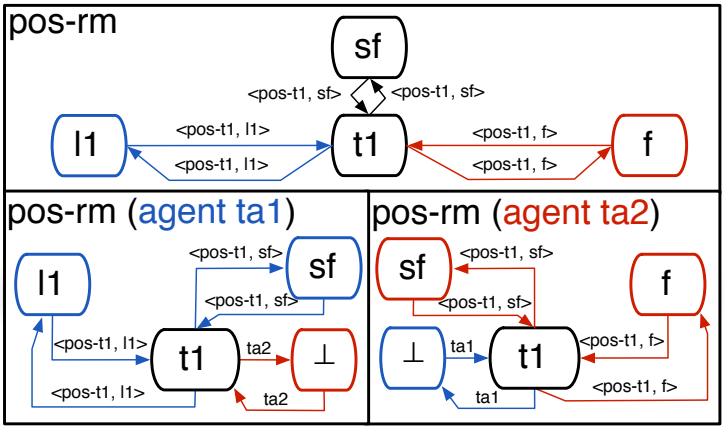

Figure 3.19: Centralized and distributed DTG of the variable $\langle pos\text{-}rm \rangle$

3.4.4.3 DTG-based Heuristic Function

The last aspect of FMAP to analyze is how agents evaluate the refinement plans. FMAP guides the search through a domain-independent heuristic function, as most planning systems (23). It uses the information provided by the frontier states to perform the heuristic evaluation of the plans contained in the tree nodes.

According to the definition shown in section 3.4.4.1, the frontier state of a plan Π, $FS(\Pi)$, can be easily computed as the finite set of fluents that results from executing the actions of the plan Π in $\mathcal{I}$, the initial state of $\mathcal{T}_{MAP}$. Since

refinement plans are not sequential plans, the actions in Δ have to be *linearized* in order to compute the frontier state. The linearization of a refinement plan Π involves establishing a total order among the actions in Δ. Given two actions $\alpha \in \Delta$ and $\beta \in \Delta$, if $\alpha \prec \beta \in \mathcal{OR}$ or $\beta \prec \alpha \in \mathcal{OR}$, we keep this ordering constraint in the linearized plan. In case that α and β are non-sequential actions, we set a total ordering among them. Since plans returned by FLEX are free of conflicts, it is irrelevant how non-sequential actions are ordered.

Frontier states allow to make use of state-based heuristics such as h_{FF}, the relaxed planning graph (RPG) heuristic of FF (54). However, the distributed approach and the privacy model of FMAP make the application of h_{FF} inadequate to guide the search. The reason is this: as none of the agents has a complete knowledge to build an RPG by itself, using h_{FF} to estimate the quality of a refinement plan involves agents building a *distributed RPG* (127). This is a costly process that entails many communications among agents to coordinate which each other, and it has to be repeated for the evaluation of each refinement plan. Therefore, the predictable high computational cost of the application of h_{FF} led us to discard this choice and opt for designing a heuristic based on Domain Transition Graphs (DTGs) (49).

A DTG is a directed graph that shows the ways in which a variable can change its value (49). Each transition is labeled with the necessary conditions for this to happen; i.e., the preconditions that are common to all the actions that induce the transition. DTGs are independent from the state of the plan, thus avoiding recalculation during the planning process.

Privacy is kept in DTGs through the use of the undefined value $\perp$. This value is represented in a DTG as the rest of values of the variables, being the only difference that transitions from/to $\perp$ are labeled with the agents that induce them.

Consider a reduced version of Example 3 depicted in Figure 3.18. In this example, both transport agents $ta1$ and $ta2$ can use truck $t1$ within their geographical areas $ga1$ and $ga2$, respectively. Figure 3.19 shows the DTG of the variable $\langle pos\text{-}rm \rangle$. In a single-agent problem (upper diagram) all the information is available in the DTG. However, in the multi-agent task (bottom diagrams), agent $ta1$ ignores the location of rm if $ta2$ transports it to f, while $ta2$ does not know the initial placement of rm, as location $l1$ lies outside $ta2$'s geographical area, $ga2$. In order to evaluate the cost of achieving $\langle pos\text{-}rm, f \rangle$ from the initial state, $ta1$ will first look up in its DTG, thus obtaining the cost of loading rm in $t1$. As shown in Figure 3.19, the transition between values $t1$ and $\perp$ is labeled with agent $ta2$. Thus, $ta1$ will request $ta2$ the cost of the path between values $t1$ and f to complete the calculation. Communications are required to evaluate multi-agent plans, but DTGs are more efficient than RPGs as they remain constant during planning, so agents can minimize the overhead by memorizing paths and distances between values.

Our DTG-based heuristic function, h_{DTG} in the following, returns, for a given plan Π, the number of actions of a relaxed plan between the frontier state $FS(\Pi)$ and the set of goals of $\mathcal{T}_{MAP}$, $\mathcal{G}$. h_{DTG} performs a backward search introducing in the relaxed plan the actions that support the goals in $\mathcal{G}$, until all their preconditions are supported. Hence, the underlying principle of h_{DTG} is similar to h_{FF}, except for the fact that we use DTGs instead of RPGs to build the relaxed plan.

The h_{DTG} evaluation of a plan Π begins by calculating the frontier state $FS(\Pi)$. Next, an iterative procedure is performed to build the relaxed plan. This procedure manages a list of fluents, *openGoals*, initially set to $\mathcal{G}$. The process iteratively extracts a fluent from *openGoals* and supports it through the introduction of an action in the relaxed plan. The preconditions of such an action are then included in the *openGoals* list. For each variable $v \in \mathcal{V}$, the procedure manages a list of

values, $Values_v$, which is initialized to the value of v in the frontier state $FS(\Pi)$. For each action added to the relaxed plan that has an effect $(v = d')$, d' will be stored in $Values_v$. An iteration of the h_{DTG} evaluation process performs the following stages:

- **Open goal selection**: From the *openGoals* set, the procedure extracts the fluent $\langle v, d_g \rangle \in openG$ that requires the largest number of value transitions to be supported.

- **DTG path computation**: For every value d_0 in $Values_v$, this stage calculates the shortest sequence of value transitions in v's DTG from d_0 to d_g. Each path is computed by applying Dijkstra's algorithm between the nodes d_0 and d_g in the DTG associated to variable v. The path with a minimum length is stored as $minPath = ((d_0, d_1), (d_1, d_2), \ldots, (d_{g-1}, d_g))$.

- **Relaxed plan construction**: For each value transition $(d_i, d_{i+1}) \in minPath$, the minimum-cost action α_{min} that produces such a transition is introduced in the relaxed plan; that is, $\langle v, d_i \rangle \in PRE(\alpha_{min})$ and $(v = d_{i+1}) \in EFF(\alpha_{min})$. The cost of an action is computed as the sum of the minimum number of value transitions required to support its preconditions. The unsupported preconditions of α_{min} are stored in $openGoals$, so they will be supported in the subsequent iterations. For each effect $(v' = d') \in EFF(\alpha_{min})$, the value d' is stored in $Values_{v'}$, so d' can be used in the following iterations to support other $openGoals$.

The iterative evaluation procedure carries on until all the open goals have been supported, that is, $openGoals = \emptyset$, and h_{DTG} returns the number of actions in the relaxed plan.

3.4.4.4 Limitations of FMAP

In this section we present some limitations of FMAP that are worth discussing. FMAP builds upon the POP paradigm, so it can handle plans with parallel actions and only enforces an ordering when strictly necessary. FMAP, however, does not explicitly manage time constraints nor durative actions yet. A POP-based planner can easily be extended to incorporate time because the application of the least-commitment principle provides a high degree of execution flexibility. Additionally, POP is independent of the assumption that actions must be instantaneous or have the same duration and allows to define actions of arbitrary duration and different types of temporal constraints as long as the conditions under which actions interfere are well defined (104). In short, POP represents a natural and very appropriate way to include and handle time in a planning framework.

FLEX involves the construction of a POP tree for each potentially supportable action (see Figure 3.17). This procedure is more costly than the operations required by a standard planner to refine a plan. However, the search trees are independent of each other, which makes it possible to implement FLEX by using multiple execution threads. Parallelization improves the performance of FLEX and the ability of FMAP to scale up. Section 3.4.5 provides more insight on the FLEX implementation.

Currently, FMAP is limited to cooperative goals, so that all the goals must be defined as global objectives to all the participating agents (see section 3.4.3). Nevertheless, an extension of FMAP to support self-interested agents with local goals is being considered as a future work.

FMAP is a general procedure aimed to solve any kind of MAP task. In particular, solving tightly-coupled tasks requires a noticeable coordination effort. Multi-agent coordination in distributed systems where agents must cooperate is always

a major issue. This dependency on coordination makes FMAP a communication-reliant approach. Agents do not only have to communicate the refinement plans that they build at each iteration, but they also have to communicate during the heuristic evaluation of the refinement plans in order to keep privacy (see subsection 3.4.4.3). The usage of a coordinator agent effectively reduces the need of communication. The experimental results will show that FMAP can effectively tackle large problem instances (see section 3.4.5). Nevertheless, reducing communication overhead while keeping the ability to solve any kind of task remains as an ongoing research topic that we consider for future developments.

Privacy management is another issue that potentially worsens the performance of FMAP. In section 3.4.3.1, we defined a mechanism to detect and address threats in partial plans, even when agents do not have a complete view of such plans. Privacy does not add extra complexity to FLEX since agents manage the undefined value $\perp$ as any other value in the domain of a variable. It does, however, make the refinement-plan communication stage more complex, because, when an agent i sends $view^j(\Pi)$ to an agent j, this implies that i must previously adapt the information of Π according to the privacy rules defined w.r.t. to j.

Privacy also affects the heuristic evaluation of the plans in terms of quality. Since a refinement plan is only evaluated by the agent that generates it and this evaluation is influenced by the view of the agent on the plan, the result may not be as accurate as if the agent had a complete view of such plan. Empirical results, however, will show that, even with these limitations, our heuristic function provides a good performance in a wide variety of planning domains (see section 3.4.5).

3.4.5 Experimental results

In order to assess the performance of FMAP, we run experimental tests with some of the benchmark problems from the International Planning Competitions[1] (IPC). More precisely, we adapted the STRIPS problem suites of 10 different domains from the latest IPC editions to a MAP context. The tests compare FMAP with two different state-of-the-art MAP systems: MAPR (11) and MAP-POP (114). We excluded Planning First (80) from the comparison because it is outperformed by MAP-POP (114).

This section is organized as follows: first, we provide some information on the FMAP implementation and experimental setup. Then, we present the features of the tested domains and we analyze the MAP adaptation performed for each domain. Next, we show a comparative analysis between FMAP and the aforementioned planners, MAPR (11) and MAP-POP (114). Then, we perform a scalability analysis of FMAP and MAPR. Finally, we summarize and discuss the results obtained by FMAP and how they compare to the other two planners.

3.4.5.1 FMAP implementation and experimental setup

Most multi-agent applications nowadays make use of middleware multi-agent platforms that provide them with the communication services required by the agents (83). The entire code of FMAP is implemented in Java and builds upon the Magentix2 platform[2] (109). Magentix2 provides a set of libraries to define the agents' behavior, along with the communication resources required by the agents. Magentix2 agents communicate by means of the FIPA Agent Communication Language (81). Messaging is carried out through the Apache QPid broker[3], which is a critical

[1] http://ipc.icaps-conference.org/
[2] http://www.gti-ia.upv.es/sma/tools/magentix2
[3] http://qpid.apache.org/

component for FMAP agents.

FMAP is optimized to take full advantage of the CPU execution threads. The FLEX procedure, which generates refinement plans over a given base plan, develops a POP search tree for each potentially supportable action of the agent's domain. As the construction of a tree is completely independent to each other, these processes run in parallel for each agent.

Agents synchronize their activities at the end of the refinement plan generation stage. Consequently, FMAP assigns the same number of execution threads to each agent, so that they all spend a similar amount of time to complete the FLEX procedure (note that if we allocate extra threads to a subset of the agents, they would still have to wait to the slowest agent to synchronize). FLEX builds as many POP search trees in parallel as execution threads agents have been allocated. The h_{DTG} heuristic is implemented in a similar way. An agent can simultaneously evaluate as many plans as execution threads it has been allocated.

All the experimental tests were performed on a single machine with a quad-core Intel Core i7 processor and 8 GB RAM (1.5 GB RAM available for the Java VM). The CPU used in the experimentation has eight available execution threads, distributed as follows: in tasks that involve two agents, FMAP allocates four execution threads per agent; in tasks with three or four agents, each agent has two available execution threads; finally, agents have a single execution thread at their disposal in tasks involving five or more agents. For instance, the three agents in Example 3 would get two different execution threads in this particular machine. Hence, in the FLEX example depicted in Figure 3.17, agent $ta1$ would be able to study two candidate actions simultaneously, thus reducing the execution time of the overall procedure.

Domain	Type	IPC	Agents	Coop. goals	Applicability	
					MAPR	FMAP MAP-POP
Blocksworld	LC	'98	robot	No	✓	✓
Driverlog	LC	'02	driver	No	✓	✓
Rovers	LC	'06	rover	No	✓	✓
Satellite	LC	'04	satellite	No	✓	✓
Zenotravel	LC	'02	aircraft	No	✓	✓
Depots	TC	'02	depot/truck	Yes	✗	✓
Elevators	TC	'11	fast-elevator/ slow-elevator	Yes	✗	✓
Logistics	TC	'00	airplane/truck	Yes	✗	✓
Openstacks	TC	'11	manager/ manufacturer	Yes	✗	✓
Woodworking	TC	'11	machine	Yes	✗	✓

Table 3.6: Features of the MAP domains

3.4.5.2 Planning domain taxonomy

The benchmark used for the experiments includes 10 different domains of the IPCs that are suitable for a multi-agent adaptation. The IPC benchmarks come from (potential) real-world applications of planning, and they have become the de facto mechanism to assess the performance of single-agent planning systems. The *elevators* domain, for instance, is inspired in a real problem of Schindler Lifts Ltd. (62); the *satellite* domain is motivated by a NASA space application (70); the *rovers* domain deals with the decision of daily planning activities of Mars rovers (16); and the *openstacks* domain is based on the *minimum maximum simultaneous open stacks* combinatorial optimization problem. Hence, all the domains from the IPCs resemble practical scenarios and they are modeled to keep, as far as possible, both their structure and complexity. In MAP, there is not a standardized collection of planning domains available. MAP approaches adapt instead some well-known

IPC domains to a multi-agent context, namely the *satellite*, *rovers* and *logistics* domains (11, 80, 114).

Converting planning domains into a multi-agent version is not always possible due to the domain characteristics. While some IPC domains have a straightforward multi-agent decomposition, others are inherently single-agent. We developed a domain-dependent tool to automatically translate the original STRIPS tasks into our *PDDL*-based MAP language.

Table 3.6 describes the main features of the 10 MAP domains included in the benchmark. Column *Type* indicates whether the MAP tasks of the domain are loosely-coupled (LC) or tightly-coupled (TC). Column *IPC* shows the last edition of the IPC in which the domain was included. *Agents* indicates the types of object used to define the agents. *Coop. goals* indicates the presence or absence of such goals in the domains tasks. Finally, the *Applicability* columns show the MAP systems capable to cope with each domain.

In order to come up with a well-balanced benchmark, we put the emphasis on the presence (or absence) of specialized agents and cooperative goals. Besides the adaptation to a multi-agent context, the 10 selected domains are a good representative sample of loosely-coupled domains with non-specialized agents and tightly-coupled domains with cooperative goals.

Privacy in each domain is defined according to the nature of the problem and the type of agents involved, while maintaining a correlation and identification with the objects in a real-world problem.

Loosely-coupled domains. The five loosely-coupled domains presented in Table 3.6 are: *Blocksworld*, *Driverlog*, *Rovers*, *Satellite* and *Zenotravel*. The prime characteristic of these domains is that agents have the same planning capabilities (operators) such that each task goal can be individually solved by a single agent.

That is, tasks can be addressed without cooperation among agents. Next, we provide some insight into the features of these domains and the MAP adaptations.

Satellite (70). This domain offers a straightforward multi-agent decomposition (80, 114). The MAP domain features an agent per satellite. The resulting MAP tasks are almost decoupled as each satellite can attain a subset of the task goals (even all the goals in some cases) without interacting with any other agent. The number of agents in the problems of this domain vary from 1 up to 12. The instruments of a satellite are private to the agent, only the orientation of each agent and the information on the images taken by the satellites is defined as public.

Rovers (70). As the *Satellite* domain, *Rovers* also offers a straightforward decomposition (80, 114). The MAP domain features an agent per rover. Rovers collect samples of soil and rock and hardly interact with each other except when a soil or rock sample is collected by an agent, and so it is no longer available to the rest of agents. The number of agents ranges from 1 to 8 rovers per task. As in the *Satellite* domain, only the information related to the collected samples is defined as public.

Blocksworld. The MAP version of this domain introduces a set of robot agents (four agents per task), each having an arm to arrange blocks. Unlike the original domain, the MAP version of *Blocksworld* allows to handle more than one block at a time. All the information in this domain is considered as public.

Driverlog (70). In this MAP domain, the agents are the drivers of the problem, ranging between 2 and 8 agents per task. Driver agents are in charge of driving the available trucks and delivering the packages to the different destinations. All the information in the domain (status of drivers, trucks and packages) is publicized by the driver agents.

Zenotravel (70). This domain defines one agent per aircraft. The simplest tasks include one agent and the most complex ones up to five agents. Aircrafts can directly transport passengers to any city in the task. As in the *Blocksworld* and *Driverlog* domains, all the information concerning the situation of the passengers and the current location of each aircraft is publicly available to all the participating agents.

Tightly-coupled domains. We also analyzed five additional domains that feature specialized agents with different planning capabilities: *Depots*, *Elevators*, *Logistics*, *Openstacks* and *Woodworking*. The features of these domains give rise to complex, tightly-coupled tasks that require interactions or commitments (46) among agents to be solved.

Depots (70). This domain includes two different types of specialized agents, depots and trucks, that must cooperate in order to solve most of the tasks' goals. This domain, which is the most complex one in our MAP benchmark, leads to tightly-coupled MAP tasks with many dependences among agents. *Depots* tasks contain a large number of participating agents, ranging from 5 to 12 agents. Only the location of packages and trucks is defined as public information.

Elevators. Each agent in this domain can be a slow-elevator or a fast-elevator. Operators in the STRIPS domain are basically the same for both types of elevators as the differences between them only affect the action costs. Elevator agents, however, are still specialized as the floors they can access are limited. This leads to tasks that require cooperation to fulfill some of the goals. For instance, an elevator may not be able to take a passenger to a certain floor, so it will stop in an intermediate floor so that the passenger can board another elevator able to reach this floor. Tasks include 3 up to 5 agents. Agents share the information regarding the location of the different passengers.

Domain	Tasks	#C	FMAP				MAPR			
			#S	#A	MS	Time	#S	#A	MS	Time
Blocksworld	34	19	19	17,79	13,68	86,17	34	1,27x	1,20x	0,04x
Driverlog	20	15	15	24,64	13,93	42,02	18	1,19x	1,53x	0,06x
Rovers	20	19	19	32,63	14,95	53,82	20	0,97x	0,85x	0,05x
Satellite	20	15	16	27,27	16,47	177,65	18	1,14x	1,03x	0,03x
Zenotravel	20	18	18	25,50	13,94	180,62	20	1,24x	1,32x	0,02x

Table 3.7: Comparison between FMAP and MAPR

Logistics. This domain presents two different types of specialized agents, airplanes and trucks. The delivery of some of the packages involves the cooperation of several truck and airplane agents (similarly to the example task introduced in this article). Tasks feature 3 up to 10 different agents. Information regarding the position of the packages is defined as public.

Openstacks (40). This MAP domain includes two specialized agents in all of the tasks; the *manager* is in charge of handling the orders, and the *manufacturer* controls the different stacks and manufactures the products. Both agents depend on each other to perform their activities, thus resulting in tightly-coupled MAP tasks with inherently cooperative goals. Most of the information regarding the different orders and products is public, since both agents need it to interact with each other.

Woodworking. This domain features four different types of specialized agents (a planer, a saw, a grinder and a varnisher), representing the machines in a production chain. In most cases, the output of a machine constitutes the input of the following one, so *Woodworking* agents have to cooperate to fulfill the different goals. All the tasks include four agents, a machine of each type. All the information on the status of the different wood pieces is publicized, since agents require this information in order to operate.

3.4.5.3 FMAP vs. MAPR comparison

This subsection compares the experimental results of FMAP and MAPR (11). MAPR is implemented in Lisp and uses LAMA (95) as the underlying planning system, making no use of a middleware platform for multi-agent systems. Each experiment is limited to 30 minutes.

Table 3.7 shows the comparative results. FMAP and MAPR $\#S$ columns refer to the number of tasks solved by each approach. The average number of actions ($\#A$), makespan or plan duration (MS) and search time ($Time$) consider only the tasks solved by both FMAP and MAPR (column $\#C$ shows the number of tasks solved by both planners). Actions, makespan and time values in MAPR are relative to the results obtained with FMAP. Values nx in Table 3.8 indicate "n times as much as the FMAP result". Thereby, an #A or MS value higher than 1x is a better result for FMAP and a value lower than 1x is a worse result for FMAP. However, a $Time$ value higher than 1x indicates a better result for FMAP.

MAPR is one of the most recent MAP systems that exhibits an excellent performance in comparison to other state-of-the-art MAP approaches (11). However, as reflected in Table 3.6, MAPR is only compatible with the loosely-coupled domains in the benchmark. This limitation is due to the planning approach of MAPR. Particularly, MAPR applies a goal allocation procedure, decomposing the MAP task into subtasks and giving each agent a subset of the task goals to solve. Each agent subtask is solved with the single-agent planner LAMA (95) such that the resulting subplans are progressively combined into a global solution. This makes MAPR an incomplete planning approach, limited to loosely-coupled tasks without cooperative goals. That is, MAPR is built under the assumption that each goal must be addressed by at least one of the agents in isolation (11).

Whereas the communication overhead is relatively high in FMAP (to a large extent, this is due to the use of Magentix MAS platform), agents in MAPR do not need to communicate during the plan construction because each agent addresses its allocated subgoals by itself. This setup has a rather positive impact in the execution times and the number of problems solved (coverage). Table 3.7 shows that, as expected, execution times in MAPR are much lower than FMAP. With respect to coverage, MAPR solves 110 out of 114 loosely-coupled tasks (roughly a 96% of the tasks), while FMAP solves 87 of such tasks (a 76%).

However, in most domains, FMAP comes up with better-quality plans than MAPR, considering the number of actions as well as makespan. MAPR is conditioned by the order in which agents solve their subtasks. The first agent that computes a subplan cannot take advantage of the potential synergies that may arise from other agents actions; the second agent has only the information of the first agent's subplan, and so on. Additionally, the allocation of goals to each agent may lead to poorly-balanced plans. Although FMAP is a more time-consuming approach, it avoids these limitations, as agents work together to build the plan action by action. Thus, FMAP provides agents with a global view of the plan at each point of the construction process, while agents in MAPR keep a local view of the plan at hand.

The *Driverlog* domain, while being loosely-coupled, offers many possible synergies between agents. For instance, a driver agent can use a truck to travel to its destination and load a package on its way, while another agent may take over the truck and complete the delivery. If the first agent acted in isolation, it would deliver the package and then go back to its destination, which would result in a worse plan. Robot agents in the *Blocksworld* domain can also cooperate to improve the quality of the plans: for instance, a robot can pick up a block so that another robot can retrieve the block below. Goal balance is also a key aspect in

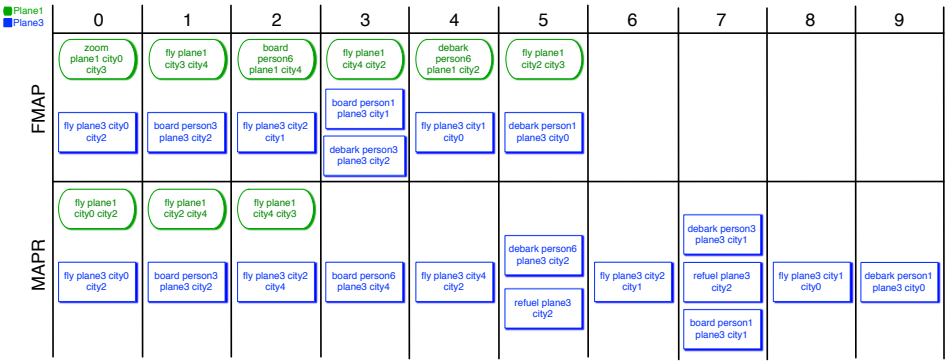

Figure 3.20: *Zenotravel* task 8 solution plan as obtained by FMAP (upper plan) and MAPR (lower plan)

Zenotravel as aircraft agents have a limited autonomy. If an aircraft solves too many goals it may be forced to refuel, worsening the plan quality.

Figure 3.20 illustrates the MAPR limitations by showing the solution plans obtained by both approaches for the task 8 of the *Zenotravel* domain. The goals of this task involve transporting three different persons and flying `plane1` to `city3`. The first three goals are achievable by all the plane agents, while the last one can only be completed by agent `plane1`.

MAPR starts with agent `plane3`, which solves all of the goals it can. Then, `plane1` receives the subplan and completes it by solving the remaining goal. The resulting joint plan is far from the optimal solution. Agent `plane3` requires 10 time units to solve its subplan, as it transports all of the passengers. The high number of `fly` actions forces the agent to introduce additional actions to refuel its tank. On the other hand, agent `plane1` flies directly to its destination without transporting any passenger.

Agents in FMAP, however, progressively build the solution plan together without using an a-priori goal allocation, which allows them to obtain much better-

Domain	Tasks	#C	FMAP				MAP-POP			
			#S	#A	MS	Time	#S	#A	MS	Time
Blocksworld	34	6	19	9,20	7,80	7,57	6	0,91x	0,74x	21,49x
Driverlog	20	2	15	9,50	7,00	0,66	2	1,11x	1,00x	949,39x
Rovers	20	6	19	32,63	14,95	53,82	6	1,01x	1,04x	29,27x
Satellite	20	7	16	17,14	12,57	16,00	7	1,03x	0,89x	0,37x
Zenotravel	20	3	18	7,67	4,33	1,25	3	1,00x	1,00x	87,54x
Depots	20	1	6	14,00	9,00	10,56	1	0,86x	1,00x	2,77x
Elevators	30	22	30	21,32	11,36	14,60	22	1,04x	1,37x	14,23x
Logistics	20	7	10	32,29	12,71	18,26	7	0,97x	0,91x	5,89x
Openstacks	30	0	23	53,13	41,78	268,62	0	-	-	-
Woodworking	30	0	22	16,50	4,45	100,88	0	-	-	-

Table 3.8: Comparison between FMAP and MAP-POP

quality plans, taking advantage of synergies between actions of different agents and effectively balancing the workload among agents. Figure 3.20 shows that, in FMAP, agent `plane1` transports `person6` to its destination, thus simplifying the activities of `plane3`, which effectively avoids refueling. The resulting plan is a much shorter and better-balanced solution than MAPR's (only 6 time steps versus 10 time steps in MAPR) and requires fewer actions (13 actions versus 16 in MAPR).

Table 3.7 shows that FMAP noticeably improves plan quality except in the most decoupled domains, namely *Rovers* and *Satellite* (in the latter, FMAP results are still better than MAPR's but not so outstanding). In these domains, synergies among agents are minimal or even inexistent. Thus, MAPR is not penalized by its search scheme, obtaining plans of similar quality to FMAP.

3.4.5.4 FMAP vs. MAP-POP comparison

We compared FMAP against another recent MAP system, MAP-POP (114). As FMAP, MAP-POP agents jointly explore the space of multi-agent plans. This setup allows MAP-POP to overcome some of the limitations of MAPR, as it is able to

tackle tightly-coupled tasks with cooperative goals. However, MAP-POP has two major disadvantages: much like MAPR, MAP-POP is an incomplete approach, as it implicitly bounds the search tree by limiting its branching factor. This may prevent agents from generating potential solution plans (114). Additionally, MAP-POP is based on backward-chaining POP technologies, thus relying on heuristics that offer a rather poor performance in most MAP domains.

Table 3.8 shows the comparison between FMAP and MAP-POP. As in Table 3.7, the average results consider only the tasks solved by both approaches (FMAP results for the *Openstacks* and *Woodworking* include all the tasks solved by this approach as MAP-POP solves none of the tasks). The figures in *FMAP* show the results obtained with FMAP for the common problems; figures in *MAP-POP* are relative to the results of FMAP.

In general, FMAP improves MAP-POP results in almost every aspect. In terms of coverage, FMAP clearly outperforms MAP-POP, solving 178 out of 244 tasks (roughly a 73% of the tasks in the benchmark), while MAP-POP solves only 54 tasks (22%). Overall, MAP-POP has issues with some of the most complex tightly-coupled domains (in particular, *Depots*, *Openstacks* and *Woodworking*), but performs well in the *Elevators* domain. With respect to the loosely-coupled domains, MAP-POP solves only the simplest tasks, ranging from three to seven tasks solved per domain.

Regarding plan quality, it is difficult to compare the results due to the low coverage of MAP-POP. Focusing on the domains in which MAP-POP solves a significant number of tasks, we observe that MAP-POP obtains slightly better solution plans than FMAP in *Blocksworld* and *Satellite*. FMAP, however, outperforms MAP-POP in *Elevators*, the domain in which both approaches solve the largest number of tasks.

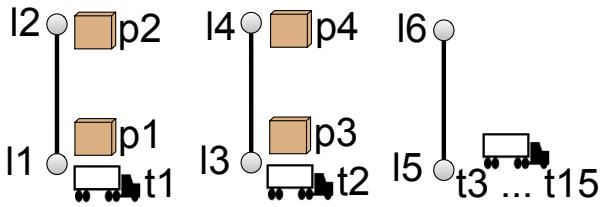

Figure 3.21: *Logistics*-like scalability task

Finally, the results show that FMAP is much faster than MAP-POP, from 5 times faster in *Logistics* to even 1000 times faster in the *Driverlog* domain. MAP-POP only improves FMAP times in the seven *Satellite* tasks.

3.4.5.5 Scalability analysis

We prepared two additional experiments to analyze the ability of FMAP and MAPR to scale up. The first test analyzes how both planners scale up when increasing the number of agents of a task, keeping the rest of the parameters unchanged. More specifically, we designed the loosely-coupled *logistics*-like transportation task shown in Figure 3.21. The basic task includes two different trucks, $t1$ and $t2$. Truck $t1$ moves between locations $l1$ and $l2$, and truck $t2$ between locations $l3$ and $l4$; there is no connection between $t1$'s and $t2$'s locations. Trucks have to transport a total of four packages, $p1 \ldots p4$, as shown in Figure 3.21. In order to ensure that MAPR is able to solve the task, both $t1$ and $t2$ can solve two of the four problem goals by themselves: $t1$ will deliver $p1$ and $p2$, while $t2$ will transport $p3$ and $p4$. Cooperation is thus not required in this task, as opposite to the IPC *logistics* domain.

We defined and ran 14 different tests of this basic task. Each test increases the number of agents in the task by one, ranging from 2 to 15 truck agents. The problems are modeled so that the extra truck agents, $t3 \ldots t15$, are placed in a

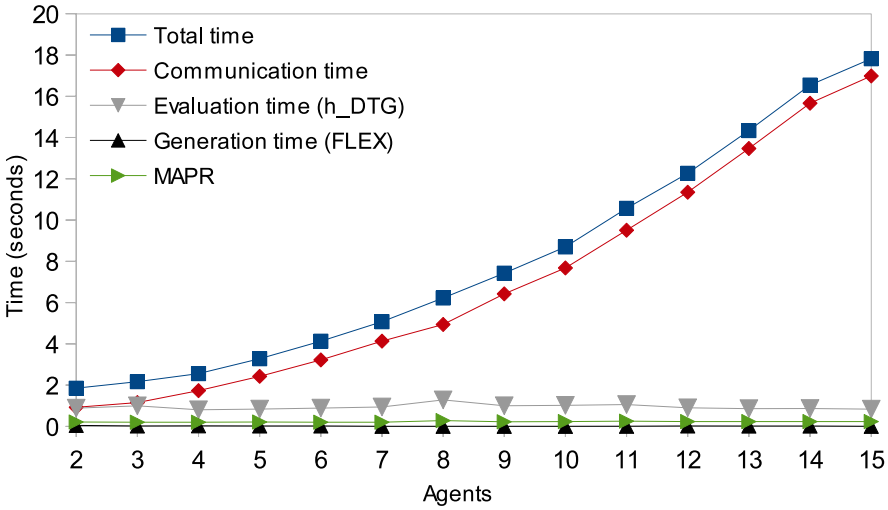

Figure 3.22: Scalability results for the *logistics*-like task

separate location $l5$, from which there is no access to the locations that $t1$ and $t2$ can move through. Therefore, these additional agents included in each task are unable to solve any of the task goals. They, however, propose refinement plans in FMAP (more precisely, they introduce an action to move to $l6$, as shown in Figure 3.21), increasing the complexity of the task in terms of both the number of messages exchanged and the branching factor of the FMAP search tree.

The plot in Figure 3.22 separately depicts the time required by each process in FMAP. We show the time of FLEX in generating the refinement plans, the time consumed by the h_{DTG} evaluation procedure and the time spent by agents to communicate and synchronize, which includes the base plan selection and the exchange of plans among agents. Every task was solved by FMAP in 14 iterations, resulting in a 12-action solution plan (six actions introduced by each truck $t1$ and $t2$).

As shown in Figure 3.22, FLEX has a noticeably low impact in the overall execution time. This proves that, even dealing with privacy and building a tree for each potentially supportable action, FLEX offers a good performance and does not limit FMAP's scalability.

Although every task only required 14 iterations to be solved, the growing number of agents increases the size of the search tree. In the two-agent task, agents generate 3.3 refinement plans per iteration in average, while in the 15-agent task, the average branching factor goes up to 11.8 refinement plans. This, however, does not affect the time consumed by h_{DTG}, which remains relatively constant in all tasks. As agents evaluate plans simultaneously, the evaluation time hardly grows when the number of participants increases.

Figure 3.22 confirms that communications among agents are the major bottleneck of FMAP. As the number of agents increases, so does the branching factor. Thus, each agent has to communicate more refinement plans to a higher number of participants. Synchronizing a larger number of agents is also more complex, which increases the number of exchanged messages. All these communications are managed by a centralized component, the QPid broker, which is affected by the communication overhead of the system.

The behaviour of MAPR remains constant in all the tests, taking about 0.2 seconds to resolve each task. Since MAPR does not require communications, the growing number of agents does not affect its performance. Notice that if we consider only the time spent by h_{DTG} (around 0.8 seconds per test) and FLEX (0.02 seconds approximately), FMAP execution times are quite similar to MAPR.

The resolution of this loosely-coupled task does not require coordination in order to be able to compare with MAPR. However, the coordination mechanism and message exchange of FMAP is equally applied to all planning tasks. Hence,

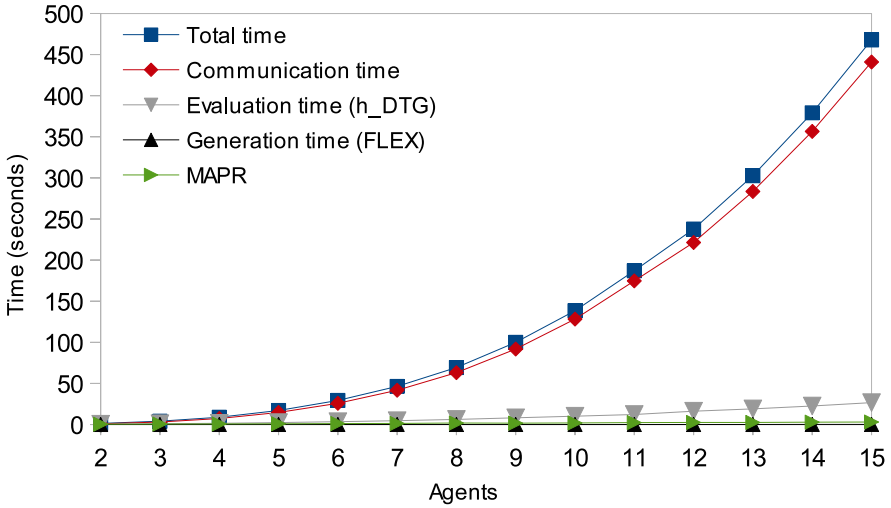

Figure 3.23: Scalability results for the *satellite* task

the ability of solving tightly-coupled tasks comes at the cost of a high coordination effort which is not suffered by MAPR.

We performed a second experiment, based on the *satellite* domain, to assess the scalability of both planners when increasing not only the number of agents, but also the number of goals and so the complexity of the task. We also defined 14 MAP tasks, ranging from 2 to 15 satellite agents. The simplest task comprises two satellite agents, $s1$ and $s2$, which must take an image of two different planets. Satellites are configured so that each of them can capture an image of a single planet. The instruments they have on board are powered on and calibrated, so the agent can directly reorient and acquire the image. Unlike the first test, each *satellite* task adds one more goal over the previous task, as well as an extra agent. Then, the additional agents, $s3 \ldots s15$, must each solve a goal by themselves. This increases the branching factor as well as the number of iterations for solving a

task.

Figure 3.23 shows the results for this scenario. The solution plans obtained by FMAP range from 4 actions (in the two-agent task) to 30 actions (in the 15-agent task). FMAP required 31 iterations to solve the 15-agent task and only 4 iterations for the two-agent task. The growing complexity also affects the average branching factor, which ranges from 25.67 to 255.06 plans.

As depicted in Figure 3.23, the complexity of the tasks does not affect FLEX, which takes less than 0.2 seconds in each task. In general, the performance of FLEX only decreases when handling very large base plans in domains with many applicable actions. We thus can conclude that FLEX is an efficient and highly scalable component of FMAP.

Regarding the h_{DTG} heuristic, evaluation times range from 0.35 seconds in the simplest task to 26.64 seconds in the most complex one. Although evaluation time is slightly higher than the generation time, we can affirm that this is a good performance considering that: 1) the branching factor and the number of iterations increase from task to task, which results in a much larger number of plans to evaluate, and 2) unlike FLEX, the evaluation h_{DTG} also involves some communications among agents, which obviously increase when the number of agents goes up. All in all, and considering just the times of h_{DTG} and FLEX, FMAP is only about 9 times slower in the 15-agent task than MAPR, which completes this task in 3 seconds.

To sum up, both tests confirm that communication overhead is the main issue of FMAP regarding scalability. Communicating plans and synchronizing agents are rather costly tasks, especially when dealing with complex tasks that combine a large branching factor and a high number of participating agents.

3.4.5.6 Discussion on the results

The experimental results support our initial claims: FMAP is a domain-independent approach that offers a good tradeoff between coverage and execution times, being able to solve any typology of MAP task.

We compared FMAP against two different state-of-the-art MAP approaches. On the one hand, MAPR is designed as a fast MAP solver. The results show that MAPR provides excellent execution times, but its performance comes at a cost: it completely rules out tightly-coupled domains that require cooperation. Many real-world domains, such as logistics or production supply-chains, require cooperation between independent entities. Hence, non-cooperative planners for solving disjoint subtasks in which agents can effectively avoid interactions are not suitable for many real-world MAP problems. All in all, MAPR solves 45% of the whole benchmark while FMAP solves 73% of the tasks.

On the other hand, MAP-POP is a general approach capable of solving any type of planning task like FMAP. The approach followed by MAP-POP is clearly influenced by the use of backward-chaining POP technologies and, in particular, by the low-informative heuristics. This planner offers the worst results in terms of coverage and execution times, thus stating that FMAP represents a step ahead in multi-agent cooperative planning.

Regarding the scalability tests, it has been proved that FMAP ability to scale up is only affected by communications. While MAPR performance is unaltered when the number of agents increase, FMAP performance is affected by its heavy dependency on the agents communications. These results lead to one of our future lines of work, studying techniques to reduce overhead communication without losing the ability to tackle any kind of MAP task.

3.4.6 Conclusions

FMAP is a general-purpose MAP model that supports inherently distributed domains and defines an advanced notion of privacy. Agents in FMAP use an internal POP procedure to calculate all possible ways to refine a plan, which guarantees FMAP completeness. Agents exchange plans and their evaluations by means of a communication mechanism governed by a coordinator agent. FMAP exploits the structure of distributed state-independent domain transition graphs for the heuristic evaluation of plan, thus avoiding recalculating estimates in each node of the POP search tree.

Privacy is maintained all along the search process. Agents only communicate the relevant information they share with the rest of agents. This advanced notion of privacy is very useful for modeling real-world problems. The experiments show that dealing with privacy has a relatively low impact in the overall performance of FMAP.

The exhaustive testing on IPC benchmarks shows that FMAP outperforms other state-of-the-art MAP frameworks as it is capable of solving tightly-coupled domains with specialized agents and cooperative goals as well as loosely-coupled problems. The performance of FMAP is only affected by the extensive communications among agents. To the best of our knowledge, FMAP is likely to be currently the most competitive domain-independent MAP system.

3.5 Global heuristics for distributed cooperative multi-agent planning

Paper data

Authors:	A. Torreño, Ó. Sapena, E. Onaindia
Conference:	25th International Conference on Automated Planning and Scheduling
Pages:	225-233
Ranking:	CORE A*
Year:	2015

Abstract *Almost every planner needs good heuristics to be efficient. Heuristic planning has experienced an impressive progress over the last years thanks to the emergence of more and more powerful estimators. However, this progress has not been translated to multi-agent planning (MAP) due to the difficulty of applying classical heuristics in distributed environments. The application of local search heuristics in each agent has been the most widely adopted approach in MAP but there exist some recent attempts to use global heuristics. In this paper we show that the success of global heuristics in MAP depends on a proper selection of heuristics for a distributed environment as well as on their adequate combination.*

3.5.1 Introduction

Cooperative Multi-Agent Planning (MAP) extends classical planning by introducing a set of individual entities or agents that plan together in a shared deterministic environment to solve a common set of goals. Agents in cooperative MAP address two basic tasks, synthesize individual plans and coordinate them to build a joint plan that solves the MAP task.

The various existing MAP approaches can be classified according to the planning and coordination models they use. Some approaches perform a pre-planning distribution of the MAP task. MAPR (11) allocates the task goals to the participating agents, which in turn individually invoke LAMA (95) to solve their assigned subtasks. The work in (22) automatically decomposes single-agent tasks into MAP problems, which are then locally solved through a centralized heuristic planner.

Other MAP techniques put the focus on plan merging. Planning First (80) is one of the first planners based on *MA-STRIPS* (14), a minimalistic multi-agent extension of the STRIPS model. Agents in Planning First individually synthesize plans through a state-based planner. The resulting local plans are then coordinated through a distributed Constraint Satisfaction Problem.

A third group of approaches directly apply multi-agent search, interleaving planning and coordination. MA-A* (78) is also a *MA-STRIPS*-based approach that performs a distributed A* search, guiding the procedure through admissible local heuristic functions. The work in (10) formulates a privacy-preserving MAP model by adapting MA-A*.

Most of the aforementioned MAP approaches resort to heuristic search at some point during the planning process, applying local heuristic search to each participating agent. Since agents usually have a limited knowledge of the task, the quality of local estimates diminish in comparison to the global heuristics applied in single-agent planning tasks.

A *global heuristic* in MAP is the application of a heuristic estimate to the MAP task carried out by several agents which have a different knowledge of the task and, possibly, privacy requirements. The design of global estimators for cooperative MAP is a challenging task (78) which has been seldom studied. Exceptions are the work in (107), which introduces a distributed version of some well-known

relaxation-based heuristics, and the application of a landmark-based global heuristic in the GPP planner (72).

The focus of the present work is to analyze the benefits of global heuristics in MAP and to study how the combination of these functions can noticeably improve the efficiency of cooperative MAP systems. For our purposes, we take FMAP as our framework (116). FMAP is a fully-distributed forward-chaining multiagent POP approach that preserves agents' privacy. Specifically, this paper presents the following contributions:

- Formalization of two distributed heuristic functions: h_{DTG} (116), a variation of the Context-Enhanced Additive heuristic (52) based on Domain Transition Graphs (49); and h_{Land}, a privacy-preserving version of the landmark extraction algorithm introduced in (55).
- MH-FMAP, a novel multi-heuristic MAP approach that combines h_{DTG} and h_{Land} orthogonally, notably improving the performance of FMAP.

This paper is organized as follows: after presenting some related work and the key notions of FMAP, we introduce the formalization of h_{DTG}, the design of h_{Land} and the combination of both heuristics into MH-FMAP. The experimental results evaluate the two heuristics and the multi-heuristic approach on various domains adapted from the International Planning Competition[1] (IPC) to a multi-agent context and compares the results with the ones obtained with GPPP.

3.5.2 Related work

Many of the existing MAP frameworks apply some form of heuristic search to guide the planning process. The use of global heuristics in MAP is, however, less

[1] http://ipc.icaps-conference.org

frequent due to the inherent features of MAP scenarios, which introduce additional requirements and make it an arduous task:

- The data of a MAP task are usually distributed across the agents; unlike single-agent planning, in MAP it does not exist an entity that centralizes the information of the task. Hence, a communication protocol among the agents is required to compute global heuristic estimates.
- Most MAP models deal with agents' privacy. The communication protocol must thus guarantee that agents are able to calculate heuristic estimates without revealing sensitive private information.

In some works, the features of the planning model force the application of a local heuristic search scheme, in which an agent calculates the heuristic value of a plan based on its local information. In (11), goals are allocated to the agents, which then solve their problems iteratively, communicating the solution of an agent to the next agent. Thus, the heuristic functions defined in LAMA, namely h_{FF} (54) and h_{Land} (95), are applied from a local standpoint.

Local search heuristics have also been used in other MAP approaches, even though their planning model is suitable to accommodate distributed functions. The work in (78) presents MA-A*, a multi-agent design of the well-known A* algorithm. Authors test different configurations of the planner with two optimal heuristic functions, Merge&Shrink (53) and LM-Cut (51). These functions are however applied locally by each agent.

Authors in (106) introduce a multi-agent design of the h_{FF} heuristic. This adaptation, based on the use of *distributed Relaxed Planning Graphs* (dis-RPGs) (127), yields the same results as the original single-agent design of h_{FF} (54). However, the construction and exploration of a dis-RPG entails many communications between agents, resulting in a computationally expensive approach.

In (107), authors present the distributed design of several relaxation heuristics, namely h_{add}, h_{max} and a relaxed version of h_{FF}. In this work, authors replace the dis-RPG by an *exploration queue*, a more compact structure that significantly reduces the need of communications among agents. The distributed version of h_{FF}, however, does not yield the same results as the original single-agent version.

Finally, in (72), authors design a distributed version of a privacy-preserving landmarks extraction algorithm for MAP, resulting in a planner named GPPP. Authors show that the Landmarks Graph used in GPPP improves the performance of the *MA-STRIPS*-based planner MAFS (79). In GPPP, the heuristic value of the plan is calculated as the sum of the local heuristic estimates computed by each agent.

3.5.3 Multi-agent planning task formalization

In this section we present the formalization of a MAP task as used in the FMAP framework (116). Agents have a limited knowledge of the planning task, and it is assumed that the information that is not represented in the agent's model is unknown to the agent. The states of the world are defined through a finite set of *state variables*, $\mathcal{V}$, each of which is associated to a finite domain, $\mathcal{D}_v$, of mutually exclusive values that refer to the objects in the world. Assigning a value d to a variable $v \in \mathcal{V}$ generates a *fluent*, a tuple of the form $\langle v, d \rangle$. A *state* S is defined as a finite set of fluents.

An *action* is of the form $\alpha = PRE(\alpha) \rightarrow EFF(\alpha)$, where $PRE(\alpha)$ and $EFF(\alpha)$ are finite set of fluents representing the preconditions and effects of α, respectively. Executing an action α in a world state S leads to a new world state S' as a result of applying $EFF(\alpha)$ over S. An effect of the form $\langle v, d \rangle$ updates S' w.r.t. S, replacing the fluent $\langle v, d' \rangle \in S$ by $\langle v, d \rangle$. Since values in $\mathcal{D}_v$ are mutually exclusive, the inclusion of $\langle v, d \rangle$ in S' implies that $\forall d' \in \mathcal{D}_v, d' \neq d, \langle v, d' \rangle \notin S'$.

Definition 3.16. *A **MAP task** is a tuple* $\mathcal{T}_{MAP} = \langle \mathcal{AG}, \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$. $\mathcal{AG} = \{1, \ldots, n\}$ *is a finite non-empty set of agents.* $\mathcal{V} = \bigcup_{i \in \mathcal{AG}} \mathcal{V}^i$, *where* $\mathcal{V}^i$ *is the set of state variables known to an agent* i. $\mathcal{I} = \bigcup_{i \in \mathcal{AG}} \mathcal{I}^i$ *is a set of fluents that defines the initial state of* $\mathcal{T}_{MAP}$. *Since specialized agents are allowed, they may only know a subset of* $\mathcal{I}$*; the initial states of two agents never contradict each other.* $\mathcal{G}$ *is a set of fluents defining the goals of* $\mathcal{T}_{MAP}$. *Finally,* $\mathcal{A} = \bigcup_{i \in \mathcal{AG}} \mathcal{A}^i$ *is the set of planning actions of the agents.* $\mathcal{A}^i$ *and* $\mathcal{A}^j$ *of two specialized agents* i *and* j *will be typically disjoint sets; otherwise,* $\mathcal{A}^i$ *and* $\mathcal{A}^j$ *may overlap.* $\mathcal{A}$ *includes two fictitious actions* α_0 *and* α_f *that do not belong to any particular agent:* α_0 *represents the initial state of* $\mathcal{T}_{MAP}$, *while* α_f *represents the goal state.*

The *view* of an agent i on $\mathcal{T}_{MAP}$ is defined as $\mathcal{T}^i_{MAP} = \langle \mathcal{V}^i, \mathcal{A}^i, \mathcal{I}^i, \mathcal{G} \rangle$. $\mathcal{V}^i$ is the set of state variables known to agent i; $\mathcal{A}^i \subseteq \mathcal{A}$ is the set of its capabilities (planning actions); $\mathcal{I}^i$ is the subset of fluents of the initial state $\mathcal{I}$ that are known to agent i, and $\mathcal{G}$ is the set of goals, which are known to all the agents in $\mathcal{T}_{MAP}$. An agent i may also have a partial view on the domain $\mathcal{D}_v$ of a variable v. We define $\mathcal{D}^i_v \subseteq \mathcal{D}_v$ as the subset of values of v known to agent i.

Agents interact by sharing information about their state variables. For a pair of agents i and j, the information they share is defined as $\mathcal{V}^{ij} = \mathcal{V}^{ji} = \mathcal{V}^i \cap \mathcal{V}^j$. Additionally, the set of values of a variable v shared by agents i and j is defined as $\mathcal{D}^{ij}_v = \mathcal{D}^i_v \cap \mathcal{D}^j_v$.

FMAP follows a forward-chaining POP approach which has been adapted to a multi-agent context.

Definition 3.17. *A **partial-order plan** or partial plan is a tuple* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$. $\Delta = \{\alpha | \alpha \in \mathcal{A}\}$ *is the set of actions in* Π. $\mathcal{OR}$ *is a finite set of ordering constraints* ($\prec$) *on* Δ. $\mathcal{CL}$ *is a finite set of causal links of the form* $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$, *where* α *and* β *are actions in* Δ. *A causal link* $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ *enforces precondition* $\langle v, d \rangle \in PRE(\beta)$ *through an effect* $\langle v, d \rangle \in EFF(\alpha)$.

An *empty* partial plan is defined as $\Pi_0 = \langle \Delta_0, \mathcal{OR}_0, \mathcal{CL}_0 \rangle$, where $\mathcal{OR}_0$ and $\mathcal{CL}_0$ are empty sets, and Δ_0 contains only the fictitious initial action α_0.

The introduction of new actions in a partial plan may trigger the appearance of *flaws*: preconditions that are not yet supported in the plan, and threats. A *threat* over a causal link $\alpha \overset{\langle v,d \rangle}{\rightarrow} \beta$ is caused by an action γ not ordered w.r.t. α or β, where $(v = d') \in EFF(\gamma)$, $d' \neq d$. A *flaw-free* plan is a threat-free partial plan without unsupported preconditions.

Agents in FMAP jointly refine an initially empty plan until a solution is reached. We define a refinement plan as follows:

Definition 3.18. *A **refinement plan** $\Pi_r = \langle \Delta_r, \mathcal{OR}_r, \mathcal{CL}_r \rangle$ over a partial plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ is a flaw-free partial plan that extends Π by introducing an action α, resulting in $\Delta_r = \Delta \cup \alpha$. All the preconditions in $PRE(\alpha)$ are supported by existing actions in Π through causal links: $\forall p \in PRE(\alpha)$, $\exists\ \beta \overset{p}{\rightarrow} \alpha \in \mathcal{CL}_r$, where $\beta \in \Delta$.*

For each refinement plan, FMAP computes the *frontier state* (6), that is, the state that results from executing the actions in the plan. Frontier states allow for the application of state-based heuristic functions.

Definition 3.19. *A **frontier state** $FS(\Pi)$ over a refinement plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ is the set of fluents $\langle v, d \rangle$ achieved by actions $\alpha \in \Delta \mid \langle v, d \rangle \in EFF(\alpha)$, such that any action $\alpha' \in \Delta$ that modifies the value of the variable v ($\langle v, d' \rangle \in EFF(\alpha') \mid d \neq d'$) is not applicable from α by following the orderings and causal links in Π.*

A *solution plan* for $\mathcal{T}_{MAP}$ is a refinement plan that achieves all the goals $\mathcal{G}$ of $\mathcal{T}_{MAP}$ by including the fictitious final action α_f and supporting all its preconditions, i.e., $\forall g \in PRE(\alpha_f)$, $\exists\ \beta \overset{g}{\rightarrow} \alpha_f \in \mathcal{CL}$, $\beta \in \Delta$.

3.5.3.1 Privacy in partial plans

Agents in FMAP carry out several distributed procedures that require communications. To keep privacy, only the information that is shared between the sender

and receiver agents is transmitted. To do so, the sender *encodes* the information that is not in the view of the receiver agent. Each variable and value has an associated unique global identifier, a positive integer that is used to mask the original variable or value when necessary.

When an agent i refines a plan Π by adding an action $\alpha \in \mathcal{A}^i$, it communicates such refinement to the rest of agents. To preserve privacy, agent i will only communicate to agent j the fluents in α whose variables are common to both agents. The information of Π that agent j receives from i configures its view of that plan. More specifically, given a fluent $\langle v, d\rangle$, where $v \in \mathcal{V}^i$ and $d \in \mathcal{D}_v^i$, FMAP identifies three cases:

- **Public fluent**: if $v \in \mathcal{V}^{ij}$ and $d \in \mathcal{D}_v^{ij}$, the fluent $\langle v, d\rangle$ is public to both agents, and thus agent i will share with agent j all the information regarding $\langle v, d\rangle$.

- **Private fluent to agent** i: if $v \notin \mathcal{V}^{ij}$, $\langle v, d\rangle$ is private to agent i w.r.t. j, and hence agent i will send j $\langle gid(v), gid(d)\rangle$, thus replacing v and d by their global identifiers, $gid(v)$ and $gid(d)$, respectively.

- **Partially private fluent to agent** i: if $v \in \mathcal{V}^{ij}$ but $d \notin \mathcal{D}_v^{ij}$, $\langle v, d\rangle$ is partially private to agent i w.r.t. j. Instead of $\langle v, d\rangle$, agent i will send j a fluent $\langle v, gid(d)\rangle$, thus replacing the value d by its global identifier $gid(d)$.

As well as keeping privacy during planning, encoding variables and values eases the design of global heuristic functions and streamlines communications among agents.

3.5.4 FMAP: multi-agent planning framework

FMAP is a fully-configurable distributed search procedure, an appropriate testbed for the integration of global heuristic functions. This section summarizes some of the key aspects of this MAP framework.

Algorithm 7: FMAP search algorithm for an agent i

```
openList ← Π_0
while openList ≠ ∅ do
    Π_b ← extractPlan(openList)
    if isSolution(Π_b) then
        return Π_b
    RP ← refinePlan(Π_b)
    for all j ∈ AG, j ≠ i do
        sendRefinements(j)
        RP ← RP ∪ receiveRefinements(j)
    for all Π_r ∈ RP do
        distributedEvaluation(Π_r, heuristic)
        openList ← openList ∪ Π_r
return No solution
```

FMAP is a cooperative refinement planning procedure in which agents jointly explore a multi-agent, plan-space search tree (see Algorithm 7). Nodes of the tree are partial plans contributed by one or several agents. The process is led by an agent that plays the *coordinator* role (this role is rotated after each iteration of the procedure).

Agents keep a common *openList* with the unexplored refinement plans prioritized according to a search criterion (by default, FMAP applies a weighted A* search, evaluating nodes through a function $f = g + 2 * h$). Agents jointly choose the best node of *openList* and then each of them individually expands the se-

lected plan through an embedded forward-chaining POP procedure, generating all the possible refinement plans. Afterwards, agents exchange the plans and apply a distributed heuristic evaluation of such plans, which are then inserted in the *openList*. The procedure ends up when a solution plan is found, or when *openList* is empty.

As in most distributed frameworks, communications play a central role in FMAP. The system is built on top of the Magentix2 MAS platform, which provides the basic libraries to define the agents' behavior, as well as the communication infrastructure required by FMAP. Agents communicate by means of the FIPA Agent Communication Language (81), and communications are managed by the Apache QPid message broker.

The communication broker acts as a post office, receiving the messages from the sender agents and forwarding them to the receivers. The use of a messaging broker offers some key advantages for the design of distributed systems since it allows agents to be launched in different machines, as long as the broker is accessible from the network. However, when the workload of messages is relatively high, the broker entails a bottleneck of the system. For this reason, the global estimators introduced in this paper have been designed and optimized to minimize the communications among agents.

3.5.5 Global heuristic functions

This section formalizes and details the distributed design of two different heuristic functions as well as a novel multi-heuristic approach to MAP that combines both functions, noticeably improving the performance of the FMAP system.

The first heuristic, h_{DTG}, is a variation of the Context-Enhanced Additive Heuristic (52) that uses Domain Transition Graphs (DTGs) to estimate the cost of the state variables. The second one, h_{Land}, computes the Landmarks Graph

(LG) of a MAP task, which is later used to calculate the number of landmarks of the partial plans. We designed a distributed version of the landmarks extraction algorithm introduced in (55).

The design of h_{DTG} and h_{Land} in FMAP aims to keep the number of messages exchanged among the agents as low as possible. Prior to the search process, we build data structures, like the DTGs or the LG, which remain immutable throughout the multi-agent search, thus reducing the communication overload during search. In contrast to other constructs, such as dis-RPGs (127), the DTGs and the LG do not need to be re-calculated during search. The use of static structures makes h_{DTG} and h_{Land} be more suitable heuristics for fully-distributed systems than other well-known heuristic functions, such as h_{FF} (106), that requires the generation of a dis-RPG at each search node.

Besides h_{DTG} and h_{Land}, we also introduce MH-FMAP, a multi-heuristic adaptation of the FMAP algorithm that alternates both heuristics, successfully improving the overall performance of the MAP system.

3.5.5.1 DTG heuristic

This is a state-based additive heuristic calculated from the DTGs (49). A DTG is a graph in which nodes represent values of a particular variable, and transitions show the changes in the values of such variable through the actions of the agents. An action of the form $\langle v, d_0\rangle \rightarrow \langle v, d_n\rangle$ induces a transition $d_0 \rightarrow d_n$ in the DTG associated to v.

Similarly to the Context-Enhanced Additive heuristic (h_{CEA}) (52), h_{DTG} builds a relaxed plan and reuses the side effects of the actions in the relaxed plan as a basis to estimate the cost of the subsequent subgoals. A plan Π of FMAP is always evaluated from its frontier state, $FS(\Pi)$, but the cost of some of the subgoals can be estimated in a state different from $FS(\Pi)$.

Formally, the formulation of h_{DTG} is very close to h_{CEA}. Given a subgoal $g = \langle v, d \rangle$, a state S and an action $\alpha \in \mathcal{A}$, where $g \in EFF(\alpha)$, $g' = \langle v, d' \rangle \in PRE(\alpha)$, $d' \neq d$, and $z = PRE(\alpha) \setminus \{g'\}$, evaluating g in S with h_{DTG} is recursively defined as follows:

$$h_{DTG}(g|S) = \begin{cases} 0 \text{ if } g \in S \\ \min\limits_{\alpha:(g',z \to g) \in \mathcal{A}} (1 + h_{DTG}(g'|S) + \\ \sum\limits_{x \in z} h_{DTG}(x|S')) \text{ otherwise} \end{cases} \tag{3.1}$$

The recursive equation 3.1 expresses that the precondition g', related to the same variable v as the fluent g, is also evaluated in S, whereas the rest of preconditions, $x = \langle v', d' \rangle \in PRE(\alpha)$, $v' \neq v$ can be evaluated in a state S' different from S.

Following, we describe in detail the h_{DTG} algorithm to clarify aspects such as the evaluation of a subgoal g, the selection and insertion in the relaxed plan of the action α that minimizes equation 3.1 or the selection of the states S' from which the preconditions x of equation 3.1 are evaluated.

Instead of exploring the Causal Graph as h_{CEA} does, h_{DTG} explores the DTGs. The algorithm maintains a *subGoals* list that stores the subgoals of the problem that are not yet evaluated (this list is initialized as $subGoals = \mathcal{G}$) and a $sideEffects$ list that maintains the side effects of the actions added to the relaxed plan (initially, $sideEffects = FS(\Pi)$). The heuristic h_{DTG} builds a relaxed plan by finding in the DTGs the shortest paths between the fluents in $sideEffects$ and *subGoals* via the application of the Dijkstra algorithm.

We first introduce some notions that are needed for the h_{DTG} algorithm:

- $minPath(v, d_0, d_n) = \{d_0, \ldots, d_{n-1}, d_n\}$ is the shortest path between $\langle v, d_0 \rangle \in sideEffects$ and $\langle v, d_n \rangle \in subGoals$, where d_0 is the initial value of the path

and d_n is the final value of the variable or subgoal to be achieved.

- $getAction(v, d_{n-1}, d_n)$ is the minimum cost action that induces a value transition $d_{n-1} \rightarrow d_n$.

Subgoals are sorted according to their cost. We define the cost of a subgoal $g = \langle v, d_n \rangle$ as follows:

$$cost(g) = \underset{\langle v, d_0 \rangle \in sideEffects}{\arg\min} |minPath(v, d_0, d_n)|$$

The cost of an action α is defined in terms of its preconditions:

$$cost(\alpha) = \sum_{p = \langle v, d_n \rangle \in PRE(\alpha)} cost(p)$$

The h_{DTG} algorithm extracts the subgoal $g = \langle v, d_n \rangle \in subGoals$ that maximizes $cost(g)$. Then, $minPath(v, d_0, d_n)$ is applied to all the values d_0 such that $\langle v, d_0 \rangle \in sideEffects$. From all the obtained paths, the algorithm chooses the shortest one, $p = \{d_0, \ldots, d_{n-1}, d_n\}$.

Once the shortest path p is known, the algorithm introduces in the relaxed plan the minimum cost action α that induces each transition in p. That is, given, for instance, the last value transition in p, $d_{n-1} \rightarrow d_n$, the algorithm applies $getAction(v, d_{n-1}, d_n)$, obtaining an action α such that $\langle v, d_{n-1} \rangle \in PRE(\alpha)$ and $\langle v, d_n \rangle \in EFF(\alpha)$.

The effects of the action α for each value transition in p are inserted in the relaxed plan and stored in $sideEffects$, and the rest of preconditions of α, $\langle v', d' \rangle$, are inserted in the $subGoals$ list. Then, a new iteration of h_{DTG} starts with a new subgoal $g \in subGoals$.

Note that, as stated in equation 3.1, the cost of all preconditions related to the same variable v is estimated from the same state as $g = \langle v, d_n \rangle$ since they

are solved in the same iteration of the process using the path p as a reference. The cost of the rest of preconditions $g' = \langle v', d' \rangle$, for variables $v' \neq v$, might be estimated from a state S' different from the state of g, depending on the fluent selected from $sideEffects$ to compute the cost of g'.

The process is completed when all the subgoals in *subGoals* are processed. h_{DTG} returns the number of actions in the relaxed plan as an estimate of the cost of the plan.

To preserve privacy, each agent i stores its own version of the DTGs according to its knowledge of the planning task. Given a state variable v, agent i only keeps the DTG nodes and transitions that involve the values in $\mathcal{D}_v^i$. The rest of transitions are replaced by a reference to the agents that can realize such transition. For instance, given a transition $d_{n-1} \rightarrow d_n$, where $\mathcal{D}_v^i = \{d_{n-1}\}$ and $\mathcal{D}_v^j = \{d_{n-1}, d_n\}$, agent i maintains a transition $d_{n-1} \rightarrow j$, which indicates agent i that it must communicate with agent j in order to retrieve the cost of the transition. This way, the calculation of h_{DTG} preserves agents' privacy.

When *minPath* is applied in a distributed context, agent i may have to resort to another agent j to find out the cost of a subpath that is not visible to i. In turn, agent j may also require the assistance of another agent k. To prevent an excessive number of messages among agents, the *recursion depth* of requests is limited during the application of h_{DTG}.

3.5.5.2 Landmarks heuristic

This heuristic uses *landmarks*, fluents that must be satisfied in every solution plan of a MAP task, as the basis of its calculation.

Agents jointly generate the Landmarks Graph (LG). Formally, LG $= \{N, V\}$, where N is a set of nodes (landmarks) and V is a set of orderings between the

nodes. Among the different types of orderings between landmarks, we use *necessary orderings*, which are directly inferred with the algorithm presented in (55). A necessary ordering of the form $l' \leq_n l$ indicates that the landmark l' should be achieved before l in all the solution plans for the task. *Single* landmarks contain only one fluent, while *disjunctive* landmarks are composed of a set of fluents, where one of them must be true in all the solution plans.

Algorithm 8 shows the distributed landmark extraction algorithm. This multi-agent procedure is described from the point of view of one agent i. In order to ensure privacy, all the fluents transmitted in Algorithm 8 are encoded as described in subsection 3.5.3.1. As a result of the execution of this algorithm, each agent i will obtain a version of the LG which includes only the landmarks that are public to i.

The algorithm is a backwards process that departs from the goals in $\mathcal{G}$. Given a landmark l, the process finds new landmarks as the preconditions that are common to all the actions that yield l as an effect. Once a landmark l' is inferred from l, a necessary ordering $l' \leq_n l$ is also established. Before their inclusion in the LG, all the single landmarks and necessary orderings must be *verified* to ensure their correctness.

An iteration of the Algorithm 8 is conducted by an agent that plays the role of coordinator (in the following, we reference in parenthesis the lines of Algorithm 8 in which each task is performed). Since actions are distributed across agents, the detection of single landmarks, from the viewpoint of an agent i, is described as follows:

- When a landmark l is extracted for its analysis (line 3), agent i calculates $candidates^i$ as the intersection of the preconditions of $producers^i$, the actions in $\mathcal{A}^i$ that yield l as an effect (lines 4-5).

Algorithm 8: LG construction algorithm for an agent i

$N \leftarrow \emptyset$, $V \leftarrow \emptyset$, $landmarks \leftarrow \mathcal{G}$
while $landmarks \neq \emptyset$ **do**
  $l \leftarrow extractLandmark(landmarks)$
  $producers^i \leftarrow \alpha \in \mathcal{A}^i \mid l \in EFF(\alpha)$
  $candidates^i \leftarrow \cap_{\forall \alpha \in producers^i} PRE(\alpha)$
  $disj^i \leftarrow groupNonCommonPrecs(producers^i)$
  if $isCoordinator(i)$ **then**
    $lm \leftarrow candidates^i$, $disj \leftarrow \{disj^i\}$
    for all $j \in \mathcal{AG}, j \neq i$ **do**
      $receive(\{disj^j, candidates^j\}, j)$
      $lm \leftarrow lm \cap candidates^j$
      $disj \leftarrow disj \cup disj^j$
    $lm \leftarrow lm \cup groupDisjLandmarks(disj)$
    $\forall j \in \mathcal{AG}, j \neq i, send(lm, j)$
  else
    $send(\{disj^i, candidates^i\}, coordinator)$
    $lm \leftarrow receive(lm, coordinator)$
  for all $l' \in lm$ **do**
    if $isDisjunctive(l') \vee verify(l') = true$ **then**
      $N \leftarrow N \cup l'$
      $V \leftarrow V \cup l' \leq_n l$
      $landmarks \leftarrow landmarks \cup l'$
  Rotate coordinator role
for all $l' \leq_n l \in V$ **do**
  if $verify(l' \leq_n l) = false$ **then**
    $V \leftarrow V \setminus \{l' \leq_n l\}$
return $LG = \{N, V\}$

- Agent i masks the fluents in $candidates^i$ according to its level of privacy w.r.t. the coordinator agent. Then, i transmits $candidates^i$ to the coordinator agent (line 16), which applies the intersection of the sets of candidates received from all the agents in order to compute the actual set of landmark candidates called lm (line 11).

Agent i groups the preconditions of $producers^i$ that are not in $candidates^i$ according to its variable in order to generate disjunctive landmarks (line 6). For instance, let $producers^i = \{(\langle v, d_{n-1}\rangle, \langle v', d'\rangle) \rightarrow \langle v, d_n\rangle, (\langle v, d_{n-1}\rangle, \langle v', d''\rangle) \rightarrow \langle v, d_n\rangle\}$; then $candidates^i = \{\langle v, d_{n-1}\rangle\}$ and $disj^i = \{(\langle v', d'\rangle, \langle v', d''\rangle\}$ Agent i sends $disj^i$ along with $candidates^i$ to the coordinator agent, which groups together the disjunctive landmarks received from the agents, inserts them in the set lm (line 13) and sends lm back to the agents (lines 14 and 17).

In the next step, agents jointly verify the single landmark candidates $l' \in lm$ (line 19). The verification of l' entails solving a relaxed problem in which the actions α such that $l' \in EFF(\alpha)$ are excluded. If the goals in $\mathcal{G}$ are not satisfied then l' is verified as a landmark. If l' is verified, it is added to the LG along with a necessary order $l' \leq_n l$ (lines 20-21). For the verification of landmarks, agents are required to jointly generate a dis-RPG (127).

Note that, in order to preserve privacy, agent i stores l' and the associated ordering $l' \leq_n l$ in its LG only if l' is public to i. This way, agents will keep different versions of the LG.

When the extraction and verification of landmarks is completed, the next step is the verification of the orderings in the LG (*forall* loop in lines 24-26). Given an ordering $l' \leq_n l$, agents create a dis-RPG excluding the actions $\alpha \in \mathcal{A} \mid l' \in PRE(\alpha) \wedge l \in EFF(\alpha)$ in order to validate it.

The LG created in Algorithm 8 is used to calculate the value of h_{Land} of a refinement plan in FMAP. Given a plan Π, $h_{Land}(\Pi)$ returns an estimate of the quality of Π, which is estimated as follows:

1. The agent i that generates Π checks which landmarks are satisfied in Π according to its LG (agent i coordinates the evaluation of Π). A refinement plan Π satisfies a landmark l iff $\exists \alpha \in \Delta(\Pi) \mid l \in EFF(\alpha)$, and $\forall l' \leq_n l \in N$, $l' \in EFF(\beta)$, where $\beta \in \Delta(\Pi)$ and $\exists \beta \prec \alpha \in \mathcal{OR}(\Pi)$; that is, a landmark l is not satisfied unless all its predecessors in the LG appear in Π as effects of the actions that precede the action α that has l in its effects.

2. Agent i communicates the verified landmarks to each agent j, $j \neq i$, masking the variables and values according to the level of privacy with agent j (see subsection 3.5.3.1). Then, agent j verifies whether Π achieves any more landmarks that are not visible in the LG of the coordinator agent i.

3. Agents mask the new found landmarks and send them to the coordinator agent i, which computes the value of $h_{Land}(\Pi)$ as the number of landmarks that are not satisfied in Π.

The communication machinery required for the calculation of h_{Land} has been integrated into FMAP by reusing the messages of the original protocol, and thus, its distributed calculation does not increase the communication overhead.

3.5.5.3 Multi-heuristic approach

Over the last years, one of the most successful research trends on single-agent state-based planning emphasizes the combination of heuristic functions. Recent studies conclude that the combination of multiple heuristics dramatically improves performance and scalability in planning (96). This conclusion is backed up by some

Algorithm 9: MH-FMAP algorithm for an agent i

```
openList ← Π_0, preferredList ← ∅
list ← true
while openList ≠ ∅ do
    if list = true then
        Π_b ← extractPlan(openList)
    else
        Π_b ← extractPlan(preferredList)
    list ← ¬list
    if isSolution(Π_b) then
        return Π_b
    RP ← refinePlan(Π_b)
    for all j ∈ AG, j ≠ i do
        sendRefinements(j)
        RP ← RP ∪ receiveRefinements(j)
    for all Π_r ∈ RP do
        distributedEvaluation(Π_r, h_DTG)
        distributedEvaluation(Π_r, h_Land)
        openList ← openList ∪ Π_r
        if h_Land(Π_r) < h_Land(Π_b) then
            preferredList ← preferredList ∪ Π_r
return No solution
```

well-known planning systems, such as Fast Downward (50) and LAMA (95), which successfully apply a multi-heuristic approach to state-based planning. Up to this date, however, the multi-heuristic approach has never been tested in MAP.

A basic question that arises when modeling a multi-heuristic approach is *how* to combine heuristics in order to maximize the performance of the resulting planner. The work in (96) experimentally compares different heuristic combination methods (sum, weighted sum, maximum, Pareto and alternation of heuristics), concluding

that the *alternation* of heuristics is by far the most efficient method.

Our multi-heuristic MAP approach, MH-FMAP, is a heuristic alternation mechanism. Rather than aggregating the heuristic values, alternation makes equal use of all the estimators, assuming that different heuristics might be useful in different parts of the search space. The most promising states are selected according to the currently used heuristic, completely ignoring all other heuristic estimates (96).

MH-FMAP is inspired by Fast Downward, which combines the FF and Causal Graph heuristics in an orthogonal way. Fast Downward maintains two open lists per heuristic: one list stores the open nodes and the other one keeps track of the *preferred successors*. While authors in (50) defined preferred successors as the ones generated by the so-called preferred operators, we define them by means of the landmark-based heuristic:

Definition 3.20. *A refinement plan* Π_r *is a* ***preferred successor*** *of a plan* Π *iff* $h_{Land}(\Pi_r) < h_{Land}(\Pi)$.

Algorithm 9 shows the FMAP basic search scheme adapted to our multi-heuristic approach, MH-FMAP. Agents now maintain two open lists: the *openList* maintains the open nodes of the search tree, ordered by $f = g + 2 * h_{DTG}$, and the $preferredList$ keeps only the preferred successors, sorted by h_{Land}. If a plan is a preferred successor, it is introduced in both open lists. Agents extract a base plan from one of the lists alternatively; if a base plan is stored in both lists, it is removed from both of them.

The results of the next section prove that MH-FMAP yields notably superior results than the individual heuristics.

Domain-Tasks	MH-FMAP					FMAP- h_{DTG}				
	Sol	#Iter	#Act	MS	Time	Sol	#Iter	#Act	MS	Time
Depots-20	**12**	614,75	31,83	24,50	141,57	8	9,46x	1,23x	0,97x	5,55x
Driverlog-20	**15**	400,73	22,93	13,07	18,24	**15**	0,62x	1,04x	1,15x	0,60x
Elevators-30	**30**	53,93	24,67	13,40	9,43	**30**	0,66x	1,00x	0,96x	0,56x
Logistics-20	**20**	128,85	69,95	20,75	100,25	10	2,83x	1,05x	1,20x	5,43x
MA-Blocksworld-34	22	2542,36	18,18	14,64	45,71	**23**	0,96x	1,09x	1,06x	0,96x
Openstacks-30	**30**	707,80	63,10	52,90	353,73	25	0,75x	1,02x	0,95x	1,14x
Rovers-20	**20**	507,50	35,05	14,35	95,55	19	1,08x	1,01x	1,03x	1,50x
Satellite-20	**19**	72,74	32,58	19,95	115,05	18	0,92x	0,99x	0,97x	0,93x
Woodworking-30	**27**	1331,74	19,48	4,81	197,78	23	0,51x	1,01x	1,02x	0,40x
Zenotravel-20	**20**	96,65	32,35	18,35	115,68	**20**	0,94x	0,99x	0,97x	0,95x
Global results	**215**	670,56	35,59	20,37	128,51	191	0,95x	1,02x	0,99x	1,06x

Domain-Tasks	MH-FMAP					FMAP- h_{Land}				
	Sol	#Iter	#Act	MS	Time	Sol	#Iter	#Act	MS	Time
Depots-20	**12**	614,75	31,83	24,50	141,57	7	2,58x	0,96x	0,81x	0,57x
Driverlog-20	**15**	400,73	22,93	13,07	18,24	7	349,37x	0,90x	1,03x	51,78x
Elevators-30	**30**	53,93	24,67	13,40	9,43	13	585,47x	0,97x	0,92x	49,28x
Logistics-20	**20**	128,85	69,95	20,75	100,25	10	8,68x	1,00x	0,98x	1,75x
MA-Blocksworld-34	22	2542,36	18,18	14,64	45,71	16	11,44x	0,98x	0,91x	7,89x
Openstacks-30	**30**	707,80	63,10	52,90	353,73	**30**	0,18x	1,02x	1,02x	0,05x
Rovers-20	**20**	507,50	35,05	14,35	95,55	6	7,14x	0,99x	1,00x	1,21x
Satellite-20	**19**	72,74	32,58	19,95	115,05	4	22,01x	1,02x	1,00x	6,42x
Woodworking-30	**27**	1331,74	19,48	4,81	197,78	17	0,95x	0,97x	1,01x	0,13x
Zenotravel-20	**20**	96,65	32,35	18,35	115,68	7	155,19x	0,97x	1,03x	20,50x
Global results	**215**	670,56	35,59	20,37	128,51	117	11,32x	1,00x	0,99x	0,53x

Table 3.9: Comparison between MH-FMAP and FMAP (using h_{DTG} and h_{Land})

3.5.6 Experimental results

We executed a wide range of experimental tests in order to assess the performance of the heuristic strategies presented in this paper[1]. Our benchmark includes the *STRIPS* suites of 10 different domains from the IPC[2], all of them adapted to a MAP context: *Depots*, *Driverlog*, *Elevators*, *Logistics*, *MA-Blocksworld*, *Openstacks*, *Rovers*, *Satellite*, *Woodworking* and *Zenotravel*. All the tasks were directly adapted from the *STRIPS* IPC suites, except for the *MA-Blocksworld* domain (11),

[1] All the experimental tests were performed on a single machine with a quad-core Intel Core i7 processor and 8 GB RAM (2 GB RAM available for the Java VM).

[2] For more details on the MAP adaptation of the planning domains, please refer to (116).

which introduces several arms that can simultaneously manipulate the blocks (4 agents per task).

The first experiment, shown in Table 3.9, compares the performance of FMAP with the h_{DTG} heuristic ($f = g+2*h_{DTG}$), the h_{Land} heuristic ($f = g+2*h_{Land}$) and MH-FMAP, our novel multi-heuristic approach based on the alternation of h_{DTG} and h_{Land}.

Due to the large amount of performed tests (244 planning tasks), we only display average results. More precisely, Table 3.9 summarizes, for each domain, the total number of solved tasks (*Sol* columns) and the average results of: search iterations (*#Iter* columns), execution time in seconds (*Time* columns), and plan quality results in terms of number of actions (*#Act* columns) and *makespan* (*MS* columns). The results of h_{DTG} and h_{Land} are relative to the results obtained with MH-FMAP, considering only the common tasks solved by both MH-FMAP and the respective single-heuristic approach. The nx values in Table 3.9 indicate "n times as much as the MH-FMAP result". Therefore, a value higher than 1x in *#Act*, *MS*, *Time* or *#Iter* is a better result for MH-FMAP.

The *Sol* columns of h_{DTG} and h_{Land} represent the number of problems solved by each heuristic, which happens to coincide, except for the *MA-Blocksworld* domain, with the number of common tasks solved by both MH-FMAP and h_{DTG} and h_{Land}, respectively. The last row of Table 3.9 displays the global average results.

MH-FMAP obtains the best coverage results in 9 out of the 10 tested domains, solving 215 out of 244 tasks (roughly 88% of the tasks). h_{DTG} solves one more problem than MH-FMAP in the *MA-Blocksworld* domain and it solves overall 191 tasks (78%). Using h_{Land} as a standalone estimator shows the worst performance, solving 117 tasks (48%).

It is worth noting that MH-FMAP tends to mimic the behaviour of the best-performing heuristic in most of the domains: for instance, in *Driverlog*, *Elevators*,

MA-Blocksworld or *Zenotravel*, the results of coverage are much better with h_{DTG} than with h_{Land} and this is also reflected in MH-FMAP. However, h_{Land} solves more problems than h_{DTG} in the *Openstacks* domain and MH-FMAP equals the results obtained with h_{Land}. Interestingly, in the domains where h_{DTG} and h_{Land} offer a similar performance (namely, *Depots*, *Logistics* and *Woodworking*), the synergy of both estimators in MH-FMAP clearly outperforms the single-heuristic approaches, even resulting in twice as much the coverage in the *Logistics* domain.

h_{Land} takes much less time to evaluate a plan than the rest of approaches (33 ms per iteration in average, while MH-FMAP and h_{DTG} take around 200 ms).This is because, unlike h_{DTG}, the integration of h_{Land} in FMAP does not require any exchange of additional messages between agents apart from those already required by the FMAP search procedure. Nevertheless, h_{Land} requires the largest amount of iterations to find solutions in most domains; for instance, in *Driverlog* and *Elevators*, h_{Land} takes 350 and 585 times more iterations than MH-FMAP, respectively. In general, the accuracy of h_{Land} depends on the quality of the Landmarks Graph (LG). Particularly, in the *Openstacks* domain, the LG almost provides an skeleton for the solution plans, which explains the great performance of h_{Land} in this domain.

MH-FMAP requires more iterations and execution time than h_{DTG} in 6 out of the 10 tested domains. However, in general, MH-FMAP shows low execution times: less than 3 minutes in most domains, and around 6 minutes in *Openstacks*, the most time-consuming domain. Moreover, MH-FMAP performs admirably well in some domains, being around 5 times faster than h_{DTG} in *Depots* and *Logistics*.

Regarding plan quality (number of actions and makespan), MH-FMAP offers a good tradeoff between h_{DTG} and h_{Land}. According to the global results in Table 3.9, the quality results of the three approaches are very similar, being $\#Act$ slightly

higher in h_{DTG}. As a whole, we can observe that the alternation of heuristics does not entail a loss of quality versus the standalone heuristics.

To sum up, h_{Land} turns out to be the fastest approach with the worst coverage. h_{DTG} is the slowest approach but it solves many more problems than h_{Land}. MH-FMAP, however, shows the potential of alternating global heuristics in MAP: it remarkably improves the coverage up to 88% of solved problems and, despite the overhead caused by the simultaneous application of two heuristics, it offers competitive execution times. Finally, the combination of heuristics does not reduce the quality of the solution plans.

The second test compares MH-FMAP to another landmark-based approach to MAP, the Greedy Privacy Preserving Planner (GPPP). GPPP is the current best-performing *MA-STRIPS* planner and it introduces PP-LM, the first distributed version of a landmark-based heuristic (72)[1].

Both PP-LM and h_{Land} build the LG and evaluate plans by counting the landmarks of the LG that are not reached yet. However, each heuristic is built upon a different planning framework (MH-FMAP and GPPP), presenting some key differences among them. PP-LM is designed for propositional *MA-STRIPS* domains, while h_{Land} supports tasks where facts are modeled through object fluents. In addition, the two heuristics are designed around a different notion of privacy: in GPPP, the private literals of an agent are occluded to the rest of agents, and the public literals are visible to all the participants. In contrast, MH-FMAP defines privacy between each pair of agents, masking the private information in preconditions and effects.

Table 3.10 compares the coverage, average execution time and plan quality of MH-FMAP and GPPP. Note that GPPP develops sequential plans, so the plan duration (makespan) equals the number of actions in this approach. Figures in Table

[1]We want to thank the authors of GPPP for their kind support.

Domain-Tasks	MH-FMAP				GPPP		
	Sol	#Act	MS	Time	Sol	#Act	Time
Elevators-30	**30**	24,04	13,25	8,90	28	26,71	0,72
Logistics-20	**20**	69,95	20,75	100,25	**20**	69,25	2,02
Rovers-20	**20**	28,88	12,29	25,63	17	32,12	3,25
Satellite-20	19	32,58	19,95	115,05	**20**	38,32	3,44
Zenotravel-20	**20**	32,35	18,35	115,68	**20**	45,00	13,86

Table 3.10: Comparison between MH-FMAP and GPPP

3.10 show average results for both approaches when running five IPC domains used in (72).

Table 3.10 shows that GPPP is much faster than MH-FMAP (up to 50 times faster in some domains), mainly because, unlike MH-FMAP, GPPP does not use any communication infrastructure. As commented before, the use of a communication broker may entail a bottleneck when agents exchange a large amount of messages.

However, this superiority is not reflected in the coverage results. Despite being slower, MH-FMAP solves 109 out of 110 tasks, five more tasks than GPPP, which outnumbers MH-FMAP in only one task in the *Satellite* domain.

With respect to plan quality, MH-FMAP returns solution plans with fewer actions than GPPP in almost all the domains. For example, in *Zenotravel*, the solution plans of MH-FMAP contain 30% fewer actions than GPPP in average. GPPP only obtains slightly better results in the *Logistics* domain. Additionally, the POP-based approach of MH-FMAP allows us to obtain much shorter solutions (better makespan) than GPPP, which is limited to sequential plans.

In conclusion, MH-FMAP proves that the alternation of global heuristics is as effective in MAP as it is in classical planning. MH-FMAP not only performs much better than the single-heuristic FMAP setups, but also outperforms GPPP in terms of coverage and plan quality.

3.5.7 Conclusions

In this paper, we have presented MH-FMAP, a multi-agent planning system that draws upon the FMAP framework and incorporates a novel multi-heuristic search scheme that alternates two global heuristics: h_{DTG} and h_{Land}. We compared the performance of MH-FMAP against the standalone heuristics and GPPP, an *MA-STRIPS*-based planner, and the results throw a very positive balance in favor of MH-FMAP: a clearly superior coverage and a much better solution plan quality. In contrast, these excellent results come at the cost of a high number of message-passings between the agents.

The take-home lessons from this paper are: a) the use of global heuristics in MAP are actually worthy as long as the gain of the heuristic pays off the communication cost; b) the alternation of heuristics shows very beneficial for planning in general and also for MAP; c) using communication infrastructures is costly and affects the execution time but it is, however, necessary in order to implement heuristics in distributed environments with private information.

All in all, a proper combination of global heuristic estimators, well-defined communication protocols and a multi-heuristic search mechanism results in an ideal approach to cooperative MAP in distributed environments.

4

General discussion on the results

This chapter discusses the contributions of the work presented in the selected papers of Chapter 3. Besides analysing the main contributions of this work, we also summarize the experimental results obtained with our MAP system and discuss the ongoing and future research trends of the MAP community. In order to perform a thorough analysis of the performance of our system, we consider not only the results presented in the selected papers of Chapter 3, which compare the performance of MAP-POP, FMAP and MH-FMAP against other state-of-the-art MAP systems, but also the outcome of the 2015 Competition of Distributed and Multi-Agent Planners (CoDMAP)[1], in which MH-FMAP took part (118).

This chapter is organized as follows: section 4.1 summarizes the contributions of our work; section 4.2 analyzes the experimental results of the MAP frameworks presented in this PhD dissertation; and finally, section 4.3 briefly discusses the future directions of the MAP research area.

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

4.1 Summary of contributions

The focus of this dissertation is on the design and development of techniques that address the cooperative Multi-Agent Planning (MAP) problem. The compendium of articles of Chapter 3 are a comprehensible analysis of the main contributions of our work and the chronological evolution of the research.

The articles in Chapter 3 provide a general overview of our MAP framework at the time the papers were published and, therefore, further clarifications are required to assess the individual achievements of this PhD thesis. This section is thus devoted to classify and summarize the principal individual contributions of this dissertation.

Aside from the state of the art in single-agent planning and MAP, presented in Chapter 2, the contributions of this work include the formal definition of a cooperative MAP task, a definition language to specify MAP tasks with private information, a complete search framework for the distributed resolution of MAP tasks, and a set of global MAP heuristic functions to guide the search process of the resolution framework. The following subsections classify and analyze these individual contributions.

4.1.1 Theoretical formalization of a cooperative MAP task

One of the first steps of this investigation was the formal definition of the elements that constitute a MAP task. Our formal definition of a MAP task, which has been progressively updated and revised across the published articles, can be found in sections 3.2.2, 3.3.5, 3.4.3 and 3.5.3.

The formalization involves three main aspects: 1) the definition of the components of a MAP task, 2) the formal modelling of our notion of privacy, and 3)

the generalization of the (backward and forward) POP search scheme to a MAP context.

We model a cooperative MAP task as an extension of a single-agent task, so that the task actions are divided among a finite set of independent planning entities or agents. As opposite to classical single-agent *STRIPS* tasks, we model the facts that describe the world through a set of state variables, each associated to a finite domain of objects of the world. The facts of the world are thus described through a set of *fluents* that assign a specific value to a state variable. State variables are a more comprehensive mechanism for the specification of a planning task; in fact, many of the techniques and algorithms developed for our MAP framework take explicit advantage of the state-variable-based task modelling.

The information of the MAP task, such as state variables, actions and initial state of the task, is distributed among the agents. The only exception is related to the task goals: since agents are fully cooperative, private objectives are not considered in our formalization, and thus, all the task goals are shared by the participating agents.

The distribution of information stresses the second key aspect of the formalization; that is, the definition of *privacy*. Each agent possesses a limited view of the world according to the state variables or values of the variables known to the agent. Fluents can be either *public* to an agent (that is, the agent knows both the variable and its assigned value), *private* (the agent does not know the variable), or *partially private* (the agent knows the variable but not its assigned value).

As described in section 2.2.3.1, most current MAP privacy models are based on MA-STRIPS, and thus, they define an agent's private information by occluding it to every other agent in the task. In contrast, we introduced a more realistic and

sophisticated theoretical approach to privacy by establishing privacy *pairwise*, so that a fluent can be known only to *a subset* of agents.

The privacy constraints established in the MAP task formalization are applied in all the algorithms and procedures of our framework, thus making privacy preservation one of the main contributions of this research work.

The final aspect of our MAP task formalization is the definition of the planning procedure that agents carry out to solve the task. Initially, the multi-agent resolution procedure was configured as a POP search, which eventually evolved to a forward-chaining POP setting (see section 4.1.3). While the search algorithm was changed and adapted accordingly to the research progress, the concept of *refinement planning* was maintained across the different versions of the MAP framework: agents jointly search the solution space by progressively refining an initially empty plan until a solution is found.

The implementation of the multi-agent search uses a compact plan representation (information of a search node) where the refinement plans are contributed with the actions of several agents. Each agent maintains its own local version of the POP search tree, where the refinement plans of the tree encode the information known to the agent according to the privacy rules of the task.

Additionally, we guarantee that the application of the privacy constraints does not compromise the soundness of the search process (see section 3.2.3.3). Therefore, all the agents expand the multi-agent search tree correctly despite their limited view of the refinement plans.

4.1.2 Multi-agent planning task definition language

The special requirements of a MAP task, such as information distribution or privacy, are not supported by the different single-agent *PDDL* versions. Moreover,

the MAP community has not agreed on a *de facto* standard language until the recent 2015 CoDMAP competition, in which *MA-PDDL* was introduced (see section 2.2.2). In 2012, we defined our own MAP specification language, whose syntax is described in detail in section 3.3.6.

Our language is defined as a multi-agent extension of *PDDL3.1* and inherits some of the functionalities of this single-agent language. Particularly, it allows us to encode MAP tasks via *SAS+*-like state variables and it also admits the classical *PDDL* predicates.

The structure of the facts that compose a MAP task are defined through two different sections of the domain: `:functions` and `:predicates`. Internally, our MAP framework translates predicates into *true/false* state variables. An additional `:multi-functions` section allows for a more compact definition of functions (see section 3.3.6).

We include in our language some special characteristics and constructs that model the particular requirements of a MAP task:

- **Factored input**: Since information on the MAP task is distributed among agents, our language allows for a factored task representation; that is, each agent receives a separate domain and problem that represent its knowledge of the MAP task.

- **Shareable information**: The privacy of the agents is established through a `:shared-data` section. This construct, integrated in the problem block, defines the information shareable with other agents.

- **Global and private goals**: Although our model only accepts the definition of global goals, a section `:private-goal` for defining private goals of agents in non-cooperative MAP models can also be written with our language.

One can find strong similarities between our language and *MA-PDDL*: both are designed as an extension of *PDDL3.1*, they are intended to be used also for factored representations, and they introduce mechanisms to define the agents' privacy. The main difference lies in the definition of private and public information: in *MA-PDDL* the user explicitly defines the private information and in our language the user defines the shareable (public) information.

The next section shows the modelling of Example 2 with our MAP language. Since this particular task was also encoded through *MA-PDDL* in section 2.2.2, we compare the two specifications to stress the differences between both languages.

4.1.2.1 MAP task specification example

Modelling the task in Example 2 with our language is very similar to the encoding of the same task with *MA-PDDL* shown in section 2.2.2. As *MA-PDDL*, our language allows for a factored input, so that each agent in the task receives a separate domain and problem. Like in section 2.2.2, we focus on the specification of agents $ta1$ and f, stressing only the fragments of the code that present major differences with respect to the *MA-PDDL* representation.

Like in *MA-PDDL*, the domain block includes the type hierarchy of the agent, as well as the predicates and the functions for describing the facts of the world. The following fragments of code show the definition of the predicates and functions for agents $ta1$ and f, respectively:

```
(:predicates
    (in-area ?p - place ?a - area)
)
(:functions
    (at ?p - package) - location
    (owner ?t - truck) - agency
    (pos ?t - truck) - location
)
(:multi-functions
```

```
    (link ?p1 - place) - place
)
```

As in the *MA-PDDL* modelling of section 2.2.2, the domain specification of agent $ta1$ includes a predicate `in-area`. Predicates `at`, `owner` and `pos` are now defined as functions, and the multi-function `link` allows for the definition of all the outgoing connections from a certain `place` with a compact notation.

```
(:predicates
    (pending ?p - product)
    (manufactured ?p - product)
)
(:functions
    (at ?p - package ?l) - location
)
```

The task modelling of agent f keeps the predicates `pending` and `manufactured` used in the *MA-PDDL* encoding of section 2.2.2, and represents `at` as a state variable.

Next, the domain specification describes the agent's operators. The following code snippets show the `drive` operator of agent $ta1$ and the `manufacture` operator of agent f, respectively:

```
(:action drive
    :parameters   (?a - agency ?t - truck ?p1 - place ?p2 - place)
    :precondition (and (in-area ?p1 ?a)(in-area ?p2 ?a)
                  (= (owner ?a) ?t)(= (pos ?t) ?p1)
                  (member (link ?p1) ?p2))
    :effect       (assign (pos ?t) ?p2)
)
```

```
(:action manufacture
    :parameters   (?f - factory ?rm - package ?fp - product)
    :precondition (and (= (at ?rm) ?f)(pending ?fp))
    :effect       (and (not (pending ?fp))(manufactured ?fp))
)
```

Whereas the encoding of these actions and their corresponding *MA-PDDL* modelling (see section 2.2.2) are very similar, one can observe some key differences:

- The operator `=` is used to define the value that must be assigned to each state variable in the preconditions of the action.
- The keyword `member` is used to check the value of a multi-function precondition. For example, the precondition `(member (link ?p1) ?p2))` holds if `?p2` is one of the values of the multi-function `(link ?p1)`.
- The traditional positive and negative *STRIPS* effects like `(not (pos ?t ?p1))` and `(pos ?t ?p2)` are replaced by state variable assignments. For instance, in order to change the position of a `truck` in the `drive` action, we `assign` a new value `?p2` to a `(pos ?t)` state variable.

Next, we show the main sections of the problem block of agents $ta1$ and f. After modelling the task objects, we declare the public information that agents $ta1$ and f can share starts with other agents via the `:shared-data` section:

```
(:shared-data
    ((at ?p - package) - location) - ta2
)
```

As in the *MA-PDDL* example, agent $ta1$ will only disclose the position of the `package`, thus occluding all the information regarding its `truck`.

```
(:shared-data
    (manufactured ?p - product) - (either ta1 ta2)
)
```

Similarly to $ta1$, agent f will only inform the rest of agents once the `final product` is manufactured; i.e., once the task goal has been successfully completed.

Note that our language allows for a more advanced definition of privacy than *MA-PDDL*: in our specification, the privacy is defined among pairs of agents. For

instance, agent $ta1$ defines the `at` predicate as public with respect to agent $ta2$, but at the same time, $ta1$ is not willing to share this predicate with agent f. This distinction is not possible in *MA-PDDL*, in which all public predicates are always shared with the rest of agents.

The initial state for agents $ta1$ and f, respectively, is defined in the following fragments of code:

```
(:init
    (= (pos t1) l1)(= (at rm) l2)(= (owner t1) ta1)
    (= (link l1) {l2 sf})(link l2 {l1 sf})(link sf {l1 l2})
    (in-area l1 ga1)(in-area l2 ga1)(in-area sf ga1)
```

```
(:init (pending fp))
```

The initial state definition for both agents is almost identical to its *MA-PDDL* counterpart. However, note that, in the case of agent $ta1$, the multi-functions offer a simpler and more compact definition of the `links` among `places`.

The last section of the problem block is preceded by the keyword `:global-goal` instead of `:goal`, but otherwise, it is identical to the *MA-PDDL* version shown in section 2.2.2.

4.1.3 Multi-agent planning framework

The main result of this PhD thesis is the development of a fully-distributed resolution framework for cooperative MAP tasks. Our MAP framework has been progressively enhanced and refined, giving rise to three different versions of the system; namely, MAP-POP, FMAP and MH-FMAP.

The following sections summarize the evolution of our framework, analyzing the main features, strengths and weaknesses of each version of the system.

4.1.3.1 Initial framework: MAP-POP

The *raison d'être* of cooperative MAP lies in the collaboration of several planning entities which assist each other in order to reach a set of common objectives that satisfies all the participants. The idea of multiple agents providing assistance to each other is illustrated in Example 2: in order for agent f to manufacture the final product, agent $ta2$ must first deliver the raw materials rm into f. In turn, agent $ta1$ must deliver rm in sf, so that the package is available in the working area of agent $ta2$.

The notion of multi-agent cooperation closely resembles the concept of causal link in Partial-Order Planning (POP). A causal link of the form $\alpha \xrightarrow{l} \beta$ is introduced in a partial-order plan to *support* an open precondition l of an action β through another action α (see section 2.1.2.2). Therefore, a causal link is a mechanism that can be naturally adapted to model the interaction of an agent providing assistance to another one in the context of a cooperative MAP task.

Hence, the single-agent POP paradigm offers the potential and flexibility to be extended to a cooperative MAP setting. Our initial MAP framework, MAP-POP, adapts the traditional backward reasoning of POP to MAP to come up with a sound and reliable MAP approach. MAP-POP is built around two basic design principles:

- **Fully-distributed system**: MAP-POP is designed to be integrated seamlessly as a reasoning module in each agent. Reasoning is fully distributed, thus avoiding the use of mediators or centralized modules. This design choice implies the development of robust communication protocols as well as other aspects such as the factored task description received by the agents.

- **Privacy preservation**: All the algorithms that make up MAP-POP are designed so as to preserve the agents' privacy established in the task de-

scription. Only the information described as shareable is exchanged during the execution of the different distributed procedures within MAP-POP. As most state-of-the-art MAP methods, both MAP-POP and its successors implement a weak form of privacy according to the classification presented in section 2.2.3.2.

Algorithm 3 shows the basic MAP-POP procedure. Before initiating the planning process, agents exchange information (see section 3.3.8 for a thorough description of this stage) and progressively share the public fluents of the task. In this stage, the planning entities perform a multi-agent grounding of the task by building a distributed Relaxed Planning Graph (dis-RPG). The dis-RPG provides an estimate of the cost of achieving the fluents of the task, a valuable information that will be later used to perform heuristic calculations at planning time.

The MAP-POP resolution process, described in section 3.3.9, is based on the iterative exploration of a multi-agent search tree. Each agent keeps its own version of the tree, storing plans according to its privacy constraints. An iteration of the search procedure is initiated when agents agree on the open node of the tree to expand; that is, the next *base plan*. The base plan is selected according to the estimated quality of the plans obtained through a heuristic function (see section 4.1.4 for a discussion on the heuristic estimators). Afterwards, agents select the most costly open goal of the plan according to the dis-RPG as the next objective to address.

Each agent individually refines the chosen base plan through an embedded POP system, and generates a successor or refinement plan per alternative of solving the open goal of the base plan. Once the successors are heuristically evaluated, agents perform a coordination stage in which the refinement plans are exchanged and incorporated into the multi-agent search tree. Agents share their refinement plans

with the rest of participants by occluding the information defined as private in their task descriptions. Therefore, every plan received by an agent preserves the privacy constraints established in the MAP task.

Refinement plans in MAP-POP are partial-order plans built backwards from the MAP task goals, whose actions can be contributed by several agents. The internal POP search of each agent creates refinement plans through the composition of the base plan and a *refinement step*. A refinement step extends a base plan by solving the open goal selected by the group, along with all the private open goals that arise from this resolution (see Definitions 3.7 and 3.10).

Limitations of MAP-POP. MAP-POP presents several limitations that diminish its overall performance. Firstly, as discussed in section 3.2.3.3, MAP-POP is not a complete approach because we implicitly prune the search space by setting an arbitrary limit on the number of refinement plans an agent can create over a given base plan. This limit is set because an agent may generate an infinite number of refinement plans of a given base plan. To illustrate this shortcoming, let us examine the MAP task described in Example 2 and modelled in section 4.1.2.1: let Π_0 be an empty plan with a single open goal `(= (at rm) f)`. Agent $ta2$ can easily refine Π_0 by posing the following sequence of actions: {`(load ta2 t2 rm sf)`, `(drive ta2 t2 sf l3)`, `(drive ta2 t2 l3 f)`, `(unload ta2 t2 rm f)`}. However, once truck `t2` is in location `l3`, $ta2$ can `unload` and `load rm` and then proceed to `f`, giving rise to another sound refinement plan. In general, $ta2$ can `load` and `unload rm` in `l3` multiple times, thus obtaining a potentially limitless number of refinement plans over Π_0.

Along with the lack of completeness, which might prevent the planner from finding solution plans, the use of a traditional, backward-chaining POP to build the refinement plans restricts the potential application of heuristic functions: it

is not possible to infer a state from a regressed plan, which rules out the use of powerful state-based heuristics. This explains the generally low performance of the heuristic guidance in MAP-POP.

The aforementioned limitations motivated the design of a revised framework that offers better theoretical properties and notably outperforms MAP-POP.

4.1.3.2 Enhanced framework: FMAP

The second version of our MAP framework, FMAP, maintains the general search scheme of MAP-POP while redesigning key aspects such as the node expansion and heuristic. These improvements give rise to a complete and much faster planner that achieves vastly superior experimental results (see section 3.4.4 for a thorough analysis of the FMAP search procedure).

FMAP also implements a multi-agent A* search in which the participanting agents jointly explore a common search tree. The multi-agent search scheme of FMAP is directly inherited from MAP-POP: agents select a base plan among the open nodes of the joint search tree according to heuristic estimates on the plan quality; the base plan is individually refined by each agent, which can contribute with several refinement plans; and, finally, the successor plans are heuristically evaluated and exchanged among the agents while preserving the privacy constraints.

The main difference with respect to MAP-POP is the replacement of the embedded POP engine of the agents by a forward-chaining POP system. The forward-chaining POP paradigm was firstly introduced in (18), just after the publication of MAP-POP, and it demonstrated the potential to overcome most of the limitations of backward-chaining POP while keeping its main strengths.

Agents in FMAP build plans forward from the initial state, which allows us to infer the *frontier state* of each plan. The frontier state represents the situation of

the world after executing the actions in the plan, which enables the use of state-based heuristic functions to evaluate the partial-order plans. The h_{DTG} heuristic function used in FMAP (see section 4.1.4) overcomes the poor heuristic guidance of MAP-POP, drastically improving the performance of FMAP.

Another key difference between both frameworks relies in the construction of the refinement plans. In FMAP, a refinement plan introduces a new action in the current base plan, fully supporting all its preconditions and solving all the *threats* (see Definition 2.6) that arise during this solving process. The introduction of this new conceptual definition of refinement plans guarantees the completeness of FMAP since, in this case, agents are able to generate all the possible refinements over a given base plan (see section 3.4.4.2).

In order to maximize the performance of FMAP, the internal forward POP engine of the agents, named FLEX, is designed as a multi-thread search algorithm that takes full advantage of the available execution threads in the CPU (see section 3.4.4.1). FLEX selects the actions in the agent's task whose preconditions are satisfied with the effects of the actions in the base plan, creating a set of *potentially supportable actions*. Then, for each action in this set, FLEX performs an independent search process to fully support the action, thus giving rise to the refinement plans (see Figure 3.17).

Note that, given a base plan and an action, FLEX can potentially insert the action at many different points of the plan, thus maintaining the flexibility of the conventional backward-chaining POP search.

Limitations of FMAP. Despite FMAP was designed to overcome the main limitations of MAP-POP, we can yet identify some room for improvement.

The performance of FMAP directly depends on the accuracy of the heuristic function h_{DTG}, which is used to select the node to expand. Relying on a single

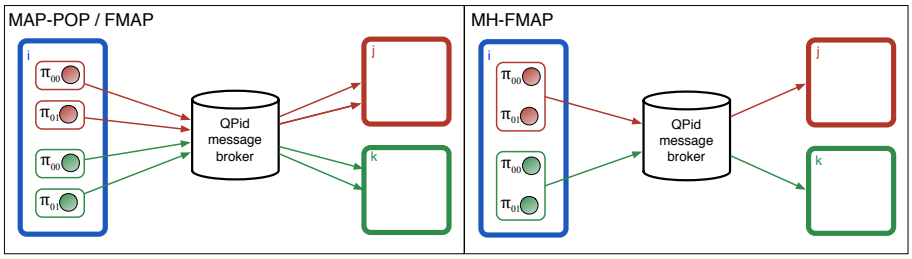

Figure 4.1: Communications during search in MAP-POP/FMAP and MH-FMAP

heuristic may jeopardize the performance of the system in some domains, since the quality of the estimates may differ from one domain to another. This fact can be observed in the experimental results of section 3.4.5. Particularly, Table 3.8 shows that the performance of FMAP degraded in domains like *Depots* and *Logistics*. Moreover, section 3.4.5.5 analyzes the impact of the communications in the overall execution time of FMAP. The results show that, in order to effectively reduce execution times, it is necessary to optimize communications, which represent the main bottleneck of FMAP and the reason that it does not scale up well (see Figures 3.22 and 3.23).

FMAP follows the same communication scheme as MAP-POP, described in Figure 4.1 (left). Once an agent i has built its refinement plans over the base plan Π_0, Π_{00} and Π_{01}, it configures the view of these plans for agents j and k according to its privacy constraints with respect to them. Each plan is individually sent to each recipient via the QPid message broker. Since the average number of refinement plans per iteration tends to be large in most MAP domains, the amount of messages exchanged compromises the performance of FMAP in most cases.

4.1.3.3 Multi-heuristic framework: MH-FMAP

The latest evolution of our approach, MH-FMAP, extends FMAP by applying, for the first time in MAP, a multi-heuristic search scheme. The literature in single-agent planning proves the efficiency of the *alternation* of heuristic functions (96). This method selects plans alternatively with each of the available heuristic functions, using equally all the estimators and obtaining much better results than any aggregation or combination of heuristics.

Alternating heuristics is specially useful for leaving *plateaus*, because if search does not progress with one of the estimators, we can use the rest of heuristic functions to quickly find another branch and leave the plateau.

MH-FMAP makes use of two different heuristics: h_{DTG}, a function already applied in FMAP, and h_{Land}, an estimator that takes into account the landmarks not reached in the plan (see section 4.1.4). We employ two different queues of plans, although MH-FMAP can be generalized to apply as many heuristic schemes as required: open nodes are ordered in the main queue according to $f = g + 2 * h_{DTG}$, while the secondary queue uses h_{Land} to arrange the *preferred successors* (see Definition 3.20).

The results in section 3.5.6 prove that the synergy of both estimators clearly outperforms FMAP, particularly in domains such as *Depots* and *Logistics*, in which the coverage results are almost twice as much as with FMAP.

In addition to the introduction of a novel multi-heuristic search approach for MAP, MH-FMAP includes a thorough optimization of the communication infrastructure in the exploration of the joint search tree and the heuristic calculations. In particular, the exchange of plans during the multi-agent search was optimized by grouping the refinement plans of an agent in one iteration in a single message, as depicted in Figure 4.1 (right).

Let us suppose that the three agents in Figure 4.1 perform n iterations of the search algorithm to solve a given MAP task, and agent i generates in average p refinement plans per iteration. In MAP-POP and FMAP, agent i sends a total of $2 * n * p$ messages to j and k to share refinement plans. However, in MH-FMAP, agent i would send only $2 * n$ messages, one message per iteration to j and k, respectively (see Figure 4.1). Therefore, MH-FMAP clearly alleviates the negative impact of communications with respect to its predecessors.

The reduction in the number of exchanged messages noticeably improves the performance of the system. The results of MH-FMAP using only the h_{DTG} heuristic (column FMAP- h_{DTG} in Table 3.9) show a superior coverage than the earlier FMAP results in Table 3.8, despite the same estimator, h_{DTG}, was used in both cases.

4.1.4 Global heuristics for multi-agent planning

The different versions of our cooperative MAP framework discussed in section 4.1.3 implement a multi-agent heuristic search to generate solution plans. The usual approach to evaluate plans in MAP is the application of heuristics locally by each agent. This implies a loss of accuracy compared to the application of a heuristic in a single-agent setting because in MAP agents usually do not have a complete knowledge of the task.

A significant part of the work of this PhD thesis focuses on the development of *global* heuristic functions for cooperative MAP that take into account the full information of the MAP task, rather than using the local task projections managed by the individual agents. The following subsections summarize the different MAP heuristic functions introduced throughout the development of this research.

4.1.4.1 Backward-chaining POP heuristic

MAP-POP adapts state-of-the-art POP heuristics to MAP. Particularly, MAP-POP applies the additive plan selection heuristic h_{add} that was also used in the VHPOP planner (126). This estimator evaluates a partial-order plan as the sum of the cost of its open goals.

For the multi-agent adaptation of h_{add}, we use the dis-RPG calculated by MAP-POP at pre-planning time. Given a fluent f of the MAP task, the fluent level of the dis-RPG in which f appears denotes the minimum cost to reach f in the relaxed task. Hence, given a plan Π, h_{add} is calculated as follows:

$$h_{add}(\Pi) = \sum_{\forall f \in openGoals(\Pi)} level(f)$$

The dis-RPG used in h_{add} is calculated from the initial state of the task. Ideally, this graph should be recalculated in every search node from the *state* derived in the plan of the node. However, since plans are built backwards, the only known state is actually the initial state. For this reason, the initial dis-RPG is used to evaluate all the refinement plans. This affects the precision of the h_{add} heuristic, which provides rather inaccurate estimates.

4.1.4.2 DTG-based heuristic: h_{DTG}

The most relevant advantage of FMAP and MH-FMAP with respect to MAP-POP is the use of a forward-chaining POP to generate refinement plans. Since plans are built in a forward-chaining fashion, we can infer the *frontier state* that results from the sequential application of the actions in a given partial-order plan (see section 3.4.4.1). In turn, frontier states can be used to obtain accurate state-based heuristic estimators to evaluate partial plans.

The first state-based global heuristic function developed in the context of this PhD thesis, h_{DTG}, is thoroughly analyzed in sections 3.4.4.3 and 3.5.5.1. h_{DTG}

is a global estimator that evaluates plans through the information available in the Domain Transition Graphs (DTGs) associated to the state variables of the MAP task. As stated in section 3.19, the design of h_{DTG} is motivated by the high communication cost of a distributed h_{FF} heuristic (54), which implies calculating a dis-RPG in each base plan. On the contrary, DTGs are built at pre-planning time and they remain unaltered during the search, which reduces the communication needs of the system.

Formally, h_{DTG} is a variation of the well-known h_{CEA} estimator (52). Given a MAP task $\Im_{MAP}$ and a partial plan Π, h_{DTG} estimates the quality of Π by calculating the number of actions of a relaxed plan between the frontier state $FS(\Pi)$ and the set of goals of $\Im_{MAP}$, G. As h_{CEA}, h_{DTG} reuses the side effects of the actions in the relaxed plan to estimate the cost of the subsequent goals. As shown in equation 3.1, given a goal $g = \langle v, d \rangle$, a state S and an action α, both g and the precondition of α related to v are evaluated in the same state S, while the rest of preconditions of α can be evaluated in different states.

As commented before, DTGs are calculated in a pre-processing stage. The DTGs are privacy-preserving data structures: the private values are replaced by the undefined value $\perp$ (see section 3.4.4.3). Therefore, given a variable v, two different agents may maintain a different version of the DTG associated to v.

The h_{DTG} plan evaluation algorithm, described in detail in section 3.4.4.3, builds a relaxed plan by progressively supporting the goals of the task from the plan's frontier state. Given a goal $\langle v, d \rangle$, the procedure applies the Dijkstra algorithm to calculate the shortest path in v's DTG between an initial value (which can be the value of v in the frontier state or an effect of an action already in the relaxed plan) and d. The relaxed plan is then extended by introducing a sequence of

actions that corresponds to the value transitions in the DTG path. The algorithm iterates until all the goals are solved.

The work in (108) compares a MA-STRIPS implementation of h_{DTG} against h_{FF} using a distributed greedy best-first search. The experimental results reveal that h_{DTG} obtains slightly better coverage results than h_{FF}, particularly surpassing the results of h_{FF} in domains such as *elevators* and *openstacks*. All in all, h_{DTG} is not only computationally more efficient than h_{FF} in a distributed setting, but it also obtains better overall results.

4.1.4.3 Landmark-based heuristic: h_{Land}

The most recent version of our MAP framework, MH-FMAP, aims to improve the performance of the planning procedure by alternating global heuristics. Along with h_{DTG}, MH-FMAP uses h_{Land}. This global estimator, thoroughly described in section 3.5.5.2, measures the quality of a plan Π according to the number of pending *landmarks* in Π. A landmark is a fluent that is necessarily satisfied in every solution plan for a given MAP task.

The h_{Land} procedure consists of two different stages. At pre-planning, agents jointly build the Landmarks Graph (LG) through the multi-agent adaptation of the landmark extraction procedure in (55) (see Algorithm 8). Similarly to the DTGs used in h_{DTG}, the LG is a privacy-preserving and immutable data structure in which the nodes (landmarks) are connected through a set of directed edges (*necessary orderings*). A necessary ordering $l \leq_n l'$ indicates that the landmark l will be obtained before l' in all the solution plans for the planning task at hand.

Due to the private information, agents may have a different version of the LG, according to the landmarks that are private. Privacy affects the plans evaluated with h_{Land}: given an agent i and a plan Π, agent i takes into account first the

satisfied landmarks in Π according to its view of the LG and Π. Next, it sends the list of verified landmarks to the rest of agents, which in turn check if they reach more landmarks in Π. Once agent i receives the rest of landmarks satisfied in Π, it evaluates the plan according to the total amount of pending landmarks.

As stated in section 3.5.6, h_{Land} closely resembles the PP-LM distributed heuristic used in the GPPP system. Both h_{Land} and PP-LM implement the landmark extraction algorithm in (55) in the form of a global privacy-preserving heuristic. The main difference between both approaches is that PP-LM is based on MA-STRIPS, while h_{Land} is built upon our planning and privacy model.

4.2 Experimental results

Throughout the selected papers in Chapter 3, the different versions of our framework have been thoroughly evaluated in order to assess their performance. Our framework has been systematically compared to other state-of-the-art MAP techniques through a benchmark based on the problem suites of the International Planning Competition (IPC).

The most recent version of our framework, MH-FMAP, was submitted to take part in both the centralized and distributed tracks of the 2015 Competition of Distributed and Multi-Agent Planning (CoDMAP). The CoDMAP provided researchers with a set of standard MAP tasks encoded through *MA-PDDL*, which will ease the task of comparing the performance of different MAP systems.

In order to perform an accurate review of the performance of our MAP system, it is thus necessary to consider both our own experimental results, showcased in the selected papers of Chapter 3, and the CoDMAP results. For this reason, this section provides a summary of the experimental results obtained for each version

of our MAP framework, followed by an in-depth review of the performance of MH-FMAP in the 2015 CoDMAP.

4.2.1 MAP-POP results

In section 3.2.4, we made use of an early version of our benchmark to analyze the performance of MAP-POP. Since at that point there were not standardized benchmarks for evaluating MAP systems, the common procedure entailed adapting the domains and problem suites of the IPC to a cooperative MAP context. For these early tests, we used the *Rovers*, *Satellite* and *Logistics* domains, which are frequently utilized to conduct experiments in other MAP-related papers (80).

The experiments of section 3.2.4 compare MAP-POP against one of the state-of-the-art MAP systems at that point, Planning First (80). The results show that MAP-POP scales up much better than Planning First, which tends to be faster in smaller problem instances but it is unable to solve any of the largest problems of the three tested MAP domains.

This conclusion is backed up by an additional scalability analysis which shows that MAP-POP scales up better than Planning First when executing several times a given MAP task with an increasing number of agents (see Figures 3.1 and 3.2). Whereas Planning First converges to a solution faster in the simplest tasks, MAP-POP offers a more stable behaviour, performing much better in instances with a high number of agents.

In section 3.3.10, we extend the performance analysis of MAP-POP by testing the system against two custom MAP domains: *Picture*, a loosely-coupled domain, and *Transportation*, which gives rise to tightly-coupled tasks. In this case, we first compare the quality of the solution plans when running MAP-POP in a single-agent and multi-agent setup.

The results show that, despite the influence of privacy in the distributed setting, MAP-POP obtains similar plan quality results in both setups, in terms of number of actions and plan duration.

The second test, whose results are depicted in Figures 3.11 and 3.12, analyzes the scalability of MAP-POP when running the aforementioned MAP domains. The results show that the number of participating agents directly increases the execution time of MAP-POP and, particularly, the number of messages passed among agents. This result stresses the high cost of communications as one of the main disadvantages of our initial MAP approach.

4.2.2 FMAP results

For the experimental evaluation of FMAP, discussed in section 3.4.5, we used an extended benchmark which includes 10 different MAP domains selected from the IPC suites and adapted to a cooperative MAP context. Table 3.6 summarizes the features of the MAP domains in our benchmark: the first 5 domains give rise to loosely-coupled tasks, while the second half of Table 3.6 consists of complex domains that result in tightly-coupled MAP tasks. Most of these tightly-coupled domains also include *heterogeneous* agents with different abilities and knowledge of the MAP task.

The first test, presented in section 3.4.5.3, compares the performance of FMAP against MAPR (11), one of the best-performing state-of-the-art MAP systems. As shown in Table 2.1, MAPR is a distributed state-based MAP system which allocates the task goals in pre-planning time, giving rise to a set of subtasks that are individually assigned to each of the participating agents. Agents perform planning sequentially: each agent receives the previous agent's solution plan and extends it incrementally by solving its endowed subtask. This resolution scheme

limits the scope of MAPR, since it can only cope with loosely-coupled tasks in which goals do not require cooperation among agents.

The results in Table 3.7 show that, even though MAPR is several orders of magnitude faster than FMAP, which gives it a superior coverage in the 5 domains tested, our approach obtains much better-quality plans. Indeed, FMAP is particularly efficient at reducing the number of actions and the makespan of the plans. Moreover, the general-purpose scope of FMAP allows it to tackle complex tightly-coupled MAP domains that are not solvable by MAPR.

The second test, described in Section 3.4.5.4, compares FMAP and MAP-POP, the previous version of our framework. As expected, the improvements introduced in several key areas of the framework (see section 4.1.3.2) give FMAP a remarkable performance boost against MAP-POP. Particularly, the coverage and execution time results in Table 3.8 clearly favour FMAP, which solves 124 tasks more than MAP-POP, being up to three orders of magnitude faster in some domains.

The experimental analysis of FMAP concludes with a thorough analysis of the scalability of the system in section 3.4.5.5. As in previous scalability tests, we try the same MAP task several times, increasing the number of agents by one in each run. In order to ensure a precise analysis, we separately measured the execution time spent by FMAP in generating, evaluating and communicating plans. Additionally, we run the test with MAPR for comparison purposes.

Figures 3.21 and 3.23 showcase the results of the two scalability experiments, based on a *Logistics* and a *Satellite* MAP task, respectively. Both tests confirm that communications take most of FMAP's execution time. The communication overhead grows up exponentially with the number of participating agents. Nevertheless, if we consider only the evaluation and generation time, the performance of FMAP is much closer to MAPR. Therefore, the impact of communications is

the main reason behind the relatively high execution times exhibited by FMAP throughout the experimental tests.

4.2.3 MH-FMAP results

The experimental evaluation of MH-FMAP, discussed in section 3.5.6, analyzes the performance of the multi-heuristic search strategy against the two single heuristics: more precisely, we compare MH-FMAP against FMAP when applying h_{DTG} and h_{Land} to evaluate plans, respectively. We make use of the 10-domain benchmark already utilized to assess the performance of FMAP in section 3.4.5.

Table 3.9 summarizes the results of this test. MH-FMAP obtains the best coverage results in almost all the domains of the benchmark, solving 24 more tasks than h_{DTG} and 98 more tasks than h_{Land}. MH-FMAP clearly benefits from the alternation of heuristics, mimicking the behaviour of the best-performing estimator in most of the domains. Additionally, MH-FMAP clearly improves the results of the single-heuristic approaches in domains in which both heuristics offer a similar performance, even doubling the coverage in the *Logistics* domain.

Regarding plan quality, the results prove that the multi-heuristic approach does not diminish the quality of the solution plans, offering similar figures to the standalone estimators. The same conclusions can be obtained regarding execution time: MH-FMAP provides a good trade-off between both heuristics, being slightly slower than h_{Land} but faster that h_{DTG} in 6 out of 10 MAP domains.

Additionally, we provide a second test that compares MH-FMAP against GPPP (72) a *MA-STRIPS*-based method that constitutes the only other cooperative MAP approach in the literature to integrate a global landmark-based heuristic function, PP-LM.

Table 3.10 showcases the experimental results obtained in this experiment. Whereas GPPP is up to two orders of magnitude faster than MH-FMAP in some domains, our approach improves the coverage results of GPPP, obtaining better figures in 2 out of 5 tested domains.

Regarding plan quality, MH-FMAP is much more efficient than GPPP, returning solutions with a lower number of actions in almost all the tested domain. Moreover, since GPPP generates only sequential plans, the solutions produced by MH-FMAP are much shorter, effectively minimizing the plans' makespan.

Domain	Tasks	MH-FMAP coverage	CoDMAP coverage
Blocksworld	19	**8**	0
Depots	20	**12**	2
Driverlog	20	15	**17**
Logistics	20	**20**	4
Rovers	11	**11**	7
Satellite	15	14	14
Zenotravel	17	**17**	14
Total	122	**97**	58

Table 4.1: Coverage results of MH-FMAP with our benchmark and in the CoDMAP

4.2.4 CoDMAP 2015 results

In order to conclude with the summary of the experimental results, this section analyzes the results obtained by MH-FMAP in the 2015 Competition of Distributed and Multi-Agent Planners (CoDMAP) (118). This event has not only introduced *MA-PDDL*, a *de facto* standard language for the definition of cooperative MAP tasks (see section 2.2.2), but it has also provided researchers with a standardized benchmark of MAP domains and tasks that will ease the experimental evaluation of ongoing and future MAP systems.

The 2015 CoDMAP benchmark includes 12 different MAP domains, most of which are directly adapted from the single-agent problem suites of the International Planning Competition (IPC). Nine of these domains were also used in our benchmark (see section 3.4.5.2), and some of the tasks in seven of these domains are shared by our benchmark and the CoDMAP.

Table 4.1 compares the coverage results obtained by MH-FMAP in the experiments presented in section 3.5.6 and the CoDMAP, respectively, considering only the common tasks of both benchmarks. As it can be immediately noted, the coverage results are in general much better in our experimental setting than in the CoDMAP, with the only exception of the *Driverlog* and *Satellite* domains.

This apparent anomaly can be explained by the different modelling of the tasks in our benchmark and the CoDMAP. Particularly, it is possible to identify some key differences regarding the task encoding and the definition of privacy.

Encoding. The *MA-PDDL*-based tasks of the CoDMAP are purely propositional, while the tasks in our benchmark are described through state variables. This different *PDDL* model may jeopardize the performance of MH-FMAP, which is designed to take full advantage of variable-based task descriptions.

More precisely, h_{DTG}, one of the heuristic functions that govern the search of MH-FMAP, bases its estimates on the DTGs associated to the MAP task, and therefore, its accuracy depends on the quality of the information provided by the DTGs.

In order to properly infer DTGs from a propositional task, our parser automatically translates the *MA-PDDL* code into our MAP language, converting the *MA-PDDL* literals into state variables with an associated binary domain $\mathcal{D}_v = \{true, false\}$. Binary variables are not descriptive enough, since they offer

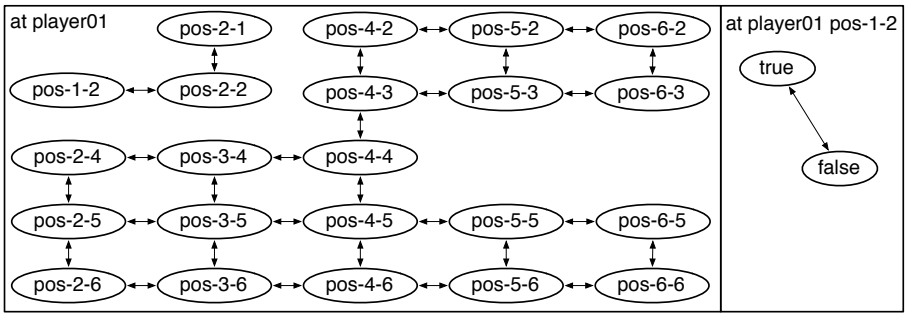

Figure 4.2: DTG for a state variable and a predicate in the `Sokoban p01` task

very poor information on the MAP task. Therefore, we presume that the differences regarding task encoding compromised the performance of MH-FMAP in the 2015 CoDMAP.

Figure 4.2 illustrates this encoding issue by depicting two DTGs inferred from one of the tasks in the CoDMAP benchmark, the `Sokoban p01` task. Our MAP language allows us to describe the position of the player `player01` through a single state variable `(pos player01)`. As shown in the left-hand diagram of Figure 4.2, the associated DTG provides rich information, displaying the complete game board and all the connections among the different `positions`.

In the CoDMAP, however, the task description includes one `(at player01 pos-x-y)` literal per `position` of the game board. MH-FMAP infers a binary state variable for each `position` in the board, resulting in a collection of rather simple DTGs as the one in the right-hand diagram of Figure 4.2.

Privacy. Another basic difference between both benchmarks is the level of privacy defined for each MAP task. In some of the domains, the CoDMAP organizers introduced restrictive privacy rules, allowing agents to share less information than our benchmark in general.

For instance, the CoDMAP version of the well-known *Satellite* domain only allows agents to share whether they have obtained an `image`. As opposite to our benchmark, agents cannot reveal the `direction` they are `pointing` to.

Privacy plays a basic role in the performance of MH-FMAP, since agents transmit the refinement plans including only the data defined as shareable, thus occluding the private information. This may have a direct impact in the performance of MH-FMAP, since increasing the amount of private information reduces the available data in the plans, which might potentially affect the precision of the heuristic estimators.

Given the aforementioned differences between both settings, it is reasonable to conclude that task encoding and privacy are the main reasons behind the poor performance of MH-FMAP in the 2015 CoDMAP. However, in order to properly test and firmly confirm our hypothesis, we designed two additional experimental tests to individually assess the impact of each factor.

Among the MAP domains in Table 4.1, we chose three domains to conduct the additional tests, namely the loosely-coupled *Rovers* and *Zenotravel*, and the tightly-coupled *Depots*.

The features of these domains make them an appropriate choice to test the impact of encoding and privacy on the performance of MH-FMAP. On the one hand, the coverage of MH-FMAP in these domains clearly diminished in the CoDMAP setting, as shown in Table 4.1. On the other hand, these domains present the complete range of possibilities regarding privacy:

- *Depots*: In this domain, both our benchmark and the CoDMAP's define the same notion of privacy. Agents can share the location of `trucks` and `crates` and whether a `surface` or `hoist` is `clear`.

- *Rovers*: Our benchmark introduces a more restrictive definition of privacy than the CoDMAP in this case. Agents can only publicize the position of a `rock` or `soil sample` and whether they `communicated` the `data` of a `sample`. In the CoDMAP, however, most of the information regarding the instruments of the `rovers`, the location of the `lander` and the visibility of the `waypoints` are publicly available.

- *Zenotravel*: The CoDMAP version of the *Zenotravel* domain is more restrictive than ours. The location of the different `persons` is public if they are in a `city`, but `plane` agents cannot publicize their list of `persons` on board. In our version, the position of a `person` is always publicly available.

The next subsections detail the configuration of the encoding and privacy tests and thoroughly analyze the results obtained.

4.2.4.1 Encoding test

One of the key factors that may affect the performance of MH-FMAP is the encoding of the tasks, due to its dependence on state variables. To perform a precise analysis of the impact of task encoding, we executed a total of 49 tasks of the *Rovers*, *Depots* and *Zenotravel* domains, modelled through *MA-PDDL* and our language[1].

All the tasks were configured using the privacy constraints defined in the CoDMAP, so that the privacy does not affect the results of this experiment.

[1] As in section 3.5.6, all the tests were performed on a single machine with a quad-core Intel Core i7 processor and 8 GB RAM, assigning up to 2 GB RAM for the Java VM. The duration of each experiment was limited to 30 minutes.

Domain	Our language (state variables)				*MA-PDDL* (propositions)			
	#Act	MS	#Iter	Time	#Act	MS	#Iter	Time
Depots	13,00	10,00	**71,00**	**5,10**	13,00	10,00	3421,00	99,64
	Solved			**12**/20	Solved			2/20
Rovers	38,43	17,86	**383,14**	**29,16**	**38,29**	**17,43**	2746,57	178,94
	Solved			**11**/11	Solved			7/11
Zenotravel	29,40	16,13	**96,00**	**49,01**	**27,73**	**12,33**	245,60	94,32
	Solved			**17**/17	Solved			15/17

Table 4.2: Experimental results of the encoding test

Table 4.2 shows the average results obtained in this test: *#Act* and *MS* columns indicate the average number of actions and duration (makespan) of the solution plans for a given domain, respectively. *#Iter* and *Time* refer to the number of iterations spent by MH-FMAP to find a solution and the execution time in seconds, respectively. The average values are computed considering only the tasks solved by MH-FMAP in both settings.

As it can be immediately noted, the *MA-PDDL* encoding does clearly compromise the coverage of MH-FMAP in the three tested domains. MH-FMAP solves 40 out of 48 tasks with our language, while it only manages to find a solution in 24 tasks in the *MA-PDDL* version of the benchmark.

The iterations and execution time figures in Table 4.2 reveal that the propositional encoding of the *MA-PDDL* tasks lowers the accuracy of the heuristic functions used in MH-FMAP: our planner requires up to 50 times more iterations with the *MA-PDDL* encoding, being up to 20 times slower in average when solving this version of the benchmark.

The plan quality figures (actions and makespan), however, show that MH-FMAP obtains shorter plans with the *MA-PDDL* version of the benchmark. This is also explained by the impact of propositional *MA-PDDL* tasks on our h_{DTG} and h_{Land} estimators: since heuristic values are not informative and accurate enough,

our system explores the MAP tree in almost a breadth fashion, expanding more nodes in each level of the tree and finding solutions at lower depths. These solutions have fewer actions and, in general, a lower makespan than the plans obtained for the tasks modelled with our language.

4.2.4.2 Privacy test

The second experimental test analyzes the impact of privacy in the performance of MH-FMAP. For this experiment, we used again the *Depots*, *Rovers* and *Zenotravel* domains, modelled through our MAP language. For each domain, we run each task with three different levels of privacy:

- **No privacy**: All the information of each MAP task is shared among the agents.
- **Our privacy**: The private information of the agents is configured as in our MAP benchmark (see section 3.4.5.2).
- **CoDMAP privacy**: The private information is defined according to the level of privacy defined in the 2015 CoDMAP.

Note that, as we mentioned in the previous section, the privacy settings defined in our benchmark and the CoDMAP coincide in the *Depots* domain, and therefore, there are no differences between the results for both configurations. All the experiments were run under the same conditions (using the same machine and time limit) than the encoding test.

Table 4.3 displays the average results for each domain in the three privacy settings established for this test. As in Table 4.2, *#Act* and *MS* refer to the average number of actions and makespan of the solution plans obtained in each domain, while *#Iter* and *Time* denote the average number of iterations and execution time, respectively.

Domain	No privacy				Our privacy				CoDMAP privacy			
	#Act	MS	#Iter	Time	#Act	MS	#Iter	Time	#Act	MS	#Iter	Time
Depots	32,17	23,08	671,67	160,96	32,17	23,08	671,67	**146,41**	32,17	23,08	671,67	**146,41**
	Solved			12/20	Solved			12/20	Solved			12/20
Rovers	43,20	18,40	450,20	103,31	43,20	18,40	450,20	61,99	43,20	18,40	450,20	**59,99**
	Solved			10/11	Solved			**11**/11	Solved			**11**/11
Zenotravel	36,53	19,82	113,24	128,08	36,53	19,82	113,24	**123,69**	36,53	19,82	113,24	124,81
	Solved			17/17	Solved			17/17	Solved			17/17

Table 4.3: Experimental results of the privacy test

The results of Table 4.3 show that the level of privacy does not have a significant impact on the accuracy of the heuristics of MH-FMAP. In the three tested domains, the average number of iterations remains constant regardless the privacy level, and so does the quality of the solution plans. Since the estimates provided by both h_{DTG} and h_{Land} do not change substantially when the privacy level is modified, the shape of the MH-FMAP search tree remains unaltered in in the three versions of a MAP task.

The only significant difference between the three privacy levels tested in this experiment lies in the average execution time. Interestingly, defining a certain amount of private information does benefit the execution time with respect to a completely public setting. In the three domains, the version without privacy is noticeably slower than the rest of variants.

The deviation on the execution time is caused by the heuristic calculations: since plans have more available information, the agent that creates a plan can perform almost all the calculations of h_{DTG} and h_{Land} by itself, without requesting the assistance of other agents. This reduces the parallelism in heuristic computation and introduces a noticeable overhead in terms of execution time.

Focusing on the privacy-preserving versions of the benchmark, Table 4.3 shows only minor differences in terms of execution time between the CoDMAP privacy

and ours. MH-FMAP obtains very similar execution times with both privacy settings, while the average number of iterations and the solution quality data remain the same in both cases.

4.2.4.3 Analysis of the results

The aforementioned results illustrate that the propositional encoding of the MAP tasks is the main factor behind the poor results of MH-FMAP in the 2015 CoDMAP.

The encoding test in section 4.2.4.1 showed major performance differences between the *MA-PDDL*-based domains and their counterparts modelled with our variable-based representation. These results are mainly caused by the binary DTGs obtained in this particular setting, which are not informative enough and have a huge impact on the precision of the h_{DTG} heuristic. As shown in Table 4.2, this issue increases the number of iterations required to find solution plans, thus clearly reducing the scalability of MH-FMAP.

On the other hand, the second test shows that privacy is not a decisive factor in the performance of MH-FMAP. The results in Table 4.3 reveal that there is only an insignificant variation between the execution times of our privacy setting and the CoDMAP's.

4.3 Ongoing trends in Multi-Agent Planning

In order to conclude with the discussion on the results, this section sketches the ongoing and future directions of the MAP research field.

We focus on three different topics which have recently captured the attention of the MAP community and constitute the current main research trends in this area; namely, privacy, MAP under uncertainty and practical applications.

4.3.1 Privacy

The issue of privacy is one of the current centric topics in MAP research. The state of the art in MAP shows a growing effort in analyzing, formalizing and taking advantage of privacy as a means to improve the performance of MAP systems.

On the one hand, section 2.2.3 summarizes the body of work devoted to formalize privacy in MAP systems. On a theoretical level, the literature includes two different models that define the distribution of public information on a privacy-preserving approach MAP (see section 2.2.3.1).

From a practical standpoint, the implementation of a MAP framework may jeopardize privacy, since in many cases an agent can infer private information from the data it receives. Section 2.2.3.2 synthesizes a four-level classification that characterize the privacy guarantees offered by actual MAP systems, ranging from *no privacy* at all to *strong privacy*, a level that guarantees the protection of private information regardless of the nature of the communication channel and the computational capabilities of the agents.

On the other hand, some recent approaches to MAP make a smart use of privacy to increase the performance of the system. A paradigmatic example of this trend is DPP (103), which calculates the Dependency-Preserving (DP) projection, an accurate public projection of the information on the MAP task which allows for a dual search: first, agents use the DP projection to obtain a robust high-level plan that is completed afterwards through the introduction of private actions. This scheme improves performance with respect to a general multi-agent search, vastly reducing the communication requirements of the agents.

4.3.2 Multi-Agent Planning under uncertainty

This PhD thesis focuses on deterministic planning, where the world is completely observable, the effects of the actions are deterministic and known and no exogenous event is assumed to occur in the environment. However, in domains such as multi-robot coordination or manufacturing, outcomes of the actions are uncertain and agents take decisions based on partially observable worlds.

It is important to highlight that uncertainty does not amount to handling agents' private information in MAP. Privacy has no impact on an agent's decisions because these depend entirely on its own information. In contrast, in non-determinism planning, agents must take decisions on the basis of uncertain or partially known information.

In the following, we summarize several approaches that deal with uncertainty and partial information in MAP settings.

Markov decision processes (MDPs). MDPs can deal with uncertainty and even partially observable worlds. MDPs calculate the optimal actions for each agent for any possible belief state or partially observable state. After taking the actions, each agent receives a local observation on the partial information about the other agents and the state of the world. Then, the environment generates a global reward that depends on the actions taken by the agents. The objective of these systems is to maximize a joint global reward function.

There exist different models of decentralized control of multiple agents under uncertainty and partially observable information. In decentralized control applications, formal models are inspired on the use of MDPs for MAP and they vary in the implicit or explicit communication of the agents' actions and representation of beliefs (102). Hence, one can find distinct variants of the decentralized partially

observable MDPs (DEC-POMDP) (7) or the multi-agent team decision problem (93).

Communications play a key role in DEC-POMDPs. By communicating their local observations before acting, agents synchronize their knowledge of the environment, and the planning problem reduces to a centralized POMDP. Therefore, it is crucial for DEC-POMDPs to rely on optimized and realistic communication models (105). This is of special relevance in many real-world applications, such as in robotics.

Other approaches. In contrast to deterministic planning, agents in contingency planning have no full knowledge of the conditions under which the plan will be executed and the outcome of the actions is not fully predictable. Under these circumstances, the planner must construct a plan that can be expected to succeed despite the unknown initial conditions and uncertain outcomes of nondeterministic actions.

Continual planning, interleaving planning and execution, is a technique widely adopted to perform planning under uncertainty in worlds undergoing continual changes (27). Since the knowledge available to the agents in continual planning is typically insufficient, agents perform plan monitoring and then repair the plan or apply replanning when the situation changes or planned actions fail.

For agents with limited perceptions and knowledge, it is necessary to explicitly model the agents' sensing capabilities as part of the planning domain (91). Other approaches, however, let the planner postpone decisions until the perceptions have actually been made. In other words, a conditional subplan is withheld until the agent has enough information to solve the contingency (15).

4.3.3 Practical applications

MAP has been used in a great variety of practical applications: industry applications, market operational frameworks, military operations or e-science applications.

In industry, MAP has been used in product assembly (e.g., car assembly). Agents plan the path of manufacturing of the product through the assembly line of the factory, which is composed of a number of resources connected to each other, and where each resource can perform a number of operations. ExPlanTech, for instance, is a consolidated framework in the area of agent-based production planning, manufacturing simulation and supply chain management (88).

MAP technology has also been used to control the flow of electricity in the Smart Grid (94). The agents' actions are individually rational and contribute to desirable global goals such as promoting the use of renewable energy, encouraging energy efficiency and enabling distributed fault tolerance. Another interesting application of MAP is the automated creation of workflows in biological pathways like BioMAS, a Multi-Agent System for Genomic Annotation (25). BioMAS uses DECAF, a multi-agent system toolkit that provides standard services to integrate agent capabilities such as plan retrieval, local scheduling, communication dispatching and execution monitoring. DECAF incorporates a GPGP (68) to coordinate multi-agent tasks.

In decentralized control problems, MAP has found application in coordination of space exploration rovers, coordinated helicopter flights, multi-access broadcast channels, and sensor network management, among others (102). MAP combined with argumentation techniques to handle belief changes about the context has been used in applications of ambient intelligence in the field of healthcare (85).

Finally, MAP has been extensively used in space applications and has inspired

the well-known *Satellite* IPC domain. The Artificial Intelligence groups of NASA have developed many multi-spacecraft missions that involve MAP, such as handling mission planning of multiple rovers, crew operations or spacecraft constellation.

5

Conclusions

The present PhD thesis introduces a novel approach to cooperative MAP based on distributed and privacy-preserving multi-heuristic search. In this document, we presented a chronologically-ordered compendium of research articles that keep record of the progressive advances and developments that led to MH-FMAP, the final version of our MAP resolution framework, which constitutes the main contribution of this research work.

Our approach presents several strengths and novel features that make it stand out among other state-of-the-art MAP techniques:

- Our formalization of a MAP task provides an advanced definition of privacy that makes our model more flexible than most of its counterparts. While the common approach defines information as either public or private for all the participants (14), our formalization establishes privacy between each pair of agents, so that a fact can be just known to a subset of agents.

  Our cooperative MAP specification language supports this richer definition of privacy, which benefits its overall expressiveness. The features of our

language coincide in general with those of the current *de facto* standard, *MA-PDDL*, which proves the validity of our language specification.

- MH-FMAP interleaves planning and coordination, which results in a general-purpose approach that efficiently solves both loosely-coupled and tightly-coupled MAP tasks, as it is proved in the experimental results.

- Unlike most approaches to MAP, MH-FMAP is based on a novel forward-chaining Partial-Order Planning engine. On the one hand, this paradigm benefits the parallelism of the actions of the different agents, effectively minimizing the *makespan* or duration of the solution plans, as can be noted in the experimentation.

  On the other hand, the forward-chaining search scheme allows for the application of accurate state-based heuristic functions that maximize the performance of the system.

- Regarding heuristic search, we contributed to the state of the art in MAP with several technical innovations. First, we developed two different heuristic estimators, the DTG-based h_{DTG} and the landmark-based h_{Land}. Both heuristics assess the quality of the plans according to the global information of the MAP task, offering an excellent accuracy while keeping agents' privacy.

  Additionally, MH-FMAP introduces the first multi-heuristic search strategy in MAP, an alternation scheme that combines both estimators. The results show that alternating h_{DTG} and h_{Land} dramatically improves the scalability and performance of the framework.

All in all, these strengths and innovations give rise to a framework capable of tackling MAP tasks of any complexity while offering a positive trade-off between solution quality and time consumption.

Nevertheless, MH-FMAP still presents some limitations which can be overcome in future developments of the model. The current weaknesses of our approach can be summarized as follows:

- The scope of this research is limited to cooperative MAP only, since one of the initial assumptions of our model is that agents are not self-interested. As mentioned in Chapter 1, MAP includes other interesting topics that are worth researching, such as self-interested agents or planning with preferences, among others.

  Additionally, our approach focuses only on suboptimal heuristic search. For this reason, the design of optimal heuristic estimators arises as a potential future line of research.

- Despite the optimizations introduced in the latest iteration of the framework, MH-FMAP still relies heavily on communications among agents. Agent coordination and synchronization is not only applied during the joint exploration of the search tree, but also during the heuristic evaluation of the plans, since global estimators entail an important communication effort.

- The analysis of the CoDMAP results in section 4.2.4 proves that the performance of MH-FMAP is limited when solving tasks described via a propositional specification. The precision of our h_{DTG} heuristic directly depends on the information provided by the DTGs, and thus, this estimator is heavily optimized for MAP tasks defined through state variables.

5.1 Future Work

According to the aforementioned strengths and limitations of our approach, we identified several potential future lines of research and development:

- **Improving the performance of the system in propositional tasks**: As described in section 4.2.4, MH-FMAP encounters performance issues when running propositional tasks described through *MA-PDDL*. This limitation is caused by the dependence of h_{DTG} on state variables: since predicates are converted into binary state variables, the resulting DTGs are not informative enough for h_{DTG} and the quality of the estimates it returns diminishes.

  In order to fix this issue, it is possible to develop and integrate a conversion tool that adapts propositional tasks to a richer state-variable-based representation, as in the Fast Downward planning system (50).

- **Optimizing the multi-heuristic strategy**: MH-FMAP constitutes our initial approach to distributed multi-heuristic search. To our knowledge, further research is required to develop our multi-heuristic scheme to its full potential. In this sense, we consider three possible lines of work:

  - **Generalizing multi-heuristic search**: It is possible to generalize the current alternation mechanism to a four-queue scheme as applied in Fast Downward (50), sorting both open nodes and preferred successors through all the available heuristic estimators.

  - **Developing additional heuristics**: Given the effectiveness of the current multi-heuristic scheme, we consider developing additional heuristic estimators to make them work in conjunction with h_{DTG} and h_{Land}. This can potentially provide MH-FMAP with more options to leave *plateaus*, thus increasing the efficiency of the multi-agent exploration.

  - **Improving existing estimators**: One of the take-home lessons of the research stay listed in section 1.2.2 is the possibility of using concepts derived from the Merge-and-Shrink heuristics to improve h_{DTG}. More

precisely, these estimators perform alternative merge and shrink procedures over an initial set of *atomic projections*, one per state variable, that guard a strong resemblance to the DTGs.

As shown in section 4.2.4, the size of the DTGs is critical to the accuracy of the h_{DTG} heuristic. For this reason, we believe that it is possible to combine (merge) DTGs to come up with more informative graphs, thus improving the overall accuracy of the h_{DTG} estimator.

- **Reducing the impact of communications**: The distributed procedures carried out by MH-FMAP entail a large amount of communications to continuously coordinate and synchronize the planning agents. In particular, agents perform a joint and complete search on the space of plans, being forced to exchange all the refinement plans they build. This results in a very demanding approach regarding communications, which notably slows down the system.

  Some works in the literature explored less demanding ways to carry out distributed search: MAD-A* allows agents to solve parts of the task concerning private information in isolation, thus reducing the communication needs (78). MAPR (11) introduces a sequential planning scheme in which each agent plans over the incremental solution produced by the previous agents, which results in an extremely fast MAP method.

  Therefore, one of our future lines of work focuses on reducing the communication needs of our approach via a smarter search scheme that takes advantage of the information distribution without losing the generality and solution quality inherent to MH-FMAP.

References

[1] E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julian, and M. Rebollo. **An abstract architecture for virtual organizations: the THOMAS approach**. *Knowledge and Information Systems*, **29**(2):379–403, 2011. 94

[2] C. Bäckström and B. Nebel. **Complexity results for SAS+ planning**. *Computational Intelligence*, **11**(4):625–655, 1995. 23, 25

[3] A. Barrett and D. S. Weld. **Partial-order planning: evaluating possible efficiency gains**. *Artificial Intelligence*, **67**(1):71–112, 1994. 26, 106

[4] A. Belesiotis, M. Rovatsos, and I. Rahwan. **Agreeing on plans through iterated disputes**. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 765–772, 2010. 86, 90, 92

[5] F. Bellifemine, A. Poggi, and G. Rimassa. **JADE: a FIPA2000 compliant agent development environment**. In *Proceedings of the 5th International Conference on Autonomous Agents (AGENTS)*, pages 216–217, 2001. 56, 94

[6] J. Benton, A.J. Coles, and A.I. Coles. **Temporal planning with preferences and time-dependent continuous costs**. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 2–10, 2012. 28, 55, 164, 203

[7] D.S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. **The complexity of decentralized control of Markov decision processes**. *Mathematics of Operations Research*, **27**(4):819–840, 2002. 259

[8] A. Blum and M.L. Furst. **Fast planning through planning graph analysis**. *Artificial Intelligence*, **90**(1-2):281–300, 1997. 25, 28, 88

[9] B. Bonet and H. Geffner. **Planning as heuristic search**. *Artificial Intelligence*, **129**:5–33, 2001. 24

[10] A. Bonisoli, A.E. Gerevini, A. Saetti, and I. Serina. **A privacy-preserving model for the multi-agent propositional planning problem**. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, pages 973–974, 2014. 198

[11] D. Borrajo. **Multi-agent planning by plan reuse**. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1141–1142, 2013. 2, 3, 39, 42, 49, 50, 53, 147, 151, 177, 180, 184, 198, 200, 217, 245, 267

[12] C. Boutilier and R.I. Brafman. **Partial-order planning with concurrent interacting actions**. *Journal of Artificial Intelligence Research*, **14**(105):136, 2001. 28, 89, 108, 156

[13] R.I. Brafman. **A privacy preserving algorithm for multi-agent planning and search**. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1530–1536, 2015. 41, 42, 43, 44, 52

[14] R.I. Brafman and C. Domshlak. **From one to many: planning for loosely coupled multi-agent systems**. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 28–35, 2008. 32, 38, 40, 47, 50, 64, 76, 86, 90, 147, 151, 160, 198, 263

[15] M. Brenner and B. Nebel. **Continual planning and acting in dynamic multiagent environments**. *Journal of Autonomous Agents and Multiagent Systems*, **19**(3):297–331, 2009. 68, 86, 90, 92, 108, 151, 156, 259

[16] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. **Planning under continuous time and resource uncertainty: a challenge for AI**. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 77–84, 2002. 179

[17] A.I. Coles, M. Fox, D. Long, and A. Smith. **Teaching forward-chaining planning with JavaFF**. In *Colloquium on AI Education, 23rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 17–22, 2008. 76, 91

[18] A.J. Coles, A.I. Coles, M. Fox, and D. Long. **Forward-chaining partial-order planning**. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 42–49, 2010. 28, 88, 89, 235

[19] J.S. Cox and E.H. Durfee. **An efficient algorithm for multiagent plan coordination**. In *Proceedings of 4th International Conference on Au-*

tonomous Agents and Multiagent Systems (AAMAS), pages 828–835, 2005. 64

[20] J.S. Cox and E.H. Durfee. **Efficient and distributable methods for solving the multiagent plan coordination problem**. *Multiagent and Grid Systems*, **5**(4):373–408, 2009. 49, 50, 150

[21] J.S. Cox, E.H. Durfee, and T. Bartold. **A distributed framework for solving the multiagent plan coordination problem**. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 821–827, 2005. 64, 90

[22] M. Crosby, M. Rovatsos, and R. Petrick. **Automated agent decomposition for classical planning**. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 46–54, 2013. 47, 51, 149, 198

[23] T. de la Rosa, A. García-Olaya, and D. Borrajo. **A case-based approach to heuristic planning**. *Applied Intelligence*, **39**(1):184–201, 2013. 171

[24] M. de Weerdt and B. Clement. **Introduction to planning in multiagent systems**. *Multiagent and Grid Systems*, **5**(4):345–355, 2009. 16, 64, 90, 146

[25] K. Decker, S. Khan, C.J. Schmidt, G. Situ, R. Makkena, and D. Michaud. **BioMAS: a multi-agent system for genomic annotation**. *International Journal of Cooperative Information Systems*, **11**(3):265–292, 2002. 260

[26] K. Decker and V.R. Lesser. **Generalizing the Partial Global Planning algorithm**. *International Journal of Cooperative Information Systems*, **2**(2):319–346, 1992. 90

[27] M.E. desJardins, E.H. Durfee, C.L. Ortiz, and M.J. Wolverton. **A survey of research in distributed continual planning**. *AI Magazine*, **20**(4):13–22, 1999. 2, 45, 64, 84, 85, 89, 92, 259

[28] Y. Dimopoulos, M.A. Hashmi, and P. Moraitis. **μ-SATPLAN: multi-agent planning as satisfiability**. *Knowledge-Based Systems*, **29**:54–62, 2012. 2, 50, 150

[29] P. Doshi. **On the role of interactive epistemology in multiagent planning**. In *Artificial Intelligence and Pattern Recognition*, pages 208–213, 2007. 93

[30] J. Dréo, P. Savéant, M. Schoenauer, and V. Vidal. **Divide-and-evolve: the marriage of Descartes and Darwin**. In *Proceedings of the 7th International Planning Competition (IPC)*, pages 29–30, 2011. 88

[31] E.H. Durfee. **Distributed problem solving and planning**. In *Multi-agents Systems and Applications: Selected tutorial papers from the 9th EC-CAI Advanced Course (ACAI) and AgentLink's 3rd European Agent Systems Summer School (EASSS)*, **LNAI 2086**, pages 118–149, 2001. 64, 84, 90

[32] E.H. Durfee and V. Lesser. **Partial Global Planning: a coordination framework for distributed hypothesis formation**. *IEEE Transactions on Systems, Man, and Cybernetics*, **21**(5):1167–1183, 1991. 2, 84, 85, 90

[33] S. Edelkamp. **Taming numbers and durations in the model checking integrated planning system**. *Journal of Artificial Intelligence Research*, **20**:195–238, 2003. 23

[34] E. Ephrati and J. S. Rosenschein. **Deriving consensus in multiagent systems**. *Artificial Intelligence*, **87**(1-2):21–74, 1996. 91

[35] K. Erol, J. Hendler, and D.S. Nau. **HTN planning: complexity and expressivity**. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, **94**, pages 1123–1128, 1994. 29

[36] R. Fikes and N.J. Nilsson. **STRIPS: a new approach to the application of theorem proving to problem solving**. *Artificial Intelligence*, **2**(3):189–208, 1971. 19, 22, 38, 102, 160

[37] R.L. Fogués, J.M. Alberola, J.M. Such, A. Espinosa, and A. García-Fornes. **Towards dynamic agent interaction support in open multiagent systems**. In *Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence (CCIA)*, pages 89–98, 2010. 56, 94, 99

[38] M. Fox and D. Long. **PDDL2.1: an extension to PDDL for expressing temporal planning domains**. *Journal of Artificial Intelligence Research*, **20**:61–124, 2003. 23

[39] H. Geffner. **Functional STRIPS: a more flexible language for planning and problem solving**. In *Logic-Based Artificial Intelligence*, pages 187–209, 2000. 19, 23

[40] A.E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. **Deterministic planning in the fifth International Planning**

Competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, **173**(5-6):619–668, 2009. 183

[41] A.E. Gerevini and D. Long. **Plan constraints and preferences in PDDL3**. *Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, 2005. 23

[42] A.E. Gerevini and D. Long. **Preferences and soft constraints in PDDL3**. In *Proceedings of the ICAPS Workshop on Planning with Preferences and Soft Constraints*, **6**, pages 46–53, 2006. 105

[43] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. **PDDL - the Planning Domain Definition Language**. *AIPS-98 Planning Committee*, 1998. 19, 22, 111

[44] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning. Theory and Practice*. Morgan Kaufmann, 2004. 1, 16, 17, 23, 27, 29, 155, 156

[45] P.J. Gmytrasiewicz and P. Doshi. **A framework for sequential planning in multi-agent settings**. *Journal of Artificial Intelligence Research*, **24**:49–79, 2005. 93

[46] A. Günay and P. Yolum. **Constraint satisfaction as a tool for modeling and checking feasibility of multiagent commitments**. *Applied Intelligence*, **39**(3):489–509, 2013. 182

[47] R.H. Guttman, A.G. Moukas, and P. Maes. **Agent-mediated electronic commerce: a survey**. *Knowledge Engineering Review*, **13**(2):147–159, 1998. 1

[48] P. Haslum and P. Jonsson. **Some results on the complexity of planning with incomplete information**. In *Proceedings of the 5th European Conference on Planning (ECP)*, pages 308–318, 1999. 93

[49] M. Helmert. **A planning heuristic based on causal graph analysis**. *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 161–170, 2004. 55, 172, 199, 207

[50] M. Helmert. **The Fast Downward planning system**. *Journal of Artificial Intelligence Research*, **26**(1):191–246, 2006. 25, 88, 215, 216, 266

[51] M. Helmert and C. Domshlak. **Landmarks, critical paths and abstractions: what's the difference anyway?** In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 162–169, 2009. 54, 200

[52] M. Helmert and H. Geffner. **Unifying the causal graph and additive heuristics**. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 140–147, 2008. 199, 206, 207, 241

[53] M. Helmert, P. Haslum, and J. Hoffmann. **Flexible abstraction heuristics for optimal sequential planning**. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 176–183, 2007. 54, 200

[54] J. Hoffmann and B. Nebel. **The FF planning system: fast planning generation through heuristic search**. *Journal of Artificial Intelligence Research*, **14**:253–302, 2001. 25, 29, 47, 54, 70, 88, 89, 91, 124, 172, 200, 241

[55] J. Hoffmann, J. Porteous, and L. Sebastiá. **Ordered landmarks in planning**. *Journal of Artificial Intelligence Research*, **22**:215–278, 2004. 199, 207, 211, 242, 243

[56] D. Jannach and M. Zanker. **Modeling and solving distributed configuration problems: a CSP-based approach**. *IEEE Transactions on Knowledge and Data Engineering*, **25**(3):603–618, 2013. 50, 151, 153

[57] N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, M. Wooldridge, and C. Sierra. **Automated negotiation: prospects, methods and challenges**. *Group Decision and Negotiation*, **10**(2):199–215, 2001. 1

[58] A. Jonsson and M. Rovatsos. **Scaling up multiagent planning: a best-response approach**. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, pages 114–121, 2011. 30, 65, 86, 92, 151

[59] S. Kambhampati. **Refinement planning as a unifying framework for plan synthesis**. *AI Magazine*, **18**(2):67–97, 1997. 67, 87, 95, 105

[60] G.A. Kaminka, D.V. Pynadath, and M. Tambe. **Monitoring teams by overhearing: A multi-agent plan-recognition approach**. *Journal of Artificial Intelligence Research*, **17**:83–135, 2002. 2, 84, 90

[61] H.A. Kautz. **Deconstructing planning as satisfiability**. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, **2**, pages 1524–1526, 2006. 29, 50

[62] J. Koehler and D. Ottiger. **An AI-based approach to destination control in elevators**. *AI Magazine*, **23**(3):59–78, 2002. 179

[63] M.T. Kone, A. Shimazu, and T. Nakajima. **The state of the art in agent communication languages**. *Knowledge and Information Systems*, **2**(3):259–284, 2000. 99

[64] D.L. Kovacs. **Complete BNF description of PDDL3.1**. Technical report, 2011. 23, 38, 69, 95, 111, 160

[65] S. Kraus. **Beliefs, time and incomplete information in multiple encounter negotiations among autonomous agents**. *Annals of Mathematics and Artificial Intelligence*, **20**(1-4):111–159, 1997. 93

[66] A. Kumar, S. Zilberstein, and M. Toussaint. **Scalable multiagent planning using probabilistic inference**. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2140–2146, Barcelona, Spain, 2011. 93

[67] J. Kvarnström. **Planning for loosely coupled agents using partial order forward-chaining**. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, pages 138–145. AAAI, 2011. 47, 64, 65, 85, 147, 149, 151, 164

[68] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. Prasad, A. Raja, et al. **Evolution of the GPGP/TAEMS domain-independent coordination framework**. *Autonomous Agents and Multi-Agent Systems*, **9**(1-2):87–143, 2004. 51, 55, 94, 151, 260

[69] N. Lipovetzky and H. Geffner. **Searching for plans with carefully designed probes**. In *Proceedings of the 21th International Conference on Automated Planning and Scheduling (ICAPS)*, 2011. 88

[70] D. Long and M. Fox. **The 3rd International Planning Competition: results and analysis**. *Journal of Artificial Intelligence Research*, **20**:1–59, 2003. 179, 181, 182

[71] D. Long, H.A. Kautz, B. Selman, B. Bonet, H. Geffner, J. Koehler, M. Brenner, J. Hoffmann, F. Rittinger, C. Anderson, D. Weld, D. Smith, M. Fox, and D. Long. **The AIPS-98 Planning Competition**. *AI Magazine*, **21**:13–33, 2000. 23

[72] S. Maliah, G. Shani, and R. Stern. **Privacy preserving landmark detection**. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, pages 597–602, 2014. 3, 39, 52, 54, 199, 201, 220, 221, 247

[73] J. McCarthy and P.J. Hayes. **Some philosophical problems from the standpoint of artificial intelligence**. *Machine Intelligence*, **4**:463–502, 1969. 19

[74] D. McDermott. **The 1998 AI planning systems competition**. *AI Magazine*, **21**(2):35–55, 2000. 22

[75] C. Micacchi and R. Cohen. **A framework for simulating real-time multi-agent systems**. *Knowledge and information systems*, **17**(2):135–166, 2008. 84

[76] N.T. Nguyen and R.P. Katarzyniak. **Actions and social interactions in multi-agent systems**. *Knowledge and Information Systems*, **18**(2):133–136, 2009. 1, 84

[77] X.L. Nguyen and S. Kambhampati. **Reviving partial order planning**. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 459–464, 2001. 28, 87, 95

[78] R. Nissim and R.I. Brafman. **Multi-agent A* for parallel and distributed systems**. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1265–1266, 2012. 53, 54, 198, 200, 267

[79] R. Nissim and R.I. Brafman. **Distributed heuristic forward search for multi-agent planning**. *Journal of Artificial Intelligence Research*, **51**:293–332, 2014. 42, 44, 51, 52, 54, 55, 201

[80] R. Nissim, R.I. Brafman, and C. Domshlak. **A general, fully distributed multi-agent planning algorithm**. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1323–1330, 2010. 2, 3, 39, 50, 58, 64, 76, 77, 79, 86, 91, 147, 150, 151, 177, 180, 181, 198, 244

[81] P.D. O'Brien and R.C. Nicol. **FIPA - towards a standard for software agents**. *BT Technology Journal*, **16**(3):51–59, 1998. 56, 177, 206

[82] J. Ota. **Multi-agent robot systems as distributed autonomous systems**. *Advanced Engineering Informatics*, **20**(1):59–70, 2006. 1

[83] P. Öztürk, K. Rossland, and O. Gundersen. **A multiagent framework for coordinated parallel problem solving**. *Applied Intelligence*, **33**(2):132–143, 2010. 177

[84] S. Pajares and E. Onaindia. **Defeasible argumentation for multi-agent planning in ambient intelligence applications**. In *Proceedings*

of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 509–516, 2012. 92

[85] S. Pajares and E. Onaindia. **Context-aware multi-agent planning in intelligent environments**. *Information Sciences*, **227**:22–42, 2013. 260

[86] M. Paolucci, O. Shehory, K. Sycara, D. Kalp, and A. Pannu. **A planning component for RETSINA agents**. *Intelligent Agents VI. Agent Theories Architectures, and Languages*, pages 147–161, 2000. 55, 94

[87] S. Parsons, C. Sierra, and N.R. Jennings. **Agents that reason and negotiate by arguing**. *Journal of Logic Computation*, **8**(3):261–292, 1998. 135

[88] M. Pechoucek, M. Rehák, P. Charvát, T. Vlcek, and M. Kolar. **Agent-based approach to mass-oriented production planning: case study**. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, **37**(3):386–395, 2007. 260

[89] E. Pednault. **ADL: exploring the middle ground between STRIPS and the situation calculus**. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 324–332, 1989. 22

[90] J.S. Penberthy and D.S. Weld. **UCPOP: a sound, complete, partial order planner for ADL**. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 103–114, 1992. 27, 89

[91] R.P.A. Petrick and F. Bacchus. **A knowledge-based approach to planning with incomplete information and sensing**. In *Proceedings of*

the 6th International Conference on Artificial Intelligence Planning Systems (AIPS), pages 212–222, 2002. 259

[92] J. Pita, M. Jain, J. Marecki, F. Ordóñez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. **Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport**. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 125–132, 2008. 1

[93] D.V. Pynadath and M. Tambe. **The communicative multiagent team decision problem: analyzing teamwork theories and models**. *Journal of Artificial Intelligence Research*, pages 389–423, 2002. 259

[94] P. Reddy and M. Veloso. **Strategy learning for autonomous agents in smart grid markets**. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1446–1451, 2011. 260

[95] S. Richter and M. Westphal. **The LAMA planner: guiding cost-based anytime planning with landmarks**. *Journal of Artificial Intelligence Research*, **39**(1):127–177, 2010. 25, 49, 54, 88, 184, 198, 200, 215

[96] R. Röger and M. Helmert. **The more, the merrier: combining heuristic estimators for satisficing planning**. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 246–249, 2010. 214, 215, 216, 238

[97] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003. 22

[98] Ó. Sapena and E. Onaindia. **Planning in highly dynamic environments: an anytime approach for planning under time constraints**. *Applied Intelligence*, **29**(1):90–109, 2008. 156

[99] Ó. Sapena, E. Onaindia, A. Garrido, and M. Arangú. **A distributed CSP approach for collaborative planning systems**. *Engineering Applications of Artificial Intelligence*, **21**(5):698–709, 2008. 47, 149

[100] Ó. Sapena, E. Onaindia, and A. Torreño. **FLAP: Applying least-commitment in forward-chaining planning**. *AI Communications*, 2014. 28

[101] E. Serrano, J.M. Such, J.A. Botía, and A. García-Fornes. **Strategies for avoiding preference profiling in agent-based e-commerce environments**. *Applied Intelligence*, pages 1–16, 2013. 33, 146

[102] S. Seuken and S. Zilberstein. **Formal models and algorithms for decentralized decision making under uncertainty**. *Autonomous Agents and Multi-Agent Systems*, **17**(2):190–250, 2008. 258, 260

[103] G. Shani, S. Maliah, and R. Stern. **Stronger privacy preserving projections for multi-agent planning**. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS)*, page In Press, 2016. 41, 44, 48, 257

[104] D.E. Smith, J. Frank, and A.K. Jónsson. **Bridging the gap between planning and scheduling**. *Knowledge Engineering Review*, **15**(1):47–83, 2000. 175

[105] M.T.J. Spaan, F.A. Oliehoek, and N.A. Vlassis. **Multiagent planning under uncertainty with stochastic communication delays**. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 338–345, 2008. 259

[106] M. Štolba and A. Komenda. **Fast-forward heuristic for multiagent planning**. In *Proceedings of the 1st Workshop on Distributed and Multi-Agent Planning (DMAP)*, pages 75–83, 2013. 54, 200, 207

[107] M. Štolba and A. Komenda. **Relaxation heuristics for multiagent planning**. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 298–306, 2014. 3, 54, 198, 201

[108] M. Štolba and A. Komenda. **Comparison of RPG-based FF and DTG-based FF distributed heuristics**. In *Proceedings of the 3rd Workshop on Distributed and Multi-Agent Planning (DMAP)*, pages 77–82, 2015. 242

[109] J.M. Such, A. García-Fornes, A. Espinosa, and J. Bellver. **Magentix2: a privacy-enhancing agent platform**. *Engineering Applications of Artificial Intelligence*, pages 96–109, 2012. 177

[110] K. Sycara and A.S. Pannu. **The RETSINA multiagent system (video session): towards integrating planning, execution and information gathering**. In *Proceedings of the 2nd International Conference on Autonomous Agents (Agents)*, pages 350–351, 1998. 55, 94

[111] M. Tambe. **Towards flexible teamwork**. *Journal of Artificial Intelligence Research*, **7**:83–124, 1997. 2, 64, 84

[112] Y. Tang, T. Norman, and S. Parsons. **A model for integrating dialogue and the execution of joint plans**. *Argumentation in Multi-Agent Systems*, pages 60–78, 2010. 91

[113] H. Tonino, A. Bos, M. de Weerdt, and C. Witteveen. **Plan coordination by revision in collective agent based systems**. *Artificial Intelligence*, **142**(2):121–145, 2002. 50, 64, 90, 150

[114] A. Torreño, E. Onaindia, and Ó. Sapena. **An approach to multi-agent planning with incomplete information**. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, **242**, pages 762–767. IOS Press, 2012. 151, 177, 180, 181, 187, 188

[115] A. Torreño, E. Onaindia, and Ó. Sapena. **A flexible coupling approach to multi-agent planning under incomplete information**. *Knowledge and Information Systems*, **38**(1):141–178, 2014. 38, 151, 160

[116] A. Torreño, E. Onaindia, and Ó. Sapena. **FMAP: distributed cooperative multi-agent planning**. *Applied Intelligence*, **41**(2):606–626, 2014. 199, 201, 217

[117] A. Torreño, Ó. Sapena, and E. Onaindia. **Global heuristics for distributed cooperative multi-agent planning**. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 225–233, 2015. 55

[118] A. Torreño, Ó. Sapena, and E. Onaindia. **MH-FMAP: alternating global heuristics in multi-agent planning**. In *Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP)*, pages 25–28, 2015. 223, 248

[119] R. van der Krogt. **Quantifying privacy in multiagent planning**. *Multiagent and Grid Systems*, **5**(4):451–469, 2009. 45, 147

[120] R. Van Der Krogt and M. De Weerdt. **Plan repair as an extension of planning**. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 161–170, 2005. 51, 90, 91, 150

[121] D.S. Weld. **An introduction to least commitment planning**. *AI Magazine*, **15**(4):27–61, 1994. 26, 89

[122] D.S. Weld. **Recent advances in AI planning**. *AI Magazine*, **20**(2):93–123, 1999. 16, 88

[123] D.E. Wilkins and K.L. Myers. **A multiagent planning architecture**. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 154–162, 1998. 55, 94

[124] F. Wu, S. Zilberstein, and X. Chen. **Online planning for multi-agent systems with bounded communication**. *Artificial Intelligence*, **175**(2):487–511, 2011. 93

[125] M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara. **The distributed constraint satisfaction problem: formalization and algorithms**. *IEEE Transactions on Knowledge and Data Engineering*, **10**(5):673–685, 1998. 153

[126] H.L.S. Younes and R.G. Simmons. **VHPOP: versatile heuristic partial order planner**. *Journal of Artificial Intelligence Research*, **20**:405–430, 2003. 27, 28, 67, 71, 87, 89, 95, 240

[127] J.F. Zhang, X.T. Nguyen, and R. Kowalczyk. **Graph-based multi-agent replanning algorithm**. In *Proceedings of the 6th Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 798–805, 2007. 54, 70, 88, 95, 123, 172, 200, 207, 213