

Document downloaded from:

<http://hdl.handle.net/10251/47255>

This paper must be cited as:

Torreño Lerma, A.; Onaindia De La Rivaherrera, E.; Sapena Vercher, O. (2014). A flexible coupling approach to multi-agent planning under incomplete information. *Knowledge and Information Systems*. 38:141-178. doi:10.1007/s10115-012-0569-7.



The final publication is available at

<http://dx.doi.org/10.1007/s10115-012-0569-7>

Copyright Springer Verlag (Germany)

A Flexible Coupling Approach to Multi-Agent Planning under Incomplete Information

Alejandro Torreño¹, Eva Onaindia¹ and Óscar Sapena¹

¹Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Valencia, Spain

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.

Keywords: Planning & scheduling; Multi-agent systems

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,

Received Jan 25, 2012

Revised Jun 15, 2012

Accepted Sep 01, 2012

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 (Nguyen and Katarzyniak, 2009), 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 (Durfee, 2001).

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 (desJardins, Durfee, Ortiz and Wolverton, 1999). 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 (Durfee and Lesser, 1991; Tambe, 1997; Kaminka, Pynadath and Tambe, 2002). 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 (Micacchi and Cohen, 2008).

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 (desJardins et al., 1999). 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 (Durfee and Lesser, 1991) 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 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 (Kvarnström, 2011) 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 (Brafman and Domshlak, 2008) 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 (Nissim, Brafman and Domshlak, 2010).

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 (Nissim et al., 2010). 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 (Jonsson and Rovatsos, 2011; Belesiotis, Rovatsos and Rahwan, 2010). 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 (Brenner and Nebel, 2009; Belesiotis et al., 2010) 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.

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 (Kambhampati, 1997) 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 (Nguyen and Kambhampati, 2001; Younes and Simmons, 2003).

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 2 summarizes some background on the main topics related to this work and reviews the most recent literature on MAP; section 3 introduces the example MAP scenario we will use to illustrate the different aspects of our framework; section 4 outlines our MAP architecture; section 5 presents the theoretical planning model upon which our system is based; section 6 outlines the planning language used to model MAP tasks; section 7 provides an overview of the MAP algorithm followed by the agents; section 8 describes the first stage of our MAP algorithm, the initial information exchange; section 9 outlines second stage of the MAP algorithm, the refinement planning and coordination protocol; section 10 presents the experimental results, and finally, section 11 concludes and summarizes our future lines of research.

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.

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 (Weld, 1999). 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 (Richter and Westphal, 2010), domain transition graphs (Helmert, 2006), forward-chaining partial-order planning (Coles, Coles, Fox and Long, 2010), probes (Lipovetzky and Geffner, 2011) or divide-and-conquer strategies (Dréo, Savéant, Schoenauer and Vidal, 2011), among others.

The work in (Blum and Furst, 1997) 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 (Hoffmann and Nebel, 2001). This technique has been integrated in many single-agent planning frameworks and has also been extended to a distributed context (Zhang, Nguyen and Kowalczyk, 2007).

While state-space planners such as Fast Forward (Hoffmann and Nebel, 2001) 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 (Penberthy and Weld, 1992; Younes and Simmons, 2003) 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* (Weld, 1994), 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 (Coles et al., 2010), 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 (Boutilier and Brafman, 2001).

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 (desJardins et al., 1999). 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 (Durfee, 2001): 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) (Belesiotis et al., 2010; Brenner and Nebel, 2009), while others apply only coordination after planning (avoiding stage 3) (Van Der Krogt and De Weerd, 2005; Cox, Durfee and Bartold, 2005).

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 (Brafman and Domshlak, 2008). 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 (Durfee, 2001; de Weerd and Clement, 2009; Tonino, Bos, de Weerd and Witteveen, 2002; Kaminka et al., 2002).

Different coordination techniques have been proposed for merging and gathering several individual plans into a single joint plan. The Partial Global Planning framework (Durfee and Lesser, 1991) and its extension, the Generalized Partial Global Planning approach (Decker and Lesser, 1992), 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 (Tonino et al., 2002) 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. (Nissim et al., 2010) introduces a cooperative MAP approach for loosely-coupled systems in which agents carry out planning individually through a state-based planner (Hoffmann and Nebel, 2001; Coles, Fox, Long and Smith, 2008). The resulting local plans are then coordinated by solving a distributed Constraint Satisfaction Problem. The approach in (Van Der Krogt and De Weerd, 2005) 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 (Ephrati and Rosenschein, 1996) or consider the communication needs that arise when plans are being executed (Tang, Norman and Parsons, 2010).

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 (Nissim et al., 2010). 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 (desJardins et al., 1999) applies the continual planning approach, which interleaves planning and execution and coordinates agents by synchronizing them at execution time (Brenner and Nebel, 2009). The approach in (Jonsson and Rovatsos, 2011) introduces the best-response planning algorithm, which iteratively improves the quality of the agents' plans through single-agent planning technology. Finally, the works in (Belesiotis et al., 2010; Pajares and Onaindia, 2012) 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 afore-

mentioned 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 (Haslum and Jonsson, 1999). 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 (Wu, Zilberstein and Chen, 2011; Kumar, Zilberstein and Toussaint, 2011); and the epistemological way, which assumes that agents have beliefs about the state of the world and beliefs over the other agents' knowledge (Kraus, 1997; Doshi, 2007). This latter approach has been widely used in games of incomplete information (Gmytrasiewicz and Doshi, 2005). 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.

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 (Wilkins and Myers, 1998) 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 (Lesser, Decker, Wagner, Carver, Garvey, Horling, Neiman, Podorozhny, Prasad, Raja et al., 2004) 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

(Fogués, Alberola, Such, Espinosa and Garcia-Fornes, 2010; Argente, Botti, Carascosa, Giret, Julian and Rebollo, 2011) or JADE (Bellifemine, Poggi and Rimassa, 2001), 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 (Sycara and Pannu, 1998) introduced a planning component (Paolucci, Shehory, Sycara, Kalp and Pannu, 2000). 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.

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 (Kambhampati, 1997) and an individual Partial-Order Planning (POP) (Nguyen and Kambhampati, 2001; Younes and Simmons, 2003). 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 (Kovacs, 2011) 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 (Zhang et al., 2007) 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. 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 appli-

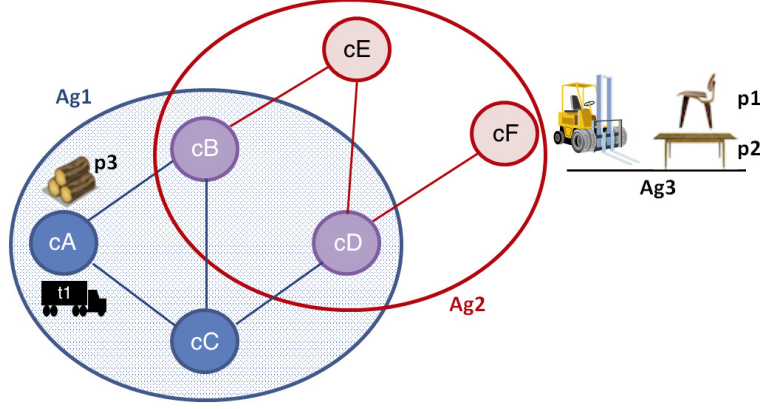


Fig. 1. Transportation and storage scenario

cation, depicted in Figure 1, 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 1 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 1 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 1, 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 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 .

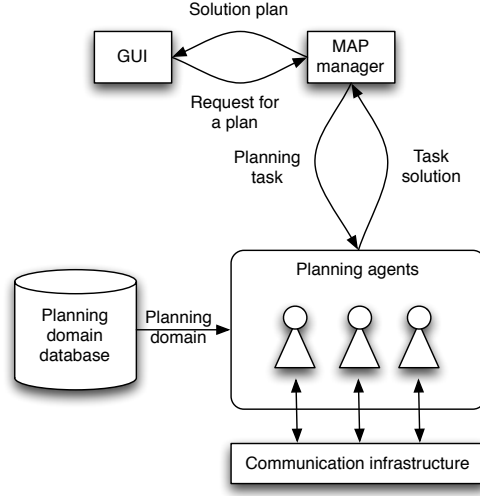


Fig. 2. MAP system architecture

4. Multi-Agent Planning architecture

The architecture of our MAP system is depicted in Figure 2. 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, two agents play the role of transport agencies and a third 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 6). The first file defines the typology and the planning capabilities of the agent,

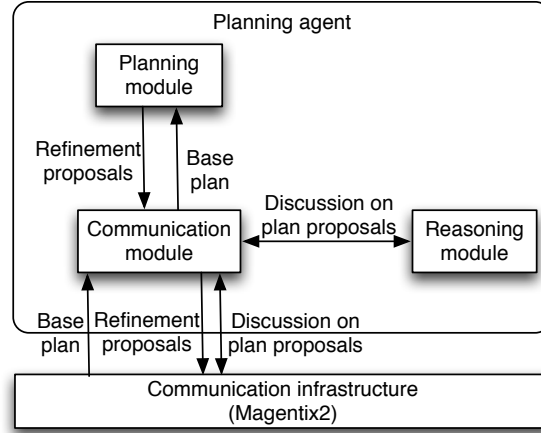


Fig. 3. Internal structure of a planning agent

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. 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 (Kone, Shimazu and Nakajima, 2000). The developed MAP system uses the Magentix2 MAS platform (Fogués et al., 2010) 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 from a set of refinement plans and communicate with each other (see Figure 3). 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 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.

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 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. Also, for the sake of clarification of some definitions, we point out the reader to the figures of plans showed in section 9.

5.1. Formalization of a MAP task

Definition 1. (MAP task) A MAP task is a tuple $\mathcal{T} = \langle \mathcal{AG}, \mathcal{O}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$. $\mathcal{AG} = \{1, \dots, n\}$ 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 (Fikes and Nilsson, 1971), 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 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 2, a fluent relates a variable with one of the values in its domain. For instance, let $(\text{at } \mathbf{t1})$ be a variable that refers to the position of a truck object $\mathbf{t1}$ in the example introduced in section 3. Possible values for this variable are the cities \mathbf{cA} , \mathbf{cB} , \mathbf{cC} , \mathbf{cD} , \mathbf{cE} and \mathbf{cF} . Then, a positive fluent $\langle \text{at } \mathbf{t1}, \mathbf{cA} \rangle$ indicates that $\mathbf{t1}$ is in \mathbf{cA} while a negative fluent $\langle \text{at } \mathbf{t1}, \neg \mathbf{cA} \rangle$ indicates that $\mathbf{t1}$ is not in \mathbf{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, $\mathcal{V} = \{\mathcal{V}^i\}_{i=1}^n$. $\mathcal{D}_v^i \subseteq \mathcal{D}_v$ 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 $\mathcal{I} = \bigcup_{i \in \mathcal{AG}} \mathcal{I}^i$. 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. Initially, Ag1 knows that the truck $\mathbf{t1}$ is in city \mathbf{cA} so the fluent $\langle \text{at } \mathbf{t1}, \mathbf{cA} \rangle$ is part of \mathcal{I}^{Ag1} . On the contrary, Ag2 does not know where $\mathbf{t1}$ is initially located, but it knows that the truck is not in city \mathbf{cB} . Hence, the fluent $\langle \text{at } \mathbf{t1}, \neg \mathbf{cB} \rangle$ 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 $\mathcal{A} = \bigcup_{i \in \mathcal{AG}} \mathcal{A}^i$. An action α is said to be public if it is shared by two or more agents, i.e. $\alpha \in \mathcal{A}^i \wedge \alpha \in \mathcal{A}^j$, $i \neq j$. $\alpha \in \mathcal{A}^i$ 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. (Planning rule or action) A **planning rule or action** $\alpha \in \mathcal{A}$ is a tuple $\langle PRE(\alpha), EFF(\alpha) \rangle$. $PRE(\alpha) = \{p_1, \dots, p_n\}$ is a set of fluents that represents the preconditions of α , while $EFF(\alpha) = \{e_1, \dots, e_m\}$ 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., $\mathcal{D}_{at\ t1}^{Ag1} = \{\mathbf{cA}, \mathbf{cB}, \mathbf{cC}, \mathbf{cD}\}$. If Ag1 knows a positive fluent $\langle \mathbf{at\ t1}, \mathbf{cA} \rangle$, it also knows the negative fluents $\langle \mathbf{at\ t1}, \neg \mathbf{cB} \rangle$, $\langle \mathbf{at\ t1}, \neg \mathbf{cC} \rangle$ and $\langle \mathbf{at\ t1}, \neg \mathbf{cD} \rangle$.
- 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 $\langle \mathbf{at\ t1}, \neg \mathbf{cB} \rangle$ 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 (Gerevini and Long, 2006), as it is not mandatory that agents achieve them.

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 (Kambhampati, 1997). An agent proposes a plan Π that typically enforces some of the goals that have not yet been supported (see definition 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) (Barrett and Weld, 1994)

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.

5.2.1. Single-agent Partial-Order Planning

Definition 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 $\alpha \xrightarrow{\langle v, d \rangle} \beta$ or $\alpha \xrightarrow{\langle v, \neg d \rangle} \beta$, where $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{A}$ are actions in Δ . $\alpha \xrightarrow{\langle v, d \rangle} \beta$ 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 $\langle v, d \rangle \in PRE(\beta)$. $\alpha \xrightarrow{\langle v, \neg d \rangle} \beta$ 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 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 $\nexists \alpha \in \Delta / \alpha \xrightarrow{og} \beta \in \mathcal{CL}$, i.e., an open goal og is a precondition of action β in the plan Π that is not yet supported by a causal link $\alpha \xrightarrow{og} \beta \in \mathcal{CL}$. $openGoals(\Pi)$ 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 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 $\alpha \xrightarrow{\langle v, d \rangle} \beta$ 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 $\alpha \xrightarrow{\langle v, \neg d \rangle} \beta$ 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 $\alpha \xrightarrow{\langle v, d \rangle} \beta$ or $\alpha \xrightarrow{\langle v, \neg d \rangle} \beta$, 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 4 in section 9 depicts a refinement plan for the example introduced in section 3. This refinement plan includes a causal link $\text{Init} \xrightarrow{\langle \text{at t1, cA} \rangle} \text{load t1 p3 cA}$. 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, $(\text{at t1} = \text{cB}) \in EFF(\text{drive t1 cA cB})$. This effect causes a threat over the previous causal link, as it introduces a fluent $\langle \text{at t1, cB} \rangle$ that affects the value of the variable (at t1) . This threat can be solved by introducing an ordering constraint $\text{load t1 p3 cA} \prec \text{drive t1 cA cB}$, i.e., demoting the threatening action drive t1 cA cB with respect to the causal link.

5.2.2. 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 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 (Brenner and Nebel, 2009). We impose the additional concurrency constraint that the preconditions of two actions have to be consistent (Boutilier and Brafman, 2001) for these two actions to be mutually consistent. This definition of concurrency is straightforwardly extended to a joint action $\alpha = \langle \alpha_1, \dots, \alpha_n \rangle$.

Definition 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, two concurrent actions **drive t1 cA cB**, planned by agent Ag1, and **drive t1 cA cC**, planned by agent Ag2, are mutually inconsistent as $(\text{at t1} = \text{cB}) \in EFF(\text{drive t1 cA cB})$ and $(\text{at t1}, \text{cC}) \in PRE(\text{drive t1 cC cB})$ (the first condition in Definition 8 holds). Concurrent actions **drive t1 cA cB** and **drive t1 cA cC** are also mutually inconsistent as $(\text{at t1} = \text{cB}) \in EFF(\text{drive t1 cA cB})$ and $(\text{at t1} = \text{cC}) \in EFF(\text{drive t1 cA cC})$ (second condition holds). Finally, concurrent actions **drive t1 cA cB** and **drive t1 cC cB** are mutually inconsistent as $(\text{at t1}, \text{cA}) \in PRE(\text{drive t1 cA cB})$ and $(\text{at t1}, \text{cC}) \in PRE(\text{drive t1 cC cB})$ (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 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 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 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 12. (Solution plan) A multi-agent concurrent plan Π is a **solution plan** for a planning task \mathcal{T} if $\text{openGoals}(\Pi) = \emptyset$ (Π is a complete plan), $\text{Threats}(\Pi) = \emptyset$, 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 6 in section 9 shows a solution plan for the MAP task presented in section 3. The different shapes of the actions indicate which agent has proposed them. The solution plan in Figure 6 is a complete, concurrent plan to which all the agents in the MAP task have contributed.

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* (Kovacs, 2011), the most recent version of *PDDL* (Ghallab, Howe, Knoblock, McDermott, Ram, Veloso, Weld and Wilkins, 1998), 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 with our language.

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 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).

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.

6.3. Encoding example

This subsection describes the encoding of the MAP task presented in section 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.

6.3.1. 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 1 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 throughout 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 outgoing 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 1 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 1), 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.

6.3.2. 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 domain 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 1). 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.

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 1 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 1 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 1, 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 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 8 describes the initial information exchanging stage performed by the agents, while section 9 details the resolution process, including both the coordination process and the individual construction of the refinement plans.

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 (Zhang et al., 2007). Unlike the proposal in (Zhang et al., 2007), 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 6). This section describes in detail the dis-RPG building process and subsection 8.1 provides a trace based on the MAP task presented in section 3 that illustrates this process.

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 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 2, 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 (Hoffmann and Nebel, 2001). 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

Algorithm 2 Dis-RPG construction for an agent i

```

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

```

of the dis-RPG begins. As depicted in Algorithm 2, 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 6.1). Agent i sends agent j the set of fluents $SF^{i \rightarrow j}$ 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 $v \in \mathcal{V}^i \cap \mathcal{V}^j$ 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 2). 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 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 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

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 1. Initial RPG built by agent Ag2

F0		A0	F1	A1
[1](at t1) 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 2. Initial RPG built by agent Ag1

the others. Following, agents start the resolution process to jointly devise a solution plan.

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. The planning agents receive the input files presented in subsection 6.3 and start the MAP algorithm by building the dis-RPG.

In the first stage of Algorithm 2, 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 1 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 1 in section 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 2). 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

F0	A0	F1	A1
[1, 2](empty cB) T	[2](link cE cB) T	[1](empty cA) T	unload t1 p3 cB
[1, 2](empty cD) T	[2](link cE cD) T	[1, 2](at t1) cB	unload t1 p3 cC
[2](empty cE) T	[2](link cF cD) T	[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. Final dis-RPG as viewed by agent Ag2

(= (at t1) cA). As Table 2 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 6.3.2, 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 2. 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 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 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.

9. Resolution process

After the information exchange, agents initiate the resolution process (see Algorithm 3), 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 3 Resolution process for an agent i

```

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

```

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 3, agent i refines Π by solving a particular open goal $g \in \text{openGoals}(\Pi)$, thus obtaining a set of valid refinement plans over Π_g , $\text{Refinements}^i(\Pi_g)$.

Our definition of refinement plan (see subsection 5.2.2) states that a refinement plan Π^i of an agent i over a base plan Π solves one of its open goals $g \in \text{openGoals}(\Pi)$, 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 $v \in \mathcal{V}^i \wedge d \in \mathcal{D}_v^i \wedge ((\forall j \neq i, v \notin \mathcal{V}^j) \vee (\forall j \neq i, d \notin \mathcal{D}_v^j)) \wedge (g^i \notin \text{openGoals}(\Pi))$.

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.

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 3 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 $g \in \text{openGoals}(\Pi)$ and each agent i starts a new individual refinement stage to compute the refinements over Π , $\text{Refinements}^i(\Pi_g)$.

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 $g \in \text{openGoals}(\Pi)$, $\text{Refinements}^i(\Pi_g)$, to the rest of agents in the task. In turn, agent i receives the refinements devised by each agent j in the task, $\text{Refinements}^j(\Pi_g)$, 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 $og \in \text{openGoals}(\Pi)$ of the form $\langle v, d \rangle$ or $\langle v, \neg d \rangle$ such that $v \in \mathcal{V}^i$ and $d \in \mathcal{D}_v^i$. The view agent i has on each refinement plan Π , $\text{view}^i(\Pi)$, 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, $\mathcal{F}(\text{view}^i(\Pi))$, 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 3). 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.

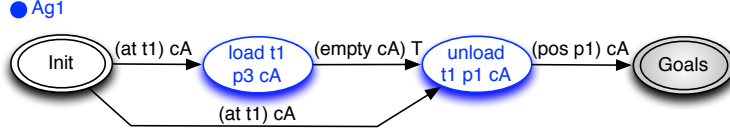
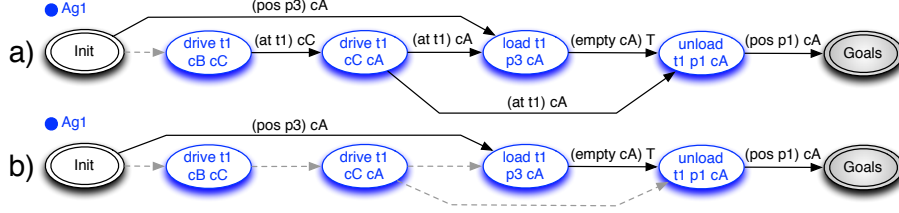
Once a refinement plan is selected, agents adopt it as the new base plan Π . If $\text{openGoals}(\Pi) = \emptyset$, a solution plan is returned. As some open goals might not be visible to some agents, all agents must confirm that Π is a solution plan according to their view of Π . Finally, the baton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

9.3. Resolution example

This subsection illustrates the resolution process by showing a partial trace that follows the trace example described in subsection 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

Fig. 4. Refinement plan Π_{00} Fig. 5. Refinement plan Π_{06} as observed by: a) Ag1 b) Ag3

MAP task. The first open goal selected by Ag1 (which takes the role of baton agent in this first iteration) for its resolution is $(= (\text{pos } p1) \text{ cA})$, as it is the most costly one according to the dis-RPG. The goals of the task are highlighted in bold in Table 3. This dis-RPG shows that $(= (\text{pos } p1) \text{ cA})$ has a cost of 5, $(= (\text{pos } p3) \text{ cF})$ has a cost of 4, and the only agent that can achieve $(= (\text{pos } p1) \text{ cA})$ is Ag1. Hence, Ag1 proposes a set of refinements over $\Pi_0, \Pi_{00}, \dots, \Pi_{09}$, 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 4 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 5). 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 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 6 depicts the solution plan for the MAP task

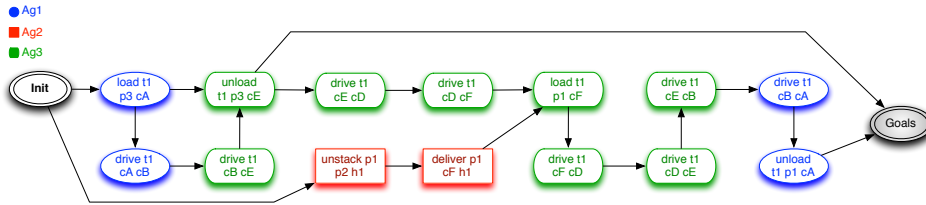


Fig. 6. Solution plan for the MAP task

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.

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.

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, 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.

10.1.1. Picture domain

This domain, adapted from the case study in (Parsons, Sierra and Jennings, 1998), 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 7 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

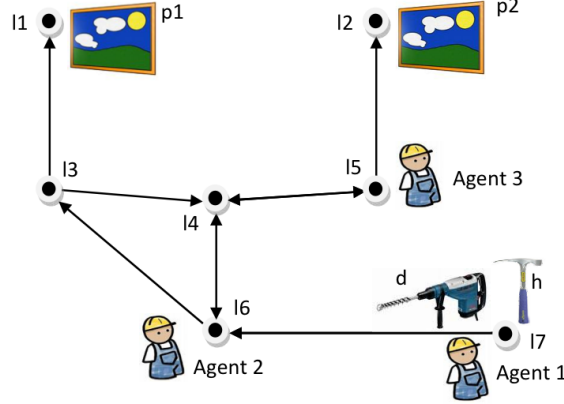


Fig. 7. Picture domain example

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.

10.2. Tests and results

The following subsections show the experimental results. We carried out two different tests¹. 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.

10.2.1. 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 10.1, the *transportation* problems present a high 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

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

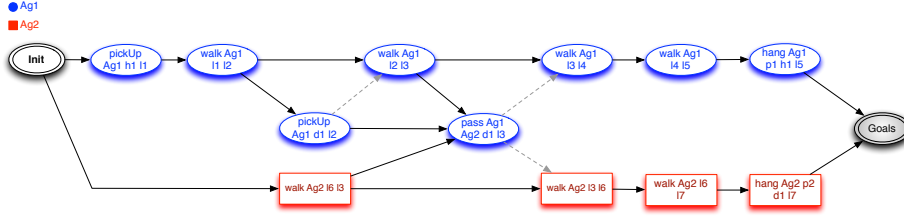


Fig. 8. Solution plan for the Picture2 MAP problem

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 4 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 8 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 8 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 ap-

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 4. Single-Agent vs. Multi-Agent Planning comparison

proaches 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 8 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 4 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 4.

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 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.

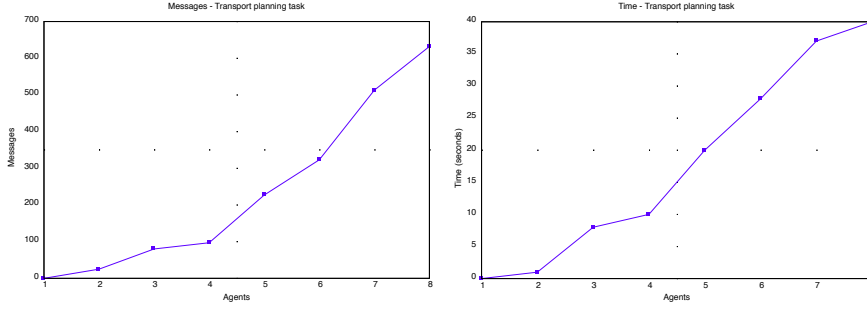


Fig. 9. Scalability results for the transportation domain

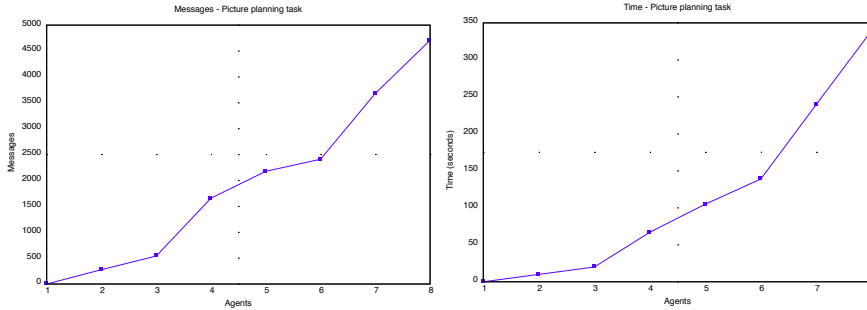


Fig. 10. Scalability results for the picture domain

10.2.2. 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 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 9 and 10 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.

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.

Acknowledgements. This work has been partly supported by the Spanish MICINN under projects Consolider Ingenio 2010 CSD2007-00022 and TIN2011-27652-C03-01, and the Valencian Prometeo project 2008/051.

References

- Argente, E., Botti, V., Carrascosa, C., Giret, A., Julian, V. and Rebollo, M. (2011), ‘An abstract architecture for virtual organizations: the THOMAS approach’, *Knowledge and Information Systems* **29**(2), 379–403.
- Barrett, A. and Weld, D. S. (1994), ‘Partial-order planning: Evaluating possible efficiency gains’, *Artificial Intelligence* **67**(1), 71–112.
- Belesiotis, A., Rovatsos, M. and Rahwan, I. (2010), Agreeing on plans through iterated disputes, in ‘Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems’, pp. 765–772.
- Bellifemine, F., Poggi, A. and Rimassa, G. (2001), JADE: a FIPA2000 compliant agent development environment, in ‘Proceedings of the 5th international conference on Autonomous Agents (AAMAS)’, ACM, pp. 216–217.
- Blum, A. and Furst, M. L. (1997), ‘Fast planning through planning graph analysis’, *Artificial Intelligence* **90**(1-2), 281–300.
- Boutilier, C. and Brafman, R. (2001), ‘Partial-order planning with concurrent interacting actions’, *Journal of Artificial Intelligence Research* **14**(105), 136.
- Brafman, R. and Domshlak, C. (2008), From one to many: Planning for loosely coupled multi-agent systems, in ‘Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)’, pp. 28–35.
- Brenner, M. and Nebel, B. (2009), ‘Continual planning and acting in dynamic multiagent environments’, *Journal of Autonomous Agents and Multiagent Systems* **19**(3), 297–331.
- Coles, A., Coles, A., Fox, M. and Long, D. (2010), Forward-chaining partial-order planning, in ‘Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)’, pp. 42–49.
- Coles, A., Fox, M., Long, D. and Smith, A. (2008), Teaching forward-chaining planning with JavaFF, in ‘Colloquium on AI Education, 23rd AAAI Conference on Artificial Intelligence’.
- Cox, J., Durfee, E. and Bartold, T. (2005), 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)’, ACM, pp. 821–827.
- de Weerd, M. and Clement, B. (2009), ‘Introduction to planning in multiagent systems’, *Multiagent and Grid Systems* **5**(4), 345–355.
- Decker, K. and Lesser, V. R. (1992), ‘Generalizing the Partial Global Planning algorithm’, *International Journal of Cooperative Information Systems (IJCIS)* **2**(2), 319–346.
- desJardins, M., Durfee, E., Ortiz, C. and Wolverton, M. (1999), ‘A survey of research in distributed continual planning’, *AI Magazine* **20**(4), 13–22.

- Doshi, P. (2007), On the role of interactive epistemology in multiagent planning, in ‘Artificial Intelligence and Pattern Recognition’, pp. 208–213.
- Dréo, J., Savéant, P., Schoenauer, M. and Vidal, V. (2011), Divide-and-evolve: the marriage of Descartes and Darwin, in ‘Proceedings of the 7th International Planning Competition (IPC)’, Freiburg, Germany.
- Durfee, E. H. (2001), Distributed problem solving and planning, in ‘Multi-agents Systems and Applications: Selected tutorial papers from the 9th ECCAI Advanced Course (ACAI) and AgentLink’s Third European Agent Systems Summer School (EASSS)’, Vol. LNAI 2086, Springer-Verlag, pp. 118–149.
- Durfee, E. H. and Lesser, V. (1991), ‘Partial Global Planning: A coordination framework for distributed hypothesis formation’, *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks* **21**(5), 1167–1183.
- Ephrati, E. and Rosenschein, J. S. (1996), ‘Deriving consensus in multiagent systems’, *Artificial Intelligence* **87**(1-2), 21–74.
- Fikes, R. and Nilsson, N. (1971), ‘STRIPS: A new approach to the application of theorem proving to problem solving’, *Artificial Intelligence* **2**(3), 189–208.
- Fogués, R., Alberola, J., Such, J., Espinosa, A. and Garcia-Fornes, A. (2010), Towards dynamic agent interaction support in open multiagent systems, in ‘Proceedings of the 2010 conference on Artificial Intelligence Research and Development: Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence’, IOS Press, pp. 89–98.
- Gerevini, A. and Long, D. (2006), Preferences and soft constraints in PDDL3, in ‘ICAPS Workshop on Planning with Preferences and Soft Constraints’, Vol. 6, Citeseer, pp. 46–53.
- Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D. and Wilkins, D. (1998), ‘PDDL - the Planning Domain Definition Language’, *AIPS-98 Planning Committee*.
- Gmytrasiewicz, P. and Doshi, P. (2005), ‘A framework for sequential planning in multi-agent settings’, *Journal of Artificial Intelligence Research* **24**, 49–79.
- Haslum, P. and Jonsson, P. (1999), Some results on the complexity of planning with incomplete information, in ‘Proceedings of the 5th European Conference on Planning (ECP)’, pp. 308–318.
- Helmert, M. (2006), ‘The Fast Downward planning system’, *Journal of Artificial Intelligence Research* **26**(1), 191–246.
- Hoffmann, J. and Nebel, B. (2001), ‘The FF planning system: Fast planning generation through heuristic search’, *Journal of Artificial Intelligence Research* **14**, 253–302.
- Jonsson, A. and Rovatsos, M. (2011), Scaling up multiagent planning: A best-response approach, in ‘Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)’, AAAI, pp. 114–121.
- Kambhampati, S. (1997), ‘Refinement planning as a unifying framework for plan synthesis’, *AI Magazine* **18**(2), 67–97.
- Kaminka, G. A., Pynadath, D. V. and Tambe, M. (2002), ‘Monitoring teams by overhearing: A multi-agent plan-recognition approach’, *Journal of Artificial Intelligence Research* **17**, 83–135.
- Kone, M., Shimazu, A. and Nakajima, T. (2000), ‘The state of the art in agent communication languages’, *Knowledge and Information Systems* **2**(3), 259–284.
- Kovacs, D. L. (2011), Complete BNF description of PDDL3.1, Technical report.
- Kraus, S. (1997), ‘Beliefs, time and incomplete information in multiple encounter negotiations among autonomous agents’, *Annals of Mathematics and Artificial Intelligence* **20**(1-4), 111–159.
- Kumar, A., Zilberstein, S. and Toussaint, M. (2011), Scalable multiagent planning using probabilistic inference, in ‘Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)’, Barcelona, Spain, pp. 2140–2146.
- Kvarnström, J. (2011), Planning for loosely coupled agents using partial order forward-chaining, in ‘Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)’, AAAI, pp. 138–145.
- Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., Prasad, M., Raja, A. et al. (2004), ‘Evolution of the GPGP/TAEMS domain-independent coordination framework’, *Autonomous Agents and Multi-Agent Systems* **9**(1), 87–143.
- Lipovetzky, N. and Geffner, H. (2011), Searching for plans with carefully designed probes, in

- ‘Proceedings of the 21th International Conference on Automated Planning and Scheduling (ICAPS)’.
- Micacchi, C. and Cohen, R. (2008), ‘A framework for simulating real-time multi-agent systems’, *Knowledge and information systems* **17**(2), 135–166.
- Nguyen, N. and Katarzyniak, R. (2009), ‘Actions and social interactions in multi-agent systems’, *Knowledge and Information Systems* **18**(2), 133–136.
- Nguyen, X. and Kambhampati, S. (2001), Reviving partial order planning, in ‘Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)’, Morgan Kaufmann Publishers, pp. 459–464.
- Nissim, R., Brafman, R. and Domshlak, C. (2010), A general, fully distributed multi-agent planning algorithm, in ‘Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)’, pp. 1323–1330.
- Pajares, S. and Onaindia, E. (2012), Defeasible argumentation for multi-agent planning in ambient intelligence applications, in ‘Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)’, pp. 509–516.
- Paolucci, M., Shehory, O., Sycara, K., Kalp, D. and Pannu, A. (2000), ‘A planning component for RETSINA agents’, *Intelligent Agents VI. Agent Theories Architectures, and Languages* pp. 147–161.
- Parsons, S., Sierra, C. and Jennings, N. (1998), ‘Agents that reason and negotiate by arguing’, *Journal of Logic and computation* **8**(3), 261.
- Penberthy, J. and Weld, D. (1992), UCPOP: A sound, complete, partial order planner for ADL, in ‘Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR)’, Morgan Kaufmann Publishers, pp. 103–114.
- Richter, S. and Westphal, M. (2010), ‘The LAMA planner: Guiding cost-based anytime planning with landmarks’, *Journal of Artificial Intelligence Research* **39**(1), 127–177.
- Sycara, K. and Pannu, A. (1998), The RETSINA multiagent system (video session): towards integrating planning, execution and information gathering, in ‘Proceedings of the 2nd International Conference on Autonomous Agents (Agents)’, ACM, pp. 350–351.
- Tambe, M. (1997), ‘Towards flexible teamwork’, *Journal of Artificial Intelligence Research* **7**, 83–124.
- Tang, Y., Norman, T. and Parsons, S. (2010), ‘A model for integrating dialogue and the execution of joint plans’, *Argumentation in Multi-Agent Systems* pp. 60–78.
- Tonino, H., Bos, A., de Weerd, M. and Witteveen, C. (2002), ‘Plan coordination by revision in collective agent based systems’, *Artificial Intelligence* **142**(2), 121–145.
- Van Der Krogt, R. and De Weerd, M. (2005), Plan repair as an extension of planning, in ‘Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)’, pp. 161–170.
- Weld, D. (1994), ‘An introduction to least commitment planning’, *AI magazine* **15**(4), 27.
- Weld, D. (1999), ‘Recent advances in AI planning’, *AI Magazine* **20**(2), 93–123.
- Wilkins, D. and Myers, K. (1998), A multiagent planning architecture, in ‘Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS)’, pp. 154–162.
- Wu, F., Zilberstein, S. and Chen, X. (2011), ‘Online planning for multi-agent systems with bounded communication’, *Artificial Intelligence* **175**(2), 487–511.
- Younes, H. and Simmons, R. (2003), ‘VHPOP: Versatile heuristic partial order planner’, *Journal of Artificial Intelligence Research* **20**, 405–430.
- Zhang, J., Nguyen, X. and Kowalczyk, R. (2007), Graph-based multi-agent replanning algorithm, in ‘Proceedings of the 6th Conference on Autonomous Agents and Multiagent Systems (AAMAS)’.

Author Biographies



Alejandro Torreño majored in Computer Science at Universitat Politècnica de València, Spain, in 2007 and received an M.Sc. degree in 2011. Since 2008 he does research work in the Grupo de Tecnología Informática - Inteligencia Artificial in the context of the Consolider-Ingenio Agreement Technologies project. He is also a Ph.D. student at the Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Spain. He has published several articles in scientific conferences and journals, such as ECAI, AAMAS and CoopIS. His research interests include Artificial Intelligence, Planning & Scheduling, Multi-Agent Systems and Computer Graphics.



Eva Onaindia is an Associate Professor of Computer Science at the Universitat Politècnica de València. She received her Ph.D. in Computer Science from the same University in 1997. She currently leads the Group of Reasoning on Planning and Scheduling where she conducts research in classical and temporal planning, and multi-agent planning. She is currently collaborating in the Agreement Technologies project where she is working on the application of new technologies and techniques, such as negotiation or argumentation, for the design and implementation of multi-agent planning systems. She has led several national research projects as well as sitting on various scientific committees in her field (IJCAI, ICAPS, AAI, ECAI, etc.). She has published about 120 articles in specialized conferences and scientific journals related to topics of Artificial Intelligence, Planning and Argumentation. Since 1994, she teaches undergraduate and master courses in Artificial Intelligence and Planning.



Óscar Sapena is an Associate Professor of Computer Science at the Politècnica de València. He obtained his Ph.D. in Computer Science in 2005 and he is a member of the Group of Reasoning on Planning and Scheduling. His main research interests are Planning and Execution and Distributed and Collaborative Planning. He has published about 40 articles in specialized conferences and scientific journals related to topics of Artificial Intelligence. He is currently collaborating in the Consolider-Ingenio Agreement Technologies project, applying new technologies, such as negotiation or argumentation, to the design of multi-agent planning systems.