Applied Intelligence (2020) 50:365–396
https://doi.org/10.1007/s10489-019-01429-0

Plan merging by reuse for multi-agent planning

Nerea Luis[1] · **Susana Fernández**[1] · **Daniel Borrajo**[1]

Published online: 24 July 2019

Abstract

Multi-Agent Planning deals with the task of generating a plan for/by a set of agents that jointly solve a planning problem. One of the biggest challenges is how to handle interactions arising from agents' actions. The first contribution of the paper is Plan Merging by Reuse, PMR, an algorithm that automatically adjusts its behaviour to the level of interaction. Given a multi-agent planning task, PMR assigns goals to specific agents. The chosen agents solve their individual planning tasks and the resulting plans are merged. Since merged plans are not always valid, PMR performs planning by reuse to generate a valid plan. The second contribution of the paper is RRPT-PLAN, a stochastic plan-reuse planner that combines plan reuse, standard search and sampling. We have performed extensive sets of experiments in order to analyze the performance of PMR in relation to state of the art multi-agent planning techniques.

Keywords Multi-agent planning · Plan reuse · Automated planning · Centralized planning · Distributed planning

1 Introduction

Consider a warehouse where everyday hundreds of orders are received and have to be delivered into the shortest period of time. First, the warehouse workers need to get the corresponding products to complete each order. In order to speed up the process, some robots are available to fetch the products for them. Workers will only focus on receiving the products and checking that the order has been completed. After that, orders need to be delivered to the clients' addresses using trucks. When reasoning about this scenario, some coordination systems are needed: (1) a system should divide up the work into the workers; (2) also robots should move autonomously and individually through the warehouse avoiding collisions; and (3) each driver needs to be assigned a set of deliveries based on the deliveries' location or some other minimization cost feature to improve efficiency. Automated Planning is the subarea of Artificial Intelligence that reasons on how to synthesize sequences of actions for solving tasks within this kind of scenarios. When multiple agents are involved (e.g robots, workers, drivers) or some coordination is needed, we talk about Multi-Agent Planning (MAP).

MAP aims at solving planning tasks for/by a set of agents. Usually, it is assumed that these agents collaborate to reach common goals. Two main approaches have been commonly used: centralized and distributed. The former builds a common plan for all agents by merely considering the agents as another planning resource. This was the usual way of dealing with MAP tasks in the Automated Planning community until recently. When MAP complexity was studied in [10], Brafman and Domshlak showed that MAP's complexity depends on the number of agents, the difficulty of their individual planning tasks and the number of interactions between agents (points where some coordination is needed). Thus, centralized planning is usually more efficient when computing a plan with a reduced number of agents and goals. On the other hand, in distributed planning, agents generate their plans either synchronously with the rest of agents as in FMAP [45] or MA-FS [40], or independently [36]. When planning synchronously, agents need to share information during planning. Thus, these approaches incur in a high communication cost. Also, they require to modify the code of an existing planner in order to accommodate the communication among agents. In the case of planning

✉ Nerea Luis
nluis@inf.uc3m.es

[1] Universidad Carlos III de Madrid, Avda Universidad 30 28911 Leganés, Madrid, Spain

independently, agents do not employ any communication among agents while planning. Therefore, they have to later merge their plans. In that case, some merging function is applied to the set of plans to generate a joint plan [17, 35]. Planning domains can vary from loosely-coupled, where there is almost no interference among the agents' plans to tightly-coupled, with higher interaction [10]. Plan merging has been shown to work best in loosely-coupled domains.

Regarding MAP in real world scenarios, in most domains the solution should be executed in the shortest possible time. Hence, concurrent execution of agents' actions is needed. One way to deal with the task of finding the plan that minimizes the number of concurrent execution steps (*makespan*) consists of planning explicitly taking into account that minimization criteria. Another alternative consists of generating a sequential plan and convert it into a parallel one. The parallel plan is partially-ordered, which means that a set of actions that do not have dependencies among them can be executed at the same time step. By doing so, we can use any standard total-order planner in the state-of-the-art to generate the set of sequential MAP plans, and then apply some parallelization algorithm to improve the *makespan*.

This paper focuses on classical deterministic planning tasks, where a set of agents should find a common plan. We describe a new MAP approach, whose objectives are to: efficiently solve MAP tasks by combining distributed and centralized MAP techniques; directly reuse existing planning techniques without further code modification; effectively apply factorization to divide the main task into subtasks regarding some minimization criteria such as *plan length* or *makespan*; and automatically adjust to the interaction level among agents and goals. In order to satisfy those objectives, we employ three off-the-shelf planners inside our first contribution: a MAP algorithm called Plan Merger by Reuse (PMR). Experimental results show that our approach is competitive with state-of-the-art techniques and planners, as the ones that participated in the first Competition of Distributed and Multi-agent Planners (CoDMAP).[1]

We also propose a novel use of planning by reuse as plan repair for plan merging. Planning by reuse has been widely employed in areas such as Case-Based Planning [7], or replanning when plan execution fails [19]. A planning-by-reuse planner works best when the invalid plan and the final plan are similar, as only a small set of actions need to be changed or added to the invalid plan for it to become valid. This situation is usually given on easy-to-solve interactions e.g grabbing the same resource or passing through the same door at the same time step. As a result, the planner will be able to efficiently generate a valid solution without generating an entire valid plan from scratch.

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

In order for PMR to perform better on both scenarios, we have also developed a new algorithm called RRPT-PLAN, which is our second contribution. RRPT-PLAN is a stochastic plan-reuse planner that combines search, sampling and plan reuse, performing one of the three stochastically on each iteration depending on the values of two parameters that represent the probability of executing each one of the three techniques. This contribution is inspired on two previous works, ERRT-PLAN [6] and RPT [1]. Experiments not only show how RRPT-PLAN can adapt itself to both scenarios but also how it can be included inside PMR as its plan-reuse planner, obtaining similar results as the other state-of-the-art plan-reuse planners and adapting better to a wide variety of scenarios.

A version of PMR was published in a workshop [33] in 2014, when the work was still into a very preliminary stage. We also participated in CoDMAP (June 2015) using that version. The main differences regarding the algorithm are that in the workshop paper:

- RRPT-PLAN had not been yet designed; instead, we were using LPG-ADAPT as plan-reuse planner.
- There was not a centralized phase in case the individual planning phase failed.
- The individual planning phase was using LAMA with Greedy-Best First Search (GBFS) and unitary costs (LAMA-UNIT-COST), instead of using GBFS with costs (LAMA-FIRST). Hence, the quality of individual plans was worse in general, as LAMA-UNIT-COST cannot minimize the cost metric.

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

– The input invalid plan sent to the plan-reuse planner (LPG-ADAPT) was the joined parallel plan instead of the joined sequential plan. The input plan is read sequentially in LPG-ADAPT and RRPT-PLAN. Thus, altering the actions' order before running a plan-reuse phase did not benefit the success of plan-reuse. Sending as input a parallel plan did not have any advantage.

The main contributions of this paper are:

– A new MAP system, PMR, whose main focus is on improving the work load among the agents, resulting in an improvement of the makespan.
– Development of a new plan-reuse planner, RRPT-PLAN, that interleaves search, sampling and plan-reuse to solve replanning tasks. It can be included as the plan-reuse planner of PMR.

We have also contributed by modelling three domains to test concrete features of PMR and RRPT-PLAN:

– Definition of a new domain, Hammers, as a proof-of-concept to exemplify and analyze the impact of different plan-reuse approaches to solve MAP tasks. This domain is explained in Section 5.2.
– A variation of the classical IPC Rovers domain called Rover-graph which changes the usual waypoints' topology. This domain is explained in Section 5.6.
– New version of the Depots-Robots [5], inspired in the Kiva-Amazon robots, which now contains the pods organized in vertical columns. This domain is explained in Section 5.6.

Finally, we contribute with an analysis on the performance of PMR and RRPT-PLAN. We have identified domains' features that characterize when these two techniques work well. This analysis has been performed from the results obtained of an extensive experimental evaluation:

– A CoDMAP rerun using the new PMR version jointly with RRPT-PLAN. We compared our contribution against the CoDMAP winners.
– Evaluation of RRPT-PLAN parameters to choose the best configuration. We created a benchmark of problems where the number of goals was increased to analyze the impact on RRPT-PLAN performance.
– Comparison of performance between RRPT-PLAN and LPG-ADAPT.
– Evaluation of PMR jointly with RRPT-PLAN in loosely-coupled and tightly-coupled domains.

This paper is organized as follows: Section 2 presents a formal definition of the MAP task and its preprocessing. Section 3 describes the PMR algorithm and its different phases and properties. Section 4 presents our second contribution, the RRPT-PLAN algorithm. Then, Section 5 shows the experiments and results of comparing both PMR and RRPT-PLAN with other similar approaches. Section 6 presents related work. And, finally, in Section 7 we present some conclusions and directions for future work.

2 Multi-agent planning formalization

In this section, we first formalize the MAP task and briefly mention some MAP languages that are being used to define those tasks. Then, we also describe the way we handle the *agentification* and factorization of the problem.

2.1 Multi-agent planning task

Here we first define the standard planning task, using a propositional description. We also describe each of its elements and its lifted representation, which is known as the domain and problem representation in the planning community. After that, we define the MAP task and its components. In Automated Planning, the planning task is defined as follows:

Definition 1 Planning Task (Single Agent). A single-agent STRIPS planning task [16] is a tuple $\Pi = \langle F, A, I, G \rangle$, where F is a set of propositions, A is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a set of goals.

Each action $a \in A$ is described by (1) a set of preconditions (pre(a)) that represent literals that must be true in a state to execute the action; (2) and a set of effects (eff(a)), which are literals that are expected to be added (add(a) effects) or removed (del(a) effects) from the state after the execution of the action. The definition of each action might also include a cost $c(a)$ (the default cost is one).

A state $s \subseteq F$ describes the current situation of the environment. In order to transit to a different state s', an action a must be applicable to s. The application of an action a in a state s is defined by the function $\gamma(s, a) = (s \backslash \text{del}(a)) \cup \text{add}(a)$ if $\text{pre}(a) \subseteq s$. Otherwise a cannot be applied in s.

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

$$\Gamma(s, \pi) = \begin{cases} \Gamma(\gamma(s, a_1), (a_2, \ldots a_n)) & if\ \pi \neq \emptyset \\ s & if\ \pi = \emptyset \end{cases} \quad (1)$$

Our planning task is encoded in the propositional fragment of the standard Planning Domain Description Language (PDDL) [18]. It is automatically generated from the PDDL description of a domain D and a problem P. The domain in PDDL contains a hierarchy of types to characterize the problem objects; a set of predicates and a set of functions, respectively, whose instantiations generate the facts in F; and a set of generalized actions, defined using variables—parameters, *par(a)*. The instantiations of those actions with problem objects generate the actions in A. A planning problem in PDDL contains a set of objects (instances of types in the domain); the initial state I; the set of goals G; and an optional metric to define the optimization criteria.

In order to work with multiple agents in Automated Planning, we have to define the Multi-Agent Planning Task.

Definition 2 Multi Agent Planning Task. We consider a multi-agent setting where a set of m agents, $\Phi = \{\phi_1, \ldots, \phi_m\}$, has to solve the task Π. We define the MAP task M as a set of planning subtasks, Π_i, one for each agent. Thus, $M = \{\Pi_1, \ldots, \Pi_m\}$ being $i = \{1...m\}$. For representation convenience, an alternative equivalent lifted representation of each single-agent planning task in PDDL would be a pair (domain, problem): $\Pi_i = \langle D_i, P_i \rangle$.

Since there does not exist yet a standard to represent MAP tasks, as PDDL for planning tasks, there have been several proposals. MAP usually considers that agents can have private information. Therefore, these proposals also include a way of describing agents' private information. MA-STRIPS [9] takes as input a standard PDDL description, as well as a list of agents, and automatically extracts the public and private components.

MA-PDDL [29], which was used in CoDMAP [43], is an extension of PDDL where agents and their private knowledge are explicitly defined in the input files.

Therefore, when working on MAP, it is common to find information that either belongs to the agents or belongs to the elements of the environment. When maintaining agents' privacy, the former is the one that should be hidden (total or partially) to other agents. We prefer to attach the property of privacy to the information available to agents (states, goals and objects) as in [5].

In that work, the information is parameterized by the user indicating the agent's type and by setting the types and predicates as private or public. We followed the same procedure in our approach but omitting privacy issues. In order to prevent confusions, we will refer to agent's information and public information. Thus, we have considered the agent's predicates as the elements that, when instantiated, will generate the propositions that belong to each agent. This process is applied during factorization to transform the original planning task Π into the MAP task M.

Once the MAP task has been defined and agents' and public information described, this is our definition of the individual planning task:

Definition 3 Individual agent task (Π_i). For each agent $\phi_i \in \Phi'$ a specific task Π_i is generated, which is described as a tuple $\Pi_i = \langle F_i, A_i, I_i, G_i \rangle$ where:

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

In Definition 3, the set of agent's propositions is identified as F_{ϕ_i} and the set of agent's actions as A_{ϕ_i}.

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

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

The set of public propositions is identified as F_{pub} and the set of public actions as A_{pub}. We assume that both the complete initial state, $I = \cup_{i=1}^{m} I_i$, and the set of goals, $G = \cup_{i=1}^{m} G_i$ are consistent; that is, they are conflict-free (there are no mutexes). In Automated Planning, a set of propositions $M \in F$ is mutually exclusive (mutex) if there is no state s that may be reached from the initial state by application of any sequence of actions in A, such that $M \subseteq s$.

MAP planners need some preprocessing to generate the set of planning tasks for each agent, Π_i. The process to obtain the MAP task from the planning task as well as the individual planning tasks is explained in the following subsection.

2.2 Agentification and factorization

In order to work on MAP environments, agentification and factorization are two methods to simplify the planning task. Agentification consists on identifying which are the agents of the given problem. This identification can be done in many different ways. If the planning task is defined using MA-STRIPS, the agents are given as input to the planner along with the domain and problem. Inside MA-STRIPS there is no information that explicitly indicates which type of agent can execute which action. Alternatively, if the planning task is defined using MA-PDDL, the agents are explicitly included on the domain and problem files. Each action has a keyword *:agent* in order to know which type of agents can execute the action.

Our approach works with both language definitions, either by explicitly receiving the list of potential agents or by getting the information through the *:agent* keyword of MA-PDDL to create the agents' list afterwards. It is important to choose as agents the elements of the domain that suit best the division of tasks. An advantage of dealing with MAP domains is that, in most of them, agentification comes naturally and the agents-to-be are immediately identified.

Factorization is the ability to divide a planning problem into subproblems using some criteria. For instance, it is common to apply factorization dividing the problem regarding goals [3, 15]. In our approach, the problem is first factorized from the agents' point of view. The aim is to transform the planning task into easier tasks so that each agent can solve its factorized problem individually. Definition 2 describes the factorization of the Planning Task in terms of agents, which gives as a result the Multi-Agent Planning Task.

Regarding goals, current MAP techniques in the Automated Planning community consider the MAP task M where goals are considered achievable by the collaboration of all agents [40, 45]. So goals are pursued by all agents and no further factorization is applied. In other areas, as in Multi-Agent Systems or robotics, some approaches first perform task allocation (assignment of each public goal to a single agent) to improve the efficiency of problem solving [13, 21]. Inspired by that, our approach has a second factorization step, which consists on dividing the goals among the agents by following some strategy. Specifically, some of the goal assignment (GA) strategies defined in a previous work [3] are included in PMR. In particular, we use:

- *All*, that assigns all goals to all agents;
- *Best-cost* (BC), where each goal $g \in G$ is assigned to the agent $\phi_i \in \Phi$ that could achieve it with the least cost; and
- *Load-balance* (LB) that first calculates $k = \left\lceil \frac{|G|}{|\Phi|} \right\rceil$, which will be the average number of goals per agent. Then it assigns each goal $g \in G$ to the best agent $\phi_i \in \Phi$ as in BC. This strategy avoids, if possible, assigning more goals than k to each agent.

As in [3], in order to assign a goal $g \in G$ to an agent $\phi_i \in \Phi$, a relaxed plan is computed using the FF heuristic [24]. Depending on the strategy selected, GA might leave some agents without any assigned goal. Hence, the output of goal allocation is a new MAP task, M', with goals assigned to a set of n agents $\Phi' \subseteq \Phi$ [5]. If the agent's cost of a goal is infinite (it cannot be reached), the goal is not assigned to that agent $\phi_i \in \Phi$ but if there is some goal that cannot be reached individually by any agent, the previous process assigns that goal to all agents. Then, $\Phi' = \Phi$ since all agents will have at least one goal assigned. Further details on the consequences of this situation will be given on Section 3.

After the second factorization step has been applied, each Π_i of the MAP task M' is formally described by Definition 3. Also, M' can be defined in terms of Π_i or using the lifted representation.

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

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

Definition 5 Lifted representation of the M' task. Considering the lifted representation of $\Pi_i = \langle D_i, P_i \rangle$ and being $n = |\Phi'|$, the M' task is defined as:

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

We now describe through a simple example how agentification and factorization work before generating and delivering to PMR the input M'.

Example (Logistics domain) Given a Logistics domain where trucks need to deliver packages to a set of locations, our example contains three trucks and four goals (package-delivery to its destination). Trucks initially located at CityA can traverse any kind of road (lined, dotted). Trucks from other cities can only traverse lined roads. Also, only trucks marked with "F" can deliver fragile "F" packages. The initial state of the problem is shown in Fig. 1. The destination of each package is described inside the square box of the figure. Costs of driving between each pair of cities are shown over the lined/dotted city connections. There is also a cost of 1 for loading and unloading a

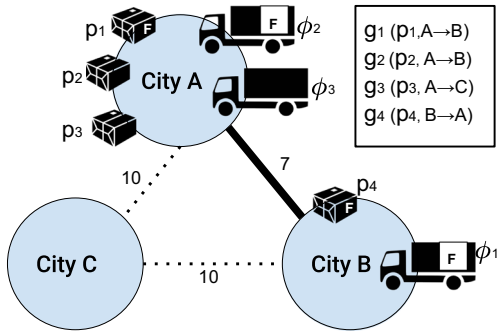

Fig. 1 Example of a simple Logistics problem where trucks (ϕ_1, ϕ_2, ϕ_3) have to deliver some packages to the destinations (g_1, g_2, g_3, g_4) specified on the square box. Trucks marked with "F" (ϕ_1, ϕ_2) are the only ones allowed to transport fragile packages (p_1, p_4). Trucks that start in CityA can traverse any kind of road (lined, dotted). Trucks from other cities can only traverse lined roads

package. Agentification comes naturally on this example, as trucks are the ones performing the actions. As a result, the first factorization generates the MAP task M, which contains an individual planning task for each truck. Table 1 shows an example of the estimated cost per truck and goal when delivering each package to its destination.

Taking the information presented on Table 1 and the three GA strategies mentioned before, goals would be assigned to each agent as follows:

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

The second factorization of the MAP task M varies depending on which GA is chosen. The GA strategy *All* assigns every goal to every agent even if the goal cannot be reached (represented with ∞ cost). There are three cases where ∞ cost is returned by h_{FF}: Cost(ϕ_1, g_3) because the truck ϕ_1 cannot traverse dotted roads as it does not start at CityA; Cost(ϕ_3, g_1) and Cost(ϕ_3, g_4) because the truck ϕ_3 cannot transport fragile packages (marked with "F"). The *Best-cost* (BC) strategy does not assign any goal to ϕ_3 even though it can reach two of the four goals. Instead, they are assigned to ϕ_2, who is the first on the list of agents that can reach (g_2, g_3) with an estimated cost of 2. Finally, *Load-balance* (LB) returns an average of k=2. Thus, it assigns two goals per agent. After assigning g_1 and g_2 to ϕ_2, ϕ_2's goal capacity has been reached. Then, g_3 is assigned to ϕ_3 even though ϕ_2 could achieve it with less cost.

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

3 Plan Merging by Reuse (PMR)

In this section, our first contribution, called PMR, is presented in detail. A general description as well as the description of each one of its components is included in the following subsections.

3.1 Algorithm

Plan Merging by Reuse (PMR) receives as input a MAP task (M'). As shown in the pseudocode in Algorithm 1, M' is formed by a number of Π_i tasks, each of them containing the information included in Definition 3. In the first step, each agent in $\phi_i \in \Phi'$ builds its plan individually (line 1). Then, we find three different scenarios:

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

Algorithm 1 High level description of the PMR algorithm.

Algorithm PMR(M', P, R)

Inputs: M': MAP task
P: planner
R: plan-reuse planner
Output: π_{PMR}: plan

```
1  Forall φ_i ∈ Φ' do π_i =plan(M'_i, P)
2  If ∀φ'_i ∈ Φ' π_i=fail
3  Then M'_joined = join-task(M')
4       π_cen =plan-centralized(M'_joined, P)
5       π_PMR =parallelize(π_cen)
6  Else π_seq =merge(π_1, ..., π_n)   /* where n = |Φ'| */
7       If valid(π_seq)
8       Then π_PMR =parallelize(π_seq)
9       Else M'_joined = join-task(M')
10           π_reuse =plan-reuse(M'_joined, π_seq, R)
11           π_PMR =parallelize(π_reuse)
12  If valid(π_PMR) Then return π_PMR
13  Else return no-solution
```

Given that we are dealing with MAP tasks, it is expected that agents can execute the actions in their plans in parallel

Table 1 Example of an estimated-cost matrix from the problem pictured in Fig. 1

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

Each number represents the cost returned by h_{FF} for an agent ϕ_i when reaching goal g_m. Cost(ϕ_i, g_m)=∞ means goal g_m is not reachable by agent ϕ_i

when possible. Thus, the aim of PMR is to minimize the makespan. In non-temporal domains, we refer to the makespan as the length of the parallel plan (number of execution steps, where several actions can be executed at each execution step). Therefore, in any of the three scenarios, if the plan is valid, it is parallelized to improve the makespan of the solution as explained later on Section 3.5.

PMR contains three off-the-shelf planners: one for each agent to plan individually, P (it can be the same or a different planner); another one capable of applying plan repair by reusing an invalid/incomplete plan (R); and the third one, which is employed by PMR when all the individual agents' planning tasks fail. In this work we use the same planner, P, used by individual agents to run the centralized phase. In order to check the validity of the plans, VAL, the validator from the International Planning Competition, has been used [25]. The following sections explain in detail the main steps and the properties of PMR.

3.2 Planning

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

When the distributed phase fails If all agents fail to generate a solution, it could mean that more than one agent might be needed to achieve the goals. In those cases, PMR resorts to a centralized planner. Centralized planners usually receive as input the lifted representation of the planning task (one domain and one problem file). Thus, the elements of the MAP task M' should be first joined as follows:

Definition 6 The M'_{joined} task.

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

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

$$= \{\bigcup_{i=1}^{n} D_i, \bigcup_{i=1}^{n} P_i\}$$

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

In Algorithm 1, we have used the same planner for the centralized phase as in the individual calls. However, since PMR is planner-independent, we could have used any other planner.

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

Example (Logistics domain) Following the example explained at the end of Section 2, we want to illustrate what would be the result after PMR's individual planning phase. The goal strategy chosen is still LB. The assignment of agents and goals was $GA = \phi_1(g_4), \phi_2(g_1, g_2), \phi_3(g_3)$. Agents are ordered by name. ϕ_1 is the first one to start planning.

The resulting individual plans from the agents are the following.

$\pi_1 = (load\ \phi_1\ g_4\ B)(drive\ \phi_1\ B\ A)(unload\ \phi_1\ g_4\ A)$

$\pi_2 = (load\ \phi_2\ g_1\ A)(load\ \phi_2\ g_2\ A)(drive\ \phi_2\ A\ B)(unload\ \phi_2\ g_1\ B)(unload\ \phi_2\ g_2\ B)$

$\pi_3 = (load\ \phi_3\ g_3\ A)(drive\ \phi_3\ A\ C)(unload\ \phi_3\ g_3\ C)$

As a result, after concatenation,

$\pi_{seq} = \{\pi_1 \oplus \pi_2 \oplus \pi_3\} = (load\ \phi_1\ g_4\ B)(drive\ \phi_1\ B\ A)$
$(unload\ \phi_1\ g_4\ A)$
$(load\ \phi_2\ g_1\ A)(load\ \phi_2\ g_2\ A)(drive\ \phi_2\ A\ B)(unload\ \phi_2\ g1\ B)(unload\ \phi_2\ g_2\ B)$
$(load\ \phi_3\ g_3\ A)(drive\ \phi_3\ A\ C)(unload\ \phi_3\ g_3\ C).$

3.3 Plan Reuse

We assume that often the current invalid plan does include most of the actions that would make it a valid plan. Thus, by using plan repair techniques we are expecting PMR to generate a plan faster than planning from scratch. In the worst case, plan repair is PSPACE-complete [38]. But, in practical terms and under our assumption of closeness between the invalid plan and the valid one, plan repair techniques have shown good results [19].

Usually, planners that perform plan repair receive three inputs: a domain, a problem and a plan. Examples of plan-repair planners are LPG-ADAPT [19] and our contribution RRPT-PLAN that will be explained on Section 4. We use such a planner to transform an invalid input plan (π_{seq}) into a valid plan. In case the plan-repair planner solves the planning task and the plan is valid (π_{reuse}), PMR parallelizes it to improve the makespan and returns it.

Example (Logistics domain) For instance, following the example explained at the end of Section 2, two scenarios could trigger PMR's plan reuse phase. The first scenario is given when solving the problem using the goal strategy *All*. This implies that each agent has to plan individually to deliver the four packages marked as goals. After concatenation (π_{seq}), ϕ_2 will not find any of the packages at the original locations, as ϕ_1 has already moved them. The same happens to ϕ_3. Thus, π_{seq} is invalid and needs to be fixed. PMR will call the plan-reuse phase. The second scenario is given when two agents need the same resource. For instance, suppose that trucks need drivers in order to transport the packages. If there was only one driver per city, trucks would need that driver to move from one city to another. Each agent would need to pick up a driver, and independently of the goal strategy selected, two trucks could pick the same driver during the individual planning phase. When concatenating the plans, ϕ_2 could have picked up the same driver as ϕ_3 (as they are located at the beginning at the same city). Thus, a failure would arise when ϕ_3 needs the driver as she is not at CityA.

3.4 Centralized planning

Centralized planning within PMR is only used when the individual planning phase fails. For instance, a situation where PMR would fail using the same example presented at the end of Section 2 is the following: if the *All* GA strategy was selected instead of LB or BC, as agents have to achieve every goal from the problem individually, ϕ_1 and ϕ_3 would not be able to make it because they cannot achieve all goals (see Table 1). ϕ_1 has an estimated cost of ∞ for *g3* as well as ϕ_3 for *g1* and *g4*. Additionally, if ϕ_2 was not able to transport fragile packages, *g1* and *g4* could not be delivered either. As a result, π_{seq} would not be generated and the centralized planner would be called instead.

3.5 Parallelization

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

It performs two steps: converting the input total-order plan into a partial-order one by a similar algorithm to [47]; and parallelizing this partial-order plan by ordering actions in the first time step that satisfies all ordering constraints in the partial-order plan. The cost of the parallelization is quadratic in the number of actions in the plan.

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

Example (Logistics domain) Following the example, the plan π_{seq} described in Section 3.2 is validated by VAL. As this was a simple example with no interaction among agents, the plan is valid and the parallelization phase starts by receiving π_{seq} as input. The resulting plan, π_{PMR}, will be:

$$
\begin{aligned}
&\pi_{PMR} = \\
&1 : (load\ \phi_1\ g_4\ B)(load\ \phi_2\ g_1\ A)(load\ \phi_3\ g_3\ A) \\
&2 : (drive\ \phi_1\ B\ A)(load\ \phi_2\ g_2\ A)(drive\ \phi_3\ A\ C) \\
&3 : (drive\ \phi_2\ A\ B)(unload\ \phi_1\ g_4\ A)(unload\ \phi_3\ g_3\ C) \\
&4 : (unload\ \phi_2\ g_1\ B)(unload\ \phi_2\ g_2\ B)
\end{aligned}
$$

3.6 Properties

As most work on plan merging, PMR performs suboptimal MAP. Even if we used optimal planners at each planning step of the algorithm, the simple merging of individually generated plans cannot assure optimality. Another reason for being suboptimal is the GA step, but it can be solved by using the *All* strategy. This GA strategy is the only one that ensures that all agents are included in the process

of planning. Thus, an optimal solution could be found. In relation to soundness, each individual (agent) plan is valid, when using sound planners. If the centralized planner is invoked, soundness is ensured by using a sound planner. However, after merging, soundness cannot be ensured because each agent plans separately for a subset of goals and the merged plan might not be valid due to the agents' interactions within the elements of the domain. Thus, PMR checks for validity, and if the plan is invalid, the soundness depends on the plan repair step. In summary, PMR is sound if the planners used are sound (both P and R).

In relation to completeness, PMR is complete if all goals are assigned to all agents and the off-the-shelf planners (planners and plan repair planner) used are complete too. Under those conditions, the first planning step of each agent (individual planning) is globally incomplete (since there might be a joint plan that is not contained in the space of each agent working separately), but it is locally complete for each agent. Once the plans are generated, PMR calls either a centralized planner or the plan repair strategy, which are both complete if the chosen planners for P and R are as well. The parallelization step does not affect the completeness. Finally, PMR has the same complexity as Automated Planning and MAP, which is PSPACE [9, 12].

4 RRPT-plan

As it was briefly mentioned in Section 1, among the spectrum of different scenarios that can be given in trying to repair a plan, the most frequent one is when the invalid input plan is very similar to the final solution. Plan reuse will be very efficient as most of the final plan's actions are already on the invalid input plan. Thus, the planner reuses most of the invalid plan actions and has only to include a small set of new actions to transform it into a valid one. However, sometimes, the solution plan should be completely changed, as when trying to solve the Rovers scenario described on Fig. 2. In order to be more efficient, combining search and plan-reuse would help any plan-reuse planner to better solve a wide variety of scenarios: from the ones that are very similar to the ones that look similar but they are not.

We have developed a stochastic plan-reuse planner, called RRPT-PLAN, that combines search, sampling and plan reuse. RRPT-PLAN means Reuse & Randomly exploring Planning Tree and it is our second contribution on this paper. We were inspired by two previous works that are explained as follows.

The first previous work is ERRT-PLAN [6], which already explored the behavior of a plan-reuse planner and proposed a solution to adapt the algorithm to a broader set of plan reuse scenarios. It builds a solution tree inspired on the Rapidly-exploring Random Trees (RRTs) [32]. ERRT-PLAN receives as input the domain and problem description, the probabilities for node expansion and the solution (plan) represented as an ordered set of pairs of actions and their weakest preconditions. Given a plan π, the weakest preconditions of any action $a_i \in \pi$ represent the set of propositions that are required to be true before applying a_i in the current state s_i so that the goals can be achieved from a_i when applying the remaining actions of the plan; the weakest preconditions act as subgoals of π. ERRT-PLAN has a probability of p to expand the tree towards the goal, a probability of r to expand towards an action of the input plan and a third probability to expand towards one of the weakest preconditions. ERRT-PLAN employs a reimplementation of METRIC-FF [23] as the heuristic planner and EHC [24] as the search algorithm.

The second work is RPT [1]. In order to implement RRPT-PLAN, the RPT algorithm was taken as a basis for developing our contribution. RPT was already able to combine two different types of search: one towards the nearest goal and another one towards a sampled state, which is any mutex-free state of the search space. RPT builds the solution tree inspired on the RRT's structure.

Our contribution, RRPT-PLAN, emulates the ERRT-PLAN behavior by receiving as input the invalid plan, the domain and problem description and the set of probabilities; the weakest preconditions that ERRT-PLAN stored are not considered on our implementation. RRPT-PLAN combines

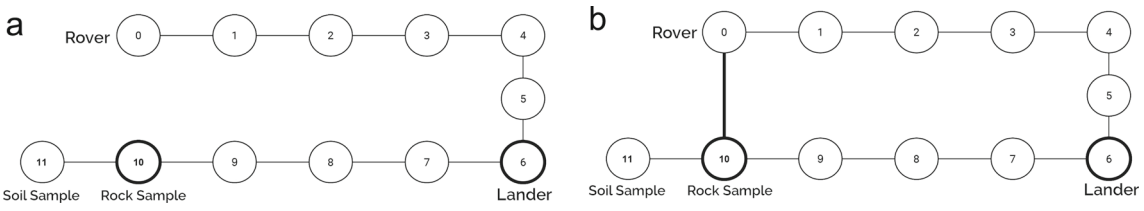

Fig. 2 Example of a simple Rovers problem. **a** depicts the initial state of a Rovers problem where the Rover agent has to take the soil and rock samples to later send them to the lander. **b** depicts the same problem as on the left, but now the Rover agent can directly traverse 0-10 to reach the samples. As both problems are very similar, any plan-reuse planner would usually choose the solution obtained from problem (**a**) to solve problem (**b**). Unfortunately, that would not be efficient, as the plan-reuse planner would ignore the existence of the new path 0-10. Thus, the plan-reuse planner will return as a solution the same one obtained on problem (**a**)

search, plan reuse and sampling using the state-of-the-art planner FAST DOWNWARD [22], which was the base planner used by RPT. Details on RRPT-PLAN and the main differences regarding RPT and ERRT-PLAN are explained later on Section 4.6.

As a result, RRPT-PLAN has three parameters:

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

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

Algorithm 2 Description of the RRPT-PLAN planning algorithm.

Algorithm RRPT-PLAN

Parameters: ϵ, p, r
Inputs: domain, problem, input-plan
Output: Plan (*solution*)

```
1  sas_operators ← translate(input-plan)
2  tree ← q_init
3  first_iteration ← true
4  while not goalReached() do
5    n ← random() /*between 0 and 1*/
//Search towards the goal
6    if (not(first_iteration) ∧ (n < p)) then
7      tree ←search( q_goal, tree, ε)
//Plan reuse
8    else if ((first_iteration) ∨ (n ≥ p ∧ n <(p+r)))
     then
9      first_iteration ← false
10     q_reuse ← unreutilized_nodes.pop()
11     tree ←reuse(q_reuse, sas_operators, tree)
//Sampling
12   else
13     q_rand ← sampleSpace(S)
14     tree ←search( q_rand, tree, ε)
15     tree ←search( q_goal, tree, ε)
16 end while
//Tracing back the solution
17 solution ← traceBack(q_goal)
18 return solution
```

The following subsections explain in detail first the current configuration of RRPT-PLAN and then each step of the algorithm. Section 4.5 describes the planning properties of RRPT-PLAN. Finally, Section 4.6 explains the differences between RRPT-PLAN, ERRT-PLAN and RPT.

4.1 Configuration

RRPT-PLAN follows the same configuration as RPT. Thus, the local planner FAST DOWNWARD was configured with greedy best-first search choosing lazy evaluation as its local search algorithm. The heuristic used is the h_{FF} heuristic [24].

As both RPT and RRPT-PLAN are inspired on RRTs, a tree structure is built during the planning process. RPT originally had local search and sampling phases. We have added the plan reuse phase. Details on the implementation of the tree are explained in [1].

As it was mentioned before, RRPT-PLAN has three parameters (p, r, ϵ). Parameters p and r set the probability of performing search, reuse or sampling on each iteration; except for the first iteration, in which plan reuse will be performed. Parameter ϵ sets the maximum number of expanded nodes per iteration.

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

- Every node of the tree (we refer to them as q in Algorithm 2), contains: a state (s_i), a pointer to the previous node (parent ρ_i), the sequence of actions that reaches s_i from the parent (subplan τ_i), the number of the last reused action of the input plan (it is zero when none of the input plan's actions has been reused yet) and the cached best supporters for every proposition $q \in F$.
- We refer to cached best supporters as the implementation previously included in RPT [1] where the actions that first achieve a given proposition in the reachability analysis are stored in order to compute h_{FF} efficiently.

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

4.2 Preprocessing

The first step of the preprocessing translates the PDDL domain and problem to the SAS+ language [2] by calling and using FAST DOWNWARD's Translate module [22]. This module also returns a list of operators, which contains every valid combination of actions and parameters that can be generated from the given domain and problem. Our algorithm RRPT-PLAN uses that list to translate the input plan to the SAS+ language. Thus, for each action of the

plan, the algorithm looks for the equivalent SAS+ operator on that list (line 1, Algorithm 2). As a result, the input plan is transformed into a sequence of SAS+ operators instead of instantiated PDDL actions. For simplicity, along the following sections we will use the word *action* when referring to these SAS+ operators.

4.3 Search-reuse-sampling

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

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

Function `search` of RRPT-PLAN

Inputs: q, *tree*, ϵ

```
1  q_near ← findNearest(tree, q)
2  q_new ← join(q_near, q, ϵ)
3  tree ← addNode(tree, q_near, q_new)
4  return tree
```

4.3.1 Search

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

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

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

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

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

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

Function `reuse` of RRPT-PLAN

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

```
1  i ← last_action_reused(q_reuse)
2  state ← getCurrentState(q_reuse)
3  q_reuse' ← createNewNode(state)
4  reuseSuccess ← false
5  applicable ← true
6  while (i <sizeof(sas_operators) ∧ applicable) do
7    current← sas_operators.get(i)
8    if isApplicable(current, state) then
9        reuseSuccess ← true
10       state' ← updateState(state, current,
         q_reuse')
11       q_reuse' ← insert(state', current, i, q_reuse')
12       if (goalReached()) then
13         tree ← addNode(tree,q_reuse, q_reuse')
14         return tree
15       i ← i + 1
16       state ← (q_reuse').state
17   else applicable ← false
18 if reuseSuccess then
19   tree ← addNode(tree,q_reuse, q_reuse')
20 return tree
```

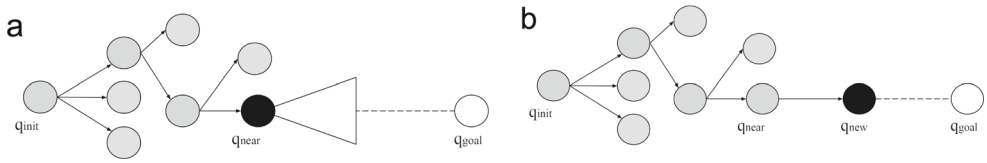

Fig. 3 The first step of RRPT-PLAN search towards the goal is shown on the left. Local search is run from q_{near}, which is the closest expanded node found to the goal so far. The second step is shown on the right. After expanding ϵ number of nodes, q_{new} is the closest node to q_{goal}. The node q_{new} stores the plan to reach q_{new}'s state from q_{near}. Finally, q_{new} is stored into the tree

4.3.2 Reuse

When $p \leq n < (p + r)$, RRPT-PLAN runs the plan reuse scenario (lines 8-11, Algorithm 2). During the first iteration, the algorithm always performs plan reuse regardless of the values of p and r. This decision is further justified on Section 5. Algorithm 4 describes the steps to run plan reuse. Figure 4 shows the process inside the tree. In order to differentiate each node on Fig. 4 and Algorithm 4, a brief description of them is given below.

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

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

- Plan reuse can be applied to a node as long as the number of the last reused action does not reach the last action of the input plan.
- There is an open list to store and provide the nodes to whom plan reuse is applied. The algorithm always extracts the first node on the list. When local search or sampling add new nodes to the tree, they are also automatically added to the plan-reuse open list.

The algorithm takes the first node of the plan reuse open list (q_{reuse}, line 1 Algorithm 4) and gets the position of the last action reused (stored on the node; by default 0 when none of the actions has been yet reused). Then, it iterates over the sequence of actions of the input invalid plan. For each one of them it checks if the action can be applied to the current state of the node (Fig. 4a). If this is true, the action ($current$) is added into the plan and the current $state$ and index i are updated (lines 10-11, Algorithm 4). In addition, when an action is added to the new plan, the algorithm checks if the goal state has been reached as well (line 12, Algorithm 4). The reuse process will be repeated until an action cannot be applied (Fig. 4b). In that case, the position of the last reused action, the current state and the sequence of the reused actions are stored into the node $q_{reuse'}$. Also, the previous node q_{reuse} is set as parent.

4.3.3 Sampling

When $(p + r) \leq n < 1$, RRPT-PLAN runs the sampling scenario. Lines 13-15 of Algorithm 2 describe the steps to run the sampling process. Figure 5 shows the process inside the tree. In order to differentiate each node on Fig. 5, a brief description of them is given below.

- q_{init}: first node on the tree. It contains the initial state of the problem.
- q_{rand}: node that contains a random valid state from the search space.
- $q_{sampling}$: closest node from the tree to the sampled state.
- q_{new}: last expanded node towards q_{rand}.
- $q_{neargoal}$: last expanded node towards the goal.

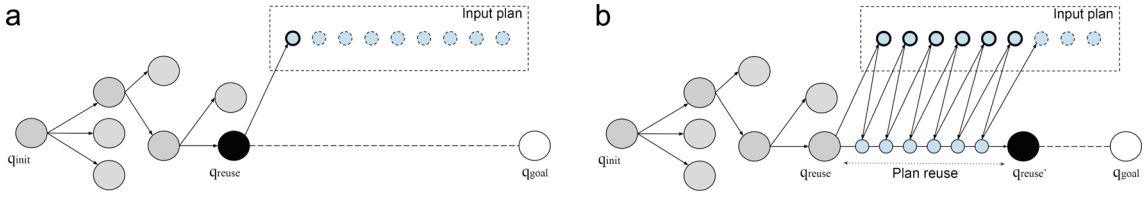

Fig. 4 The first step of plan reuse in RRPT-PLAN is shown on Fig. 4a. The state of q_{reuse} is evaluated to see if the first action from the input plan can be directly applied. The second step of plan reuse in RRPT-PLAN is shown on Fig. 4b. As long as there are actions that can be applied to the current state, they will be stored inside a new node $q_{reuse^{prime}}$ which at the end of the process, when no more actions can be reused, will be the child of q_{reuse}

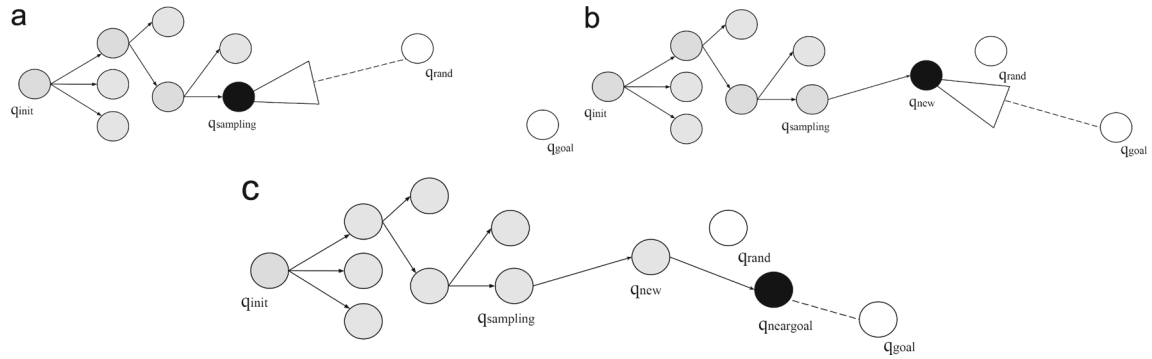

Fig. 5 The first step of sampling in RRPT-PLAN is shown on the left. Once the sampled node q_{rand} is obtained, a local search is run from $q_{sampling}$ towards that node. $q_{sampling}$ represents the closest node from the tree to the sampled node. The second step of sampling in RRPT-PLAN is shown on the right. After local search is performed, q_{new} is added into the tree. Now a local search towards the goal (q_{goal}) is run. The third step of sampling in RRPT-PLAN is shown below the previous steps. After expanding ϵ number of nodes, $q_{neargoal}$ is the closest node to q_{goal} so it is stored into the tree

– q_{goal}: node that contains the goal state of the problem. When it is reached, it means that a plan has been found.

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

– A new node is added to the tree at every iteration. If the solution is not reached during sampling, the last expanded node is added to the tree.
– After sampling, when the new node is obtained (q_{new}), a new local search is performed towards the goal until the limit ϵ is reached. This is equivalent to the *Extend* phase of RRTs.

First, the algorithm obtains a random valid state (q_{rand}) after sampling the search space ($\mathcal{S}$). Then, the closest node to the sampled state is found ($q_{sampling}$, Fig. 5a) by computing h_{FF}. Details about how the random state and the distance are computed can be found in [1]. After $q_{sampling}$ is identified, a local search is performed from there towards q_{rand}. However, q_{rand} might not be reached because of the limit ϵ. If this happens, the last expanded node is stored on the tree (q_{new}). The next step is to perform a new local search from q_{new} towards the goal (q_{goal}, Fig. 5b). This is the *Extend* phase, already implemented in RPT[1]. The last expanded node from the *Extend* local search is then stored into the tree ($q_{neargoal}$, Fig. 5c).

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

4.4 Tracing back the solution

In order to retrieve the solution plan π, the algorithm has to check if q_{goal}'s state s_{goal} satisfies every proposition in G, so that $G \subseteq s_{goal}$. If so, RRPT-PLAN starts tracing back the solution plan from the node q_{goal}. From q_{goal}, RRPT-PLAN obtains the link (ρ_{goal}) to the parent's node and repeats the process until the initial node (q_{init}) has been reached. Figure 6 illustrates the process. Solution nodes are stored into a list called $Tree_{sol} = \{q_{init}, ..., q_{goal}\}$ where q_{init}'s state $s_{init} = I$ and q_{goal}'s state $G \subseteq s_{goal}$. Nodes' subplans are concatenated to obtain $\pi = \{\tau_{init} \oplus \tau_2 \oplus ... \oplus \tau_{goal}\}$. Each τ_i contains at least one action of the solution plan. The concatenation gives as a result the solution plan $\pi = \{a_1, a_2, ..., a_m\}$.

4.5 Properties

RRPT-PLAN is an algorithm that performs suboptimal planning. Optimality is out of the scope of this work. Also, RRPT-PLAN is incomplete, as it cannot assure that a solution will always be found. For instance, extreme cases where $p = 1.0$ (only search) or $r = 1.0$ (only plan-reuse) or both have 0.0 values (only sampling); these are some configurations where RRPT-PLAN could fail to find a solution - specially if there exist some timeout. Regarding only search ($p = 1.0$ $r = 0.0$), RRPT-PLAN can get stuck after exploring and expanding every existing node of the

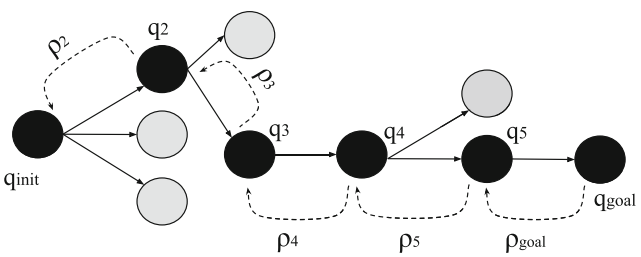

Fig. 6 Tracing back the solution in RRPT-PLAN. The algorithm starts on q_{goal} and goes backwards on the *Tree*, obtaining on each step a new solution node through the link ρ_i. Black nodes of the Figure represent the solution nodes. They are stored in order on a list called $Tree_{sol}$

search space and still not being able to obtain a solution. Regarding only plan reuse ($p = 0.0\ r = 1.0$), if no action of the input plan can be reused or every action has been reused but it is not enough to solve the problem, RRPT-PLAN will constantly enter into the reuse phase and will never obtain a solution to the problem. Regarding only sampling ($p = 0.0\ r = 0.0$), it is the same case as only applying search. Sampling explores the search space randomly, but after exploring and expanding every node of the search space, the solution might still not be found.

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

$$\begin{aligned}\pi &= \{\tau_{init} \oplus \tau_2 \oplus ... \oplus \tau_{goal}\} \\ &= \{\overbrace{a_1, a_2, ..., a_k}^{\tau_2}, \overbrace{a_{k+1}, ...a_p}^{\tau_3}, ..., \overbrace{a_{r+1}, ..., a_m}^{\tau_{goal}}\}\end{aligned} \tag{2}$$

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

$$S = \left\langle \underbrace{s_{init}}_{q_{init}}, \overbrace{(s_{init}^1, s_{init}^2 ... s_{init}^{k-1})}^{\tau_2}, \underbrace{s_2}_{q_2}, \overbrace{(s_2^1, s_2^2 ... s_2^{p-1})}^{\tau_3}, s_3, ..., \overbrace{(s_r^1, s_r^2 ... s_r^{m-1})}^{\tau_{goal}}, \underbrace{s_{goal}}_{q_{goal}} \right\rangle \tag{3}$$

Thus, when the subplan τ_2 is applied to s_{init} following (1), the resulting state is s_2.

$$\Gamma(s_{init}, \tau_2) \models s_2 \tag{4}$$

Theorem 1 RRPT-PLAN *is sound*

Proof By induction: Base case: $s_{init} = I$ is reachable from the initial state by q_{init} construction.

Inductive step: *if* s_i is reachable from s_{init}, s_{i+1} is a reachable state from s_{init}.

By construction:

$$\Gamma(s_{init}, \tau_2 \oplus \tau_3 ... \oplus \tau_i) \models s_i \tag{5}$$

$$\Gamma(\underbrace{s_{init}, \tau_2 \oplus \tau_3 ... \oplus \tau_i}_{s_i} \oplus \tau_{i+1}) \models s_{i+1} \tag{6}$$

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

$$\Gamma(s_{init}, \underbrace{\tau_2 \oplus ... \oplus \tau_{goal}}_{\pi}) \models s_{goal}, where\ G \subseteq s_{goal} \tag{7}$$

□

4.6 Differences of RRPT-PLAN regarding previous works

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

- First, RRPT-PLAN presents a more clear bias towards search than ERRT-PLAN. While ERRT-PLAN considered the search step as adding one more node to the tree, RRPT-PLAN search algorithm works by expanding ϵ nodes in the same step, where we have found that ϵ should take big values.
- Second, ERRT-PLAN sampling of goals was directed by the computation of weakest preconditions from the input plan, while RRPT-PLAN sampling uses RPT sampling procedure instead. However, RRPT-PLAN does not use RPT's computation of *h2* mutexes for that task. By avoiding that computation, the aim is to speed up the process.
- Third, ERRT-PLAN and RPT considered the goal sampling step as equally relevant. On the contrary, RRPT-PLAN assigns a very small role to goal sampling by assigning a very low probability of using sampling.

All these differences are due to the diverse uses of ERRT-PLAN, RPT and RRPT-PLAN. In the case of ERRT-PLAN, we were studying the effects of different strategies of plan reuse (replanning from scratch vs. eager use of the previous plan) in a wide variety of scenarios. In RRPT-PLAN we are interested in a very particular kind of plan reuse scenario, where in most cases, the reuse strategy is a mixture between the two extremes. Finally, the

obvious difference with respect to RPT is that RRPT-PLAN can partially reuse a previous plan and that the sampling phase is not considered as important as the other two.

5 Experiments

This section presents some experiments on the performance of our two contributions, PMR and RRPT-PLAN, and their comparison with other state-of-the-art planners. We have divided the experiments and results in six different sections structured as follows.

First, Section 5.1 describes metrics and configuration environments. Section 5.2 presents some experiments specifically designed to explain the flexible behavior of RRPT-PLAN in different plan-reuse scenarios. Section 5.3 analyzes RRPT-PLAN in terms of parameters to select the best configuration. Section 5.4 shows the results of running the CoDMAP competition with different configurations of PMR and other state-of-the-art, multi-agent and centralized, planners. Also, Section 5.5 shows the performance of PMR when scaling the number of agents. Section 5.6 shows the performance of PMR against the same set of planners previously used on Section 5.4. However, the set of problems is harder to solve, containing a reasonable amount of agents and goals to reach. We have also included three new domains specifically designed to evaluate the makespan metric in order to show PMR's potential. Finally, Section 5.7 describes some general remarks and conclusions extracted from the extensive set of experiments.

5.1 Experimental setup

For each set of experiments described in the following sections, results of coverage (number of solved problems), quality (cost and makespan) and planning time are shown. In order to compute these metrics, we have used the scores of the International Planning Competition (IPC)[2]. Coverage is computed increasing by one each time a planner solves a given problem of a given domain. Makespan refers to the number of execution steps, where several actions can be executed at each execution step. Cost refers to the sum of costs of all the actions contained in the plan. For computing both metrics, makespan and cost, we use Q_{best}/Q, where Q_{best} is the cheapest value obtained for a specific problem by the set of planners and Q is the value obtained for the same problem by one of those planners. In order to compute the time score, we use $1/(1+log10(T/T_{best})$, where T_{best} is the lowest time in which a specific problem has been solved by any planner and T is the time in which one of those planners has solved the specific problem. The time bound to solve each problem is 1800s. The quality/makespan/time score of a planner is the sum of its scores for all problems. For every domain presented on the tables, 20 problems were run, except for the ones of Section 5.3 where 15 problems per domain were run instead (the reason is explained on that Section). All the experiments were run on an Intel(R) Xeon(R) X3470 2.93GHz with 8 GB RAM.

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

- Every configuration of PMR is using LAMA-FIRST as the planner P of the algorithm. LAMA-FIRST corresponds to the first search that LAMA performs, using greedy-best-first with unit costs for actions [41]. We have also used LAMA-FIRST for the centralized planning step.
- We have used LPG-ADAPT [19] and RRPT-PLAN for the plan reuse experiments. When they have been used inside PMR we refer to them as PMR-LPG-ADAPT or PMR-RRPT-PLAN. Otherwise it means they were executed outside PMR. LPG-ADAPT has always been run in the *speed* mode. Additionally, we have compared RRPT-PLAN against ERRT-PLAN [6] and RPT [1], as our contribution is an evolution of those approaches combined. RPT has been configured with p=0.3 and ERRT-PLAN with p=0.3 and r=0.6. The reason is explained on Section 5.3.
- RRPT-PLAN has three additional parameters set on each configuration. We refer to them as p, r and ϵ. The way these parameters affect RRPT-PLAN is explained on Section 4.
- Our configurations of PMR have been evaluated using different goal assignment strategies. We refer to them as BC (*Best-cost*), LB (*Load-balance*) and ALL i.e. PMR-LPG-ADAPT-BC means that PMR was executed using *Best-cost* as goal assignment and LPG-ADAPT as plan reuse planner.
- We have tested our PMR configurations against two centralized planners, LAMA-FIRST and YAHSP [48].
- Also, we have compared our configurations with three multi-agent planners (CMAP-T [4], ADP-L [14] and SIW [37]). They will be later explained in Section 5.4. They were chosen because of their remarkable performance on some metrics of CoDMAP.

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

The aim of this section is to analyze in detail the behavior of RRPT-PLAN. A simple multi-agent scenario was designed to force PMR to use its plan reuse planner to fix the plan (either RRPT-PLAN or LPG-ADAPT on these experiments). The Load-Balance (LB) strategy was chosen for this experiment

[2] ipc.icaps-conference.org

in order to assign goals using as many agents as possible and balancing the number of assigned goals per agent at the same time.

We are calling this domain Hammers, for which we have generated a set of problems. The aim was to show how interactions are handled when a set of agents has to share a set of resources. Thus, the designed domain contains robots (agents), hammers, nails and paintings. The robot needs first to find and grab a hammer and a nail. It is not allowed for robots to grab more than one hammer and nail at the same time. Once the robot has grabbed the pair hammer-nail it should go to some room that contains a painting and hang it up. Each scenario is completely solved when all paintings are hanged up.

The ideal multi-agent situation (reflected on Fig. 7a) has as many robots, hammers and nails as paintings. Thus, each robot will be in charge on hanging one painting up when goals are divided among the set of agents. However, the bottleneck of the problem is the number of hammers; if the robots need to share the hammer(s), many interactions arise and need to be fixed by a plan reuse planner. As agents (robots) plan individually on PMR's first step, all of them are forced to use a hammer on their individual plans.

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

The first scenario presents six robots starting in a common room, C, plus a hammer and a nail per room (Fig. 7a). When computing this ideal scenario for MAP with either PMR-RRPT-PLAN or PMR-LPG-ADAPT using the LB strategy, the planner will assign to each robot one of the goals. Thus, each robot will move to a different room, pick up the hammer and the nail and hang the painting on the wall. As a result, after the individual planning step, the concatenation and parallelization of the resulting plan will return a valid plan. There is no need for replanning in this first case as there is no interaction among agents.

The second scenario is the same as the previous one but the difference lies in the number of hammers: it has just one hammer in C (Fig. 7b). This reflects a common MAP issue on how agents deal with a shared resource (hammer) and how the planner deals with agents' interactions.

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

We can see in this example that calling a pure plan-reuse planner to fix a plan might not always be a good choice. It will always either reuse as many actions as possible from the input invalid plan, or, when this is not possible, it generates

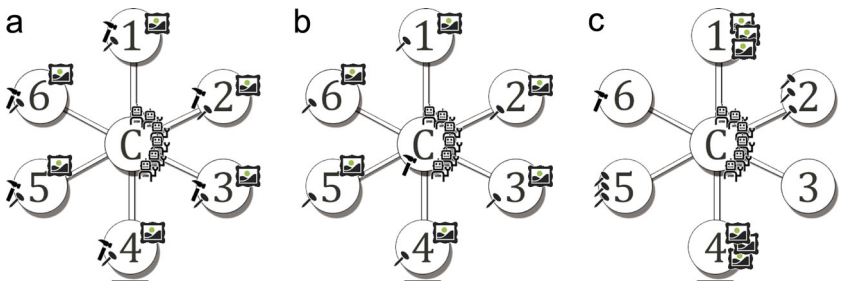

Fig. 7 **a** presents the first scenario (6 robots, 6 nails, 6 hammers, 6 paintings). This problem is an ideal multi-agent planning scenario. There are as many hammers as robots. Thus, they do not need to share any resource during planning. In the second scenario (**b**) there is only one hammer for six robots. Paintings and nails are placed as in case (**a**). Interactions arise when merging the robots' plans as they all use the hammer in their individual plans. The third scenario (**c**) not only shows hammer interaction issues, but also interactions caused by nails. Nails are grouped into two rooms. Thus, multiple robots could have used the same nail in their individual plans

new ones to be later reused. This issue can generate noise on the plan's actions, later explained on the third scenario. If the aim is to obtain an efficient valid plan fast, plan reuse needs to be combined with search to solve these specific multi-agent scenarios. Also, when a shared resource is limiting the number of agents that can perform actions, the best solution will probably be to involve a number of agents close to the number of available shared resources.

The third case presents six robots starting in C, six nails and paintings equally divided among two rooms and one hammer at Room 6 (Fig. 7c). After obtaining the concatenated plan and checking that it is not valid, several parts of the plan need to be fixed. Not only the actions related with grabbing and dropping the hammer must be fixed, but also the ones related with looking for nails (Table 2).

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

On the other hand, when using the PMR-RRPT-PLAN approach, the set of actions performed by robot1 will be directly reused until it finishes hanging the first painting. When moving robot2 to room6 and realizing that the hammer is not there, the plan-reuse phase ends and the search phase starts, discarding the rest of the actions from the input plan. As a result, again only robot1 is used to hang all the paintings up and redundant actions are avoided except for the one action from robot2 moving to room6. Even though only one robot is used to execute the whole plan, the plan length is 43. That value is still lower than the one from PMR-LPG-ADAPT, which returns a plan of length 149. Regarding makespan, PMR-LPG-ADAPT returns 118 and PMR-RRPT-PLAN, 38.

In Table 2, the first three rows are related to the three cases explained above. The following rows represent new problem-variations generated from the Fig. 7c scenario where paintings and nails are equally divided into two sets of different rooms and hammers' location vary. The aim is

Table 2 Besides the three problems previously generated to explain three different cases of plan-reuse, a set of four problems were generated based on Fig. 7c's distribution to show the impact of increasing the number of paintings and hammers while having the same number of rooms and agents

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

The main difference between a and b versions is where the hammers are placed: (a) hammers are equally divided in the set of rooms 1, 4, 6; (b) all hammers are placed in room 6. Paintings are equally divided between rooms 1 and 4 as in Fig. 7c. Nails are also grouped into two rooms (rooms 2 and 5)

Table 3 This table shows the plan length (L) and makespan (M) obtained for every scenario of the Hammer domain with multi-agent planners PMR-RRPT, PMR-LPG-AD, PMR-LAMA, CMAP-T, ADP-L, SIW and centralized planners LAMA, YAHSP and RPT

	PMR-RRPT		PMR-LPG-AD		PMR-LAMA		CMAP-T		ADP-L		SIW		LAMA		RPT	
	L	M	L	M	L	M	L	M	L	M	L	M	L	M	L	M
Fig. 7a	24	3	24	3	24	3	24	3	24	23	24	3	24	3	24	3
Fig. 7b	24	24	34	34	24	24	24	24	24	24	24	24	24	24	24	24
Fig. 7c	38	38	159	133	43	38	43	38	38	38	34	32	43	38	43	38
1a	76	46	122	119	85	74	–	–	74	74	96	49	–	–	85	76
1b	89	84	121	85	–	–	–	–	74	74	96	49	–	–	79	32
2a	148	82	235	235	157	146	157	146	146	146	134	91	157	146	157	146
2b	161	150	–	–	–	–	–	–	146	146	95	68	–	–	157	146
3a	221	118	380	371	229	218	229	218	218	218	184	45	229	218	229	218
3b	233	223	–	–	–	–	–	–	218	218	180	74	–	–	241	111
4a	294	154	497	491	301	290	301	290	–	–	237	30	301	290	303	292
4b	299	163	–	–	–	–	–	–	290	290	226	51	–	–	280	131

Results of YAHSP were not included as it was not able to solve any problem. Makespan values of ADP-L and SIW were obtained after paralellizing their resulting plans with our parallelization algorithm

to show the evolution of plan length, makespan and time when increasing number of paintings and hammers. Due to this fact, the complexity of the problem increases depending on where the hammers are initially placed and the number of interactions that have arisen.

Tables 3 and 4 show the plan length, makespan and time obtained for each PMR configuration, other multi-agent and centralized planners on each Hammer problem. As Table 3 shows, PMR-LAMA and PMR-LPG-ADAPT are not able to solve most of the problems where all hammers are placed on the same room (version *b* problems). PMR-LAMA gets lost on the search space as there is a huge number of possible combinations of movements with the same heuristic estimation (known as plateaux) to solve the problem. On the other hand, PMR-LPG-ADAPT gets lost when reusing the actions of the invalid plan, as most of them are valid, but they still do not solve the problem. This ends up generating redundancy and causes the planner to search for new actions, while always looking up at the ones on the invalid plan again on each iteration. However, PMR-RRPT-PLAN performs better than the other two configurations, as it has the opportunity to change between plan reuse, search and sampling to solve the problem and this has an impact on the number of problems solved as well as on the makespan metric. PMR-RRPT-PLAN is more flexible and in summary obtains the best performance on the three metrics regarding the three PMR configurations. Regarding multi-agent planners CMAP-T, ADP-L and SIW, the best

Table 4 Time in seconds that each configuration of PMR, CMAP-T, ADP-L, SIW, LAMA and RPT took to solve the Hammer domain scenarios

	PMR-RRPT	PMR-LPG-AD	PMR-LAMA	CMAP-T	ADP-L	SIW	LAMA	RPT
Fig. 7a	1.80	2.25	1.79	1.00	1.00	1.00	1.00	1.00
Fig. 7b	1.00	1.66	1.64	1.00	1.00	1.00	1.00	1.00
Fig. 7c	1.08	1.63	1.12	1.00	1.00	1.00	1.00	1.00
Prob 1a	43.84	4.40	40.05	2.83	1.24	2.48	2.07	2.67
Prob 1b	4.01	39.20	–	–	1.23	4.54	–	3.91
Prob 2a	107.15	16.81	115.97	13.16	6.47	14.18	10.55	16.71
Prob 2b	14.37	–	–	–	6.37	14.27	–	22.11
Prob 3a	159.07	43.89	140.42	48.30	22.43	43.00	39.02	91.75
Prob 3b	59.81	–	–	–	30.69	65.66	–	110.62
Prob 4a	173.96	90.48	182.78	136.98	–	94.81	110.20	329.29
Prob 4b	126.81	–	–	–	77.31	103.73	–	329.18

YAHSP results are not shown as it was not able to solve any problem. Time limit per problem was 1800s. When (-) appears on a cell means that the problem was not solved by the planner on time. Problems solved in less than 1 second are all considered as 1.00s

configuration is SIW. SIW's serialization of goals allows the planner to use efficiently most of the agents resulting in the improvement of makespan at the same time. ADP-L and CMAP-T behave very similar to LAMA, the centralized planner. These planners (except for SIW) try to solve the problem with the smallest possible number of agents, as a consequence of minimizing the plan length. ADP-L is the only one able to avoid the plateaux of the search space and does not get lost on version *b* problems. The centralized planner YAHSP was run as well, but it was not able to solve any of the problems.

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

Regarding times on Table 4, PMR-LPG-ADAPT is faster when it is capable of solving the problems but PMR-RRPT-PLAN is more regular on performance.

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

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

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

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

As it was mentioned in Section 4, RRPT-PLAN has three parameters that change the behaviour of the algorithm. Parameters p and r control the probability of running local search, plan reuse or sampling, e.g. values $p = 0.6, r = 0.3$ cause RRPT-PLAN to have a probability of 0.6 to run the local search phase, 0.3 to run plan-reuse and 0.1 to run the sampling one. In addition, the parameter ϵ limits the number of expanded nodes per iteration during local search, e.g a value of $\epsilon = 1000$ means that 1000 nodes will be expanded at most per local search iteration.

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

1. $p = 0.3$, $r = 0.3$; this configuration gives equal probability to search and reuse and 0.4 to sampling.
2. $p = 0.3, r = 0.6$; this configuration benefits plan-reuse over search and leaves a probability of 0.1 to sampling.
3. $p = 0.3$, $r = 0.7$; same to the previous one but RRPT-PLAN will not perform the sampling phase.
4. $p = 0.6$, $r = 0.3$; this configuration benefits local search over plan-reuse and leaves a probability of 0.1 to sampling.

Each of these parameter configurations was tested for $\epsilon = 1000$ and $\epsilon = 10000$ to analyze the impact on the solution when allowing a smaller or greater number of nodes to be expanded during search. For this experiment, the execution of plan reuse on the first iteration of RRPT-PLAN is avoided in order to focus on the impact of these probabilities. We also show the comparison against LAMA-FIRST, LPG-ADAPT, ERRT-PLAN and RPT. Results of ERRT-PLAN are not shown on the tables below, as it was not able to solve any

of the problems in 1800 seconds. ERRT-PLAN uses Enforced Hill Climbing (EHC) as the search method [24]. Since EHC performs a form of hill climbing without backtracking, the heuristic can guide the search towards dead-end states, big plateaux, or very long paths. Therefore, ERRT-PLAN could not solve any problem.

Results of this experiment are shown on the following tables. Table 5 shows the number of problems solved per configuration. Table 6 shows the summary of the obtained quality score per configuration. Finally Table 7 shows the summary of the time scores per configuration.

Table 5 shows that coverage is very similar between RRPT-PLAN and LPG-ADAPT. LAMA-FIRST and RPT perform a bit worse, especially on high-interaction domains (e.g. depots, logistics, driverlog). As they do not reuse a previous plan, they had to run each modified problem from scratch and those domains are harder to solve. RRPT-PLAN ($p = 0.6, r = 0.3$) is the best configuration followed by RRPT-PLAN ($p = 0.3, r = 0.6$).

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

On the other hand, regarding time, the fastest configuration is LPG-ADAPT. If we considered only planning time, RRPT-PLAN results would be similar to those of LPG-ADAPT. Before starting to plan, our planner, first translates the domain and the problem and computes mutexes and disambiguation (which is useful for search and sampling). Thus, our planner's performance is worse than LPG-ADAPT in time (not in quality). LPG-ADAPT's preprocessing phase only translates the domain and problem. LAMA-FIRST applies the same translation process as RRPT-PLAN but it does not compute mutexes and disambiguation. RPT performs an exhaustive computation of mutexes and disambiguation, which is why its time results are the worst ones. As it can be seen on Table 8, the fastest preprocessing is performed by LPG-ADAPT. Also, the impact on computing mutexes and disambiguation can be seen by comparing RRPT-PLAN, RPT and LAMA-FIRST preprocessing results. On the other hand, even though it is not explicitly reflected on Tables 7 and 8, the harder the problem, the closer the time score gets RRPT-PLAN to the one of LPG-ADAPT. The reason is the following: when the number of added goals to the problem increases, plan reuse performance decreases, as the input plan is not similar anymore. On the other hand, search becomes more useful. In Table 7 is shown the poor performance of a centralized planner (LAMA-FIRST, RPT) in comparison with plan-reuse ones. When the problems are similar, reusing input plans is faster than planning from scratch. Table 8 shows the average time employed on preprocessing by each planner on each domain.

Table 5 Coverage score obtained on each domain

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

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

Table 6 Quality score obtained on each domain

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

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

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

5.4 Results in CoDMAP problems

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

Tables 9 and 10 show the obtained results in coverage (number of problems solved). We have used the three different goal allocation strategies: *Best-cost* (BC), *Load-balance* (LB) and *All*. Since PMR can solve each problem in three different ways, we show the coverage obtained on each one separately: merge (M, when plans are valid after merging), centralized planning (C, when no agent could generate any plan) or plan reuse (R, when the merged plan was invalid). The configuration PMR ALL was added in order to make PMR complete. It shows a similar behavior as a centralized algorithm. Thus, when PMR ALL fails in the M and R steps, it usually solves the problems in the C step. On the other hand, PMR LB and BC solve more problems in the M and R steps than PMR ALL. The aim of PMR is to solve as many problems as it can executing the M and R phases.

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

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

Low interaction domains Zenotravel, Rovers and Satellites. PMR often solves problems in these domains by merging the individual agents' plans, because these plans are mostly independent. However, it depends on the type of goal allocation strategy selected. Take, for instance, the

Table 7 Time score obtained on each domain

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

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

Table 8 Average time (in seconds) spent on preprocessing each problem per planner and domain

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

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

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

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

A key related issue is how domains are modeled. For instance, if the only agents considered in the Logistics domain are the airplanes, then BC and LB would solve all problems just by either merging the resulting individual plans or by plan repair. Similarly, in the case of Elevator and

Table 9 PMR with LPG-ADAPT and RRPT-PLAN configuration in combination with up to three goal assignments: BC (*Best-cost*); LB (*Load-balance*) and *All*

	PMR-LPG-ADAPT									PMR-RRPT-PLAN $p = 0.3\ r = 0.6\ \epsilon = 1000$								
	BC			LB			ALL			BC			LB			ALL		
	M	R	C	M	R	C	M	R	C	M	R	C	M	R	C	M	R	C
Driverl.	19	1	0	8	12	0	0	20	0	19	1	0	8	12	0	0	20	0
Zenotra.	20	0	0	20	0	0	0	20	0	20	0	0	20	0	0	0	20	0
Elevators	1	0	19	1	0	19	0	0	20	1	0	19	1	0	19	0	0	20
Logistics	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20
Rovers	20	0	0	19	0	0	9	3	8	20	0	0	19	0	0	9	3	8
Satellites	20	0	0	20	0	0	16	0	4	20	0	0	20	0	0	16	0	4
Sokoban	1	3	7	0	4	7	0	0	17	1	3	7	0	2	7	0	0	17
Taxi	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20
Blocks	20	0	0	0	20	0	0	20	0	20	0	0	0	20	0	0	20	0
Wireless	0	0	2	0	0	2	0	0	5	0	0	2	0	0	2	0	0	5
Depots	0	0	17	0	0	17	0	0	17	0	0	17	0	0	17	0	0	17
Woodw.	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20	0	0	20
Partial	101	4	105	68	36	105	25	63	131	101	4	105	68	34	105	25	63	131
Total	210			209			**219**			210			207			**219**		

In PMR, *M*: merging; *R*: plan-reuse; *C*: centralized. Partial is the total score in coverage of each step in PMR. Each domain has 20 problems. It is the same set of problems used on CoDMAP

Bold represent the best configuration

Table 10 IPC scores in Coverage per configuration in CoDMAP domains

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

Bold represent the best configuration

Taxi, one could define fast elevators or taxis respectively as the only agents and PMR would solve all problems without the centralized planner.

We have compared our contributions against the winner of CoDMAP in coverage, ADP-L [14]. Note that ADP-L does not preserve privacy and it uses a different agents' configuration than the one proposed in the competition. We also compare against CMAP-T [4] that obtained the best coverage and time score from the planners that preserved privacy. Finally, we also included SIW [37] as it was one of the best planners of the competition. Results of this comparison are shown on Table 10.

Regarding Table 10, ADP-L obtains the best coverage followed up by SIW and PMR-ALL (both with LPG-ADAPT and RRPT-PLAN). YAHSP coverage is the worst. Related to coverage in the official CoDMAP, after the summer-run[3] PMR-ALL would share the first position with ADP-L, which was 219.

Regarding time scores (Table 13 in the Appendix), the fastest planner is ADP-L followed very closely CMAP-T. The performance on time of PMR configurations is homogeneous. The time score in PMR was computed using the total amount of time spent by the whole process. Since the individual agents' planning processes were executed in sequence, the total time is computed as the sum of all these processes. However, considering that the merging phase could be implemented fully distributed, the total time would be less by using the maximum planning time among all agents instead of the sum.

In relation to quality scores, Tables 14 and 15 (in the Appendix) show the results of makespan and cost of plans, respectively. As LAMA and ADP-L did not compute the makespan metric, our parallelization algorithm was applied to their resulting plans to fairly compare the results of makespan. SIW makespan results are not shown in the table because our parallelization algorithm could not support the use of constants that they include in the PDDL domain and problem to respect the privacy of objects and fluents after they transform the MA-PDDL files into PDDL [37]. This issue does not happen when the domain and problem are directly given in PDDL.

Regarding PMR configurations, it can be seen how BC is better in cost and LB in makespan. The difference between these numbers lies in the number of agents involved in the planning process. BC often includes the minimum necessary number of agents to plan (the ones that expectedly achieve goals with the minimum cost), so the plans' cost will usually be low (good score). In the extreme, some problems were solved using only one agent. However, the makespan is penalized given that the same agent is achieving all goals, so many actions cannot be executed in parallel. On the other hand, LB tries to include as many agents as possible, as long as they can solve at least one of the goals of the problem. The makespan will be better than that of BC, because actions can be easily parallelized. But, the cost is penalized when choosing LB, as it uses agents whose plans

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

are worse in terms of cost. Potentially more interactions need to be solved (causing an increase in the number of plans solved by R instead of M).

PMR-RRPT-PLAN configurations perform better on quality and PMR-LPG-ADAPT configurations, better on time. Moreover, PMR-LPG-ADAPT-ALL, which was one of the best configurations in coverage, is the worst of our configurations in quality scores. The *All* strategy uses all agents in the problem to plan leading to long merged plans. PMR-RRPT-PLAN-ALL performs better in quality than PMR-LPG-ADAPT-ALL because it obtains better results mainly in Driverlog and Blocks thanks to the combination of plan-reuse and search. Although ADP-L obtains a better value in makespan than all PMR-LB configurations, it is mainly due to the difference in solved problems in Wireless and Sokoban. It is also remarkable the result obtained by LAMA-FIRST (best configuration in cost and makespan) because it is just a centralized planner. This clearly shows that the CoDMAP problems are not hard to solve even for classical planners, as there is no huge impact during the planning process even when agents try to solve all goals at once because goals are not assigned specifically.

5.5 Results scaling the number of agents

In order to analyze how PMR configurations scale with the number of agents, we generated one medium-sized problem in the Zenotravel domain. It has 63 goals. Then, we increased the number of agents. For each instance of the Zenotravel problem, agents were increased by 10 starting on 10 agents and stopping at 70. The configuration used for this experiment was PMR-RRPT-PLAN. Zenotravel belongs to the low interaction group identified in Section 5.4. The domains that belong to these groups are the ones where generally PMR performs better, as most problems are solved during merging or plan-reuse phases, avoiding the centralized planning step. Thus, we were interested in exploring in detail the evolution of makespan and plan length when increasing the number of agents. Results are shown in Fig. 8. X axis represents the maximum number of agents that can plan per problem. In Zenotravel, BC uses 10, 16, 18, 21, 21, 22 and 23 agents per problem while LB uses 10, 20, 30, 34, 37, 59 and 62. As Zenotravel belongs to the low interaction level group, the bigger the number of agents is, the better the makespan obtained by LB. The performance decreases drastically using *ALL* (only 3 problems out of 7 were solved in 1800s), because all goals are assigned to all agents in the problem, which makes the process of solving the interactions harder during plan-reuse.

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

5.6 Multi-agent hard problems

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

In this section we show some experiments on eight domains whose problems are harder to solve than the ones used on previous experiments. Tables 11 and 12 show the IPC scores on coverage and makespan respectively. Table 12 also contains the summary of time scores. For these experiments we have used the same planners and configurations shown on the CoDMAP experimentation (Section 5.4). In order to select the domains on this experimentation, we decided to include some domains from the group of low interaction (Zenotravel, Satellite, Rover), and some from the group of medium interaction (Driverlog, Blocks). We excluded the ones that had higher interaction (Elevator, Logistics) as they were only solved on the centralized phase, so the work-load distribution of PMR would not affect the result.

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

The Rover-graph domain is an evolution of the usual Rover domain where two huge grids of waypoints are generated independently and then joined by an edge between those grids. This simulates environments where there is a bottleneck in the connection between two areas. The aim was to test how the planners' performance changes when a different configuration of the environment penalizes the use of a single agent to solve the whole problem. The number of goals in the set of problems oscillates between 150-200. The minimum number of predicates per problem is 5100. Goals are the same as in the classical Rover domain (e.g. communicate soil, image, rock data etc.). The number of agents varies from 2 to 6. The two grids contain around 400 waypoints in total. On this variation of the usual Rover domain, planners (centralized-based ones specially) get lost because of the size of the search space. Instead of applying factorization to alleviate the agents'

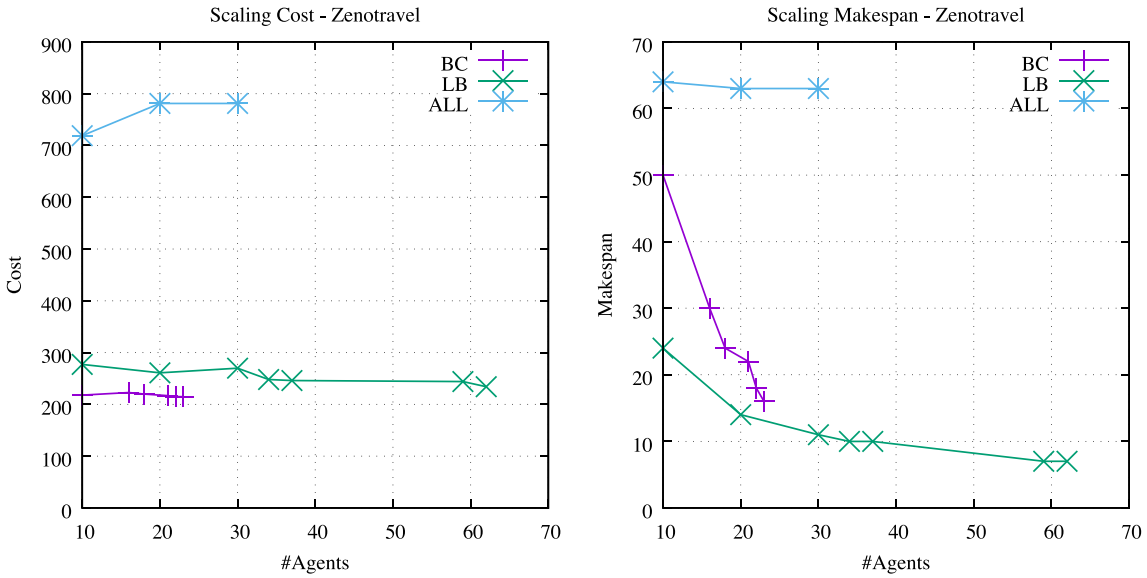

Fig. 8 Evolution of Cost and Makespan in a Zenotravel problem when increasing the number of agents gradually (x axis). The configuration used was PMR-RRPT-PLAN with *Best-cost* (BC), *Load-balance* (LB) and *All*

individual planning process, those planners employ the smallest possible number of agents to solve the problem. As a result, an agent has to deal with almost all the goals, which makes its planning task harder to solve.

Depots-Robots is the planning version of a warehouse environment inspired on the Kiva Robots[49] where robots have to deliver to humans a list of products from the storage pods to complete a list of delivery orders. This new version works over a grid of waypoints where pod storages follow a classical structure of a warehouse by being stored in columns. There is an empty column between each pair of pod-occupied columns for robots to move. The first and last rows of the grid are empty. The last row is where humans are situated for the reception of products. The original version, where restrictions on the placements of pods and empty rows are not applied, is described in [5]. Robots are spread through the grid. Thus, PMR can indirectly assign a specific zone of the grid to a robot to pick up the nearby packages. As a result, planners that factorize the problem for each robot regarding human goals and pods' locations will obtain better results.

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

Table 11 Coverage results in hard and specific problems

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

Bold represent the best configuration

Table 12 Makespan score in hard and specific problems

	PMR-RRPT-PLAN						
	BC	LB	CMAP-T	ADP-L	SIW	LAMA-FIRST	YAHSP
Zenotravel	10.09	20.00	14.03	8.51	0.53	15.36	0.00
Satellite	6.93	20.00	10.78	3.98	0.97	11.76	0.00
Rover	2.52	20.00	4.25	2.49	0.00	3.88	0.00
Driverlog	0.00	2.84	4.85	5.94	0.00	6.85	0.00
Blocks	4.59	2.36	6.57	6.57	7.10	6.04	14.00
Rover-graph	16.66	19.44	13.81	11.40	9.66	6.90	0.00
VRP	6.28	19.00	1.83	4.45	0.15	3.49	0.00
Depots-robots	9.48	11.11	10.00	6.37	9.55	9.27	0.00
Total Mkspn	56.57	**114.75**	66.12	49.72	27.97	63.55	14.00
Total time	97.89	79.68	88.53	**113.27**	31.92	97,47	7.57

Bold represent the best configuration

Coverage results on Table 11 show how a planner like SIW that had promising results on CoDMAP now cannot solve more than half of the problems due to their complexity. The rest of the planners, including our configurations, except for YAHSP have a similar coverage.

Table 12 shows that our LB configuration outperforms the rest of the planners regarding makespan. These planners were again multi-agent (CMAP-T, ADP-L, SIW) and centralized (LAMA, YAHSP) planners. Hence, even though our configurations might be slower, they are still capable of solving harder problems by involving multiple agents. Dividing the number of goals as much as possible among the agents has a direct impact on makespan. In VRP, Rover-graph and Depots-robots domains, PMR-RRPT-PLAN-LB increases the makespan score more than the other planners; problems can be easily divided, creating balanced subtasks for each agent. In VRP, PMR-RRPT-PLAN outperforms the rest of planners, as it is a good example of domain, where multi-agent planning can improve through factorization. This can also be seen on Rover-graph with CMAP-T and LAMA. The main difference between them is factorization. CMAP-T is able to solve every problem while LAMA cannot scale enough to solve a task of 200 goals. Domains such as Zenotravel, Satellite and Rover, that have a low number of interactions, are also good for our LB configuration.

Table 16 in the Appendix shows the time score, where ADP-L and LAMA are the fastest ones. PMR is slower, because of the goal assignment phase, as our algorithm spends some time on identifying which agent solves best each goal. The fastest configurations usually assign all goals to all agents by default. This strategy works well when optimizing for time or coverage. However, we claim that the makespan score should be the main performance criteria if the goal is to generate plans in a real multi-agent environment, where a big number of agents is available to work.

5.7 Discussion on the experiments' results

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

- The performance of PMR and its adaptability cannot be appreciated on easy MAP tasks. For instance, results on CoDMAP (Section 5.4) reflect that the centralized planner LAMA was able to solve more problems than PMR's BC, LB and CMAP-T; even its coverage results are very close to those of ADP-L and SIW. This indicates that MAP planners do not usually have a remarkable advantage over centralized approaches on easy MAP tasks.
- Regarding interactions among agents, PMR works best on low and medium interaction domains. Those are generally identified as loosely-coupled domains by the planning community. Some of those domains are: Rovers, Satellite, Zenotravel, Depots, Hammer and VRP.
- PMR biases towards optimizing the makespan metric, independently of the MAP task received as input. Thus, the more the task can be factorized and work equally distributed among agents, the better the makespan obtained will be. Usually, those tasks are the loosely-coupled ones.
- PMR scales well on hard loosely-coupled planning tasks, and also in those that contain a topology that can be factored for the agents to work independently. This can be appreciated on the results from Rovers-graph or VRP.
- The goal assignment strategy Load-Balance works best for loosely-coupled domains, as it balances the amount of goals among the agents. In turn, this has a direct

impact on improving the makespan. Best-cost biases towards improving the plan length and works best for tightly-coupled domains. As less agents are used during planning, it will reduce the number of conflicts to solve.
- RRPT-PLAN behaves very similar to LPG-ADAPT, which is important regarding that LPG-ADAPT is considered the state-of-the-art replanner.

In addition, RRPT-PLAN covers the full range of reuse scenarios from the best case for reuse (when the input incorrect plan is very similar to the final correct one) and the worst case (when the input and final plans are very different).
- The plan-reuse phase of RRPT-PLAN is more useful when it is selected first. If we assume that the input plan is similar to the final plan, running plan-reuse first will automatically include most of the input plan actions into the final plan, boosting the performance of RRPT-PLAN as a result. However, in case the input plan is completely different from the final plan, we expect that the plan-reuse will fail fast and RRPT-PLAN will switch to the search phase. This issue was tested on the experiments with increasing number of goals in Section 5.3.

Also, some limitations have been identified:

- The main issue that faces PMR with tightly coupled domains is the number of interactions to solve. If the planning task presents many interactions, the plan-reuse phase could be potentially solved better by planning from scratch. Examples of such planning tasks are the ones defined in the IPC for the Driverlog or Sokoban. However, one must take into account that the distribution of problems generated by the IPC organizers for each domain, albeit randomly generated, usually focus on a specific subarea of the set of potential problems that can be generated. Therefore, it is easy to see that even in these two tightly-coupled domains, one can generate problems with lower interaction. For instance, one could have several rooms in Sokoban, each with its own set of robots, or one could have different network subgraphs in Driverlog, each one with its own set of drivers and vehicles. So, the property of being tightly-coupled is connected to the planning task (domain and problem) and not only to the domain.
- There exists a bottleneck on the goal assignment process. When the MAP task contains a considerable amount of goals, the time spent on estimating the cost per goal-agent can be heavily increased if the search space of the problem is big. In real-life environments, this issue can be solved by including external information to boost the cost estimation process.
- Given that our objective was to focus on loosely-coupled tasks, PMR cannot deal with joint actions (actions that require more than one agent to be executed, as moving a table by using two agents).

6 Related work

MAP lies between the automated planning and multi-agent communities, with strong implications in other areas, as robotics. As it was discussed in the introduction, approaches range from centralized to distributed planning. In case of distributed planning, some papers employ a distributed coordinated approach when generating plans [26, 39, 42, 46], while others delay coordination and perform plan merging after generating the individual plans [17].

MAP is an active topic within the planning community as shown by the organization of the CoDMAP competition and the wide range of planners that participated [43]. The planners vary from strong privacy preservation to no privacy preservation, from fully distributed to centralized; there are many ways of classifying MAP algorithms. Different classifications are explained in the survey [44]. PMR automatically changes its behavior from a purely distributed planner to a centralized planner depending on the input planning task. It is also capable of maintaining weak privacy.

In relation to plan merging, Mali devised algorithms for performing plan merging by removing or rearranging actions [35], while PMR can also modify the input merged plan by adding actions. Furthermore, PMR can handle plans where the same action appears in several individual plans. Britanik and Marefat proposed to perform plan merging within HTN planning [11]. Merging appears at different levels of abstraction by decomposing a plan into subplans. Our approach, PMR, does not work on HTN planning and neither has different levels of abstraction. Instead of decomposing a plan into subplans, we focus on decomposing the problem into subtasks. In the field of temporal planning, Mudrova et al. [36] propose an algorithm that merges partial order plans with durative actions for solving robotic tasks. A different approach close to plan merging is conflict solving. Jordan presents one such approach where conflicts are identified while agents generate the solution [27]. The algorithm works at the same time on a joint plan and penalizes the agents that generate a conflict.

Regarding factorization on MAP, some works in MAP deal with plan decomposition. Brafman and Domshlak propose a decomposition method of the planning domain [8]. Crosby *et al.* present a centralized total ordered planning algorithm that decomposes, with the help of heuristics, a loosely coupled problem into agents and subtasks [15]. In a recent work, Mali and Puthiyattil

transform a MAP task into a set of subplans encoded in SAT to be later assigned to the agents [34]. Our approach, PMR, decomposes the problem into subtasks by assigning goals to agents. Then each subtask is given to a different agent. We are also helped by heuristics to proceed with the goal assignment.

Plan Reuse has also been studied in Automated Planning. Gerevini *et al.* define a new domain-independent planning system in [20]. They use two techniques: the first one divides the actions of the plan to repair in subgroups to later solve the conflicts with a Planning Graph; the second technique uses Action Graphs to reduce the number of inconsistencies reflected in the plan that needs to be adapted. Related with MAP, van der Krogt *et al.* apply a plan repair system to a MAP problem so that agents can adapt their plans iteratively by exchanging goals in an auction until all plans generated are valid [30]. The main difference with respect to our work is that we use a plan reuse phase after all plans from agents are merged and parallelized and plan reuse is executed only once. Krogt et al. also presented an extension of the VHPOP planner called POPR [31]. Their approach computes a set of partial plans similar to the given input plan. Then, it analyzes the dependencies of predicates and actions of those plans by generating removal trees and uses a heuristic to compute the most promising candidate. Finally, once the candidate plan is selected, plan reuse is applied to it. As it can be seen, the procedure is very different from RRPT-PLAN, but it is one of the first works that combines search and plan reuse on the same planner. The first plan reuse planner that incorporated heuristic search inside the replanning process was SHERPA [28]. It stores knowledge about both previous plans and previous plan-construction processes. RRPT-PLAN does not use heuristic search inside the plan reuse phase. We just try to reuse as many actions as possible from the input plan but we do not modify them. If plan reuse is not applicable anymore, classical heuristic search is performed from that point.

7 Conclusions and future work

We have presented PMR, an algorithm capable of solving a MAP task by merging individual plans and applying plan-reuse techniques, if needed. A key feature of PMR is that it automatically adapts to the interaction level among agents and goals, varying its behavior from distributed to centralized. It generates many valid plans in the merge phase (M) with a low computational effort if the domain has a low degree of interaction. Otherwise, it uses plan reuse (R) in domains with more interaction and resorts to centralized planning (C) in case of domains with strong interactions.

Another advantage of PMR is that it only includes off-the-shelf planners in its three phases (M, C and R). Hence, we can trivially improve the performance of PMR by just changing the planners used by better ones once they are developed. Moreover, PMR can easily be configured to target coverage, cost or makespan by just changing its goal allocation strategy.

When testing the set of planners against the last benchmark of hard problems, the results of PMR-RRPT-PLAN have shown that it easily adapts to any type of MAP problem, independently of the problem's features (e.g. number of agents, goals, interactions). PMR-RRPT-PLAN maintains good results on coverage and time and remarkable results on makespan, specially in combination with the *Load-balance* strategy.

Regarding plan reuse, we have presented another contribution, RRPT-PLAN, which is an algorithm that combines plan reuse, sampling and local search to solve a planning problem. RRPT-PLAN receives the domain, the problem and an input plan (usually invalid) from which it will try to reuse actions to include in the final plan. We have shown in the experiments that RRPT-PLAN adapts to diverse plan reuse scenarios, including the ones that are not usually considered by state-of-the-art plan reuse planners. Also, we have shown experiments of RRPT-PLAN outside of PMR and the results were very similar to the state-of-the-art planner LPG-ADAPT.

As future work, we would like to define new techniques for plan repair, improving the way probabilities change the behavior of RRPT-PLAN and also to work on different plan merging strategies for PMR to merge more efficiently the individual plans obtained from the agents. A third option would be to make more efficient, in terms of time, the goal-assignment process.

Acknowledgements This work has been partially supported by the MINECO projects TIN2017-88476-C2-2-R, RTC-2016-5407-4, and TIN2014-55637-C2-1-R and MICINN project TIN2011-27652-C03-02.

Appendix

A CoDMAP extra results

The following tables show specific results on Time, Makespan and Cost obtained on the CoDMAP experiments from Section 5.4.

Table 13 IPC scores in Time per configuration in CoDMAP domains

	PMR-RRPT-PLAN							
	BC	LB	ALL	CMAP-T	ADP-L	SIW	LAMA-FIRST	YAHSP
Driverlog	18.92	17.44	16.54	19.66	17.29	16.04	18.40	14.53
Zenotravel	19.31	18.00	15.74	19.84	18.80	18.75	18.64	10.71
Elevators	16.78	16.73	16.45	19.40	20.00	19.32	18.48	1.58
Logistics	20.00	19.65	19.07	20.00	20.00	20.00	20.00	20.00
Rovers	17.16	15.56	16.26	19.87	19.34	17.68	18.10	2.97
Satellites	18.54	17.70	17.04	20.00	19.31	17.96	17.18	7.46
Sokoban	8.82	6.56	11.32	12.13	15.16	15.52	13.14	5.76
Taxi	19.84	19.83	17.77	20.00	20.00	20.00	20.00	19.03
Blocks	20.00	15.25	14.98	20.00	18.42	16.56	18.19	14.88
Wireless	0.77	0.75	1.77	1.79	8.95	3.24	3.35	15.17
Depots	13.48	12.75	13.01	14.27	12.73	15.03	12.91	12.60
Woodworking	13.26	13.26	15.91	14.76	17.59	11.75	18.75	2.00
Total	186.87	173.48	175.86	201.72	**207.59**	191.84	197.14	126.69

Bold represent the best configuration

Table 14 IPC scores in Makespan per configuration in CoDMAP domains

	PMR-RRPT-PLAN						
	BC	LB	ALL	CMAP-T	ADP-L	LAMA-FIRST	YAHSP
Driverlog	11.82	12.65	8.86	10.99	13.84	12.43	16.00
Zenotravel	11.22	17.03	17.03	11.08	10.62	12.88	12.00
Elevators	16.25	16.41	16.41	17.96	16.30	17.44	2.00
Logistics	15.09	15.09	15.09	15.39	15.28	16.14	20.00
Rovers	10.56	9.86	9.75	17.51	11.06	16.39	4.00
Satellites	7.68	16.01	15.95	9.82	7.31	9.08	10.00
Sokoban	7.78	6.50	6.50	11.43	13.64	14.36	7.00
Taxi	13.37	13.37	13.37	13.37	13.43	13.32	20.00
Blocks	8.38	6.02	6.95	7.79	11.10	9.69	17.00
Wireless	1.89	1.89	1.89	4.52	7.21	4.55	18.00
Depots	10.13	10.76	11.00	9.49	7.19	10.00	14.00
Woodworking	13.43	13.43	13.43	10.80	19.67	13.43	0.90
Total	127.60	139.02	136.24	140.12	146.66	**149.72**	140.90

Bold represent the best configuration

Table 15 IPC scores in Cost per configuration in CoDMAP domains

	PMR-RRPT-PLAN							
	BC	LB	ALL	CMAP-T	ADP-L	SIW	LAMA-FIRST	sYAHSP
Driverlog	16.90	15.57	14.28	17.27	18.30	14.40	17.92	14.64
Zenotravel	19.38	16.31	18.26	17.07	18.09	13.64	18.61	10.06
Elevators	15.95	15.87	15.11	14.25	15.23	14.59	15.74	2.00
Logistics	18.91	18.91	18.94	19.17	19.34	13.34	19.60	19.82
Rovers	18.64	16.87	18.10	19.43	19.11	16.94	18.41	3.75
Satellites	16.44	14.17	13.26	19.51	17.29	18.54	18.21	7.98
Sokoban	8.54	6.64	15.06	11.65	13.61	16.39	14.54	6.71

Table 15 (continued)

	PMR-RRPT-PLAN							
	BC	LB	ALL	CMAP-T	ADP-L	SIW	LAMA-FIRST	sYAHSP
Taxi	18.94	18.94	18.94	18.94	18.67	16.03	18.62	16.11
Blocks	13.35	8.00	13.35	12.62	12.81	18.98	12.54	14.52
Wireless	1.93	1.93	4.44	4.44	9.00	6.39	4.47	17.44
Depots	15.07	14.54	14.54	14.62	9.05	14.66	12.97	13.25
Woodworking	17.57	17.57	17.57	15.20	17.85	15.18	17.57	2.00
Total	181.61	165.33	181.87	184.17	188.34	179.05	**189.22**	128.27

Bold represent the best configuration

B Hard problems extra results

Table 16 shows results on the time score obtained on the Hard problems experiments from Section 5.6.

Table 16 Time score score in hard and specific problems

	PMR-RRPT-PLAN						
	BC	LB	CMAP-T	ADP-L	SIW	LAMA-FIRST	YAHSP
Zenotravel	16.66	15.37	14.03	17.15	0.40	15.63	0.00
Satellite	16.61	15.58	15.53	19.86	0.53	18.72	0.00
Rover	10.34	10.25	10.22	19.40	0.00	19.87	0.00
Driverlog	0.00	2.05	4.37	5.22	0.00	7.56	0.00
Blocks	14.37	4.46	15.13	10.13	7.54	9.19	7.57
Rover-graph	8.33	8.33	9.60	20.00	11.73	6.23	0.00
VRP	20.00	12.16	10.79	13.57	1.56	11.03	0.00
Depots-robots	11.57	11.48	8.86	7.94	10.16	9.23	0.00
Total	97.89	79.68	88.53	**113.27**	31.92	97.47	7.57

Bold represent the best configuration

References

1. Alcázar V, Veloso MM, Borrajo D (2011) Adapting a rapidly-exploring random tree for automated planning. In: SOCS
2. Bäckström C, Nebel B (1995) Complexity results for sas+ planning. Comput Intell 11(4):625–655
3. Borrajo D (2013) Plan sharing for multi-agent planning. In: Nissim R, Kovacs DL, Brafman R (eds) Proceedings of the ICAPS'13 DMAP Workshop, pp 57–65
4. Borrajo D, Fernández S (2015) MAPR and CMAP. In: Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15), Jerusalem (Israel), http://agents.fel.cvut.cz/codmap/results/CoDMAP15-proceedings.pdf
5. Borrajo D, Fernández S (2018) Efficient approaches for multi-agent planning. KAIS 14:253–302
6. Borrajo D, Veloso MM (2012) Probabilistically reusing plans in deterministic planning. In: Inproceedings of ICAPS'12 Workshop on HSDIP
7. Borrajo D, Roubíčková A, Serina I (2015) Progress in case-based planning. ACM Comput Surv 47(2):35:1–35:39. https://doi.org/10.1145/2674024
8. Brafman RI, Domshlak C (2006) Factored planning: How, when, and when not. In: Proceedings of AAAI-2006, pp 809–814
9. Brafman RI, Domshlak C (2008) From one to many: Planning for loosely coupled multi-agent systems. In: Proceedings of the 18h International Conference on Automated Planning and Scheduling, ICAPS, pp 28–35, http://www.aaai.org/Library/ICAPS/2008/icaps08-004.php
10. Brafman RI, Domshlak C (2013) On the complexity of planning for agent teams and its implications for single agent planning. Artif Intell 198:52–71. https://doi.org/10.1016/j.artint.2012.08.005
11. Britanik J, Marefat M (1995) Hierarchical plan merging with application to process planning. In: Proceedings of IJCAI - Volume 2, pp 1677–1684, http://dl.acm.org/citation.cfm?id=1643031.1643116
12. Bylander T (1994) The computational complexity of propositional strips planning. Artif Intell 69(1-2):165–204

13. Conitzer V (2010) Comparing multiagent systems research in combinatorial auctions and voting. AMAI 58(3-4):239–259. https://doi.org/10.1007/s10472-010-9205-y
14. Crosby M (2015) Adp an agent decomposition planner. In: Proceedings of the CoDMAP-15, pp 4
15. Crosby M, Rovatsos M, Petrick RPA (2013) Automated agent decomposition for classical planning. In: ICAPS
16. Fikes RE, Nilsson NJ (1971) Strips: A new approach to the application of theorem proving to problem solving. In: Proceedings of IJCAI, pp 608–620, http://dl.acm.org/citation.cfm?id=1622876.1622939
17. Foulser D, Li M, Yang Q (1992) Theory and algorithms for plan merging. Artif Intell 57(2-3):143–181
18. Fox M, Long D (2003) PDDL2.1: An extension to PDDL for expressing temporal planning domains. JAIR 20
19. Fox M, Gerevini A, Long D, Serina I (2006) Plan stability: Replanning versus plan repair. In: Proceedings of ICAPS'06, pp 212–221
20. Gerevini A, Serina I (2000) Fast plan adaptation through planning graphs: Local and systematic search techniques. In: AIPS, pp 112–121
21. Gerkey BP, Matarić M (2004) A formal analysis and taxonomy of task allocation in multi-robot systems. Int J Robot Res 23(9):939–954
22. Helmert M (2006) The fast downward planning system. JAIR 26:191–246
23. Hoffmann J (2003) The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. J Artif Int Res 20(1):291–341. http://dl.acm.org/citation.cfm?id=1622452.1622463
24. Hoffmann J, Nebel B (2001) The FF planning system: Fast plan generation through heuristic search. JAIR 14:253–302
25. Howey R, Long D, Fox M (2004) VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In: ICTAI 2004, pp 294–301
26. Jonsson A, Rovatsos M (2011) Scaling up multiagent planning: A best-response approach. In: Proceedings of ICAPS'11, pp 114–121
27. Jordán J, Torreño A, de Weerdt M, Onaindia E (2018) A better-response strategy for self-interested planning agents. Appl Intell 48(4):1020–1040. https://doi.org/10.1007/s10489-017-1046-5
28. Koenig S, Furcy D, Bauer C (2002) Heuristic search-based replanning. In: Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, AIPS'02, pp 294–301, http://dl.acm.org/citation.cfm?id=3036884.3036923
29. Kovacs DL (2012) A multi-agent extension of pddl3.1. In: Proceedings of the 3rd Workshop on the International Planning Competition, IPC, ICAPS, pp 19–27
30. van der Krogt R, de Weerdt M (2005) Self-interested planning agents using plan repair. In: Workshop on Multiagent Planning and Scheduling, ICAPS-2005, pp 36–44
31. Krogt RVD, Weerdt MD (2005) Plan repair as an extension of planning. In: Inproceedings of ICAPS-05, pp 161–170
32. LaValle SM, Kuffner JJ (2001) Randomized kinodynamic planning. I J Robotics Res 20(5):378–400
33. Luis N, Borrajo D (2014) Plan Merging by Reuse for Multi-Agent Planning. In: Proceedings of the 2nd ICAPS Distributed and Multi-Agent Planning workshop (DMAP)
34. Mali A, Puthiyattil R (2013) Fully-automated instance decomposition and subplan synthesis for parallel execution. In: ICTAI, pp 322–329, https://doi.org/10.1109/ICTAI.2013.56
35. Mali AD (2000) Plan merging & plan reuse as satisfiability. In: Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning. ECP'99, pp 84–96
36. Mudrova L, Lacerda B, Hawes N (2016) Partial order temporal plan merging for mobile robot tasks. In: Proceedings of ECAI 2016, IOS Press, vol 285, pp 1537–1545
37. Muise C, Lipovetzky N, Ramirez M (2015) MAP-LAPKT: Omnipotent multi-agent planning via compilation to classical planning. In: Competition of DMAP, http://www.haz.ca/papers/muise_CoDMAP15.pdf
38. Nebel B, Koehler J (1995) Plan reuse versus plan generation: A theoretical and empirical analysis. AIJ 76:427–454
39. Nissim R, Brafman RI (2013) Cost-optimal planning by self-interested agents. In: Proceedings of AAAI'13
40. Nissim R, Brafman RI (2014) Distributed heuristic forward search for multi-agent planning. JAIR 51:293–332. https://doi.org/10.1613/jair.4295
41. Richter S, Westphal M (2010) The LAMA planner: Guiding cost-based anytime planning with landmarks. JAIR 39:127–177
42. Stolba M, Komenda A (2017) The madla planner: Multi-agent planning by combination of distributed and local heuristic search, vol 252. https://doi.org/10.1016/j.artint.2017.08.007, http://www.sciencedirect.com/science/article/pii/S0004370217301042
43. Stolba M, Komenda A, Kovacs DL (eds) (2015). In: Proceedings of the Competition of DMAP (CoDMAP-15)
44. Torreño A, Onaindia E, Komenda A, Štolba M (2017) Cooperative multi-agent planning: A survey. ACM Comput Surv 50(6):84:1–84:32. https://doi.org/10.1145/3128584
45. Torreño A, Onaindia E, Sapena Ó (2014) Fmap: Distributed cooperative multi-agent planning. Appl Intell 41(2):606–626. https://doi.org/10.1007/s10489-014-0540-2
46. Torreño A, Sapena Óscar, Onaindia E (2018) Fmap: a platform for the development of distributed multi-agent planning systems. Knowl-Based Syst 145:166–168. https://doi.org/10.1016/j.knosys.2018.01.013
47. Veloso MM, Pérez MA, Carbonell JG (1990) Nonlinear planning with parallel resource allocation. In: Proceedings of the DARPA Workshop on IPSC, Morgan Kaufmann, pp 207–212
48. Vidal V (2004) The YAHSP planning system: Forward heuristic search with lookahead plans analysis. In: Proceedings of the 4th IPC, pp 59–60
49. Wurman PR, Raffaello D, Mountz M (2007) Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In: Proceedings of the 19th IAAI - Vol 2, pp 1752–1759, http://dl.acm.org/citation.cfm?id=1620113.1620125

Nerea Luis is a PhD in Artificial Intelligence by the Universidad Carlos III de Madrid. Her thesis is about combining multi-agent planning and plan-reuse to easily adapt to different scenarios. In 2016, she was a visitor student at Manuela Veloso's CORAL group for 4 months in Carnegie Mellon University (USA). Her current research interests are multi-agent planning, machine learning and robotics. She was awarded the Women Techmakers scholarship by Google.

Susana Fernández is currently an Associate Professor at the Universidad Carlos III de Madrid, since 2007. She received the B.A. degree in Physics (Universidad Complutense de Madrid, 1989), and a M.Phil. degree in "Logic, Text and Information Technology" (University of Dundee, Scotland, 1991). She worked for 10 years as a software engineer in two private companies, INDRA and Eliop, S.A. She received the Ph.D. degree in Computer Science (Universidad Carlos III de Madrid, 2006). Her current research interest is in Artificial Intelligence, particularly automated planning, machine learning and multi-agent systems. She has participated in several competitive projects related to automated planning, problem solving or agents. She has published more than 20 papers.

Daniel Borrajo received his Ph.D. from Universidad Politcnica de Madrid (1990). Currently, he is a Professor of Computer Science at Universidad Carlos III de Madrid. He has published over 200 journal and conference papers, mainly in the fields of problem solving methods (heuristic search and automated planning) and machine learning. He has been the Program Co-chair of the International Conference of Automated Planning and Scheduling (ICAPS'13), Conference co-chair of the Symposium of Combinatorial Search (SoCS'12, SoCS'11) and ICAPS'06, and PC member (including Area chair and Senior PC) of the main conferences on Artificial Intelligence (e.g., IJCAI, AAAI, ICAPS, ...). His current research interests lie in goal reasoning, Multi-agent Planning, and Machine Learning applied to Planning.