Decentralised Planning for Multi-Agent Programming Platforms

Rafael C. Cardoso
University of Liverpool
Liverpool, United Kingdom
rafael.cardoso@liverpool.ac.uk

Rafael H. Bordini*
School of Technology, PUCRS
Porto Alegre, RS, Brazil
rafael.bordini@pucrs.br

ABSTRACT

Extensive research has been done on planning for single-agent problems, but multi-agent planning has not as yet been thoroughly explored in agent development platforms. These platforms typically provide various mechanisms for runtime coordination, which are often useful in online planning. In this context decentralised multi-agent planning can be efficient as well as effective, especially in loosely-coupled domains, whilst also ensuring important properties in agent systems such as privacy and autonomy. In this paper, we describe the DOMAP planning framework and its integration with a multi-agent programming platform to support the achievement of social goals. Our planning framework has separate phases for goal allocation and individual HTN planning, whilst relying on available runtime coordination. Experiments on three different multi-agent systems implemented in JaCaMo show that DOMAP outperforms four other state-of-the-art multi-agent planners with regards to both planning and execution time.

KEYWORDS

multi-agent planning; hierarchical task network; multi-agent development platforms; goal allocation

ACM Reference Format:
Rafael C. Cardoso and Rafael H. Bordini. 2019. Decentralised Planning for Multi-Agent Programming Platforms. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

Multi-Agent Planning (MAP) has received significant attention from the relevant research communities [19, 35] as it tackles new and complex multi-agent problems that require decentralised solutions. By allowing agents to do their own individual planning the search space is effectively pruned, and agents also get to maintain their autonomy and keep some privacy within the system. A prevailing problem in the area is to combine planning and execution [21]. Multi-agent programming languages and development platforms that cover the social and environment dimensions [2, 31] of Multi-Agent Systems (MAS) as well as the agent dimension [3] make multi-agent oriented programming especially suited for solving complex problems that require highly social, decentralised autonomous software.

One such multi-agent development platform is JaCaMo[1] [2]. A platform for the development of MAS that explores the use of three programming dimensions, each operated by a dedicated approach with its own programming model: organisation (Moise [17]), agent (Jason [4]), and environment (CArtAgO [25]). The JaCaMo platform connects them all together via a global meta-model. At the top-most level, the organisation dimension is composed of roles, missions, and schemes. The roles are adopted by Belief-Desire-Intention (BDI) agents that populate the agent dimension. At the bottom-most dimension, the environment houses artifacts that provide information about the environment, grouped into workspaces that agents can access. These workspaces can be distributed across multiple network nodes, effectively distributing the MAS.

JaCaMo has been used in many applications, such as in the production control of printed circuit boards [26], applying ontologies in the area of health care [13], and in the simulation of social production and management processes in urban environments [24]. JaCaMo was also used by several top-ranking teams in the Multi-Agent Programming Contest (MAPC) [1], an annual competition to stimulate research in multi-agent programming. The MAPC introduces complex scenarios that can be used to compare the effectiveness of multi-agent platforms towards solving these problems.

Our main contribution in this paper is a framework for Decentralised Online Multi-Agent Planning (DOMAP), including the design details, its implementation in JaCaMo, and evaluation of the framework. DOMAP is divided into several main components: (i) a multi-agent factored representation, i.e., a multi-agent planning formalism that contains information about the world according to each agent's point of view; (ii) a contract net mechanism for goal allocation; and (iii) individual Hierarchical Task Network (HTN) planning. Our approach combines MAP with MAS, allowing for dynamic execution of solutions found through planning, and making it easier to transition from planning to execution and vice-versa. Solutions found by DOMAP are generally sub-optimal, since planning at runtime often requires fast response, although finding optimal solutions is possible, depending also on optimal goal allocation.

We use three scenarios in our experiments to compare our framework against other implemented multi-agent planners: the well-established Petrobras domain [34], the classical Rovers domain, and the Floods domain [9]. In our experiments agents are fully cooperative and norm-complying entities that aim to achieve their organisations' plans. To evaluate our implementation, we selected four state-of-the-art planners that took part in a multi-agent planning competition. DOMAP outperforms those other planners in the largest planning problems and has the best overall results in both planning and execution experiments. For the latter, we use JaCaMo to execute the solution found by all planners.

*Currently on a sabbatical leave funded by CAPES at the Universities of Genoa and Oxford.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

[1] http://jacamo.sourceforge.net/

2 RELATED WORK

Much work [6, 14, 16, 19, 28] has been done to add planning into autonomous agents, but to the best of our knowledge there is no other recently implemented multi-agent online planner that takes advantage of the various programming dimensions in MAS. As such, in this section we describe some alternative approaches that share a few similarities with our work.

The work on the CANPLAN language provided the first operational semantics for incorporating HTN planning into BDI agent programming languages [28]. However, there is no available implementation of CANPLAN, making it difficult to use and compare it with current approaches. IndiGolog [14] is an agent-oriented programming language based on the situation calculus and implemented in Prolog. It adds planning in the form of high-level program execution, allowing planning to be incremental by interleaving planning and execution. IndiGolog agents are not strictly BDI-based, even though it has been shown that Golog is capable of capturing BDI semantics [27]; the language does not contain support for programming environments or organisations as first-class abstractions, one of the key motivators in our choice of development platform. The GOAL [15] agent programming language was also extended to include a subset of PDDL (Planning Domain Definition Language) planning in [16]. GOAL focuses on classical single-agent planning with PDDL, whilst we use HTN.

By extending the AgentSpeak(L) interpreter as proposed in [19], BDI agents are able to call a classical planner to create new plans at runtime in order to respond to unforeseen circumstances at design time. Annotating the expected effects of actions at design time allows them to be used during the creation of new plans. The translation between the agent and the planner is very limited due to the use of classical planning, effectively losing information that could potentially be used in planning and execution.

There was also pioneering work on RETSINA [32], TAEMS [12], and Machinetta [29]. Goals are pre-allocated in RETSINA, that is, there is no goal allocation mechanism. TAEMS and Machinetta both use scheduling techniques to coordinate tasks with no explicit planning component. These three approaches were discontinued and are no longer being developed, which limits their practical use, particularly their use in new experiments such as the ones conducted in Section 4.

A Multi-Agent Planning Language (MAPL) is proposed in [6]. Agents expressed in this language interleave planning, acting, sensing, and communication. Their approach is based on sharing knowledge to ensure the synchronous execution of joint plans. Our approach is aimed towards loosely-coupled domains where agents can keep their knowledge private.

3 A FRAMEWORK FOR DECENTRALISED ONLINE MULTI-AGENT PLANNING

DOMAP is a general-purpose domain-independent framework for multi-agent planning in loosely-coupled MAS. It consists of several main components: *input language* — a description of the language that is used as input for planning and to describe the output plans that are sent for execution; *goal allocation* — a mechanism used to allocate goals to agents; *individual planning* — the planner to be used during each agent's planning phase. In this paper, we focus on loosely-coupled domains, that is, domains with a low number of interactions that can cause conflicts. We assume that there are social laws [30] in place to solve any potential conflict.

Multiple agents interact in an environment to exchange information and carry out actions. These agents are part of an organisation where they can adopt roles, follow norms, and work towards social plans. A social plan is a structured collection of social goals (specifically a Moise scheme, in the case of JaCaMo). One such social plan achieves one particular global goal of the system through the work of various agents trying to achieve the social goals forming the social plan. In Moise, a scheme has a tree structure and the social goals at the leaves are pursued by one particular agent individually. One of our assumptions is that achieving all the leave goals of a social plan counts as achieving the global goal, although this might not be true in all domains.

Planning in DOMAP starts in any of the following situations: (*a*) new global goals are created for which there are no known social plans; or (*b*) the execution of a social plan fails, prompting the dropping of all current social goals and intentions related to that social plan. Then, all of the remaining social goals that are still active in that plan are announced as new social goals to be allocated to the available agents. Note that because we do not know the cause of the failure, it is not unreasonable to discard information that was provided by the designer of the social plan. We extract only the leave goals from the failed social plan (i.e., a Moise scheme) and discard all other goal-related information (e.g., roles, cardinalities, and missions). We assume that we have at our disposal artifacts that will support runtime coordination to make up for the lack of coordination that the designer aimed to enforce with the scheme, which failed.

In our experiments failure would often result from unpredictable changes in the environment that had an impact on the entire system. Thus, replanning for all related goals in a social plan was more reliable than replanning for only the one that failed. DOMAP can also be used to plan for private goals, which is reduced to single-agent planning and only uses the individual planner component. However, our framework does not provide scheduling between private and social goals; they will be executed in the order that solutions were originally found.

When called, DOMAP goes through the following steps:

(1) *Allocate goals*: Agents receive the announcement of new social goals. A Contract Net Protocol (CNP) mechanism is used to allocate goals to agents that have the best (estimated) chance of finding a potential solution.
(2) *Translate factored representation interface*: Each agent has its own representation with all the information about the MAS that it has access to at the time. This is translated into input for the SHOP2 HTN planner [22].
(3) *Agents start their individual planner*: Each agent runs an instance of the SHOP2 planner to search for a plan to achieve all of the goals it has been allocated. If an agent was not able to find a solution to an allocated goal, the process returns to step (1) to reallocate any remaining social goals that have failed during planning.
(4) *Solution translation*: After all social goals have been allocated the solution found by each agent's planner goes through an

inverse translator, parsing it into plans and adding them to their respective agent's plan library.

(5) *Plan execution and coordination*: Agents execute their newly found plans in accordance with the social laws embedded into the coordination artifacts.

Figure 1 illustrates the steps above, and show the implementation and runtime of DOMAP[2] in JaCaMo. Boxes with observable properties and operations represent CArtAgO artifacts. Agents that focus on an artifact receive the observable properties as beliefs and are able to execute the artifact's operations (i.e., actions). The CArtAgO infrastructure artifacts are omitted from the figure to improve readability. More details about CArtAgO infrastructure can be found in [23, 25].

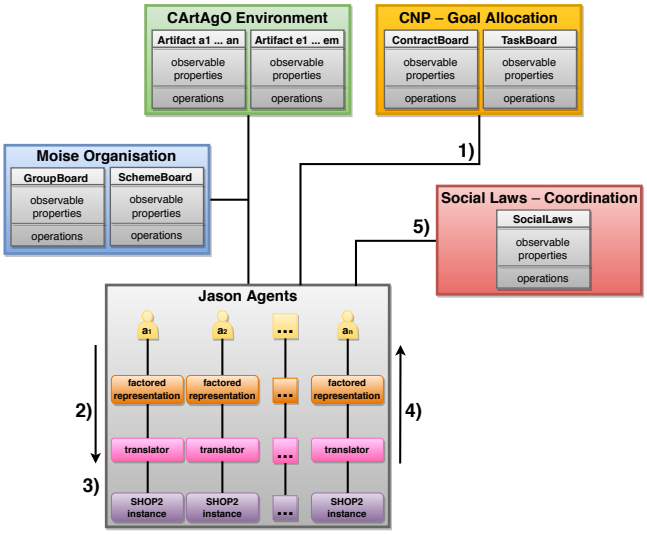

Figure 1: Overview of DOMAP's implementation in JaCaMo.

To facilitate the translation of the agents' knowledge we added one artifact per agent (`Artifact` a_1 ... a_n) that contains the information and actions that only its associated agent would have. Note, however, that this is not how MAS are traditionally programmed in JaCaMo. Normally these things would either be directly represented within the agent's code or in a public environment artifact.

3.1 Multi-Agent Factored Representation

Planning input (i.e., domain and problem representation) and output (i.e., the solution) in DOMAP are regulated by a *factored representation* of each agent's knowledge. Each agent's factored representation serves as an interface between the MAS and DOMAP. Agents use their interface to extract information from plans in their plan library and from CArtAgO environment artifacts. The correlations between JaCaMo elements, our factored representation, and SHOP2 syntax are expressed in Table 1.

We use a HTN translator to parse information from the factored representation interfaces to SHOP2 syntax. Briefly, SHOP2 syntax consists of a set of operators and a set of methods. *Operators* (also known as primitive tasks) are action descriptors that can be executed given some preconditions, causing a list of postconditions to become true. *Methods* are non-primitive tasks that impose constraints into the domain in order to guide the search for solutions through task decomposition. A HTN problem description contains a list of atoms that are true during the initial state of the system, as well as the initial task network to be decomposed (i.e., the goals).

[2]DOMAP's source code can be found at https://github.com/smart-pucrs/DOMAP

Table 1: Correlations between different representations.

JaCaMo elements	factored represent.	SHOP2 syntax
observable property	belief	state
operation	action	operator
operation's precond.	action's precond.	operator's precond.
operation's effects	action's effects	operator's add/delete
plan	plan	method
plan's precond.	plan's precond.	method's precond.
Moise leaf goals	social goals	initial task network

The translator generates the problem and domain representation for each agent based on information available in their factored representation interface. Environment information is collected from the CArtAgO artifacts observable properties into initial states. Social goals are retrieved from their respective announcements. Operators are created from all of the artifact operations that the agent has access to; their preconditions are obtained from any conditional tests in an artifact operation; and their add and delete lists are acquired from addition and deletion of observable properties. Methods are generated from the plans found in that agent's plan library, with the preconditions parsed from the context of the plan, and the task list parsed from actions and subgoals found in the body of the plan.

Although we do not enforce privacy, our agents also do not violate it unless specifically programmed to do so. Enforcing privacy would require mechanisms that ensure that an agent cannot access information that it is not privy to. We also do not impose any restrictions in agent communication, agents can freely exchange information via message passing.

3.2 Goal Allocation using CNP

Centralised planners assign goals to agents during the search for a solution to a planning problem. In a multi-agent setting, this can constrain the autonomy of the agents and potentially violate their privacy. By using task allocation protocols the agents themselves can decide who is better suited to take each goal, then later plan individually (provided coordination mechanisms are in place), giving a higher degree of autonomy and privacy during planning.

A high-level overview of DOMAP's goal allocation is shown in Algorithm 1. It starts with the initiator announcing a contract for each outstanding social goal. For implementation purposes, we used an agent to represent the organisation as the initiator during goal allocation. The *excluded* list contains the names of agents that have previously failed planning for a particular social goal and, as such, should not receive the same contract again. This list starts empty and is populated if any reallocation rounds are necessary, that is, if any agent fails planning for a social goal. Due to space constraints, Algorithms 1 and 3 are shown from a system's point of view, although both implementations are entirely decentralised.

We provide a domain-independent function that agents can use to calculate their bid (line 5 in Algorithm 1). This function performs a breadth-first expansion of a social goal using the agent's plan library, essentially exploring structures similar to goal-plan trees [33] in the process. For the purpose of comparing goal-plan trees between agents we ignore goals and subgoals from the tree. We also opted for not saving the whole goal-plan tree, instead, we update the measurements that will be part of the bid and save only the plans that need to be immediately expanded. This makes the expansion process faster and uses less memory, as shown in [7].

Algorithm 1: Goal allocation from the system's point of view.

```
1 Function allocate (SocialGoals, Agents, excluded, deadline)
2     foreach agent ∈ Agents do
3         foreach socialgoal ∈ SocialGoals do
4             if (agent, socialgoal) ∉ excluded then
5                 bid_value ← expand (socialgoal, deadline);
6                 bid (agent, socialgoal, bid_value);
```

During the expansion of a social goal our algorithm ignores the preconditions of plans, that is, we do not apply an action theory. This would involve performing lookahead, which is essentially planning, and would have a high computational cost. Instead, we opted for a quick expansion of the plan library by selecting all plans that are related to the social goal while ignoring the context of these plans. This relaxation allows for quick computation of the statistics for the goal allocation phase, sacrificing allocation quality for faster allocation times.

Agents place a multi-valued bid, a 5-tuple containing the following criteria: recursion (0 or 1), total number of actions expanded, total number of plans expanded, and maximum tree depth and width found while expanding their goal-plan tree. The "*recursion*" criteria indicates whether any recursive plan was expanded during the bid calculation for a social goal. Although we considered limiting or ignoring these recursive plans, we found out from our experiments that this can be a valuable information to have, and the disadvantage of infinite loops are surpassed by using deadlines to prevent initiators to wait indefinitely for bids. If an agent did not expand any recursive plan, then this indicates that the agent may potentially solve the social goal in less steps than agents with recursive plans. These criteria are used by the initiator to allocate social goals to agents with the best (according to the heuristic chosen for that application) bids. Our algorithm for domain-independent bid calculation is shown in Algorithm 2.

The expansion starts with the social goal at the root of the tree. The agent uses a *relevant_plans* function that returns all plans in the agent's plan library that can be used to achieve that particular goal. If the set of such *Plans* is empty, then the agent is not eligible to bid for this contract. Otherwise, the following statistics are initialised: *rec* is the presence of recursion; *n_actions* is the total number of actions found in all of the plans that were expanded; *n_plans* is the total number of plans that were expanded; *m_depth* is the maximum depth of the tree; and *m_width* is the maximum width of the tree, which initially receives the cardinality of the *Plans* set, indicating the initial width of the tree. The *Subplans* set is initially empty. Parameter *deadline* is the maximum time after the start of the expansion that the algorithm can run for. It is expected that when calling the *expand* function, the deadline given for the algorithm is slightly lower than the CNP deadline, so that the agent has time to communicate its bid to the initiator.

Algorithm 2: Breadth-first expansion of a social goal.

```
1  Function expand (goal, deadline)
2      Plans ← relevant_plans (goal);
3      if Plans = ∅ then
4          return "not eligible";
5      rec, n_actions, n_plans, m_depth ← 0;
6      m_width ← |Plans|;
7      Subplans ← ∅;
8      while there exists {plan} ∈ Plans do
9          if time() ≥ deadline then
10             return
                (rec, n_actions, n_plans, m_depth, m_width);
11         if {plan} is recursive then
12             rec ← 1;
13         n_actions ← n_actions + count_actions (plan);
14         n_plans ← n_plans + 1;
15         Goals ← goals (plan);
16         Subplans ← Subplans ∪ relevant_plans (Goals);
17         Plans ← Plans \ {plan};
18         if Plans = ∅ then
19             if |Subplans| > m_width then
20                 m_width ← |Subplans|;
21             m_depth ← m_depth + 1;
22             Plans ← Subplans;
23             Subplans ← ∅;
24     return (rec, n_actions, n_plans, m_depth, m_width);
```

While there remains any *plan* in the *Plans* set: we check if the plan has recursion; increase the counter of total plans expanded; add the number of actions found in the body of the plan to the total number of actions; and assign all the subgoals found in the body of the *plan* to the *Goals* set. Then, the agent uses the *relevant_plans* function to get all relevant plans, but now for each of the subgoals in the *Goals* set. The *plan* that was expanded is removed from the *Plans* set. If this was the last plan and the *Plans* set is now empty, then the agent checks if the cardinality of the *Subplans* set is higher than our *m_width*, in which case the cardinality of the *Subplans* set becomes the new maximum width. Then, we increase the maximum depth counter and move the *Subplans* set to the *Plans* set. When both sets are empty, or the deadline is past, the algorithm returns the multi-valued bid with the information collected during the expansion of the goal-plan tree for the given social goal.

We also allow concurrent contract net announcement and bidding. To avoid situations where an agent could win all social goals, we added priorities to the bid selection heuristics used by initiators.

The initiator awards contracts as follows: after bidding has finished for all active contract nets, the initiator first prioritises processing the results of contracts that had agents not eligible to accomplish the task, since contracts that had valid bids from all agents are easier to allocate and result in a more even distribution of goals. The first bidding attribute to be processed is *recursion*, giving priority to agents that did not expand any recursive plans. If there are any such agents, then the contract is awarded to the agent with the lowest *total number of actions*. Otherwise, the contract goes to the agent with the higher *maximum tree width*. In both cases, agents that have not been awarded any social goal yet are prioritised, and in case of any ties, the first bid that was processed wins.

3.3 Individual Planning with SHOP2

In Algorithm 3, we show the high-level function for individual planning that highlights the reallocation mechanism of DOMAP. If the individual planning of one (or more) allocated social goal(s) fail, the agent sends the respective social goal(s) back to the initiator. When all agents finish their individual planning, the initiator will either start a new round of goal allocation and individual planning (if it received at least one social goal back), or allow agents to start their execution (if no social goals were received). The name of agents who have failed planning are added to a list called *excluded*, which is sent together with the social goal when it is announced again in a new allocation round. The number of reallocation rounds depends on the efficiency of the bid selection heuristics that are being used and how restrictive the domain's goals are. For example, goals that can only be completed by a select handful of agents can cause a negative impact in performance, since more reallocation rounds may be required (see the Rovers experiments in Section 4.2).

Algorithm 3: Planning from the system's point of view.

```
1 Function plan (AllocatedGoals, Agents, excluded, deadline)
2     foreach agent ∈ Agents do
3         start_HTN_planner (agent, AllocatedGoals);
4         if planning_failed (agent, AllocatedGoals) then
5             foreach socialgoal ∈ AllocatedGoals do
6                 excluded ← excluded ∪ {(agent, socialgoal)};
7     if excluded ≠ ∅ then
8         allocate (AllocatedGoals, Agents, excluded, deadline);
```

Execution of the solution starts only when all agents who were awarded social goals finish their individual planning. This includes any reallocation rounds that might be necessary. The newly generated plans are translated into ground plans that are added to the respective agent's plan library.

4 EVALUATION

In our experiments we implemented coordination at runtime to be enforced in two different cases: joint plans involving multiple agents by using Moise schemes to define plans that have to be executed in parallel; or plans where an agent's actions can cause conflicts in other agents' plans. To solve the latter, we used the following social law: when two or more agents are trying to execute conflicting actions in the same time step, one of the agents is arbitrarily chosen to continue its execution, while all remaining involved agents hold their actions (e.g., do nothing) until the next time step. This process continues until there are no longer any conflicting actions trying to be executed in the same time step. These mechanisms were sufficient for our loosely-coupled scenarios, but tightly-coupled domains may require more sophisticated coordination mechanisms.

To the best of our knowledge, DOMAP is the only recently developed multi-agent online planner that is able to take advantage of the different programming abstractions in MAS (agent, environment, and organisation). Thus, to evaluate our framework we isolated the online components of DOMAP and compared against four state-of-the-art multi-agent offline planners, all of which took part in the 2015 Competition of Distributed and Multi-Agent Planners (CoDMAP-15) [35]. *SIW+ -then-BFS(f)* [20] (SIW) was the top performing planner of CoDMAP-15 with regards to planning time, out of 17 planners. *CMAP-t* [5] obtained second place, *ADP-legacy* [10, 11] third place, and *PMR* [18] eighth place.

Although DOMAP is the only one to use HTN rather than PDDL, we first created PDDL descriptions for all of our experiments and then translated them into suitable MAS that were later parsed to HTN during DOMAP's planning. This way there is no additional domain information being provided to SHOP2. We made sure that the HTN encoding matched the PDDL encoding to be able to provide a fair comparison, guaranteeing that the extra expressiveness of HTN would not cause any impact on the results. All HTN and PDDL encodings for our experiments can be found in the DOMAP repository at ⟨https://github.com/smart-pucrs/DOMAP⟩.

We created 10 problem variations for each scenario increasing the number of agents, goals, and initial states. We gave all planners a time limit of 60 minutes for each problem. We ran each of the five planners 20 times, resulting in a total of 3000 executions. Our measurements include: average time spent planning, average plan size, parallelisation, and average time spent executing the generated plans. Parallelisation is defined by the variance of the plan size of each individual agent, thus indicating how well the actions are spread across all agents (i.e., if the loads are balanced). To obtain execution time measurements we translated the solutions found by each planner into plans to be executed in a MAS in JaCaMo. All actions had the same execution time of 100 milliseconds.

Our experiments in this paper were made to evaluate the DOMAP framework as a whole, and not the individual planner we chose to use. Previous experiments [8] have shown that running SHOP2 by itself has much worse performance results than running it within the DOMAP framework.

The computer used to run the experiments has the following specification: Intel Xeon Processor E5645 (12M Cache, 2.40 GHz, 6 cores, 12 threads), 32 GB of memory, Ubuntu 16.04 operating system, and Java 8.

4.1 Petrobras Scenario

The Petrobras domain [34] is targeted at modelling planning and scheduling of ship operations in petroleum platforms and ports. The problem to be solved is the transportation and delivery of cargo from ports to platforms in the ocean, respecting any constraints that are imposed such as vessel load and fuel capacity. The goal

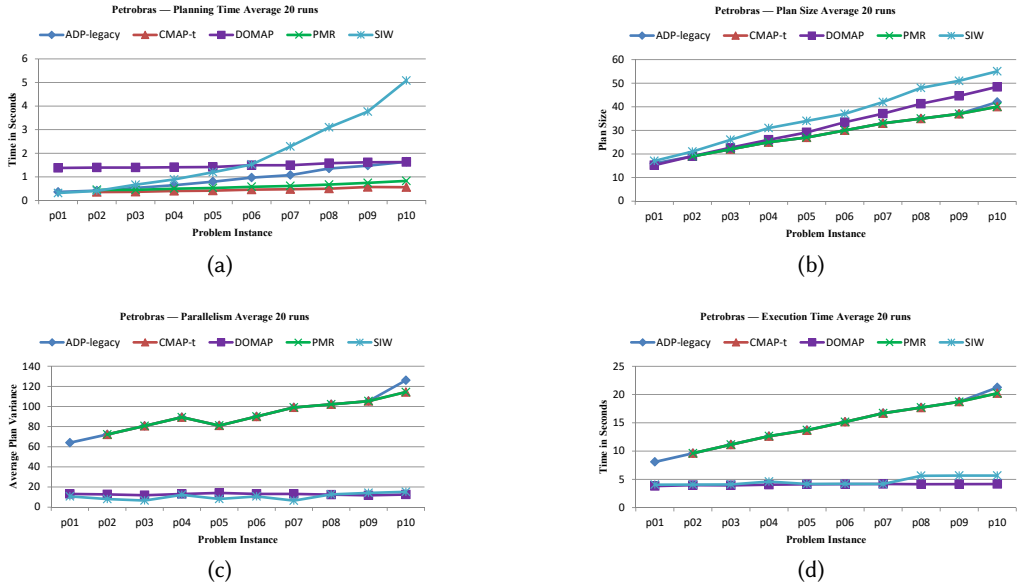

Figure 2: Results of experiments in the Petrobras domain.

is to optimise execution cost of the resulting schedule. The first problem has 4 vessels (agents) and 4 cargo (goals). Each subsequent problem adds one agent and one goal, thus, the last problem has 13 vessels and 13 cargo.

Results for time spent in planning are shown in Figure 2a. SIW scales poorly with the increase in the number of agents and goals. The other planners achieve comparable results. ADP-legacy and CMAP-t are not shown in problem 1 (p01) in all graphs of the Petrobras domain results because they had an error while parsing this problem, and thus, could not find any solution.

We can observe the plan size results in Figure 2b. SIW had the worst plan lengths across all problems in the Petrobras domain. Parallelism results are shown in Figure 2c. SIW manages to compete directly with DOMAP, even surpassing DOMAP in some problems.

The impact of parallelism is even more apparent with the execution time results in Figure 2d. The planners that had the best parallelism, SIW and DOMAP, achieve the faster execution times. The combination of planning and execution times for the last five problems is shown in Figure 3.

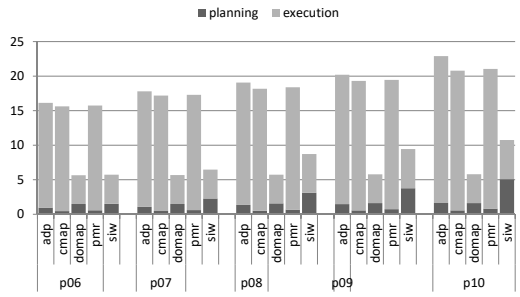

Figure 3: Planning and execution in the Petrobras domain.

Regarding parallelism, DOMAP achieves an impressive variance of zero in problems 1 and 3 in at least one of the runs. This indicates optimal goal allocation in regards to fairness with tasks distributed evenly across all agents. This is reflected in the execution results, which when combined with the planning time results show that DOMAP outperforms all other planners in the last four problems.

4.2 Rovers Scenario

Rovers is a classical domain in automated planning that is inspired by NASA exploratory space missions on Mars. Rovers are vehicles equipped with different capabilities to explore Mars' surface collecting samples and taking pictures of objectives. All data collected is then transmitted back to a lander spacecraft. Problem 1 starts with 4 agents and 8 goals, and each problem after it adds 2 agents and 4 goals, hence Problem 10 has 22 agents and 44 goals.

In Figure 4a we show the results for average planning time in the Rovers domain. Time in seconds is on the y-axis and in logarithmic scale to improve readability. The first five problems have a slow increase in planning time across all planners, but they all have similar performances with differences in the order of milliseconds. One of the 20 runs from DOMAP for the first, fifth, and sixth problems suffered from a high number of reallocation rounds due to only a few agents being able to solve some of the goals. This increased the average planning time, but other runs were much faster.

SIW could not solve problem 8 under the time limit of 60 minutes. ADP-legacy had some of the best times for the first six problems, but it was followed closely by most other planners. Once again, SIW scales poorly when increasing the number of agents and goals in each problem. In the last four problems ADP-legacy and PMR performance drop considerably; CMAP-t stays competitive with DOMAP, but does not seem to scale quite as well.

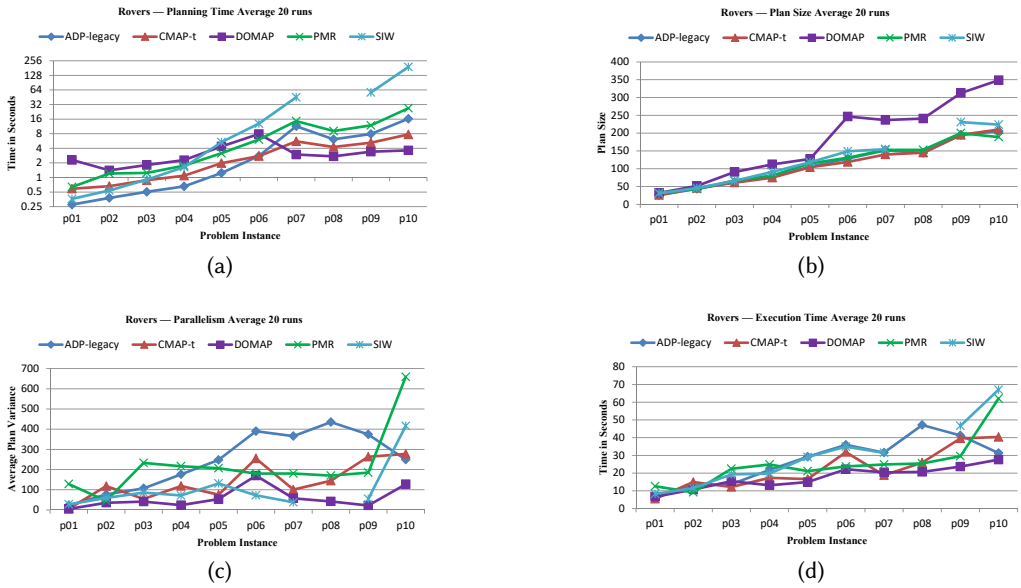

Figure 4: Results of experiments in the Rovers domain.

Plan size results are shown in Figure 4b. DOMAP has good plan sizes in problems 1–5, but in problems 6, 9, and 10 DOMAP's plan length scales very poorly; the other planners remain competitive across all problems. However, in Figure 4c, DOMAP achieves the best performance in regards to parallelism.

The time spent in execution is shown in Figure 4d. DOMAP once again shows excellent scalability and the best performance overall; other planners are inconsistent and their execution times fluctuate depending on the problem.

In Figure 5, DOMAP has the best results when combining planning and execution. SIW is not included because its total time was much higher than other planners and would affect readability.

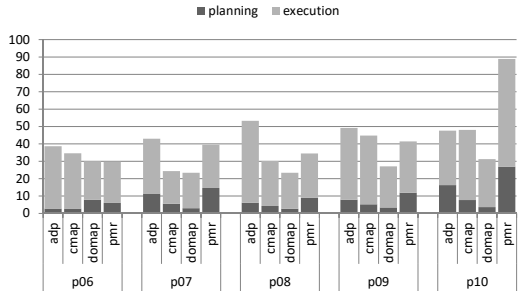

Figure 5: Planning and execution in the Rovers domain.

This is not a particularly good scenario for DOMAP. Agents are very homogeneous, their capabilities often overlap, and many goals can only be solvable by a handful of agents due to each rover having their own traversable paths. Problems 5 and 6 had the highest planning time for DOMAP, since all 20 runs in both problems required reallocation; problem 5 varied between 2 and 3 rounds of reallocation, and problem 6 between 3 and 4 rounds. Regardless of this disadvantage, DOMAP outperforms other planners in the last four problems, even though there were some runs where DOMAP had to reallocate goals and replan a few times.

4.3 Floods Scenario

The Floods domain [9] consists of a team of heterogeneous autonomous robots (unmanned aerial, ground, and surface vehicles) that are dispatched to monitor flooding activity and execute search and rescue operations within a geographical *region* divided into several interconnected *areas*. The Centre for Disaster Management (CDM) establishes bases of operation in the region that is being monitored. Problem 1 starts with 9 agents (3 of each role), 15 areas, 2 CDMs, and 9 goals. Each subsequent problem added 3 agents (1 of each type), 5 areas, and 3 goals. Every other problem had one CDM added. The last problem, p10, has 36 agents, 60 areas, 6 CDMs, and 36 goals.

Results for planning time are shown in Figure 6a. The y-axis is in logarithmic scale to improve readability. SIW had very large planning times, up to an average of 3135 seconds for problem 10. CMAP-t has lower planning times for most of the problems, except for the largest problems, where DOMAP performs best.

Plan size is shown in Figure 6b. CMAP-t, ADP-legacy, and PMR found the lowest cost plans. PMR does much worse on problems 6 and 9, when it has to use a second planner, but it is able to find the lowest cost plan for the largest problem (p10). Results of plan length were much more spread out compared to the results of plan length in the previous domains, with each planner generating the lowest cost plan for at least one problem.

DOMAP's goal allocation, of prioritising fairness among agents pays off when we consider concurrent actions (see Figure 6c). Our

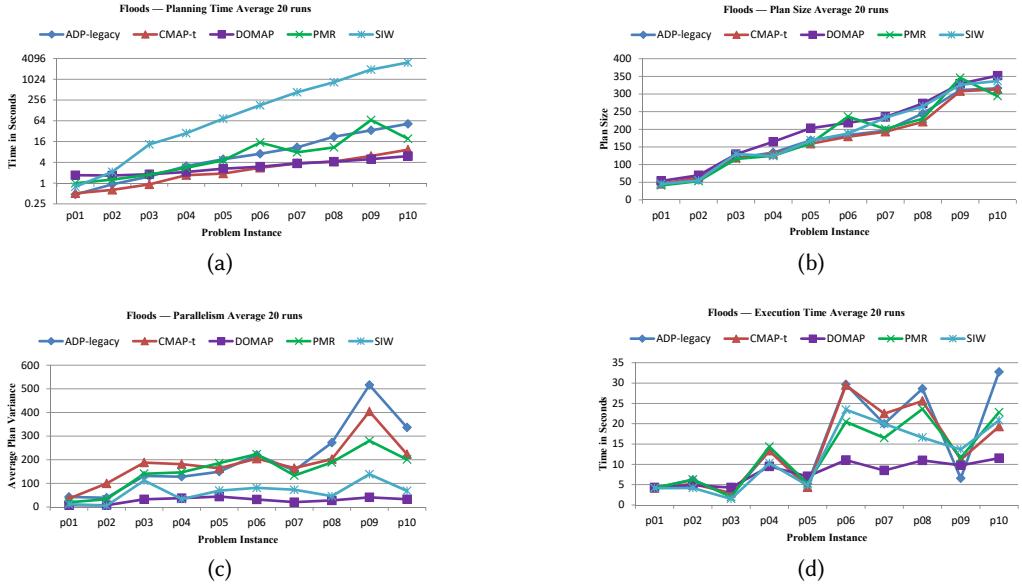

Figure 6: Results of experiments in the Floods domain.

results show that DOMAP has excellent parallel solutions, dominating other planners on most of the problems.

Time spent in execution is shown in Figure 6d. DOMAP is the only planner that scales reliably due to its good use of fairness during goal allocation. Even though DOMAP had some of the longest plans, because it allocated social goals as evenly as possible it still managed to finish execution faster than all other planners.

Figure 7: Planning and execution in the Floods domain.

When combining planning and execution (Figure 7) DOMAP scales well and obtains the best results; CMAP-t, for the most part, is able to follow closely in some problems; the other planners do not perform as well. We removed SIW for readability as its planning times for these problems were too high.

5 CONCLUSION

We described the DOMAP framework and its implementation in the JaCaMo MAS development platform. Our results show that DOMAP scales rather well, provides excellent parallel solutions, and is the fastest multi-agent planner for the largest problems. The results we obtained indicate that allocating goals before planning (only SIW does not do so) can lead to significant improvement with regards to planning time. Moreover, the DOMAP, CMAP-t, and PMR planners take advantage of multiple cores by decentralising planning, which also led to faster planning times.

Our experiments have shown that when concurrent actions are possible shorter plan lengths do not result in lower execution times, and that fairness can be as important as the overall plan length in some multi-agent settings. While most other multi-agent planners favour the *makespan* metric (largest plan size between all agents), we have shown that our *parallelism* metric is a better indicator for execution performance. Our fairness heuristic for bid selection proved to be very efficient in regards to execution time.

One of our goals with DOMAP is to turn it into a general-purpose domain-independent framework. As such, we designed DOMAP to be an open platform where other alternatives for modular components can be used by swapping the main components, allowing the developer to choose the approach that is more suited for their particular problem. This modularity would allows us to add a more complex coordination mechanism for dealing with tightly-coupled domains, while also being able to revert back to a simpler mechanism to remove any overhead for solving loosely-coupled domains. Future work also include the automatic generation of a Moise scheme to coordinate and activate the plans found by DOMAP during planning. Finally, we aim to experiment with more complex multi-agent domains.

ACKNOWLEDGEMENTS

The first author is supported by UK Research and Innovation, and EPSRC Hubs for Robotics and AI in Hazardous Environments: EP/R026092 (FAIR-SPACE) and EP/R026084 (RAIN). This research was also partially funded by CNPq and CAPES.

REFERENCES

[1] Tobias Ahlbrecht, Jürgen Dix, and Niklas Fiekas. 2018. Multi-agent programming contest 2017. *Annals of Mathematics and Artificial Intelligence* 84, 1 (Oct. 2018), 1–16. https://doi.org/10.1007/s10472-018-9594-x

[2] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. 2013. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming* 78, 6 (Jun 2013), 747–761. https://doi.org/10.1016/j.scico.2011.10.004

[3] Rafael H. Bordini and Jürgen Dix. 2013. Programming Multiagent Systems. In *Multiagent Systems* (2nd. ed.), Gerhard Weiss (Ed.). MIT Press, Cambridge, USA, Chapter 11, 587–639.

[4] Rafael H. Bordini, Michael Wooldridge, and Jomi Fred Hübner. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason.* John Wiley & Sons, Chichester, UK.

[5] Daniel Borrajo and Susana Fernandez. 2015. MAPR and CMAP. In *Proceedings of the Competition of Distributed and Multi-Agent Planners.* ICAPS, Jerusalem, Israel, 1–3.

[6] Michael Brenner and Bernhard Nebel. 2009. Continual Planning and Acting in Dynamic Multiagent Environments. *Autonomous Agents and Multi-Agent Systems* 19, 3 (Dec. 2009), 297–331. https://doi.org/10.1007/s10458-009-9081-1

[7] Rafael C. Cardoso and Rafael H. Bordini. 2016. Allocating Social Goals Using the Contract Net Protocol in Online Multi-Agent Planning. In *5th Brazilian Conference on Intelligent System.* IEEE, Recife, Pernambuco, Brazil, 199–204. https://doi.org/10.1109/BRACIS.2016.045

[8] Rafael C. Cardoso and Rafael H. Bordini. 2016. A Distributed Online Multi-Agent Planning System. In *4th Workshop on Distributed and Multi-Agent Planning.* ICAPS, London, United Kingdom, 15–23. http://icaps16.icaps-conference.org/proceedings/dmap16.pdf#page=18

[9] Rafael C. Cardoso and Rafael H. Bordini. 2017. A Multi-Agent Extension of a Hierarchical Task Network Planning Formalism. *Advances in Distributed Computing and Artificial Intelligence Journal* 6, 2 (May 2017), 5–17. http://dx.doi.org/10.14201/ADCAIJ201762517

[10] Matthew Crosby. 2015. ADP an Agent Decomposition Planner. In *Proceedings of the Competition of Distributed and Multi-Agent Planners.* ICAPS, Jerusalem, Israel, 4–7.

[11] Matthew Crosby, Michael Rovatsos, and Ronald P. A. Petrick. 2013. Automated Agent Decomposition for Classical Planning. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling.* AAAI Press, Rome, Italy, 46–54. https://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/paper/view/6051

[12] Keith S. Decker. 1996. TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. In *Foundations of Distributed Artificial Intelligence*, N. Jennings and G. O'Hare (Eds.). John Wiley & Sons, Inc, New York, NY, USA, 429–448.

[13] Artur Freitas, Daniela Schmidt, Alison Panisson, Rafael H. Bordini, Felipe Meneguzzi, and Renata Vieira. 2015. Applying Ontologies and Agent Technologies to Generate Ambient Intelligence Applications. In *Agent Technology for Intelligent Mobile Services and Smart Societies.* Springer Berlin Heidelberg, Berlin, Germany, 22–33. https://doi.org/10.1007/978-3-662-46241-6_3

[14] Giuseppe Giacomo, Yves Lespérance, Hector J. Levesque, and Sebastian Sardiña. 2009. IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents. In *Multi-Agent Programming: Languages, Tools and Applications*, Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni (Eds.). Springer, New York, NY, USA, Chapter 2, 31–72. https://doi.org/10.1007/978-0-387-89299-3_2

[15] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. 2000. Agent Programming with Declarative Goals. In *Proceedings of the 7th International Workshop on Agent Theories and Architectures.* Springer Berlin Heidelberg, Berlin, Germany, 228–243. http://dx.doi.org/10.1007/3-540-44631-1_16

[16] Koen V. Hindriks and Tijmen Roberti. 2009. GOAL as a Planning Formalism. In *Proceedings of Multiagent System Technologies.* Springer Berlin Heidelberg, Berlin, Germany, 29–40. https://doi.org/10.1007/978-3-642-04143-3_4

[17] Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. 2007. Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering* 1, 3 (Dec. 2007), 370–395. http://dx.doi.org/10.1504/IJAOSE.2007.016266

[18] Nerea Luis and Daniel Borrajo. 2015. PMR: Plan Merging by Reuse. In *Proceedings of the Competition of Distributed and Multi-Agent Planners.* ICAPS, Jerusalem, Israel, 11–13.

[19] Felipe Meneguzzi and Michael Luck. 2013. Declarative planning in procedural agent architectures. *Expert Systems with Applications* 40, 16 (Nov. 2013), 6508–6520. https://doi.org/10.1016/j.eswa.2013.05.058

[20] Christian Muise, Nir Lipovetzky, and Miquel Ramirez. 2015. MAP-LAPKT: Omnipotent Multi-Agent Planning via Compilation to Classical Planning. In *Proceedings of the Competition of Distributed and Multi-Agent Planners.* ICAPS, Jerusalem, Israel, 14–16.

[21] Dana S. Nau, Malik Ghallab, and Paolo Traverso. 2015. Blended Planning and Acting: Preliminary Approach, Research Challenges. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence.* AAAI Press, Austin, Texas, 4047–4051. https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9746

[22] Dana S. Nau, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20, 1 (Dec. 2003), 379–404. https://doi.org/10.1613/jair.1141

[23] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. 2008. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 17, 3 (Dec. 2008), 432–456. https://doi.org/10.1007/s10458-008-9053-x

[24] Flávia Pereira dos Santos, Diana Adamatti, Henrique Rodrigues, Glenda Dimuro, Esteban De Manuel Jerez, and Graçaliz Dimuro. 2016. A Multiagent-Based Tool for the Simulation of Social Production and Management of Urban Ecosystems: A Case Study on San Jerónimo Vegetable Garden - Seville, Spain. *Journal of Artificial Societies and Social Simulation* 19, 3 (June 2016), 1–29. https://doi.org/10.18564/jasss.3128

[25] Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. 2009. Environment Programming in CArtAgO. In *Multi-Agent Programming: Languages, Tools and Applications*, Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni (Eds.). Springer, New York, NY, USA, Chapter 8, 259–288. https://doi.org/10.1007/978-0-387-89299-3_8

[26] Mário L. Roloff, Marcelo R. Stemmer, Jomi F. Hübner, Robert Schmitt, Tilo Pfeifer, and Guido Hüttemann. 2014. A multi-agent system for the production control of printed circuit boards using JaCaMo and Prometheus AEOlus. In *12th IEEE International Conference on Industrial Informatics.* IEEE, Porto Alegre, Brazil, 236–241. https://doi.org/10.1109/INDIN.2014.6945514

[27] Sebastian Sardiña and Yves Lespérance. 2010. Golog Speaks the BDI Language. In *Proceedings of the International Workshop on Programming Multi-Agent Systems.* Springer Berlin Heidelberg, Berlin, Germany, 82–99. https://doi.org/10.1007/978-3-642-14843-9_6

[28] Sebastian Sardiña and Lin Padgham. 2011. A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems* 23, 1 (July 2011), 18–70. https://doi.org/10.1007/s10458-010-9130-9

[29] Paul Scerri, Yang Xu, Elizabeth Liao, Justin Lai, and Katia Sycara. 2004. Scaling Teamwork to Very Large Teams. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems.* IEEE Computer Society, Washington, DC, USA, 888–895. https://dl.acm.org/citation.cfm?id=1018841

[30] Yoav Shoham and Moshe Tennenholtz. 1995. On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence* 73, 1-2 (Feb. 1995), 231–252. https://doi.org/10.1016/0004-3702(94)00007-N

[31] Munindar P. Singh and Amit K. Chopra. 2010. Programming Multiagent Systems without Programming Agents. In *Programming Multi-Agent Systems.* Springer Berlin Heidelberg, Berlin, Germany, 1–14. https://doi.org/10.1007/978-3-642-14843-9_1

[32] Katia P. Sycara and Anandeep Pannu. 1998. The RETSINA Multiagent System: Towards Integrating Planning, Execution and Information Gathering. In *Proceedings of the 2nd International Conference on Autonomous agents.* ACM, Minneapolis, USA, 350–351. http://doi.acm.org/10.1145/280765.280858

[33] John Thangarajah, Lin Padgham, and Michael Winikoff. 2003. Detecting and avoiding interference between goals in intelligent agents. In *International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers Inc., Acapulco, Mexico, 721–726. http://dl.acm.org/citation.cfm?id=1630659.1630764

[34] Tiago Stegun Vaquero, Gustavo Costa, Flavio Tonidandel, Haroldo Igreja, Jose Reinaldo Silva, and J. Cristopher Beck. 2012. Planning and Scheduling Ship Operations on Petroleum Ports and Platforms. In *Proceedings of the Scheduling and Planning Applications woRKshop.* ICAPS, Atibaia, Brazil, 8–16.

[35] Michal Štolba, Antonín Komenda, and Daniel L. Kovacs. 2016. Competition of Distributed and Multiagent Planners (CoDMAP). In *Proceedings of the 30th AAAI Conference on Artificial Intelligence.* AAAI Press, Phoenix, Arizona, USA, 4343–4345. https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11905