Research Workshop of the
Israel Science Foundation

Proceedings of the

# Competition of Distributed and Multi-Agent Planners (CoDMAP-15)

http://agents.fel.cvut.cz/codmap/

Edited By:

**Michal Štolba, Antonín Komenda and Daniel L. Kovacs**

Jerusalem, Israel 7/6/2015

## Organizers

**Michal Štolba**
Czech Technical University in Prague, Czech Republic

**Antonín Komenda**
Czech Technical University in Prague, Czech Republic

**Daniel L. Kovacs**
Budapest University of Technology and Economics, Hungary

## Foreword

The 3rd Workshop on Distributed and Multi-Agent Planning (DMAP) at the ICAPS 2015 conference was accompanied by a multi-agent planning competition, CoDMAP (Competition of Distributed and Multi-Agent Planners). The competition was meant to be a preliminary version of a possible future DMAP track at the International Planning Competition (IPC) and focused on comparing domain-independent, offline planners for multiple cooperative agents. The DMAP problems the planners have to solve during the competition were discrete-time, deterministic and fully observable, except for the introduction of privacy, as described in MA-STRIPS, a multi-agent extension of the STRIPS model. To cope with multiple agents and privacy, the base language of deterministic IPC, PDDL3.1, and its multi-agent extension, MA-PDDL, were extended with privacy in order to serve as a base language of CoDMAP. For maximal openness, CoDMAP consisted of two tracks: a centralized highly compatible "transitional" track for various multi-agent planners and an "experimental" proper track for distributed multi-agent planners. Altogether 9 planners were submitted to the competition and 3 of them were apt to compete in both tracks, the rest competed only in the centralized track.

We thank the authors of the planners for their effort and willingness to prepare and submit their planners to the competition. We also thank the chairs of the ICAPS conference for their continuous support throughout organization of the competition.

– Michal Štolba, Antonín Komenda, Daniel L. Kovacs
CoDMAP Organizers

## Acknowledgments

This competition was partially supported by the Czech Science Foundation (grant no. 15-20433Y), Israel Science Foundation and the International Conference on Automated Planning & Scheduling 2015. We also thank to Peter Benda from FEE, CTU in Prague for IT support.

## Table of Contents

Centralized Track

MAPR and CMAP ........ 1
*Daniel Borrajo and Susana Fernández*

ADP an Agent Decomposition Planner CoDMAP 2015 ........ 4
*Matthew Crosby*

MAPlan ........ 8
*Daniel Fišer, Michal Štolba and Antonín Komenda*

PMR: Plan Merging by Reuse ........ 11
*Nerea Luis and Daniel Borrajo*

MAP-LAPKT: Omnipotent Multi-Agent Planning via Compilation to Classical Planning ........ 14
*Christian Muise, Nir Lipovetzky and Miquel Ramirez*

A First Multi-agent Planner for Required Cooperation (MARC) ........ 17
*Sarath Sreedharan, Yu Zhang and Subbarao Kambhampati*

MADLA: Planning with Distributed and Local Search ........ 21
*Michal Štolba and Antonín Komenda*

MH-FMAP: Alternating Global Heuristics in Multi-Agent Planning........ 25
*Alejandro Torreño, Óscar Sapena, Eva Onaindia*

PSM-based Planners Description for CoDMAP 2015 Competition........ 29
*Jan Tožička, Jan Jakubův and Antonín Komenda*

Distributed Track

MAPlan ........ 8
*Daniel Fišer, Michal Štolba and Antonín Komenda*

MH-FMAP: Alternating Global Heuristics in Multi-Agent Planning........ 25
*Alejandro Torreño, Óscar Sapena, Eva Onaindia*

PSM-based Planners Description for CoDMAP 2015 Competition........ 29
*Jan Tožička, Jan Jakubův and Antonín Komenda*

Author Index ........ 33

## Additional References

Detailed description of the competition can be found in [1, 2]. Results of the competition are available at [3].

[1] Michal Štolba, Antonín Komenda and Daniel L. Kovacs, Competition of Distributed and Multiagent Planners (CoDMAP). In *ICAPS Proceedings of 4th Workshop on The International Planning Competition (WIPC-15)*, pp. 24–28. 2015.
[2] Online: *http://agents.fel.cvut.cz/codmap/*
[3] Online: *http://agents.fel.cvut.cz/codmap/results/*

# MAPR and CMAP

**Daniel Borrajo** and **Susana Fernández**
Universidad Carlos III de Madrid
Av. Universidad, 30
28911 Leganés, Spain
dborrajo@ia.uc3m.es;sfarregu@inf.uc3m.es

## Abstract

We have developed a Multi-Agent Planning (MAP) framework to solve classical planning tasks for multiple cooperative agents with private information. It includes two new fully configurable MAP algorithms. In particular, we present to the CoDMAP competition three different configurations whose objectives are: minimize planning time; maximize quality of plans; and maximize the privacy level among agents. The main motivation for these systems is to be able to use any current state-of-the-art planner as the baseline problem solver; that is, our approach is planner-independent. Therefore, we can automatically benefit from advances in state-of-the-art planning techniques. They also avoid the extensive communication effort of other approaches.

## Introduction

We have developed a MAP framework that can be configured in many different ways. First, we can choose between two MAP algorithms: MAPR and CMAP. Second, they can use several methods for assigning public goals to agents. Finally, we can select the underlying planner to be used. In this paper, we describe briefly all these options and at the end we report the actual submitted planners. This work has been partially reported in (Borrajo 2013b; 2013a).

The paper is structured as follows. Section  describes MAPR and CMAP, and Section  details the configuration that participated in the competition.

## Multi-Agent Planning in our Framework

We have devised two alternative MAP algorithms: MAPR and CMAP. MAPR stands for Multi-Agent Planning by plan Reuse, while CMAP stands for Centralized MAP.

### MAP Algorithms

The main steps of the algorithms are (for further details, we refer the reader to (Borrajo 2013b)):

1. Compilation of MA-PDDL into PDDL. It translates the input MA-PDDL format of CoDMAP into a PDDL file. As a side effect it extracts the agents' types and private predicates
2. Generation of $m$ obfuscated PDDL domain and problem files (one for each agent $\phi_i$)
3. Assignment of public goals to agents, while each agent might also have an additional set of private goals. As a side effect, a subset of agents are selected to plan for.
4. Planning using a state-of-the-art planner.
   - MAPR
     - It calls the first agent to provide a solution (plan) that takes into account its private and public goals.
     - Then, it iteratively calls each agent with the solutions provided by previous agents, augmented with domain and problem components needed to reproduce the solution (goals, literals from the state and actions). Each agent receives its own goals plus the goals of the previous agents. Thus, each agent solves its own problem, but taking into account the previous agents' augmented solutions.
     - Since previous solutions might consider private information, all private information from an agent is obfuscated for the next ones.
   - CMAP
     - It merges all the domains of the selected agents in step 3 into a single domain file and all the problem files into a single problem file. In both files, private information of each agent has been obfuscated.
     - It calls any single agent standard planner to solve the new problem

### Goal Assignment

For each goal in $g \in G$ and agent in $\phi_i \in \Phi$, MAPR computes a relaxed plan from the initial state of the agent, $I_i$, following the relaxed plan heuristic of FF (Hoffman & Nebel 2001). If the relaxed plan heuristic detects a dead-end, then the cost of $\phi_i$ achieving $g$ is $c(g, \phi) = \infty$. Otherwise, $c(g, \phi) = c(RP)$ where $c(RP)$ is the number of actions in the relaxed plan. All these costs define a cost matrix, $c(G, \Phi)$. We have devised eight goals assignment strategies: `all-achievable`, `rest-achievable`, `best-cost`, `load-balance`, `contract-net`, `all`, `subset` and `subgoals`. Next, we only describe the ones used in the configurations participating in the competition.

- `rest-achievable`: MAPR first assigns to the first agent $\phi_1$ all goals that it can reach (cost less than $\infty$). Then, it removes those goals from the goals set, and assigns to the second agent all goals that it can reach from the remaining set of goals. It continues until the goals set is empty.
- `subset`: for each public goal $g_i \in G$, it computes the relaxed plan. However, instead of just computing the cost, it computes the subset of agents that appear in the relaxed plan for that goal, $\Phi_i \subseteq \Phi$. Those are agents that could potentially be needed to achieve the goal $g_i$. It is computed as the subset of agents that appear as arguments of any action in the relaxed plan. Then, it assigns $g_i$ to all agents in $\Phi_i$.
- `subgoals`: it is defined for MAPR only. The other assignment strategies assume that for each goal, there exists at least one agent that can achieve the goal by itself. This is not true in domains as logistics when agents are considered to be trucks and airplanes. In that case if an object has to move from the post-office of one city to the post-office of another city, none of the assignment strategies would allow MAPR to solve the goal individually for a given agent. Thus, we have devised a method that is inspired in (Maliah, Shani, & Stern 2014; Crosby, Rovatsos, & Petrick 2013). During goal assignment, a goal policy is computed. The goal policy is an ordered list of agent and subgoals that it needs to achieve at a given time step. At planning time, MAPR considers one policy step at a time, forcing the corresponding agent to achieve the goals specified at that policy step.

## Obfuscation

In MAPR, if the first agent solves the problem, then it cannot pass the private information directly to the rest of agents. So, it obfuscates the private parts and outputs an augmented obfuscated planning problem. We have implemented different obfuscation levels. First, a *simple obfuscation* consists of a random substitution $\sigma$ for the names of all private predicates, actions and objects. This is done when the individual domain and problem files are generated at the beginning. Second, a *further obfuscation* removes arguments from literals and also removes static predicates. Third, MAPR can learn and share macro-operators, so that privacy-preserving is further augmented by not sharing the individual private actions (either between two public actions, or all). We have defined two alternative methods: only-one-macro that generates one macro-operator for the complete plan of each agent; and several-macros that generates several macro-operators.

## Plans Parallelization

We have implemented an algorithm to transform a sequential plan into a parallel plan. First, a suboptimal algorithm generates a partially-ordered plan from a totally-ordered one by using a similar algorithm by Veloso et al. (Veloso, Pérez, & Carbonell 1990). Then, a parallel plan is extracted from the partially-ordered plan. The parallelization algorithm is planner independent. It receives two inputs: a planning task, $\Pi$, and a sequential plan, $\pi$, that solves the task. It outputs a parallel plan that is one of the potential parallelizations of the sequential plan. This helps improving makespan, since in MAP that is a useful criteria for optimizing.

Table 1: Summary of properties for MAPR and CMAP.

| Planner | sound | complete | optimal | privacy |
|---|---|---|---|---|
| MAPR | $\surd$ | X | X | strong$^1$ |
| CMAP | $\surd$ | $\surd^2$ | $\surd^3$ | weak |

## Properties

Table 1 shows a summary of the properties of MAPR and CMAP. The notes mean further constraints are needed:

- 1: strong privacy is achieved if only-one-macro is used. Otherwise, the privacy level is medium if more than one macro is used.
- 2: if we use `all` goal assignment (which we did not use in the version of the CoDMAP)
- 3: if we use `all` goal assignment (which we did not use in the version of the CoDMAP) and an optimal planner (which we did not use in the version of the CoDMAP)

## Participating Configurations in CoDMAP

As the base planner of both MAPR and CMAP, we have used Fast-Downward code (Helmert 2006) to build a simplified version of LAMA-2011. It only performs the first greedy best-first search with the FF and LM-COUNT heuristics and preferred operators. Regarding cost models, we have used unit cost (all actions have a cost of one); we have named it LAMA-UNIT-COST.

We have submitted three configurations of our system:

- *planner1*: CMAP algorithm with the `subset` goal-assignment strategy and LAMA-UNIT-COST as the base planner. It aims at optimizing planning time and coverage.
- *planner2*: CMAP algorithm with the `subset` goal-assignment strategy and LAMA-2011 as the base planner. It aims at optimizing plans' quality and coverage.
- *planner3*: MAPR algorithm with the `subgoals` goal-assignment strategy, the `min-goals` sort-agent scheme, LAMA-UNIT-COST as the base planner and learning only-one-macro. It aims at maximizing privacy among agents.

## Acknowledgments

This work has been partially supported by the Spanish government through MICINN project TIN2011-27652-C03-02.

## References

Borrajo, D. 2013a. Multi-agent planning by plan reuse. Extended abstract. In *Proceedings of the AAMAS'13*, 1141–1142.

Borrajo, D. 2013b. Plan sharing for multi-agent planning. In *Preprints of the ICAPS'13 DMAP Workshop on Distributed and Multi-Agent Planning*, 57–65.

Crosby, M.; Rovatsos, M.; and Petrick, R. P. A. 2013. Automated agent decomposition for classical planning. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of ICAPS'13*. AAAI.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffman, J., and Nebel, B. 2001. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Maliah, S.; Shani, G.; and Stern, R. 2014. Privacy preserving landmark detection. In *Proceedings of ECAI'14*, 597–602.

Veloso, M. M.; Pérez, M. A.; and Carbonell, J. G. 1990. Nonlinear planning with parallel resource allocation. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 207–212. San Diego, CA: Morgan Kaufmann.

# ADP an Agent Decomposition Planner CoDMAP 2015

**Matthew Crosby**
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
m.crosby@ed.ac.uk

**Abstract**

ADP (an Agent Decomposition-based Planner) is designed to deduce agent decompositions from standard PDDL-encoded planning problems and then to exploit such found decompositions to generate a heuristic used for efficient planning. The decomposition process partitions the problem into an environment and a number of agents which act on and influence the environment, but can not (directly) effect each other. The heuristic calculation is an adaptation of the FF relaxation heuristic to incorporate multiagent information. Relaxed planning graphs are only ever generated for single-agent subproblems. However, when cooperation is necessary, an agent's starting state may include facts added by others.

## Introduction

ADP is a complete, satisficing (non-optimal) centralised planning algorithm that attempts to compute and utilise agent decompositions for the sole purpose of improving planning time. As such, it does not take into account common multiagent concerns such as privacy, trust or strategic considerations. It has been shown (Crosby, Rovatsos, and Petrick 2013; Crosby 2014) that useful decompositions can be found and successfully utilised in around forty percent of IPC domains (IPC 2011), a collection of domains which are not explicitly designed to be multiagent, yet contain some obviously multiagent settings.

The first section of this paper provides a brief high-level overview of the ADP algorithm, while the following section provides more detail including explicit discussion of the decomposition process and heuristic calculation. The third section presents a summary of the agent decomposition results for the domains used in CoDMAP 2015 after which some limitations and future plans for ADP are discussed. Technical details of the planner and other information relevant to CoDMAP 2015 can be found in the final section.

## ADP Overview

ADP is split into two components, a decomposition phase and a heuristic calculation. The decomposition phase processes the planning problem and attempts to find a useful

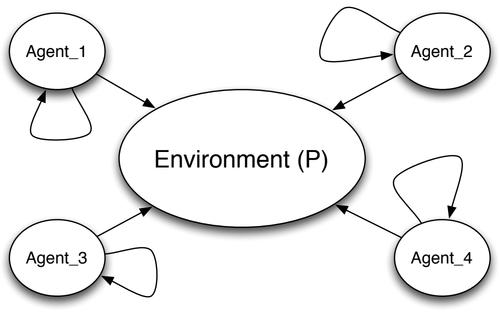


Figure 1: A depiction of an agent decomposition of the variables in a domain. Agents can only change the state of themselves and/or the environment.

multiagent decomposition. As depicted in Figure 1, a decomposed domain is made up of a number of agents, each with an internal state, and an environment that the agents are acting upon. Actions that can influence or depend upon the internal state of an agent are assigned to only that agent. Actions that only involve the environment are assigned to all agents. The decomposition algorithm is guaranteed to return a decomposition in which no actions that influence (or rely upon) multiple agents' internal states exist. This means that agents can only influence themselves or the environment (thought the actions available to an agent in a given state may depend on how others have influenced the environment).

When a decomposition is found, ADP calculates a heuristic value (used to guide a greedy best first search) that attempts to exploit the multiagent structure to find plans faster than the alternative single-agent approach. If no suitable decomposition can be found, then ADP defaults to the the single-agent FF heuristic (Hoffmann and Nebel 2001).

The main idea behind the multiagent heuristic calculation is to only ever generate planning graphs for a single agent subproblem at a time. In 'coordination points' each agent computes its heuristic value for each goal proposition by generating its relaxed planning graph from the current state. Cooperation is achieved (where necessary) by combining all individually reachable states and using this as input for successive rounds of individual relaxed planning graph generation. As a result of this process, a coordination point is as-

**Algorithm 1:** High-level Overview of ADP

```
Input  : MPT $\langle V, I, G, A\rangle$
Output: Plan or $\perp$
1 Calculate Agent Decomposition $\Phi$
2 $S \leftarrow I$
3 CoordPoint($S$) [initialises $S.agent$, $S.goals$ and $S.macro$]
4 Greedy BFS from S using h_adp heuristic [When the successor of S is generated it copies $S.agent$, $S.goals$ and $S.macro$ from its predecessor]
```

**Algorithm 2:** Decomposition Algorithm

```
Input  : MPT $\langle V, I, G, A\rangle$, Causal Graph $CG$, $\Phi = \{\}$
Output: Agent Decomposition $\Phi = \{\phi_1, \dots, \phi_n\}$
1 $\Phi \leftarrow \{\{v\} : v \in V \wedge$ v root node of CG$\setminus$2-way cycles$\}$
2 repeat
3     foreach $\phi_i \in \Phi$ do
4         $\phi_i \leftarrow \phi_i \cup \{v \in V : v$ only successor of $\cup \phi_i\}$
5     $\Phi \leftarrow \Phi$ where agents sharing joint actions are combined
6 until $\Phi$ can no longer be refined
```

signed an agent (the one with the most individually achievable goals), a set of agent goals (all propositions the assigned agent can achieve alone along with all propositions necessary for other agents to achieve goals requiring cooperation) and a macro heuristic value that estimates the coarse distance to the goal. At non-coordination points, the heuristic value is only updated by the currently assigned agent's progress towards its currently assigned goals.

## ADP Details

Algorithm 1 gives a high-level pseudo-code overview of ADP. The algorithm takes a multi-valued planning task (MPT) calculated by the Fast-Downward Planning System (Helmert 2006) as input and consists of an initial preprocessing decomposition phase, followed by greedy best-first search using the ADP heursitic.

An MPT is represented by $\Pi = \langle V, I, G, A\rangle$, where:

- V is a finite set of state variables $v$, each with an associated finite domain $D_v$,
- $I$ is a state over $V$ called the initial state,
- $G$ is a partial variable assignment over $V$ called the goal, and
- $A$ is a finite set of (MPT) actions over $V$.

In what follow we assume the standard definition of preconditions *pre(a)* and effects *eff(a)*.

### Decomposition

The decomposition algorithm creates a partitioning of the variable set $V$ into variable subsets $\phi_i$ for each agent $i$, which leaves a public variable set $P$ that contains the variables that pertain to the environment. An overview of the decomposition algorithm is shown in Algorithm 2.

**Algorithm 3:** h_adp Calculation

```
Input  : State $S$ with $S.agent$, $S.goals$ and $S.macro$
Output: $h\_adp$
1 $S.micro \leftarrow hff(V|_{S.agent}, S|_{S.agent}, G|_{S.goals}, A|_{S.agent})$
2 if $S.micro$ == 0 or deadend then
3     CoordPoint($S$)
4     $S.micro \leftarrow hff(V|_{S.agent}, S|_{S.agent}, G|_{S.goals}, A|_{S.agent})$
5 h_adp = S.macro + S.micro
```

First, a list of potential agents is found from the causal graph by taking every root node that exists in the graph once all two-way cycles have been removed. These root nodes must still have at least one successor node to be considered.

In the next step, the potential decomposition is refined by first extending agent sets to be as large as possible and then reducing the number of sets based on the existence of any joint actions. Agent sets are extended by recursively adding any node of the causal graph that is a successor of only variables already included in that agent set.

Actions in the domain can be categorised based on potential decompositions (partitions). An action is said to be *internal* to agent $i$ for a given decomposition $\Phi = \{\phi_1, \dots, \phi_n\}$ iff: $\exists v \in pre(a) : v \in \phi_i$, and$v \in pre(a) \rightarrow v \in \phi_i \cup P$. In other words, the preconditions of $a$ must depend on an internal state variable of $i$ and can only change $i$'s internal state variables or the domain's public variables.

A *public action* is any action where: $v \in pre(a) \rightarrow v \in P$, i.e., where the preconditions do not depend on the internal state of any of the agents. An agent's action set is the set of all internal actions of that agent, denoted by $Act_i$.

An action is *joint* between agents $i$ and $j$ given decomposition $\Phi = \{phi_1, \dots, \phi_n\}$ iff. $\exists v \in pre(a) : v \in \phi_i$, and$\exists v \in pre(a) : v \in \phi_j$. An action can be joint between multiple agents. In the second stage of the algorithm all agents involved in any joint actions are merged.

In a decomposition returned by ADP, the sets $\phi_i$ are guaranteed to have the *agent property*. In particular, a variable set $\phi_i$ (as part of a full decomposition $\Phi$) has the agent property when for all $a \in A$ and variables $v \in V$:

$$v \in \phi_i \wedge v \in eff(a) \rightarrow a \in Act_i.$$

In other words, any agent variable can only be modified by an action of that agent. The proof of this can be found in (Crosby 2014).

### Heuristic Calculation

The h_adp heuristic calculation is formed of two parts as shown in Algorithm 3. There is a global *coordination point* calculation that is performed infrequently and is used to pick out a single agent and a set of goals for that agent to attempt to achieve. There is also a micro single-agent heuristic calculation that is identical to that used by FF (Hoffmann and Nebel 2001) on the planning problem restricted to the current chosen agent and its current set of goals and subgoals.

**Algorithm 4:** Coordination Point Calculation

```
  Input   : State $S$
  Output: $S.agent, S.goals$ and $S.macro$
1 Iterated Relaxed Planning Graph Generation
2 if $Max\ layer > 0$ then
3 |   Calculate Subgoals
4 Assign Goals
  $S.agent \leftarrow$ agent with most goals with min $h\_add$
  $S.goals \leftarrow$ all goals achievable by $S.agent$
  $S.macro \leftarrow N \times |G \setminus S.goals|$
```

**Coordination Point Calculation** An overview of the coordination point calculation is shown in Algorithm 4. It associates a chosen agent, a goal set and a macro heuristic value with the current state. This extra state information is carried over whenever a successor state is generated and (re)calculated whenever the current agent's goal set is completed or becomes impossible.

**Iterated Relaxed Planning Graph Generation**. Given a state, each agent generates their full relaxed planning graph for their restricted problem from that state. That is, each agent uses an iterative process to create a graph containing all possible actions it can perform (ignoring delete effects) and all possible propositions that can be reached by performing those actions.

It may happen that some propositions can only be reached if agents cooperate. For example, one agent may need to unlock a door before another can pass through. Because the agents create their own planning graphs (using only their own subproblems) they will not include any parts of the search space only reachable by cooperation. If not all goals are reachable by at least one agent after the first round of relaxed planning graph generation, the collected final state of all the agents is formed and used as input for a subsequent *layer* of relaxed planning graphs. Each successive iteration introduces a new *layer* with the first being *layer* 0. Repeating this process until no more states are added by any agent is guaranteed to cover every reachable state in the full (not decomposed) problem.

**Calculate Subgoals:** Any time a goal proposition appears for the first time in a layer above 0 this means that it cannot be reached by an agent on its own. In this case, subgoals are calculated. Plan extraction is used to find out which propositions are necessary from a previous layer in order to achieve each goal. These propositions are called subgoals. All subgoals from layer 0 are added as to the agent that achieved them first. Using the door example, if the second agent needs to pass through the door to achieve a goal, the subgoal of unlocking the door will be assigned to the first agent.

**Assign Goals:** The next part of the coordination point calculation chooses which agent is going to be performing the local search. First, each goal is assigned to the agent that can achieve it with the lowest estimated cost from the relaxed planning graphs. That is, it is assigned to the agent that added it with the lowest $h\_add$ value (Hoffmann and Nebel 2001). An agents goal set is then formed of all goals and subgoals assigned to it. The agent with the largest goal set is chosen as $S.agent$ along with all of its goals (and any subgoals that it may have been assigned).

As a final part of the coordination point calculation the value $S.macro$ is calculated. This is used to provide a global heuristic estimate of the distance through the overall search space. This is calculated as $N \times |G \setminus S.goals|$ where $N$ is some large number chosen such that it dominates the single-agent FF heuristic value of a state.

## Decomposition of CODMAP Domains

This section discusses the decompositions that ADP finds for the competitions domains used in CODMAP 2015. The results are shown in Table 1. The second column of the table 'Usable' reports whether or not ADP managed to find a usable decomposition for the domain. ADP found a suitable decomposition for nine of the twelve domains in the competition, failing to find one in blocksworld, driverlog and sokoban. This does not mean that no sensible decomposition exists for this domain, but that ADP could not find one that respects the *agent property* and contains no joint actions.

In general, decompositions returned by ADP are consistent across all problem instances for each particular domain. The one exception is Woodworking for which there was at least one problem instances in which ADP could not find a decomposition (represented by an asterisk in the table).

The third column of the table 'Joint' shows whether or not ADP found a decomposition that includes joint actions. The only domain for which this differs is Driverlog for which a decomposition was found (drivers+trucks) but was not used. There is no theoretical reason why the APD heuristic cannot be applied when joint actions exist, however in such domains, the algorithm tends to perform very poorly and much worse than if no decomposition is returned. Note that for some domains a decomposition including joint actions was found part-way through the decomposition algorithm, but the final decomposition did not include any after agents were merged.

The final three columns of the table compare the predefined decompositions for the problems to the decomposition that ADP finds. ADP found the same decomposition in exactly half of the ten remaining domains. In Driverlog and Taxis, ADP found a very similar decomposition including the trucks as well as the drivers in Driverlog and only including the taxis and not the passengers in Taxis. In Depot, ADP finds a completely different decomposition which treats the trucks as agents and also contains a separate agent that includes every single crate in the domain. The two remaining domains Woodworking and Wireless return fairly odd looking decompositions and it is expected that ADP will perform poorly on these domains.

ADP also found some extended agent sets not reported in the table. For example, in satellites the decomposition found by ADP includes the variables representing the state of the instruments' of each satellite with each individual satellite. In the Zenotravel domain the agent sets include the fuel levels for each plane.

| Domain | Usable | Joint | Predefined Decomp | ADP Decomp | Match |
|---|---|---|---|---|---|
| blocksworld | ✗ | ✗ | agents | na | – |
| depot | ✓ | ✓ | drivers + dists + depots | trucks + crates* | ✗ |
| driverlog | ✗ | ✓ | drivers | drivers + trucks | ✗ |
| elevators | ✓ | ✓ | lifts(fast/slow) | lifts(fast/slow) | ✓ |
| logistics | ✓ | ✓ | trucks + airplanes | trucks + airplanes | ✓ |
| rovers | ✓ | ✓ | rovers | rovers | ✓ |
| satellites | ✓ | ✓ | satellites | satellites | ✓ |
| sokoban | ✗ | ✗ | players | na | – |
| taxi | ✓ | ✓ | taxis + passengers | taxis | ✗ |
| wireless | ✓ | ✓ | nodes + bases | messages by node + messages by base | ✗ |
| woodworking | ✓(*) | ✓ | different tools | boards(available) + saw | ✗ |
| zenotravel | ✓ | ✓ | planes | planes | ✓ |

Table 1: The decompositions found by ADP on the CODMAP problem domains.

## Limitations and Future Work

This section briefly states some of the current limitations of ADP and current plans for improvement and extension of the planner in the future. The decomposition algorithm is known to return a decomposition with the agent property but it is currently unknown if it will always find such a decomposition if one exists. Further theoretical work and experimentation with different possible decomposition definitions is planned along with the release of a a standalone decomposer that can be used to find decompositions of PDDL-encoded planning problems.

The ADP heuristic calculation is based on the FF heuristic. However, there are a large number of other successful planning heuristics that have since been developed and it is likely that the overall multiagent approach can be applied to some of these techniques. It will also be interesting to extend ADP to include reasoning about action costs, numeric fluents and other extensions of PDDL or to include more multiagent aspects of the planning problems.

Finally, ADP is currently implemented as a single-threaded process. As each agent is always acting only working on their own internal problem, there is clearly scope for a multi-threaded version in which agents explore the search space independently.

## ADP Details Summary

This final section presents a summary of the details of ADP relevant to CoDMAP 2015. ADP is a complete, satisficing (non-optimal) centralised single-threaded planning algorithm. ADP was implemented as a heuristic plug-in for the Fast-Downward planning system (Helmert 2006). The preprocessing of the planning problem into an MPT and the search algorithm are left unchanged. ADP simply calculates a decomposition and provides a heuristic value for each state that is queried during search and also stores a macro-heuristic value, agent, and goalset for each state. ADP is called with the option cost_type=2 and using the lazy_greedy search algorithm. Source code for ADP can be found online at the authors homepage.

ADP ignores the agent factorization presented in the MA-PDDL files, instead determining the agents present (if any) itself. The private/public separation is therefore also ignored. ADP does have an internal representation for private and public facts and actions used for decomposition calculation but does not use this directly for search.

Sometimes ADP will not be able to find an agent decomposition (defaulting to single-agent planning behaviour) or find a different decomposition as to that specified in the CoDMAP files. The details of these cases are explained in an earlier section of this paper.

Two versions of ADP were submitted to the CodMAP planning competition. Planner1 is the ADP implementation described in this paper. Planner2 is a legacy version of the code that is functionally identical except that instead of storing the macro-heuristic value and agent assignment, it (incorrectly) assumes that this can be carried over from the previously searched state. This legacy version was entered out of curiosity and the fact that it has produced some interesting results with some recent work in the planning community showing the possible value of including some element of randomness (essentially what the legacy version does) in the search process.

## References

Crosby, M.; Rovatsos, M.; and Petrick, R. P. A. 2013. Automated Agent Decomposition for Classical Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 46–54.

Crosby, M. 2014. *Multiagent Classical Planning*. Ph.D. Dissertation, University of Edinburgh.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

IPC. 2011. http://www.plg.inf.uc3m.es/ipc2011-deterministic/. Web Site.

# MAPlan

**Daniel Fišer** and **Michal Štolba** and **Antonín Komenda**
Department of Computer Science, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic
*danfis@danfis.cz*, *{stolba,komenda}@agents.fel.cvut.cz*

## Abstract

In this paper, we present a brand new multi-agent planner called MAPlan. The planner implements state space heuristic search on factorized problems retrieved from either unfactored or factored version of MA-PDDL files. It can run both in a multi-thread and distributed configuration with a communication over network or within a process. This paper briefly describes the details of the MAPlan configurations used in the 2015 CoDMAP competition.

## Introduction

MAPlan is a multi-agent planner implementing multi-threaded as well as distributed state space heuristic search. The MAPlan planner further expands on the ideas introduced in the MAD-A* planner (Nissim and Brafman 2012). The basic search scheme is following. All operators retrieved from the SAS$^+$ representation of the problem are divided between individual agents so that each operator belongs to only one agent. Each agent expands the state space only by its own operators, but whenever a public operator is used, the resulting state (public state) is sent to all other agents. As the public operators are considered those that have some common facts in their preconditions or effects with operators owned by other agents. So the planner can explore the state space with its own operators without communication with any other agent if the results of the applied operators cannot influence others, but the states useful to other agents are shared among all agents.

Moreover, an agent can have access to the projected operators if it is needed for a computation of heuristics. The projected operator is an image of the public operator owned by the other agent that preserves only those preconditions and effects that are already known to the agent. In other words, each agent is aware only of those facts that are in the preconditions and effects of its own operators and the projected operators are images of other agents' public operators with deleted facts that the agent cannot understand.

The planner is implemented in C and is able to run a multi-threaded as well as distributed search. The communication between agents can be inner process in case of multi-threaded search or over TCP/IP protocol. Both factored and unfactored versions of MA-PDDL files can be used as input problem definitions. In the following three sections, the heuristics used in the configurations submitted to the 2015 CoDMAP competition are described and more details about the planner specific to the centralized and the distributed track are provided.

## Heuristics

MAPlan planner uses altogether four different heuristics in configurations submitted to 2015 CoDMAP competition. The first one is the LM-cut heuristic (Bonet and Helmert 2010). The heuristic runs on projected operators that are extracted either from unfactored or factored version of MA-PDDL domain definitions.

The second one is the distributed version of LM-cut heuristic (Štolba, Fišer, and Komenda 2015a) which was designed to provide provably equal estimates as the centralized version of LM-cut heuristic on all operators not only the projected ones. This property comes with a cost of increased computational burden causing less expanded states per time unit. Nevertheless, the heuristic provides more accurate estimates than LM-cut on projected operators which should compensate the disadvantage of a slower state expansion. Both versions of LM-cut heuristics are admissible so they are used in the optimal planning track.

The next one is the distributed Fast-Forward (FF) heuristic (Štolba and Komenda 2014), specifically the lazy variant. The computation of a heuristic estimate starts with an exploration of the relaxed problem on projected operators. The following step is an extraction of the relaxed plan which is performed in a distributed manner. Once a projected operator is reached during the extraction, the owner of the operator is asked to provide its local part of the relaxed plan leading to that operator. If the local part of the plan contains a projected operator, the path to that operator must be extracted too. This would lead to a distributed recursion. The implementation used in MAPlan avoids a distributed recursion by collecting the local parts of the relaxed plan by the initial agent and requesting the other agents consecutively until all projected operators are not solved.

The last heuristic function used in CoDMAP competition is the distributed version of FF heuristic based on Domain Transition Graphs (DTGs) (Štolba, Fišer, and Komenda 2015b). This heuristic is based on an exploration of agents' local DTGs constructed from projected operators. The unknown (or more precisely possibly unknown) preconditions and effects of projected operators are recorded into DTGs with a special symbol. Once that symbol is reached during extraction of the relaxed plan, a distributed recursion is executed and the cost of the relaxed plan is computed with the help of the owners of the projected operators. Moreover, partial plans for each fact can be cached and later reused without re-computation of the full estimate. It makes this heuristic very fast even though it is built upon a distributed recursion.

## Centralized Track

MAPlan configuration for the centralized track uses the unfactored version of MA-PDDL definition files. A factorization of a problem to particular agents is made as suggested in the MA-PDDL files. The definition files are translated to $SAS^+$ representation by a slightly modified translator from fast-downward planner (Helmert 2006). The factorization itself is made by MAPlan from $SAS^+$ representation before any agent is started and it is based on splitting the operators according to an emergence of particular agents' names in parameters of operators. Privateness of facts and operators is inferred also from $SAS^+$ representation. The facts that are stated only in preconditions or effects of operators of only a single agent are considered as private and the operators that have only private preconditions and effects are considered as private. Similarly, projected operators are created simply by omitting the private facts from preconditions and effects.

Since the translation from MA-PDDL is performed in a centralized manner before any agent is started, all agents share the same representation of a state. This considerably simplifies a communication of public states between agents because each agent can directly use the received state without any additional processing. The disadvantage of this approach, of course, resides in insufficient preservation of private information because private parts of public states are freely communicated between agents. This could be easily solved if the private parts would be somehow scrambled when transmitted and the receiving agent would be able to unscramble its own private parts. Nevertheless, even though this (or any similar) mechanism is not implemented in the planner, each agent "understands" only its own private facts and does not use any private information of other agents in any way. So the privateness is at least preserved in this way. In fact, we consider this problem only as an implementation detail, but of course this property should be considered in a comparison with other planners.

In centralized track, each agent runs in a separate thread and communication channels between agents are created within a common process context. Two configurations for optimal planning are submitted, both using $A^*$ search algorithm – one with the projected LM-cut heuristic and one with the distributed LM-cut heuristic. One configuration for satisficing planning with best-first search algorithm is submitted where both inadmissible heuristics, distributed FF and distributed DTG-based FF, are used, each for a half of the maximum allowed time. All planners are complete.

## Distributed Track

For the distributed track, the factored version of MA-PDDL files is used. In contrast to the centralized track, the translation to $SAS^+$ has to be done separately by each agent from its own MA-PDDL factor. As in the previous case, the translation is done by the translate tool from fast-downward planner but this time it had to be modified more than slightly. The translation to $SAS^+$ has to be distributed over all agents because particular factors are available only to the corresponding agents and it is not possible for a single agent to perform concise grounding of the problem only from its own factor.

The translation consists of two main phases. In the first phase, the concise grounding of the problem is made by a coordinate effort of all agents that communicate in a ring, i.e., an absolute ordering of agents must be provided and each agent sends messages only to the agent that is next in the ordering (and the last agent sends messages to the first one in ring). This way, the messages circle around the established ring of agents. The grounding starts with the first agent in the ring, which uses a Datalog program (Helmert 2009) implemented in fast-downward's translate tool for grounding of its local problem. All public facts that are returned by the Datalog program are sent to the next agent. The next agent adds the received public facts to the initial state and continues with the same procedure. It runs the Datalog program and the public facts from its output sends to the next agent. This whole procedure continues until the first agent in the ring does not receive the same public facts that it already transmitted. In this moment all agents know all grounded facts that are public.

In the second phase, $SAS^+$ variables and their values must be inferred from the public and also private facts. This is done from fact invariants (Helmert 2009). This might be a little bit tricky in this case because each agent can identify different invariants that can even overlap each other. So the invariants viable for all agents are identified via a distributed coordination of agents. The ring of agents that is already established is used and the first agent sends its invariants of the public facts to the next agent. The next agent uses its own invariants to split the received invariants to preserve an invariant property and so on. At the end of this procedure all agents have the same invariants of public facts.

The resulting $SAS^+$ variables are created so that all agents share the exactly same representation of the public part of a state and the private parts differ. In other words, the private facts translate to a separate variables. Although some private facts could share a variable with some public facts because they can be an invariant together, this design considerably simplifies communication of states between agents. This way, all agents share the exactly same representation of the public part of a state but each agent still can privately manage the private variables without communicating it directly to other agents.

The obvious disadvantage of this approach is that each agent must somehow reconstruct the full state with its own private part from the received public state. In MAPlan, this problem is solved by attaching an identification of the full state to the public part that is sent to other agents. The receiving agent must preserve this identification and send it along the next state that is created via expansions from the original received state. Thus sets of state identifications from all agents travel as tokens with all public states that are communicated and the receiving agent can always reconstruct the full state, i.e., find out its private part for the state. The same approach is used for distributed heuristics whenever a state has to be sent to other agent.

The advantage is that the privacy of states is preserved because private information is never transmitted; not even during translation of the problem to SAS$^+$. The only transmitted information linked to the private part of a state is its identification number. Nevertheless, that number has no meaning to any other agent than the one that created it.

The agents in the distributed track run in separate processes and communicate over network via TCP/IP. The configurations are the same as in the case of the centralized track.

## Conclusion

In this paper, a brand new multi-agent planner, called MA-Plan, was introduced. The planner is based on ideas introduced by the MAD-A* planner and it is implemented in C. The planner accepts both factored and unfactored versions of MA-PDDL and it can utilize two different privacy preserving schemes with their different advantages and disadvantages. The planner is complete and it can be used both as an optimal planner and a satisficing planner. It can run in one process as a multi-threaded application or it can be distributed over a network of computers. Although the planner implements a moderate set of heuristics that can be used locally with projected operators, it also contains a rich set of distributed heuristics.

### Acknowledgments

This research was supported by the Czech Science Foundation (grant no. 15-20433Y), the Grant Agency of the CTU in Prague (grant no. SGS14/202/OHK3/3T/13) and USAF EOARD (FA8655-12-1-2096). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the program Projects of Large Infrastructure for Research, Development, and Innovations (LM2010005), is greatly appreciated.

## References

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 329–334. IOS Press.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence* 173(5-6):503–535.

Nissim, R., and Brafman, R. I. 2012. Multi-agent a* for parallel and distributed systems. In *Proc. of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, 1265–1266.

Štolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *Proc. of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, 298–306.

Štolba, M.; Fišer, D.; and Komenda, A. 2015a. Admissible landmark heuristic for multi-agent planning. In *Proc. of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*.

Štolba, M.; Fišer, D.; and Komenda, A. 2015b. Comparison of RPG-based FF and DTG-based FF disrtibuted heuristics. In *Proc. of the 3rd Workshop on Distributed and Multi-Agent Planning (DMAP)*.

# PMR: Plan Merging by Reuse

**Nerea Luis** and **Daniel Borrajo**
Departamento de Informática
Universidad Carlos III de Madrid
nluis@inf.uc3m.es, dborrajo@ia.uc3m.es

**Abstract**

In this paper we describe the planner and the configuration that was submitted to the centralized track of the Competition of Distributed and Multiagent Planners (CoDMAP) 2015. We have submitted the plannner called Plan Merger by Reuse, PMR. Given a multi-agent planning problem, PMR lets the agents build their individual plans separately. Then, it concatenates all the plans to build a parallel plan. As plans are not always valid, specially in the case of tightly-coupled domains, PMR executes a replanner with the invalid plan as input. The replanner performs planning by reuse and is able to generate a sound plan from the input invalid plan.

## PMR

Plan Merging by Reuse is a centralized, single threaded, multi agent planner that combines two different techniques to solve a multi agent planning (MAP) task: plan merging (Foulser, Li, and Yang 1992) and plan reuse (Fox et al. 2006). It receives as input a MAP task, which consists of a domain and a problem to solve.
The PMR algorithm uses three off-the-shelf planners: one for each agent to plan individually (it can be the same planner or a different planner). Another one capable of applying plan reuse, and the third one, only used by PMR in case all agents failed in the planning process (centralized planning).
Given that in CoDMAP the received input is in MA-PDDL, before the algorithm starts to plan, both domain and problem are translated to PDDL. Then, the private information included in them is obfuscated.
The first step of the PMR algorithm is (as shown in Figure 1) to assign goals to agents. When PMR assigns goals to agents, it also decides which agents are going to plan. There is no need for the *n* agents to be involved in the planning process if they do not have goals assigned. After that, the subset of agents (*m*) with assigned goals start to build each plan individually. PMR concatenates all these plans and checks if the concatenated plan is empty. An empty plan is obtained when no agent has been capable of achieving its assigned goals, in which case a centralized planner will be called to solve the MAP task. If the concatenated plan is not empty, PMR checks if it is sound. Then, if the plan is sound, PMR parallelizes it. If not, the invalid concatenated plan will be the input plan for the plan reuse phase, where a replanner will try to find a solution based on the actions of the input plan. Next sections explain in more detail some parts of PMR.

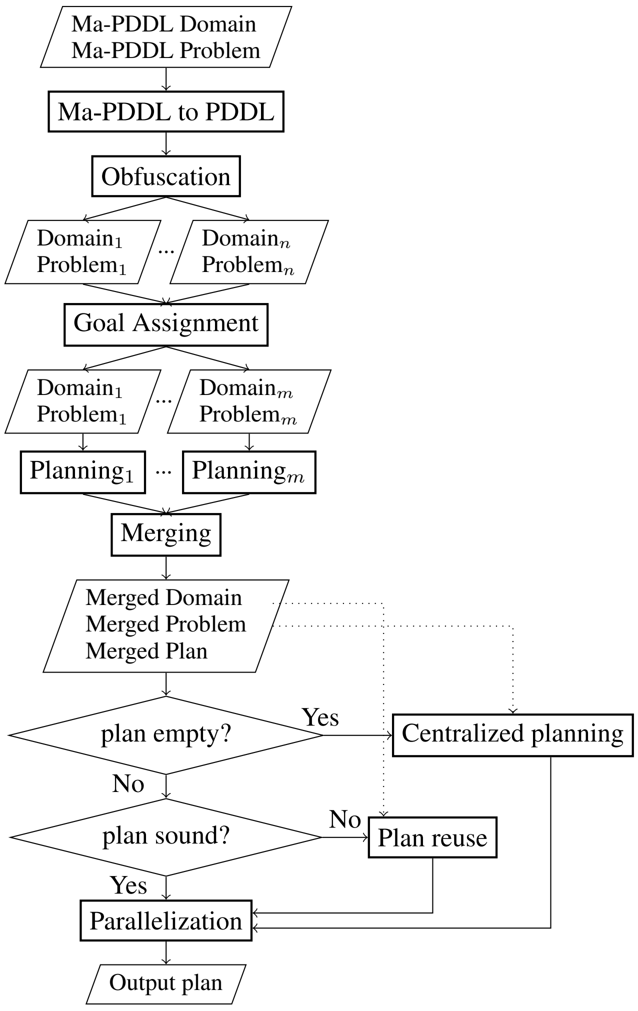


Figure 1: Flow diagram of PMR.

## Goal Assignment

In a MAP task, the way the public goals are assigned to the different agents affects directly the efficiency and the performance of the planner. PMR uses the goal assignment (GA) strategies taken from MAPR (Borrajo 2013). First, for each agent and public goal, a relaxed plan is computed (Hoffmann and Nebel 2001). Thus, the algorithm knows if the goal can be reached from the initial state of the agent and with what estimated cost. Then, as this process is repeated, cost values from the relaxed plans are stored into a matrix. After the matrix is completely filled, a GA strategy is applied in order to assign the public goals to the agents.

## Planning

In the second step of PMR, each agent receives as an input the obfuscated description of its domain and problem (public and private parts joined). As public goals were already assigned, they are included in the problem files. Then, each agent invokes a planner to solve its planning task. Any state-of-the-art planner can be used for this task. In the end, each agent will have a partial solution to the overall MAP task. Then, PMR concatenates all subplans. The concatenation process has three outputs: the merged obfuscated plan, the merged obfuscated domain and the merged obfuscated problem. These two last outputs have been obtained by merging each partial domain and problem that each agent has received as input. At this point, the initial obfuscation for each domain and problem is still preserved in the merged files.

## Plan Reuse

If the merged plan was not sound, we can obtain the benefit of transforming the merged plan into a valid plan; the planner to be used in this step must allow PMR to start the search for a sound plan from an input plan. In case it can solve the planning task and the planner is sound, it will generate a new valid plan.

## Parallelization

When a valid plan is obtained, PMR parallelizes it to obtain a parallel plan. The parallelization of a plan consists on transforming a totally ordered plan into a partial order plan. Thus, more than one agent can execute an action in the same step.

## Properties

PMR performs suboptimal incomplete planning, since we are using suboptimal planners, working separately on subsets of goals and choosing a subset of agents to plan for public goals.

A key issue in MAP is agents privacy. Our obfuscation process consists on replacing the names of private predicates, actions and objects by other random names. At the beginning of PMR, each agent builds its plan without sharing private information, so privacy is preserved. Thus, an obfuscated plan is returned as a result. After that, private information is still preserved because when all plans are merged, also the obfuscated problems and domains generated for each agent are merged into a unique pair of obfuscated domain and problem. These files are used first by the validator to validate the obfuscated merged plan and then by the plan reuse planner in case the obfuscated plan needs to be fixed. When PMR finally obtains a valid plan, it des-obfuscates the information and returns a readable PDDL version of the plan.

## Implementation and Configuration setup

PMR has been entirely developed using Bash and Common LISP. More details on the PMR algorithm can be found in (Luis and Borrajo 2014). The submitted configuration calls MAPR in the distributed phase (Borrajo 2013) and RRPT in the plan by reuse phase. RRPT is an stochastic plan reuse planner that we have developed based on two previous works (Alcázar, Veloso, and Borrajo 2011), (Borrajo and Veloso 2012). If the distributed phase fails and the returned plan is empty, PMR calls LAMA-FIRST (Richter and Westphal 2010) to find a solution using a centralized approach. LAMA-FIRST corresponds to the first solution that LAMA (Richter and Westphal 2010) returns, using greedy best first with costs.

In addition, there are a few parameters set to certain values, which were chosen because of the results obtained in the benchmark of the competition in number of solved problems (coverage), length of the plans (quality) and planning time:

- We decided to assign the public goals of a problem to the agents following the Best-Cost GA strategy. Thus the algorithm assigns each public goal to the agent that can potentially achieve it with the least cost.
- Each agent inside MAPR uses LAMA-FIRST to solve its MAP task.
- PMR also needs VAL (Howey, Long, and Fox 2004) to validate the plan that MAPR returns, in order to choose which part of the algorithm executes after that.

## Acknowledgments

This work has been partially supported by Spanish MICINN project TIN2011-27652-C03-02.

## References

Alcázar, V.; Veloso, M. M.; and Borrajo, D. 2011. Adapting a rapidly-exploring random tree for automated planning. In *SOCS*.

Borrajo, D., and Veloso, M. M. 2012. Probabilistically reusing plans in deterministic planning. In *In Proceedings of ICAPS'12 workshop on Heuristics and Search for Domain Independent Planning*. AAAI Press.

Borrajo, D. 2013. Plan sharing for multi-agent planning. In Nissim, R.; Kovacs, D. L.; and Brafman, R., eds., *Preprints of the ICAPS'13 DMAP Workshop on Distributed and Multi-Agent Planning*, 57–65.

Foulser, D.; Li, M.; and Yang, Q. 1992. Theory and algorithms for plan merging. *Artificial Intelligence* 57(2-3):143–181.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of* *the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS'06)*, 212–221.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In Khoshgoftaar, T. M., ed., *ICTAI 2004: 16th IEEE International Conference on Tools with Artificial Intelligence*, 294–301.

Luis, N., and Borrajo, D. 2014. Plan merging by reuse for multi-agent planning. *3rd Workshop on Distributed and Multi-Agent Planning (DMAP) of ICAPS* 38.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

# MAP-LAPKT: Omnipotent Multi-Agent Planning via Compilation to Classical Planning

**Christian Muise*, Nir Lipovetzky*, Miquel Ramirez†**

*Department of Computing and Information Systems, University of Melbourne
†College of Engineering and Computer Science, Australian National University
*{christian.muise,nir.lipovetzky}@unimelb.edu.au, †miquel.ramirez@anu.edu.au

## Introduction

In this paper we describe three related entries submitted to the CoDMAP planning contest (Stolba, Komenda, and Kovacs 2015). All three entries are configurations of the classical planning framework LAPKT (Ramirez, Lipovetzky, and Muise 2015), and all three use the same pre-compiled input. Our approach is based on the following insight:

*The task of planning for multiple agents with heterogeneous access to fluent observability can be solved by classical planners using the appropriate encoding.*

The general approach is quite simple: we convert the unfactored domain and problem file into a classical planning problem such that the privacy of fluents and objects are respected. The translation is both sound and complete, and we solve the encoded problem using a centralized classical planner. None of the factorization is passed to the classical planner, because the encoded problem contains all the necessary information as part of the problem itself.

In the remainder of the document, we outline (1) the simple encoding that we use to create a classical planning problem, (2) the planning framework that we use to solve the encoded problems, and (3) the configurations that we submitted to the CoDMAP contest.

## Encoding

The model of privacy used for the CoDMAP contest restricts the number of objects and fluents that an agent has access to. Any action that uses a fluent or object (in a precondition or effect) that is not either (1) private to agent $i$ or (2) public to all agents, cannot be executed by agent $i$. That is, every fluent and object mentioned in an action in order for agent $i$ to execute must be known by that agent. Crucially, the privacy of objects and fluents are static, and thus we can use classical planning techniques as long as *any action that violates the privacy restrictions is not allowed to occur*.

We have two options for filtering out any action that violates the multi-agent privacy: (1) modify the planner to use only those actions that adhere to the privacy; and (2) modify the domain description so that any valid grounding respects the privacy. We chose the latter option for our approach.

Every object $o$ and agent $ag$ in the domain has a corresponding fluent (K-obj $ag$ $o$) added. If an action that is executed by agent $ag$ uses object $o$, then a precondition of the action is for (K-obj $ag$ $o$) to hold. Similarly, we add fluents (K-fluent-foo $ag$) for every agent $ag$ and fluent $foo$ in the domain, and update the action preconditions accordingly.

The final step is to translate the privacy model provided in the multi-agent description of the domain (unfactored representation), into the initial state that fully defines which agents have access to which fluents and objects. Everything added to the model is encoded as action preconditions and initial state fluents, and any modern classical planner will strip away these auxiliary fluents because they are all static.

Any classical planner can use the resulting encoding, and the solutions that the planner produces will correspond precisely to those plans for the original domain that do not violate the privacy model for the agents. While simple conceptually, this transformation makes it possible to apply any existing classical planner to the multi-agent setting used for the centralized track of the CoDMAP competition.

## LAPKT Planner Description: Heuristic and Search Description

The algorithm *Iterated Width*, or *IW*, consists of a sequence of calls $IW(i)$ for $i = 0, 1, \dots, |F|$ until the problem is solved. Each iteration $IW(i)$ is a breadth-first search that immediately prunes any states that do not pass a *novelty* test; namely, for a state $s$ in $IW(i)$ *not* to be pruned there must be a tuple $t$ of at most $i$ atoms such that $s$ is the first state generated in the search that makes $t$ true. The time complexities of $IW(i)$ and $IW$ are $O(n^i)$ and $O(n^w)$ respectively where $n$ is $|F|$ and $w$ is the problem width. The width of existing domains is low for atomic goals, and indeed, 89% of the benchmarks can be solved by $IW(2)$ when the goal is set to any one of the atoms in the goal (Lipovetzky and Geffner 2012). The width of the benchmark domains with conjunctive goals, however, is not low in general, yet such problems can be serialized.

*Serialized Iterative Width*, or *SIW*, uses *IW* for serializing a problem into subproblems and for solving the subproblems. SIW uses IW greedily to achieve one atomic goal at a time until all atomic goals are achieved jointly. In between, atomic goals may be undone, but after each invocation of IW, each of the previously achieved goals must hold. SIW will never call IW more than $|G|$ times where $|G|$ is the number of atomic goals. SIW compares surprisingly well to a baseline heuristic search planner using greedy best-first search and the $h_{add}$ heuristic (Bonet and Geffner 2001), but does not approach the level of performance of the most recent planners. Nonetheless, *SIW* competes well in domains with no dead-ends and simple serializations.

While the blind-search *SIW* procedure competes well with a greedy best-first planner using the additive heuristic, neither planner is state-of-the-art. To narrow the performance gap, we use two simple extensions. The first involves computing a relaxed plan once before moving on to the next subgoal. This makes the pruning in the *breadth-first procedure* less aggressive, while keeping IW exponential in the width parameter. This new procedure called $IW^+(i)$, computes a *relaxed plan* once from the initial state $s$ so that states $s'$ generated by $IW^+(i)$ keep a count on the number of atoms $m$ in the relaxed plan from $s$ achieved on the way to $s'$. For the state $s'$ in the breadth-first search underlying $IW^+(i)$ *not* to be pruned, there must be a tuple $t$ with at most $i$ atoms, such that $s'$ is the first state among the states in the search *that achieved m fluents from the initial relaxed plan* that makes the tuple $t$ true. The serialized algorithm *SIW* that uses $IW^+$ is called $SIW^+$. The second extension involves changing the *greedy search* for achieving the goals one at a time, by a *depth-first search* that is able to backtrack. The planner that incorporates both extensions is called $DFS^+$ (Lipovetzky and Geffner 2014). Notice that while $DFS^+$ computes a relaxed plan once for each $IW^+$ call, $DFS^+$ does not use the relaxed plan for computing heuristic estimates. Thus, $DFS^+$ remains a blind search planner, which does not use any standard techniques such as heuristics, multiple-search queues, helpful actions or landmarks.

In contrast with $DFS^+$, we developed an additional standard *forward-search best-first planner* guided by an evaluation function that combines the notions of novelty and helpful actions (Lipovetzky and Geffner 2012; Hoffmann and Nebel 2001). In this planner, called *BFS(f)* (Lipovetzky and Geffner 2012), ties are broken lexicographically by two other measures: (1) the number of subgoals not yet achieved up to a node in the search, and (2) the additive heuristic, $h_{add}$. The additive heuristic is delayed for non-helpful actions.

## Implementation

All the planners have been implemented using the automated planning toolkit LAPKT[1] (Ramirez, Lipovetzky, and Muise 2015). The toolkit is an extensible C++ framework that decouples search and heuristic algorithms from PDDL parsing and grounding modules, by relying on planner "agnostic" data structures to represent (ground) fluents and actions. We consider LAPKT to be a valuable contribution in itself since it enables the community to develop planners, while relying on a collection of readily available implementations of search algorithms and planning heuristics. These resulting planners are independent from specific parsing modules and grounding algorithms. For planners that acquire descriptions of planning tasks from PDDL specifications, the toolkit provides the means to plug in either FF (Hoffmann and Nebel 2001) or FAST-DOWNWARD (Helmert 2006) parsers. Alternatively, and more interestingly, the planner can be embedded into complex applications, *directly*, if the "host" application is written in C++, or *indirectly* when the host is written in an interpreted language, such as PYTHON, by wrapping the planner with suitably generated marshalling code.

[1]Source code available from `http://www.lapkt.org`

## Entry Variations

Three variations of the LAPKT planning framework were submitted to test their distinctive behaviour on the encoded domains. Here, we briefly describe each in turn:

1. Anytime-LAPKT: The first configuration is sought by $SIW^+$. Failing this, BFS($f$) is called. After a first solution is computed, RWA* is invoked with the appropriate bound, and solution quality is improved iteratively. The motivation behind this configuration is to try and find high quality plans within the time limit. This variation is both sound and complete. In the limit, it is also optimal.
2. $SIW^+$-then-BFS($f$): The second configuration attempts first to solve the problem using $SIW^+$. Failing this, BFS($f$) is invoked and will run until a solution is found. The motivation behind this configuration is to try and find a solution as fast as possible, while retaining completeness. This variation is both sound and complete.
3. $DFS^+$: The third and final configuration tries to find a solution extremely quickly using only $DFS^+$. The motivation behind the third configuration is to see how many problems can be solved using this simple approach. This final variation is sound, but incomplete.

## Summary

We have described three variations of a planner submitted to the CoDMAP contest. All three take as input a converted version of the multi-agent problem such that the privacy of objects and fluents are respected by any plan. Each variation has a different motivation that explores aspects such as striving for plan quality versus speed to completion.

The specific characteristics that our planners have, as defined by the CoDMAP organizers, are as follows:

1. Planner complete? Configurations 1 and 2 are complete.
2. Planner optimal? Configuration 1 is optimal in the limit.
3. Is the agent factorization in the MA-PDDL files used? Yes, the translation uses the agent factorization to determine which agents can execute the appropriate actions.
4. Is the private/public separation presented in the MA-PDDL files used? Yes, the translation uses this information to determine the initial configuration of K-obj and K-fluent fluents.
5. Is the planner using MA-STRIPS private/public separation? Yes, as part of the translation.
6. What private information (or its derivative), in what form, and how is it communicated in the planner? Nothing, other than the newly encoded problem (i.e., the planner is unaware it is solving a multi-agent planning problem).
7. What is the architecture of the planner (centralized or distributed; single or multi-threaded)? Centralized and single thread for all three configurations.

**Acknowledgements** This research is partially funded by Australian Research Council Discovery Grant DP130102825, *Foundations of Human-Agent Collaboration: Situation-Relevant Information Sharing* and the Australian Research Council Linkage Grant LP11010015, *Making the Pilbara Blend: Agile Mine Scheduling through Contingent Planning*.

## References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129:5–33.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. 540–545.

Lipovetzky, N., and Geffner, H. 2014. Width-based algorithms for classical planning: New results. In *Proc. ECAI*, 1059–1060.

Ramirez, M.; Lipovetzky, N.; and Muise, C. 2015. Lightweight Automated Planning ToolKiT. `http://lapkt.org/`. Accessed: 2015-05-19.

Stolba, M.; Komenda, A.; and Kovacs, D. L. 2015. Competition of Distributed and Multiagent Planners (CoDMAP). `http://agents.fel.cvut.cz/codmap/`. Accessed: 2015-05-24.

# A First Multi-agent Planner for Required Cooperation (MARC)

**Sarath Sreedharan, Yu Zhang and Subbarao Kambhampati**
Department of Computer Science, Arizona State University, Tempe AZ

**Abstract**

In this report, we describe the first multi-agent planner for required cooperation (MARC) as submitted into CoDMAP competition 2015. MARC is a centralized multi-agent planner, which is complete but non-optimal. MARC is designed to use the theory of required cooperation to solve a subset of large multi-agent problems by compiling them into "smaller" problems (smaller in terms of the number of agents) by using the notion of transformer agents. We aim to improve upon the planning time by solving these simpler problems and then expanding them to a plan for the original problems. Through the use of this approach we have also observed that the competition domains only represents a subset of possible multi-agent planning problems.

## Introduction

Multi-agent planning is a research area that has been receiving considerable interest from the planning community in recent years. Most of the current approaches concentrate on using the interaction between agents to efficiently produce plans. An equally important question we need to ask ourselves is, when is the use of multiple agents required to solve the problem? Even when multiple agents are not required they may still be used in a planning problem for efficiency reasons. Our earlier work (Zhang et al 2014) asked this fundamental question, and we had identified methods to establish if a problem truly requires cooperation among agents. That work also introduced an approach of solving a multi-agent problem by solving an equivalent problem containing a smaller number of agents. This equivalent problem consists of special virtual agents called transformer Agents (TA) - As the name suggests, these agents are capable of transforming into any agent from the original domain (thus be able to use all of their capabilities).

In this report, we introduce the first multi-agent planner for required cooperation (MARC), which is an attempt to implement some of the ideas from this earlier work. Our current implementation compiles a multi-agent planner problem into a transformer agent problem, provided the problem can be solved by a single transformer agent; otherwise, the problem is solved by an existing planner. Our tests on the current domains provided by the CoDMAP organizers showed that nine out of ten domains satisfy the the requirements of this compilation. By adopting this approach we have been able to create a complete (but non-optimal) planner that has given us good performance on a number of domains used in the competition. As we explain in the later sections of the report this in fact means, that the competition domains only explores a subset of the possible required cooperation problems.

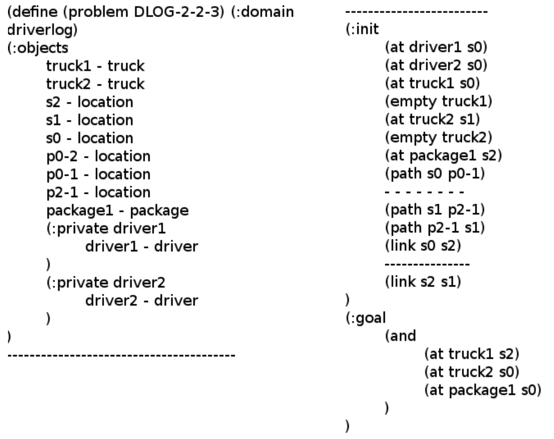
## Required Cooperation

One important concept that (Zhang et al 2014) introduced was that of required cooperation (RC). A Multi-agent problem (MAP) is said to have required cooperation if the problem is not single agent solvable. The work further goes to define two classes of cooperation namely type 1 for Heterogeneous agents and type 2 for homogeneous agents. In type 1 RC, the heterogeneity could arise from domain, variables or capabilities (DVC) of the agents. All problems with RC which have no domain, variables or capabilities heterogeneity gets classified under the type 2 RC. We also showed that for a given problem of type 1 in which the RC can only be caused by DVC, it can be solved by a single transformer agent, provided the state space of all the agents are connected. (Zhang et al 2014) suggests the use of connectivity graphs in order to test for the state space connectivity, where the connectivity graph is an undirected graph in which each agent is represented by a graph node and any two nodes are connected if the agents have connected state spaces. The work also assumes that all the goals are independent of the agents.

Our planner MARC solves a multi-agent problem by first compiling it into a transformer agent problem. Once we have the plan for the transformer agent problem, we expand this plan to produce a plan for the original problem.

## MARC

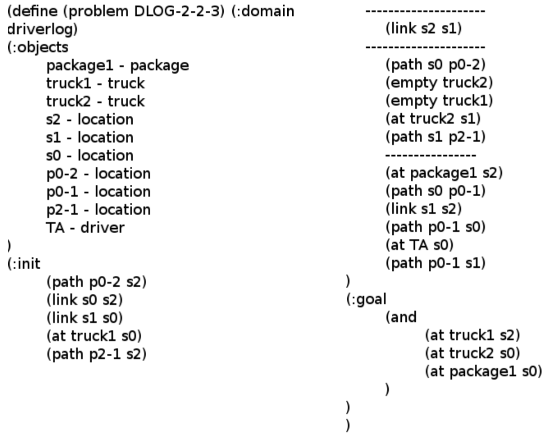
As mentioned above MARC draws from the theory of required cooperation as presented in (Zhang et al 2014)



```
(define (problem DLOG-2-2-3) (:domain
driverlog)
(:objects
        truck1 - truck
        truck2 - truck
        s2 - location
        s1 - location
        s0 - location
        p0-2 - location
        p0-1 - location
        p2-1 - location
        package1 - package
        (:private driver1
                driver1 - driver
        )
        (:private driver2
                driver2 - driver
        )
)
----------------------------------------
```

```
------------------------
(:init
        (at driver1 s0)
        (at driver2 s0)
        (at truck1 s0)
        (empty truck1)
        (at truck2 s1)
        (empty truck2)
        (at package1 s2)
        (path s0 p0-1)
        - - - - - - - -
        (path s1 p2-1)
        (path p2-1 s1)
        (link s0 s2)
        --------------
        (link s2 s1)
)
(:goal
        (and
                (at truck1 s2)
                (at truck2 s0)
                (at package1 s0)
        )
)
```

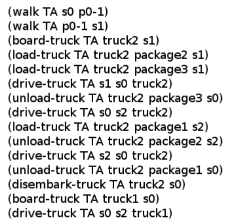
(a)



```
(define (problem DLOG-2-2-3) (:domain
driverlog)
(:objects
        package1 - package
        truck1 - truck
        truck2 - truck
        s2 - location
        s1 - location
        s0 - location
        p0-2 - location
        p0-1 - location
        p2-1 - location
        TA - driver
)
(:init
        (path p0-2 s2)
        (link s0 s2)
        (link s1 s0)
        (at truck1 s0)
        (path p2-1 s2)
```

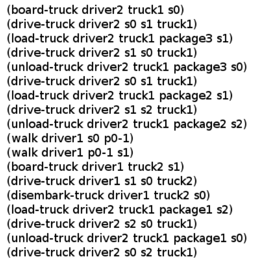
```
    --------------------
        (link s2 s1)
    --------------------
        (path s0 p0-2)
        (empty truck2)
        (empty truck1)
        (at truck2 s1)
        (path s1 p2-1)
        ----------------
        (at package1 s2)
        (path s0 p0-1)
        (link s1 s2)
        (path p0-1 s0)
        (at TA s0)
        (path p0-1 s1)
)
(:goal
        (and
                (at truck1 s2)
                (at truck2 s0)
                (at package1 s0)
        )
)
)
```

(b)



```
(walk TA s0 p0-1)
(walk TA p0-1 s1)
(board-truck TA truck2 s1)
(load-truck TA truck2 package2 s1)
(load-truck TA truck2 package3 s1)
(drive-truck TA s1 s0 truck2)
(unload-truck TA truck2 package3 s0)
(drive-truck TA s0 s2 truck2)
(load-truck TA truck2 package1 s2)
(unload-truck TA truck2 package2 s2)
(drive-truck TA s2 s0 truck2)
(unload-truck TA truck2 package1 s0)
(disembark-truck TA truck2 s0)
(board-truck TA truck1 s0)
(drive-truck TA s0 s2 truck1)
```

(c)



```
(board-truck driver2 truck1 s0)
(drive-truck driver2 s0 s1 truck1)
(load-truck driver2 truck1 package3 s1)
(drive-truck driver2 s1 s0 truck1)
(unload-truck driver2 truck1 package3 s0)
(drive-truck driver2 s0 s1 truck1)
(load-truck driver2 truck1 package2 s1)
(drive-truck driver2 s1 s2 truck1)
(unload-truck driver2 truck1 package2 s2)
(walk driver1 s0 p0-1)
(walk driver1 p0-1 s1)
(board-truck driver1 truck2 s1)
(drive-truck driver1 s1 s0 truck2)
(disembark-truck driver1 truck2 s0)
(load-truck driver2 truck1 package1 s2)
(drive-truck driver2 s2 s0 truck1)
(unload-truck driver2 truck1 package1 s0)
(drive-truck driver2 s0 s2 truck1)
```

(d)

Figure 1: (a) Original problem (b) Compiled problem (c) Transformer agent plan (d) Multi agent plan produced from transformer agent plan

to produce a multi-agent planner. We rely on our system's ability to simplify a complex multi-agent problem to a simpler transformer Agent problem to produce plans faster. In our current implementation, this works only if the given problem can be solved by a single transformer agent and has at least one non-agent variable in the goal. There could be problems where these assumptions are not met. So we make sure that if the process fails to perform the compilation or fails to produce a plan for the compiled domain, we switch to an existing planner to solve the original problem. Our use of a second planner to cover the failure cases ensures completeness, given that the planner that we use is complete. Due to the approach we adopt in expansion, our plans are not guaranteed to be optimal. The Figure 1.a-1.d shows an example problem being solved using our planner. We take a problem from the driverlog domain, compile it to a transformer agent problem, solve it and use the resultant plan to create a plan for the original problem.

The different components in our planner are as follows:

- Compiler : This component is responsible for creating a transformer agent problems, this component determines the agents to be combined and the initial state of the transformer agent.
- Transformer agent solver : We use Fast Downward (Cenamor et al 2014) or IBACOP (Helmert 2006) to solve the transformer agent problem created in the last step.
- Transformer agent plan expander : This step takes the transformer agent plan and creates an equivalent multi-agent plan. It does this by expanding each step in the transformer agent plan to an equivalent multi-agent plan segment. At the end we perform an extra planning step to make sure all the required goal predicates are satisfied by the expanded plan.

The algorithm for the planner is provided in Algorithm 1:
Here there are two possible ways the planning process could have failed:

- The planning problem could have failed to be compiled to a transformer agent problem. One possible reason could be that there are no agent independent goals. We solve this by solving the problem directly using IBACOP.
- The planning of transformer agent problem could fail ( The planner could have timed out or planner might have failed to find a solution). In such cases we rely on IBACOP to solve the original problem.

## Compiler

This is the component responsible for converting the MA-PDDL problems provided to us into a transformer agent problem. Here we make use of a modified version of the *convert.py* provided to us by the competition organizers. This modified version produces the list

**Algorithm 1** Algorithm for MARC

```
Input: $P$ (the original planning problem)
$P' = Compile\_to\_transformeragent(P)$
Try solving the problem $P'$ using Fast Downward with a timeout of 5 minutes to get a plan $\pi$
if Above step times out then
Try solving the problem $P'$ using IBACOP with a timeout of 5 minutes to get a plan $\pi$
    if the above step times out then
        Try solving the problem $P'$ using Fast Downward with a timeout of 10 minutes to get a plan $\pi$
    end if
end if
if All the previous steps timesout or fail to produce a plan then
    Solve the problem P using IBACOP to get a plan $\pi'$
else
    $\pi' = Expand\_plan(\pi, P')$
end if
```

of agents, the corresponding PDDL problem, a corresponding transformer agent problem and a mapping of each agent to their specific private objects. Also in our approach we do not use all the agents specified in the problem for creating the transformer agent, in fact we only use the agents which has at best one action which is private (i.e doesn't interact with any other agents) (Brafman et al 2008).

Once we have the new agent list, the next step is to create a transformer agent object that can represent any one of the original agents. We define this new transformer agent object to belong to all the agent types. For example if the original agents in the problem were of the type Truck and Airplane, then our new transformer agent would be of both type Truck and type Airplane. Next we update the initial state of the problem by replacing the agent object in each agent specific predicate with the transformer agent object. For the transformer agent problem goal we only consider the agent independent goals. Once we have the new problem we move onto the next step.

## Transformer agent solver

Here we just use the standard Fast downward planner running lazy greedy search with FF and CEA heuristic. We have a five minute time out on this instance of the planner, On timeout we switch to IBACOP planner (Cenamor et al 2014) (modified to use less number of planners) that again tries to solve the problem for another 5 minutes. If both planners fail the first time, we retry using Fast Downward for 10 minutes before just moving on to using IBACOP to solve the original MAP.

## Transformer agent plan expander

Next we move on to the expansion component, the purpose of this component is to take the transformer agent plan and expand it to a multi-agent plan for the original problem. Here we start with an empty final plan and then for each step in the transformer agent agent plan, we create a small Multi-agent plan equivalent to that step and append it to the final plan. To achieve this, each action of the transformer agent plan is converted into multiple agent specific actions. We create these actions by replacing the transformer agent by a specific agent. We also replace any private objects referenced by that action with those of the specific agent we are considering.

Next we look at the agent independent effects of each of these new actions. We then run a small planning instance to find out the sequence of actions to achieve the effect of each one of these new agent actions. Once we have these plans we choose the smallest one amongst them to be appended to the final plan. Here for these smaller planning instances we chose to use Metric-FF (Hoffmann 2003) to solve these problems. The choice was based on the fact that it has a smaller pre-processing time compared to the other planners we considered. This is especially important as we will be rerunning this planner many times in this component.

We also set a time out of one minute for each instance of planning problem. In case any of these problems are not solved, we carry over any goal predicate present in the effects to the planning problem for the next step. This ensures that the expanded plan can satisfy all the goals, but we only perform this carry over of goal predicates if that specific step is the last one capable of producing that specific goal predicate. Once we have expanded all the steps, if we have any goal predicates remaining, we try to plan for all the remaining goals. The plan produced in each of these steps together form the final plan. Since we run each of these planning instances independently, we need to make sure that none of these plans violate any of the already achieved final goal predicates. We do this by appending the achieved final goal predicates into the goal list of each smaller planning problem. Once this stage is completed we should have a complete plan for the original problem.

## Conclusion

We have presented MARC, a multi-agent planning method based on the recent work on required cooperation. Based on the test domains provided to us in the competition, our approach has been proven to produce good results on nine out of ten domains. Here MARC was able to improve the performance of planning on these nine domains by compiling them down to a simpler transformer agent problem. This shows that the problems of these nine domains were in fact transformer agent solvable. There remains a large set of possible RC types which are not represented by the problem domains considered in this competition (related to type-2 RC in Zhang et. al). These problems are expected to be harder to solve compared to the DVC RC problems, which forms the majority of the competition domains.

## Acknowledgments

This research is supported in part by the ARO grant W911NF-13-1-0023, and the ONR grants N00014-13-1-0176, N00014-13-1-0519 and N00014-15-1-2027.

## References

[Zhang et al 2014] Zhang, Yu, and Subbarao Kambhampati. A Formal Analysis of Required Cooperation in Multi-agent Planning. arXiv preprint arXiv:1404.5643 (2014).

[Cenamor et al 2014] Cenamor, Isabel, Toms de la Rosa, and Fernando Fernndez. IBACOP and IBACOP2 planner. IPC, 2014.

[Helmert 2006] Helmert, Malte. The Fast Downward Planning System. J. Artif. Intell. Res.(JAIR) 26 (2006): 191-246.

[Hoffmann 2003] Hoffmann, Jörg. The Metric-FF Planning System: Translating "Ignoring Delete Lists"to Numeric State Variables. Journal of Artificial Intelligence Research (2003): 291-341.

[Brafman et al 2008] Brafman, Ronen I., and Carmel Domshlak. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. ICAPS. 2008.

# MADLA: Planning with Distributed and Local Search

**Michal Štolba** and **Antonín Komenda**
{stolba,komenda}@agents.fel.cvut.cz
Department of Computer Science, Faculty of Electrical Engineering,
Czech Technical University in Prague, Czech Republic

**Abstract**

In this short paper we present the MADLA (Multi-agent Distributed and Local Asynchronous) Planner which entered the centralized track of the Competition of Distributed and Multi-Agent Planners (CoDMAP). The planner combines two variants of FF heuristic, a projected (local) variant and a distributed variant, in a multi-heuristic state-space search. The main idea of the search is that while waiting for the computation of the distributed heuristic, the local one asynchronously continues the search.

## Introduction

In most domain-independent multi-agent planners based on heuristic search one of two types of heuristic are used. First is a projected (local) heuristic, that is a heuristic computed only on the particular agent's part of the problem, second is a distributed heuristic typically using some way of sharing information between the agents to compute a global heuristic estimate. Often, one of the approaches performs better on some benchmarks.

The MADLA (Multiagent Distributed and Local Asynchronous) Planner is a domain-independent deterministic multi-agent state-space heuristic search planner based on the idea of combining projected (local) and distributed heuristics in multi-heuristic search. The planner is centralized, it runs in multiple threads (one thread per agent + communication thread) in a single process on a single machine. The agents can communicate either using in-process communication or via TCP-IP local loopback (the communication could possibly run over a network).

The MADLA planner is based on the MA-STRIPS formalism (privacy, factorization), although internally it uses a multi-valued representation of the states. As an input, the planner can use the unfactored MA-PDDL as input, but it translates it to plain PDDL + list of agents, which is then internally factored based on the MA-STRIPS rules.

The MADLA planner aims to preserve the privacy of the participating agents. The agents exchange no private information, except for the following two exceptions:

**i)** The description of the state contains the private parts of all agents, although other agents cannot interpret the private parts of other agents (they are not in their domain).

**ii)** The Set-Additive Lazy FF heuristic used exchanges hash codes of private actions, that is the other agents knows the number and identity (in the sense it knows whether the same private action was already used) of other agent's private actions, but they do not know the action's preconditions, effects, and signature.

The planner uses a simple distributed plan extraction procedure and stops whenever any of the agents finds a solution. Also, the planner uses a variant of greedy best-first search and the FF heuristic and thus is not optimal. Apart form the MADLA configuration, it can be used to use only the projected heuristic, only the distributed heuristic and multiple other configurations. The planner is written in Java and is available under the GNU-LGPL licence at https://github.com/stolba/MADLAPlanner.

## Planner Architecture

In the current implementation, the planner is written as monolithic, each agent running in its own thread. The architecture and processing of input is shown in Figure 1. The planning process starts with either MA-PDDL, which is translated to plain PDDL and ADDL (in fact a list of agents), or directly with PDDL and ADDL. The PDDL is then translated to SAS+ using the Fast-Downward (Helmert 2006) translator. Both SAS+ and ADDL is then fed to the centralized starting point (a Java executable), which parses the SAS+ input and performs a factorization based on the MA-STRIPS rules.

Actions are assigned to agents based on the first action parameter equal to some of the agent objects provided. Facts shared among agents are then determined and treated as public. Each agent is then started and provided with its factor of the problem - that is its actions, projections of other agent's public actions, a set of variables and their domains, the initial state and the goal condition. Although the states are shared including the full information, each agent projects each state to its set of variables and its portions of the domains.

## Distributed FF Heuristic

The MADLA planner uses two variants of the well known Fast-Forward heuristic (Hoffmann and Nebel 2001). The projected heuristic, denoted as $h_{\mathrm{FF}}^{\alpha_i}$ where $\alpha_i$ is the particular agent, uses only the part of the problem known to $\alpha_i$, that is its actions, facts and $\alpha_i$-projections of other agent's actions. An $\alpha_i$-projection of some action $a$ is $a^{\alpha_i} = \langle \mathsf{pre}(a) \cap P_i, \mathsf{add}(a) \cap P_i, \mathsf{del}(a) \cap P_i \rangle$ where $P_i$ are the facts known to $\alpha_i$ and $\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)$ are the preconditions, add effects and delete effect of action $a$ respectively.

The distributed heuristic is a Set-Additive (SA) variant of the Lazy FF heuristic (Štolba and Komenda 2013; 2014), denoted as $h_{\mathrm{SAlazyFF}}$. We take inspiration from the Set-Additive variation of the FF heuristic (Keyder and Geffner 2008), where instead of cost of reaching a fact $p$ in a planning problem $\Pi = \langle P, A, I, G \rangle$, each fact $p$ is associated with a relaxed plan $\pi_p^+$ solving a relaxed planning problem where $p$ is the goal $G = \{p\}$. The overall relaxed plan $\pi^+$ is then constructed by computing a set unions of the respective fact relaxed plans

$$\pi^+ = \bigcup_{p \in P} \pi_p^+, \tag{1}$$

which is possible as the order of the actions in a relaxed plan can be arbitrary and using any action more than once is redundant.

The general idea of Lazy FF is that some agent $\alpha_i$ starts the computation of a projected FF heuristic. The resulting relaxed plan may contain some projected action $a^{\alpha_i}$ such that $a$ belongs to some other agent $\alpha_j$. Notice, that $a^{\alpha_i}$ may ignore some preconditions of $a$ private to $\alpha_j$ and thus the action may not be applicable in the global (relaxed) problem. This can also be seen that the cost of $a$ is underestimated. In Lazy FF, a request is sent to $\alpha_j$ to obtain the true cost. When the request is received, $\alpha_j$ computes the FF heuristic to a restricted relaxed problem, where the goal is the set of private preconditions of $a$. Agent $\alpha_j$ then returns cost of the found relaxed plan, together with a set of (public) actions of other agents, which appear in the relaxed plan and need to also be resolved. This ensures, that only the initiating agent sends requests while other agents only compute replies, thus preventing (flattening) a distributed recursion.

The distributed process of Lazy FF may compute relaxed sub-plan for a single fact multiple times, resulting in overcounting of actions and worse estimate. In Set-Additive Lazy FF, the agents do not send the intermediate costs $c^{\alpha_j}, c^{\alpha_k}, \ldots$, but the relaxed plans $\pi^{\alpha_j+}, \pi^{\alpha_k+}, \ldots$ which are then merged by the initiating agent $\alpha_i$. The resulting heuristic estimate $c^{\alpha_i}$ is the cost of the merged relaxed plan. The SA Lazy FF estimator in the recursive form follows:

1. The agent $\alpha_i$ initiating the estimation locally computes a projected relaxed plan $\pi^{\alpha_i+}$ which is a solution of a relaxed projected problem $\Pi^{\alpha_i+}$.
2. For each projected action $a^{\alpha_i+} \in \pi^{\alpha_i+}$, the initiator agent $\alpha_i$ sends a request to the action's owner agent $\alpha_j$. Upon receiving, $\alpha_j$ computes partial RP $\pi^{\alpha_j}$ as a solution of a relaxed projected problem $\Pi_a^{\alpha_j+}$ and sends $\pi_a^{\alpha_j}$ to the initiator agent as a reply.
3. The agent $\alpha_j$ may need to ask other agent(s) $\alpha_k$ in the same manner, resulting in a distributed recursion merging the partial relaxed plans: $\pi^{\alpha_j+} \leftarrow \pi^{\alpha_k+} \cup \pi^{\alpha_k+}$.
4. The initiator agent merges the received relaxed plan $\pi^{\alpha_j}$ with the initial RP $\pi^{\alpha_i+} \leftarrow \pi^{\alpha_i+} \cup \pi^{\alpha_j+}$ and returns its cost: $\sum_{a \in \pi^{\alpha_i+}} c(a)$.

In the implementation, the recursion is flattened the same way as described for the Lazy FF heuristic.

Similarly to the Lazy FF estimator, we have to explicitly consider that in the 2nd step, no solution for $\Pi^{\alpha_j+}, \Pi^{\alpha_k+}, \ldots$ may exist. Such situation indicates a dead-end on preconditions $\mathsf{pre}(a)$. As it is not cheaply possible to find out if another relaxed plan $\pi^{\alpha_i+}$ could be used, we ignore the information and return the heuristic as if the action was reachable (and the replied relaxed plan $\pi^{\alpha_j+}, \pi^{\alpha_k+}, \ldots$ empty). This causes the heuristic to report non-$\infty$ estimates for dead-end states (will not be *safe*). Since the FF heuristic is not safe by itself and the practical efficiency gains are substantial, we conclude it is not a (practical) problem.

## MADLA Search

The observation of different coverage results of the projected and distributed FF heuristics led to the consideration of a search scheme possibly combining both heuristic variants in a positive manner. The classical multi-heuristic search as used in the Fast-Downward (Helmert 2006) and LAMA (Richter and Westphal 2010) planners did not show promising results as the pair of the $h_{\mathrm{FF}}^{\alpha_i}$ and $h_{\mathrm{SAlazyFF}}$ heuristics does not have the important property of orthogonality, that is giving significantly different results in the same states. But there are interesting properties of this heuristic pair which may be exploited. Such properties are defined as follows.

**Definition 1.** A heuristic estimator of a heuristic function $h$ for a planning problem $\Pi^G$ run by agent $\alpha_i$ is *non-blocking* iff the computation of $h$ does not block the computation process of agent $\alpha_i$ for the whole duration of the computation of $h$.

For example a global heuristic estimator can require computation of parts of the heuristic estimate by other agents. If such heuristic algorithm is non-blocking, it does not wait for responses from other agents and allows the agent to run asynchronously while waiting for the responses, that is, the agent can use the time to perform some other computations.

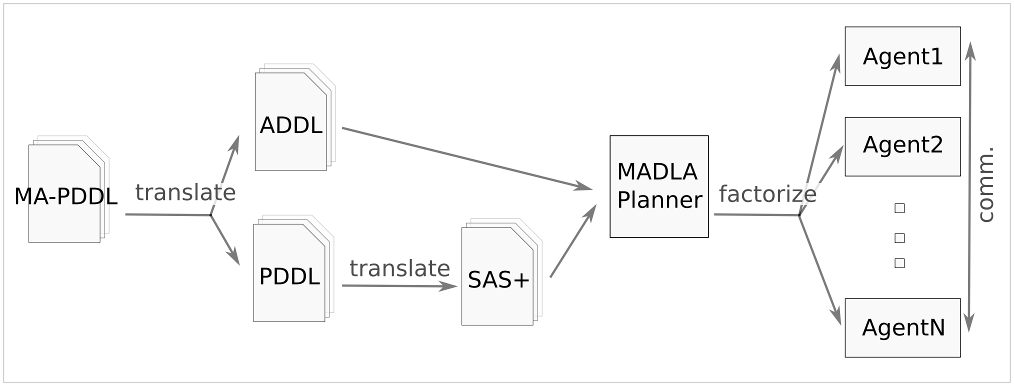


Figure 1: MADLA Planner architecture

A definition of dominance is the same as in the classical planning:

**Definition 2.** A heuristic function $h_1$ *dominates* a heuristic function $h_2$ for a planning problem $\Pi$ iff for all states $s \in 2^P$ hold that $h_1(s) \geq h_2(s)$.

Finally, we need a relation between two heuristics describing their relative computational hardness:

**Definition 3.** A heuristic estimator of a heuristic function $h_1$ is computationally *harder* than heuristic estimator of a heuristic function $h_2$ for a planning problem $\Pi$ iff for all states $s \in 2^P$ holds that computation of $h_1(s)$ takes the same or longer time than computation of $h_2(s)$.

With the help of the three definitions, we can define properties of a pair of heuristics which is required for the MADLA search:

**Definition 4.** For a multi-agent planning problem $\Pi$, let $h_\mathsf{L}$ be a heuristic function for $\Pi$ and let $h_\mathsf{D}$ be a heuristic functions for the global problem $\Pi^G$. The heuristics $h_\mathsf{L}$ and $h_\mathsf{D}$ are MADLA heuristic pair iff $h_\mathsf{D}$ uses a non-blocking estimator by Definition 1, $h_\mathsf{D}$ dominates $h_\mathsf{L}$ by Definition 2, and $h_\mathsf{D}$ estimator is computationally harder then $h_\mathsf{L}$ estimator by Definition 3.

The intuition behind the properties is that the search combines a computationally fast heuristics $h_\mathsf{L}$ working not necessarily with all the information in the problem therefore likely with worse estimation and a computationally slower heuristics $h_\mathsf{D}$ working with global information such that the agent using the estimator can get additional information from other agents (e.g, in the form of increasing cost of some actions). Therefore the global heuristic estimate is higher than the local estimate. Such combination can work smoothly only if $h_\mathsf{D}$ does not block the agent using it, therefore the last property required allows the agents to continue the search using $h_\mathsf{L}$ whilst waiting for responses from other agents computing parts of $h_\mathsf{D}$.

As a local $h_\mathsf{L}$, we can use the projected FF ($h^{\alpha_i}_{\mathrm{FF}}$) and as a distributed (global) $h_\mathsf{D}$ we can use the Set-Additive Lazy FF ($h_{\mathrm{SAlazyFF}}$), because this pair of heuristics comply with the Definition 4.

## The principle of MADLA Search

The classical multi-heuristic search works on the principle of having a separate open list for each heuristic used. The states are extracted from the open list in an alternating manner, each state is evaluated by all used heuristics and placed in respective open lists with the computed value.

Unlike the classical multi-heuristic search, in MADLA search, the extracted states are not evaluated by both heuristics, the used heuristic evaluator depends instead on the state of the distributed heuristic $h_\mathsf{D}$. If the distributed heuristic is not busy (not computing a heuristic for some state) state is evaluated by $h_\mathsf{D}$. If $h_\mathsf{D}$ is in a process of computing a heuristic for some state (that is, waiting for replies for other agents), the state is evaluated by $h_\mathsf{L}$. The distributed heuristic is always preferred. This approach is most reasonable if $h_\mathsf{D}$ dominates $h_\mathsf{L}$, which holds for the local FF and SA Lazy FF heuristics.

The local heuristic search is performed only when the distributed heuristic search is waiting for the distributed heuristic estimation to finish. This principle makes sense only if finishing a estimation of $h_\mathsf{D}$ takes longer than of $h_\mathsf{L}$ and if computation of the $h_\mathsf{D}$ estimator does not block the search process (incl. $h_\mathsf{L}$ estimations). These two requirement hold for the local FF and SA Lazy FF as well.

Separation of the searches using two open lists ($O_\mathsf{D}$ for $h_\mathsf{D}$ and $O_\mathsf{L}$ for $h_\mathsf{L}$) has the benefit of using two heuristics in parallel, but if some information between the two searches could be shared[1], most importantly the heuristically best state found so far, it could boost the efficiency of the search. The direction $O_\mathsf{D} \rightarrow O_\mathsf{L}$ is straightforward, thanks to the fact that $h_\mathsf{D}$ dominates $h_\mathsf{L}$. We can add all nodes evaluated by $h_\mathsf{D}$ also to $O_\mathsf{L}$ without ever skipping a better state evaluated by $h_\mathsf{L}$ with a worse state evaluated by $h_\mathsf{D}$.

The other direction $O_\mathsf{L} \rightarrow O_\mathsf{D}$ is trickier. If we added a node $s$ evaluated by $h_\mathsf{L}$ to $O_\mathsf{D}$ it would skip many

[1]The two searches both run on one agent, therefore the question of privacy is irrelevant here.

nodes which are actually closer to the goal only because the local, less informative heuristic, will give a lower estimate. The way at least some information can be shared in this direction is whenever the open list $O_{\mathsf{D}}$ becomes empty and the heuristic estimator $h_{\mathsf{D}}$ is not computing any heuristic, a state $s$ is pulled from the local open list $O_{\mathsf{L}}$ and evaluated by the distributed heuristic $h_{\mathsf{D}}$ and its successors are added to both open lists, as already described. This way, the nodes in $O_{\mathsf{D}}$ are evaluated only by $h_{\mathsf{D}}$, but sometimes, the best node from $O_{\mathsf{L}}$ is taken. The situation is illustrated in Figure 2.

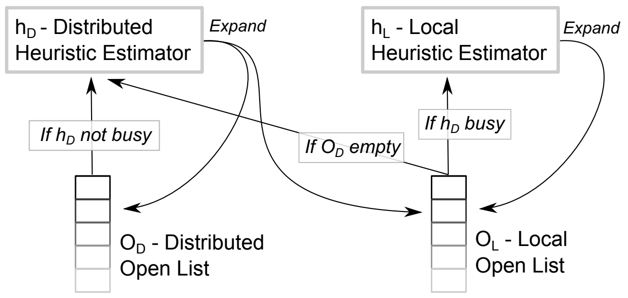

hD - Distributed Heuristic Estimator
Expand
hL - Local Heuristic Estimator
Expand
If hD not busy
If OD empty
If hD busy
OD - Distributed Open List
OL - Local Open List

Figure 2: Distributed/Local Search - OPEN lists and heuristic estimators.

The implementation of the distribution in the MADLA search follows the principles of MAD-A* (Nissim and Brafman 2014), i.e., broadcasts are used to inform other agents about states reached by public actions. Additionally, information to which open list the state should be added in is included (whether it is the local or the distributed one).

## Final Remarks

The MADLA Planner combines search using a projected and distributed heuristic in a non-trivial way. Also the experiments we have conducted shows, that the results (in terms of coverage) are not a trivial maximum of the results of pure projected and pure distributed heuristics. In many cases, the coverage is even higher, that the maximum which suggests positive synergy of the two searches. There are some cases, where the MADLA planner performs worse than the pure projected heuristic, which is in cases where the distributed heuristic performs poorly (is too communication intensive and does not give enough benefits). That is because the distributed heuristic is preferred, but still, if the projected heuristic starts computation, it is never interrupted.

**Acknowledgments** This research was supported by the Czech Science Foundation (grant no. 13-22125S) and by the Grant Agency of the CTU in Prague (grant no. SGS14/202/OHK3/3T/13). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the program "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

## References

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, 588–592.

Nissim, R., and Brafman, R. 2014. Distributed heuristic forward search for multi-agent planning. *JAIR* 51:293–332.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39(1):127–177.

Štolba, M., and Komenda, A. 2013. Fast-forward heuristic for multiagent planning. In *Proceedings of the 1st ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP'13)*, 75–83.

Štolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, 298–306.

# MH-FMAP: Alternating Global Heuristics in Multi-Agent Planning

**Alejandro Torreño, Óscar Sapena, Eva Onaindia**
Universitat Politècnica de València
Camino de Vera s/n
46022 Valencia, SPAIN

**Abstract**

MH-FMAP is a fully-distributed Multi-Agent Planning system where each agent is designed as a forward-chaining partial-order planner. MH-FMAP applies a complete and suboptimal distributed A* search alternating two heuristic functions. This distributed nature enabled MH-FMAP to participate in the two tracks of the 2015 Competition of Distributed and Multiagent Planners.

## Introduction

Cooperative Multi-Agent Planning (MAP) introduces a set of planning entities which work together in a shared deterministic environment in order to attain a set of common goals. Agents in MAP synthesize individual plans and coordinate them to come up with a joint solution plan. The state of the art in MAP includes a wide variety of methods that range from centralized to distributed planning and coordination.

The application of pre-planning decomposition methods is commonly used in MAP. In ADP (Crosby, Rovatsos, and Petrick 2013), single-agent tasks are decomposed into MAP tasks and then solved with a centralized state-based planner, and MAPR (Borrajo 2013) allocates the task goals among agents, which then solve their problems sequentially: one agent at a time solves its subgoals using the plans previously computed by the prior agents.

A large number of MAP techniques opt for a post-planning coordination or plan merging scheme where agents' local plans are merged or coordinated into a global plan that solves the MAP task. In Planning First (Nissim, Brafman, and Domshlak 2010), for instance, agents use a state-based planner to compute their local plans and these plans are later coordinated through a distributed Constraint Satisfaction Problem.

A third group of approaches directly apply multi-agent search, interleaving planning and coordination. MA-A* (Nissim and Brafman 2012) performs a distributed A* search, guiding the procedure through admissible local heuristic functions.

In this paper we outline MH-FMAP (Torreño, Sapena, and Onaindia 2015), our contribution to the 2015 Competition of Distributed and Multi-Agent Planning (CoDMAP in the following). MH-FMAP is a fully-distributed MAP system that can be run either in a centralized or distributed manner and participated in both tracks of the CoDMAP.

MH-FMAP performs a cooperative heuristic-based multi-agent A* search. Each node in the search tree is a partial-order plan possibly contributed by more than one of the participating agents in a forward-chaining fashion.

MH-FMAP is the first MAP method to apply a multi-heuristic search scheme. Agents assess the quality of the partial-order plans by alternating two state-based distributed heuristic functions: $h_{DTG}$ (Torreño, Onaindia, and Sapena 2014), a variation of the Context-Enhanced Additive heuristic (Helmert and Geffner 2008) based on Domain Transition Graphs (Helmert 2004); and $h_{Land}$, a privacy-preserving version of the landmark extraction algorithm introduced in (Hoffmann, Porteous, and Sebastia 2004).

Agents are autonomous planning entities in MH-FMAP. This way, an agent is initialized as an independent execution thread/process with an embedded planning machinery and a communication infrastructure. An agent can make use of more than one execution thread or the whole CPU if available. This flexibility enabled our planner to participate in both CoDMAP tracks without major core modifications.

On the other hand, privacy is maintained throughout the planning process. Agents preserve privacy by occluding information regarding the literals in preconditions and effects. Therefore, the notion of privacy in MH-FMAP complies with the requirements established in the CoDMAP guidelines.

This paper is organized as follows: first, we briefly present the key notions of MH-FMAP, including the search scheme and the heuristic functions. Next, we detail the main changes and adaptations carried out to comply with the competition rules. Finally, we present a brief discussion on how the adaptation of MH-FMAP to the CoDMAP affected its performance.

## The MH-FMAP Planning Framework

MH-FMAP is a complete and suboptimal cooperative MAP system in which agents jointly explore a plan-space search tree. The nodes of the tree are partial plans contributed by one or several agents. Agents estimate the quality of each partial plan by means of the alternation of two state-based heuristic functions. This section summarizes the main features of MH-FMAP.

**Input language.** MAP tasks in MH-FMAP are described by using a customized definition language based on *PDDL3.1* (Kovacs 2011). Tasks are encoded through a factored description so that each agent receives its own domain and problem file.

The world states in MH-FMAP are modeled through state variables instead of predicates. The initial state of the task, along with the preconditions and effects of the operators, are described as variable-value tuples or *fluents* in the input files.

The problem files include an additional `:shared-data` section to model the agents' privacy. By default, all the information regarding the agent's fluents is considered private. The `:shared-data` section specifies the fluents that an agent can share with each other agent in the task.

**Privacy.** Agents keep privacy throughout the MAP process by sharing only the information of the fluents that are specifically defined as shareable in the `:shared-data` section of the input files. The mechanisms used to occlude private information are detailed in (Torreño, Sapena, and Onaindia 2015).

**Search process.** Agents in MH-FMAP apply a complete and suboptimal distributed A* search, which generates a common search space or multi-agent search tree for all agents (see Algorithm 1). The search process is led by an agent that plays the role of *coordinator*, which is rotated after each iteration of the planning procedure.

Agents estimate the quality of the plans by using two different heuristic functions, $h_{DTG}$ and $h_{Land}$ (see next subsection for details). Recent works in single-agent planning have proven the benefits in terms of performance and scalability of combining multiple heuristics (Röger and Helmert 2010). This conclusion is backed up by the well-known planning systems Fast Downward (Helmert 2006) and LAMA (Richter and Westphal 2010), which successfully apply a heuristic alternation approach to state-based planning. The alternation technique has been experimentally proven to be more efficient than other heuristic combination methods, such as sum, weighted sum, maximum or Pareto (Röger and Helmert 2010).

Agents in MH-FMAP alternate $h_{DTG}$ and $h_{Land}$ by maintaining two different lists of open nodes, $openList$ and $preferredList$. The $openList$ maintains all the open nodes of the multi-agent search tree ordered according to the evaluation function $f(\Pi) = g(\Pi) + 2 * h_{DTG}(\Pi)$. The $preferredList$ keeps only the *preferred successors*, sorted by $f(\Pi) = h_{Land}(\Pi)$. We consider a refinement plan $\Pi_r$ to be a *preferred successor* of a plan $\Pi$ iff $h_{Land}(\Pi_r) < h_{Land}(\Pi)$.

Agents start an iteration of the MH-FMAP procedure by alternatively selecting the most promising open node from $openList$ or $preferredList$ as a base plan $\Pi_b$ (see the first *if-else* instruction of the *while* loop in Algorithm 1). Agents individually expand $\Pi_b$ and every generated node is inserted in a set of refinement plans called $RP$. Each agent is prepared to synthesize refinement plans autonomously through an embedded forward-chaining partial-order planner.

Once the refinement plans in $RP$ are obtained, agents exchange them and apply a distributed heuristic evaluation of the refinement plans using both $h_{DTG}$ and $h_{Land}$. Once evaluated, the plans in $RP$ are stored in $openList$. If a plan in $RP$ is a preferred successor of $\Pi_b$, it is introduced in both $openList$ and $preferredList$. The procedure ends up when a solution plan is found, or when all the open nodes have been explored.

**Algorithm 1:** MH-FMAP algorithm for an agent $i$

```
$openList \leftarrow \Pi_0, preferredList \leftarrow \emptyset$
$list \leftarrow true$
while $openList \neq \emptyset$ do
    if $list = true$ then
        $\Pi_b \leftarrow extractPlan(openList)$
    else
        $\Pi_b \leftarrow extractPlan(preferredList)$
    $list \leftarrow \neg list$
    if $isSolution(\Pi_b)$ then
        return $\Pi_b$
    $RP \leftarrow refinePlan(\Pi_b)$
    for all $j \in \mathcal{AG}, j \neq i$ do
        $sendRefinements(j)$
        $RP \leftarrow RP \cup receiveRefinements(j)$
    for all $\Pi_r \in RP$ do
        $distributedEvaluation(\Pi_r, h_{DTG})$
        $distributedEvaluation(\Pi_r, h_{Land})$
        $openList \leftarrow openList \cup \Pi_r$
        if $h_{Land}(\Pi_r) < h_{Land}(\Pi_b)$ then
            $preferredList \leftarrow preferredList \cup \Pi_r$
return No solution
```

**Heuristic functions.** MH-FMAP alternates $h_{DTG}$ and $h_{Land}$, two state-based heuristic functions, to improve the overall performance of the system. The first heuristic function, $h_{DTG}$, is an additive heuristics that uses Domain Transition Graphs (DTGs) to estimate the cost of a plan. A DTG is a graph in which nodes represent values of a particular variable, and transitions show the changes in the values of such variable through the actions of the agents.

Similarly to the $h_{CEA}$, $h_{DTG}$ builds a relaxed plan and reuses the side effects of the actions in the relaxed plan as a basis to estimate the cost of the subsequent subgoals. A plan $\Pi$ of FMAP is always evaluated from its frontier state, $FS(\Pi)$, but the cost of some of the subgoals can be estimated in a state different from $FS(\Pi)$. $h_{DTG}(\Pi)$ returns the number of actions in the relaxed plan as an estimate of the cost of the plan $\Pi$.

The second heuristic function, $h_{Land}$, computes the Landmarks Graph (LG) of the MAP task, which is later used to calculate the number of landmarks achieved by the partial plans. $h_{Land}$ is used to determine the preferred successors of each base plan. This is a very effective resource when the primary heuristic gets stuck in a plateau.

$h_{Land}$ uses *landmarks*, fluents that must be satisfied in every solution plan of a MAP task, as the basis of its calculation. Prior to the planning process, agents jointly generate the Landmarks Graph, LG $= \{N, V\}$, where $N$ is a set of landmarks and $V$ is a set of *necessary orderings* between the landmarks. A necessary ordering $l' \leq_n l$ implies that the landmark $l'$ should be achieved before $l$ in all the solution plans for the task. The LG is then used to estimate the quality of the refinement plans in MH-FMAP. Given a plan $\Pi$, $h_{Land}(\Pi)$ returns an estimate of the quality of $\Pi$ as the number of single landmarks not satisfied in $\Pi$.

$h_{DTG}$ and $h_{Land}$ are designed to minimize the number of messages exchanged among the agents. The data structures of $h_{DTG}$, the DTGs, and of $h_{Land}$, the LG, remain unalterable throughout the multi-agent search, thus reducing the communication overhead. The use of static structures makes $h_{DTG}$ and $h_{Land}$ suitable heuristics for fully-distributed systems.

Private information is guaranteed during the heuristic evaluation of the plans since agents do not transmit private information regarding the plan at hand. Only heuristic estimates are transmitted during the evaluation process.

## Adaptations for the CoDMAP competition

Several adaptations were made in MH-FMAP to comply with the CoDMAP rules. Due to the complexity of the partial plans it synthesizes, MH-FMAP was originally designed as a graphical tool[1]. However, an additional command-line user interface was added to facilitate the use of scripts. Aside from the new user interface, other updates related to the input, output and communication infrastructure were applied in MH-FMAP.

**Input.** In order to give support to the new MA-PDDL specification, we developed a pre-parser that translates the factored MA-PDDL input into our customized language. More specifically, the private information specified in the MA-PDDL files is compiled into the `:shared-data` section.

Since the `:shared-data` section represents the public information of the task, the pre-parser automatically infers the non-private predicates from the MA-PDDL input and incorporates them into the `:shared-data` section of the problem files.

As a result of this change, MH-FMAP is able to support factored MA-PDDL and complies with the privacy rules established in the tasks of the CoDMAP benchmarks.

**Output.** MH-FMAP returns partial-order solution plans in which no total order is forced among the actions. In order to make the solutions compatible with the plan validator, we performed a conversion of the planner's output.

[1]http://users.dsic.upv.es/grupos/grps/tools.php

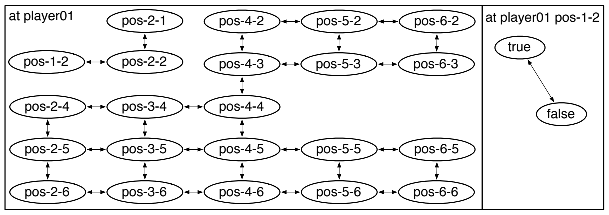


Figure 1: DTG for a state variable and a predicate

In the centralized track, plans are linearized as a sequence of actions. Whenever parallel actions are found, the conversion algorithm forces an arbitrary ordering among them.

Regarding the distributed track, the guidelines allow for parallel actions of different agents. In some cases, partial-order plans can contain parallel actions of the same agent. For this reason, in case the plan includes parallel actions of an agent, we prioritize the action that is followed by the longest sequence of actions. This way, we effectively minimize the makespan of the resulting plan as much as possible.

**Communications.** As in most distributed frameworks, communications play a central role in MH-FMAP. Agents communicate by means of the FIPA Agent Communication Language (O'Brien and Nicol 1998). Originally, the system used the Apache QPid[2] message broker, a component of the Magentix2[3] MAS platform, to handle communications.

During the early tests in the CoDMAP server, we noticed that QPid caused an important slowdown. For this reason, we rewrote the communications using sockets. This way, the current implementation of MH-FMAP is faster and more compact, since it does not rely on external components.

## Limitations of MH-FMAP in CoDMAP

The setup provided by the organizers of the CoDMAP has undeniable advantages. On the one hand, MA-PDDL has the potential to become a de facto standard for MAP. On the other hand, CoDMAP has provided the community with a standard benchmark of MAP tasks that can be used to compare the performance of the different approaches.

Nevertheless, the CoDMAP setup is far from being ideal for MH-FMAP. Despite MA-PDDL is an extension to *PDDL3.1*, the CoDMAP benchmarks are purely propositional. This fact directly affects the performance of MH-FMAP since its main heuristic, $h_{DTG}$, makes extensive use of the state variables. In order to adapt MH-FMAP to CoDMAP, we converted literals into binary variables within a true-false domain. Binary variables offer very poor information of the MAP task, which compromises the precision of $h_{DTG}$.

Figure 1 illustrates this issue by depicting two DTGs inferred from the `sokoban p01` task. The first DTG, synthesized from the state variable that keeps track of the position of agent `player01`, shows the complete game board for

[2]http://qpid.apache.org

[3]http://www.gti-ia.upv.es/sma/tools/magentix2

this task and all the connections among positions. In turn, in the propositional version of the task, MH-FMAP will infer a set of true-false DTGs, one per position in the board (Figure 1 on the right displays the DTG for the position `pos-1-2`).

In (Torreño, Sapena, and Onaindia 2015), we evaluated MH-FMAP through a benchmark that contains most of the domains of the CoDMAP. In these domains, state variables were extensively used to define initial states, preconditions and effects. However, the early tests performed for the CoDMAP setup, with the conversion of fluents into literals, showed worse results in terms of coverage and execution time. Therefore, a more sophisticated conversion procedure from literals to state variables is actually required to improve the performance of MH-FMAP when running it in the CoDMAP setup.

The quality of the plans returned by MH-FMAP is also limited by the competition rules. MH-FMAP is highly effective at parallelizing actions and minimizing the makespan of the solution plans (Torreño, Sapena, and Onaindia 2015). In order to comply with the CoDMAP rules, the makespan of the solution plans is noticeably increased in most cases.

Since the rules of the centralized track determine that the planner must return sequential plans, the makespan in this case always equals the number of actions of the solution plan. The distributed track allows for parallel actions of different agents; however, each agent must return a sequence of actions. Since MH-FMAP is designed to reason with parallel plans, the quality of the solution plans in the CoDMAP benchmarks diminishes in some cases.

## Conclusions

In this paper, we have presented MH-FMAP, the MAP system we submitted to the centralized and distributed tracks of the 2015 CoDMAP. MH-FMAP is a fully-distributed system that performs a complete and suboptimal A* search and synthesizes partial-order plans contributed by several agents built in a forward-chaining fashion. MH-FMAP incorporates a novel multi-heuristic search scheme that alternates two global heuristics, $h_{DTG}$ and $h_{Land}$. These heuristics require the interaction of agents in order to assess the quality of the plans and rely on immutable and privacy-preserving structures that are calculated in pre-planning time.

MH-FMAP takes factored MA-PDDL tasks as an input and maintains the privacy defined in the task description throughout the planning process. During planning, agents communicate partial plans and occlude the preconditions and effects that are private according to the MA-PDDL files.

The CoDMAP setup presents some disadvantages for our planner. On the one hand, the CoDMAP benchmarks consist of propositional planning tasks. This worsens the performance of MH-FMAP, which is optimized to deal with tasks defined through state variables. On the other hand, the quality of the solution plans is also affected by the rules, since we are forced to provide either a sequential plan or a sequence of actions per agent.

## Acknowledgments

This work has been partly supported by the Spanish MICINN under projects Consolider Ingenio 2010 CSD2007-00022 and TIN2014-55637-C2-2-R, the Valencian Prometeo project II/2013/019, and the FPI-UPV scholarship granted to the first author by the Universitat Politècnica de València.

## References

Borrajo, D. 2013. Multi-agent planning by plan reuse. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1141–1142.

Crosby, M.; Rovatsos, M.; and Petrick, R. 2013. Automated agent decomposition for classical planning. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, 46–54.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, 140–147.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)* 161–170.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Kovacs, D. L. 2011. Complete BNF description of PDDL3.1. Technical report.

Nissim, R., and Brafman, R. 2012. Multi-agent A* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1265–1266.

Nissim, R.; Brafman, R.; and Domshlak, C. 2010. A general, fully distributed multi-agent planning algorithm. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1323–1330.

O'Brien, P., and Nicol, R. 1998. FIPA - towards a standard for software agents. *BT Technology Journal* 16(3):51–59.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39(1):127–177.

Röger, R., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 246–249.

Torreño, A.; Onaindia, E.; and Sapena, O. 2014. FMAP: Distributed cooperative multi-agent planning. *Applied Intelligence* 41(2):606–626.

Torreño, A.; Sapena, O.; and Onaindia, E. 2015. Global heuristics for distributed cooperative multi-agent planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 225–233.

# PSM-based Planners Description for CoDMAP 2015 Competition

**Jan Tožička, Jan Jakubův, Antonín Komenda**
Agent Technology Center, Czech Technical University, Prague, Czech Republic,
{jan.tozicka, jan.jakubuv, antonin.komenda}@agents.fel.cvut.cz

**Abstract**

One possible approach to domain independent multiagent planning (DMAP) is to equip all the agents with information about public actions of other agents. This allows every agent to plan public actions for other agents. This approach can be used together with *planning state machines* (PSM) which provide a compact representation of sets of plans. In a PSM-based planner, every agent keeps generating plans and stores the generated plans in a PSM. This process continues until a plan acceptable for all the agents is found.

We describe PSM-based planners submitted to the Competition of Distributed and Multiagent Planners (CoDMAP) held at DMAP Workshop of ICAPS 2015. We describe two configurations of a PSM-based planner which differ in the set of used features, and in the amount of private information revealed to other agents. Both of these two configurations can be used both in a distributed and a centralized setting. Hence we describe altogether four PSM-based planners submitted to the competition.

## Introduction

In domain independent multiagent planning (DMAP) several cooperative agents tries to find out a distributed plan which leads to a common goal. The most widely used STRIPS-based model of DMAP is MA-STRIPS (Brafman and Domshlak 2008) which defines basic concepts of *public* and *internal* information. In MA-STRIPS, agents can either plan only with their own actions and facts and inform the other agents about public achieved facts, as for instance in the MAD-A* planner (Nissim and Brafman 2012), or can also use other agents' public actions provided that the actions are stripped of the private facts in preconditions and effects. Thus agents plan actions, in a sense, for other agents and then coordinate the plans (Tožička, Jakubův, and Komenda 2014).

Recently introduced planning approach (Tožička, Jakubův, and Komenda 2014) equip all the agents with projections of public actions of other agents. These projections are called *external actions*. *Planning state machines* (PSM) are basically a finite automatons representing a set of plans.

Two basic operations are naturally defined on PSM's: (1) public projection (restriction to public information) of a PSM, and (2) intersection of two public PSM's. A generic distributed PSM-based planner keeps generating new plans for every involved agent. The generated plans of one agent are stored in a PSM from which a public projection can be computed. Public projections of PSM's of all the agents contain only public information a thus can be freely exchanged among the agents. Hence their intersection can be computed and an non-empty intersection constitutes a solution of the problem at hand.

Basic architecture of a PSM-based planner was described recently (Tožička, Jakubův, and Komenda 2014). The following improvements and modifications were implemented in the planners submitted to CoDMAP.

**(EXT-V)** Extension with plan analysis and verification of generated plans (Jakubův, Tožička, and Komenda 2015).

**(EXT-R)** Extension with an initial (delete-)relaxed plan landmark. This extension is briefly described in the next section.

**(EXT-D)** Extension with analysis of internal dependencies of public actions introduced quite recently (Tožička, Jakubův, and Komenda 2015).

All the above extensions are briefly described in the following section. The extensions can be furthermore combined together and we have decided to use the following two planner configurations for the competition.

**(PSM-VR)** Basic PSM planner with extensions EXT-V and EXT-R.

**(PSM-VRD)** Basic PSM planner with all extensions, that is, EXT-V, EXT-R, and EXT-D.

Both the planner configurations can be used both in a centralized and a distributed setting which gives us together four planners submitted to the competition. All the planners are **complete** but **not optimal**. All the planners use **MA-STRIPS privacy classification**. In the centralized setting, planners compute MA-STRIPS classification themselves, while in the distributed setting, the classification provided in input files is used. All the planners use **agent factorization from the input files**. Differences and features are described in the following sections.

## Generic PSM Planner Architecture

PSM planner main idea is based on planning with external actions, where a *local planning problem* is constructed for every agent $\alpha$. A *local planning problem* $\Pi \triangleright \alpha$ of agent $\alpha$, also called *projection of* $\Pi$ to $\alpha$, is a classical STRIPS problem containing all the actions of agent $\alpha$ together with external actions, that is, public projections other agents public actions. The local problem of $\alpha$ is defined only using the facts of $\alpha$ and public facts, hence no private information is revealed.

## Theoretical Background

This section briefly describes conditions which allow us to compute a solution of the original MA-STRIPS problem $\Pi$ from solutions of local problems $\Pi \triangleright \alpha$.

A *plan* $\pi$ is a sequence of actions. A *solution* of $\Pi$ is a plan $\pi$ whose execution transforms the initial state to a superset of the goals. A *local solution* of agent $\alpha$ is a solution of $\Pi \triangleright \alpha$. Let $\mathsf{sols}(\Pi)$ denote the set of all the solutions of MA-STRIPS or STRIPS problem $\Pi$. A *public plan* $\sigma$ is a sequence of public actions. The *public projection* $\pi \triangleright \star$ *of plan* $\pi$ is the restriction of $\pi$ to public actions.

A public plan $\sigma$ is *extensible* when there is $\pi \in \mathsf{sols}(\Pi)$ such that $\pi \triangleright \star = \sigma$. Similarly, $\sigma$ is $\alpha$*-extensible* when there is $\pi \in \mathsf{sols}(\Pi \triangleright \alpha)$ such that $\pi \triangleright \star = \sigma$. Extensible public plans give us an order of public actions which is acceptable for all the agents. Thus extensible public plans are very close to solutions of $\Pi$ and it is relatively easy to construct a solution of $\Pi$ once we have an extensible public plan. This process is described below.

The following theorem establishes the relationship between extensible and $\alpha$-extensible plans. Its direct consequence is that to find a solution of $\Pi$ it is enough to find a local solution $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ which is $\beta$-extensible for every agent $\beta$.

**Theorem 1.** *Public plan $\sigma$ of $\Pi$ is extensible if and only if $\sigma$ is $\alpha$-extensible for every agent $\alpha$.*

The theorem above suggests a distributed multiagent planning algorithm described in Algorithm 1. Every agent executes this algorithm, possibly on a different machine (in centralized version, only the loop is executed in parallel by all the agents). Every agent keeps generating new solutions of its local problem and stores solution projections in set $\Phi_\alpha$. These sets are exchanged among all the agents so that every agent can compute their intersection[1] $\Phi$. Once the intersection $\Phi$ is non-empty, the algorithm terminates yielding $\Phi$ as the result. Theorem 1 ensures that every public plan in the resulting $\Phi$ is extensible.

The sets of plans $\Phi_\alpha$ are represented using *planning state machines*, PSMs, which are based on common finite state machines. PSMs allows to effectively represent large sets of plans and also to effectively implement the intersection of different agents sets of plans. Consult (Tožička, Jakubův, and Komenda 2014) for more details on algorithms and implementation.

[1]In our competition implementation, the intersection is computed only once by a specialized *broker* agent that also mediates the communication among the agents.

**Algorithm 1:** PSM Planner Algorithm.

```
1  Function PSMPlanner($\Pi \triangleright \alpha$) is
2      $\Pi_\alpha \leftarrow$ GroundProblem($\Pi \triangleright \alpha$);
3      if EXT-D then
4          $\Pi_\alpha \leftarrow$ ComputeDependencies($\Pi_\alpha$);
5          if all agents published dependencies then
6              return any solution of $\Pi_\alpha \triangleright \star$;
7          end
8      end
9      if EXT-R then
10         $\pi_R \leftarrow$ CreateRelaxedPlan($\Pi$);
11         $\Pi_\alpha \leftarrow$ AddLandmarks($\Pi_\alpha, \pi_R \triangleright \alpha$);
12     end
13     $\Phi_\alpha \leftarrow \emptyset$;
14     loop
15         $\pi_\alpha \leftarrow$ GenerateNewPlan($\Pi_\alpha$);
16         if EXT-V then
17             $\pi_\alpha \leftarrow$ VerifyPlan($\pi_\alpha \triangleright \star$);
18         end
19         $\Phi_\alpha \leftarrow \Phi_\alpha \cup \{\pi_\alpha \triangleright \star\}$;
20         exchange public plans $\Phi_\beta$ with other agents;
21         $\Phi \leftarrow \bigcap_{\beta \in \mathsf{agents}(\Pi)} \Phi_\beta$;
22         if $\Phi \neq \emptyset$ then
23             return $\Phi$;
24         end
25         forall the $\beta \in \mathsf{agents}(\Pi)$ do
26             $\Pi_\alpha \leftarrow$ AddLandmarks($\Pi_\alpha, \Phi_\beta$);
27         end
28     end
29 end
```

The rest of this section describes methods of the basic version of the PSM algorithm (that is, without EXT-D, EXT-R, and EXT-V). The extensions are briefly described in the next section.

## Problem Grounding

All the input problems are provided in a PDDL files which might contain parametric actions. Our algorithm requires grounded actions and hence the input is grounded as the first step. Specific implementation of method `GroundProblem` is crucial. It has to generate all possible usable action instances but it should not generate any action instance that cannot be really performed. Otherwise, other agent could use that action in their plans. Since this method is not in the center of our current research, it is a bottleneck which significantly influences planner performance in certain domains. Implementations of a grounding algorithm differ in the centralized and distributed settings.

**Centralized version.** First step of the centralized grounding algorithm is to merge all the problems of all the agents into a single problem. Then the *translate.py* algorithm of

*Fast Downward* is used to find all reachable grounded action instances. Agent problems are then separately grounded to match these actions and the algorithm continues with this grounded multi-agent problem.

**Distributed version.** In distributed versions, method `GroundProblem` protects agents privacy. Agents create distributed planning graph and then ground their problem to actions that appear in this planning graph. The planning graph is additionally reused by extension EXT-R described in the next section.

## Public Solution Search

Every agent keeps generating new plans and shares their public projections with other agents. Local planning problems are common STRIPS problems which can be solved by any classical STRIPS planner. The only special requirement on the underlaying planner is that the planner must be able to generate a plan which differs from the previously generated plans. In our implementation, method `GenerateNewPlan` uses a modified version of FastDownward[2] planner (Jakubův, Tožička, and Komenda 2015).

The generation of new local plans is further guided by the knowledge obtained from PSM's of other agents so that the plan generation is driven towards the plans generated by other agents. The knowledge from PSM's of other agents is compiled into the local problem by function `AddLandmarks` using the principle of *soft action landmarks* and *action costs* (Jakubův, Tožička, and Komenda 2015).

## Global Solution Reconstruction

The algorithm ends with a public solution $\sigma$ which is an extensible public plan. To reconstruct a solution of the original problem $\Pi$, each agent $\alpha$ creates a classical STRIPS *reconstruction* problem $\Pi_\alpha$ which contains only internal actions of agent $\alpha$. To problem $\Pi_\alpha$, the generated public solution $\sigma$ is added as action landmarks just like in method `AddLandmarks`. The only difference is that *hard action landmarks* are used to ensure that all the landmark actions are executed. The public plan is $\alpha$-extensible and thus problem $\Pi_\alpha$ is solvable. Agents then merge together solutions of all the reconstruction problems and create a solution of the original problem $\Pi$.

## PSM Planner Extensions

Basic PSM algorithm, where agents iteratively generate new plans, exchange their public PSMs and check whether their intersection is nonempty, can be extended to achieve better results. In this section we describe three implemented extensions. First extension, EXT-V, significantly reduces number of misleading landmarks by use of plan verification. Second extension, EXT-R computes a relaxed solution $\pi$ and uses its public projection $\pi \triangleright \star$ as initial landmarks. The last extension, EXT-D, allows agents to exchange some abstract rules describing their internal knowledge.

[2] `http://www.fast-downward.org/`

## EXT-V: Plan Verification and Analysis

PSM planner from Algorithm 1 uses public plans generated by other agents as landmarks to guide future plan search. However, it is desirable to use only extensible plans to guide plan search because non-extensible plans can not lead to a non-empty public PSMs intersection. Every generated plan should be verified by other agents in order to determine its extensibility. However, extensibility (or $\alpha$-extensibility) checking is expensive and thus we propose only an approximative method of plan verification. This extension of a PSM planner has already been proposed with promising results (Jakubův, Tožička, and Komenda 2015).

In order to approximative $\alpha$-extensibility we use generic process calculi type system scheme POLY✶ (Makholm and Wells 2005; Jakubův and Wells 2010) to determine the least provably unreachable action in a public plan. The result of POLY✶ analysis is either indeterminate, or the index an action which is guaranteed unreachable. However, some action prior to the indexed action might be actually unreachable as well because POLY✶ analysis provides only a polynomial time approximation of $\alpha$-extensibility.

Described $\alpha$-extensibility approximation suggests the following implementation of method `VerifyPlan(`$\pi_\alpha \triangleright \star$`)`. When agent $\alpha$ generates a new plan $\pi_\alpha$, it sends its public projection $\pi_\alpha \triangleright \star$ to all the other agents. Once other agent $\beta$ receives $\pi_\alpha \triangleright \star$, it runs the above $\beta$-extensibility check on it, and sends the result back to agent $\alpha$. Agent $\alpha$ collects analysis results from all the other agents and strips the plan $\pi$ to prefix acceptable for all agents. Finally only the stripped plan is used as a landmark to guide future plan search. We can even further speed up convergence of EXT-V by forbidding all plans with public prefixes matching plans already refused by other agents.

## EXT-R: Initial Relaxed Plan Landmark

The delete effect relaxation, where delete effects of actions are ignored, has proved its relevance both in STRIPS planning (Hoffmann and Nebel 2001), and recently also in MA-STRIPS planning (Stolba and Komenda 2014). It is known that to find a solution of a relaxed problem is an easier task than to find a solution of the original problem.

Our algorithm first creates solution $\pi_R$ of the relaxed problem and then transforms it into *initial landmarks*, which are used by all the agents. When $\pi_R \triangleright \star$ is extensible then every agent $\alpha$ is likely to generate local solution $\pi_\alpha$ such that $\pi_\alpha \triangleright \star = \pi_R \triangleright \star$ in the first iteration. In that case the algorithm terminates directly in the first iteration causing a dramatic speed-up. Otherwise, the initial landmark is forgotten by all the agents and the algorithm continues by the second iteration as before.

**Centralized EXT-R.** Actions of all agents are relaxed and merged into a single problem. One agent then solves this relaxed problem. In this implementation, every agent **reveals all its private information** to the agent which computes the relaxed solution.

**Distributed EXT-R.** Agents compute distributed planning graph by trying to fulfill goals or preconditions of reachable actions of other agents. In the case of domains where actions model use of *limited resources*, it is better to find solutions where each agent fulfill few of the goals, because limiting resources are relaxed away. Therefore, during the extraction of the relaxed plan from the distributed planning graph, agents are lazy and try to fulfill only one goal and they pass the remaining goals together with new subgoals to another agent. More effective implementation would make use of *exploration queues* (Stolba and Komenda 2014).

Distributed implementation of EXT-R **respects privacy by not revealing private information** in the form of internal facts or actions to other agents.

## EXT-D: Internal Dependencies of Public Actions

One of the benefits of planning with *external actions* is that every agent can plan separately its local problem which involves planning of actions for other agents (external actions). Other agents can then only verify whether a plan generated by another agent is $\alpha$-extensible for them. A con of this approach is that agents have only a limited knowledge about external actions because internal facts are removed by projection. Thus it can happen that agent $\alpha$ plans external actions inappropriately in a way that the resulting public plan is not $\beta$-extensible for some other agent $\beta$. A method to overcome this con was introduced quite recently (Tožička, Jakubův, and Komenda 2015).

With EXT-D extension, agents are equipped by extended information about applicability of external actions in the form of *dependency graphs* (Tožička, Jakubův, and Komenda 2015). The information encapsulated in a dependency graph is equivalent to a set of conditions of the form $\langle a, S, F\rangle$ meaning that action $a$ must be preceded by some action from $S$ not followed by any action from $F$. Agents not capable of expressing dependencies of their public actions in the above way simply do not publish anything.

Information from dependency graphs is compiled into local problems using additional facts. These additional facts might resemble some internal facts of agents. However, in all but the most trivial cases, published information is so condensed that original internal facts cannot be reconstructed.

## Summary

We have submitted the following four PSM-based planners to the CoDMAP competition.

(1) Centralized PSM-VR (extensions EXT-V and EXT-R)

(2) Centralized PSM-VRD (EXT-V, EXT-R, and EXT-D)

(3) Distributed PSM-VR (extensions as (1))

(4) Distributed PSM-VRD (extensions as (2))

As noted in the introduction, all the planners are **complete** but **not optimal**. All the planners use MA-STRIPS privacy classification and all the planners use agent factorization from the input files.

In Distributed PSM-VR, no private information is explicitly reveled in the form of internal facts and actions. In Distributed PSM-VRD, a limited private information encoded in dependency graphs is revealed to other agents. Furthermore, some additional private information might be implicitly revealed by the way agents act and respond during planning, plan verification, and relaxed planning graph construction.

In Centralized versions, all private information in the form of all the internal facts and actions is revealed to other agents for the reasons of problem grounding and relaxed planning graph construction. Apart from this, centralized versions respect privacy as the corresponding distributed versions.

## Acknowledgements

This research was supported by the Czech Science Foundation (grant no. 13-22125S) and by the Ministry of Education of the Czech Republic within the SGS project no. SGS13/211/OHK3/3T/13.

## References

Brafman, R., and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of ICAPS'08*, volume 8, 28–35.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)* 14:253–302.

Jakubův, J., and Wells, J. B. 2010. Expressiveness of generic process shape types. In *TGC'10*, volume 6084 of *LNCS*, 103–119. Springer.

Jakubův, J.; Tožička, J.; and Komenda, A. 2015. Multiagent Planning by Plan Set Intersection and Plan Verification. In *Proceedings ICAART'15*.

Makholm, H., and Wells, J. B. 2005. Instant polymorphic type systems for mobile process calculi: Just add reduction rules and close. In *ESOP'05*, volume 3444 of *LNCS*, 389–407. Springer.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*, 1265–1266.

Stolba, M., and Komenda, A. 2014. Relaxation heuristics for multiagent planning. In *24th International Conference on Automated Planning and Scheduling (ICAPS)*, 298–306.

Tožička, J.; Jakubův, J.; and Komenda, A. 2014. Generating multi-agent plans by distributed intersection of Finite State Machines. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, 1111–1112.

Tožička, J.; Jakubův, J.; and Komenda, A. 2015. On internally dependent public actions in multiagent planning. In *DMAP Workshop of International Conference on Automated Planning and Scheduling (ICAPS)*.

# Author Index

Borrajo, Daniel …… 1, 11

Crosby, Matthew …… 4

Fernández, Susana …… 1

Fišer, Daniel …… 8

Jakubův, Jan …… 29

Kambhampati, Subbarao …… 17

Komenda, Antonín …… 8, 21, 29

Lipovetzky, Nir …… 14

Luis, Nerea …… 11

Muise, Christian …… 14

Onaindia, Eva …… 25

Ramirez, Miquel …… 14

Sapena, Óscar …… 25

Sreedharan, Sarath …… 17

Štolba, Michal …… 8, 21

Torreño, Alejandro …… 25

Tožička, Jan …… 29

Zhang, Yu …… 17