# Multi-Agent Planning by Plan Reuse

# (Extended Abstract)

Daniel Borrajo
Departamento de Informática
Universidad Carlos III de Madrid
dborrajo@ia.uc3m.es

## ABSTRACT

Generating plans for a single agent has been shown to be a difficult task. If we generalize to a multi-agent setting, the problem becomes exponentially harder in general. The centralized approach where a plan is jointly generated for all agents is only possible in some applications when agents do not have private goals, actions or states.

We describe in this paper an alternative approach, MAPR (Multi-Agent Planning by plan Reuse), that considers both the agents private and public information. We have been inspired by iterative Multi-Agent Planning (MAP) techniques as the one presented in [3]. MAPR first assigns a subset of public goals to each agent, while each agent might have a set of private goals also. Then, MAPR calls the first agent to provide a solution (plan) that takes into account its private and public goals. MAPR iteratively calls each agent with the solutions provided by previous agents. Each agent receives its own goals plus the goals of the previous agents. Thus, each agent solves its problem, but taking into account the previous agents solutions. Since previous solutions might consider private data, all private information from an agent is obfuscated for the next ones. Since each agent receives the plan from the previous agent that implicitly considers the solutions to all previous agents, instead of starting the search from scratch, it can also reuse the previous whole plan or only a subset of the actions. Experiments show that MAPR outperforms in several orders of magnitude state-of-the-art techniques in the tested domains.

## Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

## Keywords

Automated planning; Multi-Agent Planning

## 1. MULTI-AGENT PLANNING TASK

A single-agent STRIPS planning task can formally defined as a tuple $\Pi = \{F, A, I, G\}$, where $F$ is a set of propositions, $A$ is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a set of goals. Each action $a \in A$ is described by a set of preconditions, and a set of effects. The solution of planning tasks are sequences of actions $\pi = (a_1, \ldots, a_n)$ such that, if applied in order, they result in a state $s$, where goals are true, $G \subseteq s$. In a multi-agent setting, the planner generates a plan for a set of agents, $\Phi = \{\phi_1, \ldots, \phi_m\}$. We define the MAP task as a set of planning subtasks, one for each agent, $M = \{\Pi_1, \ldots, \Pi_m\}$. Each planning subtask can be defined as a single-agent planning task, $\Pi_i = \{A_i, F_i, I_i, G_i\}$. All these components have a public part, that can be shared with the rest of agents, and a private part. In order to differentiate those parts, we provide each agent with the list of domain predicates and types that are private. Thus, all actions, states and goals of each agent can have a private and a public part.

**Appears in:** *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.


## 2. MULTI-AGENT PLANNING IN MAPR

Figure 1 shows a high-level description of the algorithm. We use @ to express obfuscated information. It takes as input a MAP task (domain, problem and agents description as explained in the previous section), a goal assignment strategy, the planner to be used by the first agent, and a second planner (it might be the same one) to be used by the following agents. The algorithm is composed of six main steps shown in bold face in the algorithm.

Function MAPR $(M, GA, FP, SP)$: plan

---

$M = \{\Pi_1, \ldots, \Pi_m\}$: MAP task
$GA$: goal assignment strategy
$FP$: first planner
$SP$: second planner

---

$M \leftarrow$ **Assign** public goals $(GA, M)$
Repeat until **Termination**
  $\pi_1 \leftarrow$**First-Plan**$(FP, \Pi_1)$
  For each remaining agent $\phi_j, 1 < j \leq m$
    $\phi_{j-1}$ **Obfuscates** its private information, $S^{@}_{j-1}$:
      the plan $\pi^{@}_{j-1}$ and
      the problem $\Pi^{@}_{j-1} = \{F^{@}_{j-1}, A^{@}_{j-1}, I^{@}_{j-1}, G^{@}_{j-1}\}$
    $\phi_{j-1}$ **Communicates** $S^{@}_{j-1}$ to agent $\phi_j$
    $\phi_j$ creates a new planning task:
      $\Pi_j \leftarrow$ **Merge**$(\Pi_j, S^{@}_{j-1})$
    $\pi_j \leftarrow$**Second-plan**$(SP, \Pi_j)$
If solved, return last plan

**Figure 1: High level description of MAPR planning algorithm.**

**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. If the relaxed plan heuristic detects a dead-end, then $c(g, \phi) = \infty$. This will define a cost matrix, $c(G, \Phi)$. Next, we have devised four goals assignment schemes. **all-achievable (AA):** MAPR assigns each goal $g$ to all agents $\phi_i$ such that $c(g, \phi_i) < \infty$. **rest-achievable (RA):** 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. **best-cost (BC):** MAPR assigns each goal $g$ to the agent that can potentially achieve it with the least cost, $\arg\min_{\phi_i \in \Phi} c(g, \phi_i)$. **load-balance (LB):** MAPR first computes the average number of goals per agent, $k = \frac{|G|}{m}$. Then, it starts assigning goals to agents as in best cost. When it has assigned $k$ goals to an agent, it stops assigning goals to that agent.

**Planning**. Once goals have been assigned to a subset of agents $\Phi' \subseteq \Phi$, planning starts by calling the first agent to solve its planning task. The task will be composed of its private planning task and its assigned public goals. If it does not solve the problem, it just passes the empty plan to the next agent. It could be either because there is no such plan, or, most frequently, because its plan needs some propositions to be achieved by the rest of agents plans. So, it will wait until, eventually, it will be called again to solve again the planning problem, but with some extra information coming from the other agents planning episodes. The following planning episodes can either use a planner or a replanner. In the latter case, apart from the domain and problem definitions, replanners take a previous solution as input [1].

**Obfuscation**. 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. Obfuscating is a two steps process. First, a random substitution $\sigma$ is generated for the names of all private predicates, actions and objects. The second step consists of applying the substitution to the plan. An augmented obfuscated solution $S_j^@$ consists of the obtained plan and the set of components that are needed by the rest of agents to regenerate that solution.

**Communication**. Each agent communicates $S_j^@$ to the next agent. We assume there is no noise in the communication. The size of the messages is linear with respect to the plan size and initial state size.

**Merging**. Each agent $a_{j+1}$ receives $S_j^@$ and builds a new planning problem by adding the instantiated actions to its actions set, the goals to its own goals, the private previous initial state to its own initial state and all new fluents to its own fluents set.

**Termination**. As soon as the last agent finds a plan achieving all goals, the whole planning process finishes. If the last agent does not find a plan and there is still time, we perform another iteration over all agents again, but with the accumulation of goals. Starting in the second iteration, as soon as an agent finds a solution, then the whole planning task finishes, since it incorporates all goals from all agents. The planning process will terminate with failure only if the time or memory bounds are reached.

# 3. EXPERIMENTS AND RESULTS

MAPR was compared with MAP-POP, since it represents the state-of-the-art on MAP [4], and a centralized approach (LAMA-FIRST).[1] Rovers and Satellite were used as in [4]. The four goal assignment strategies defined were also used. LAMA-FIRST (LF) was selected as the first planner, and LAMA-FIRST and LPG-ADAPT [1] as second planners. Time bound was 600 seconds.[2] Table 1 shows the results in the Rovers domain. MAPR is able to solve all IPC instances with almost all configurations in up to three orders of magnitude less time. Also LPG-ADAPT improves over LAMA-FIRST. Similar results were obtained in the Satellite domain (all problems solved with up to two orders of magnitude improvement on time).

| | | LF | | | | LPG-ADAPT | | | |
|---|---|---|---|---|---|---|---|---|---|
| MAP-POP | LF | AA | RA | BC | LB | AA | RA | BC | LB |
| 0.44 | 0.00 | 0.17 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 | 0.16 | 0.15 |
| 0.34 | 0.00 | 0.15 | 0.15 | 0.15 | 0.15 | 0.16 | 0.15 | 0.15 | 0.16 |
| 0.8 | 0.00 | 0.36 | 0.36 | 0.36 | 0.35 | 0.37 | 0.36 | 0.37 | 0.37 |
| 0.9 | 0.00 | 0.36 | 0.35 | 0.35 | 0.35 | 0.38 | 0.36 | 0.39 | 0.38 |
| 2.15 | 0.00 | 0.40 | 0.41 | 0.38 | 0.38 | 0.40 | 0.39 | 0.38 | 0.39 |
| 2.17 | 0.00 | 0.44 | 0.43 | 0.43 | 0.43 | 0.43 | 0.41 | 0.41 | 0.42 |
| 3.75 | 0.00 | 0.69 | 0.18 | 0.39 | 0.62 | 0.65 | 0.18 | 0.40 | 0.59 |
| 60.35 | 0.01 | 1.01 | 0.42 | 0.64 | 0.90 | 0.92 | 0.40 | 0.64 | 0.90 |
| 15.77 | 0.01 | 1.01 | 0.69 | 0.96 | 0.90 | 0.94 | 0.65 | 0.89 | 0.89 |
| | 0.01 | 1.09 | 0.46 | 0.45 | 0.96 | 0.96 | 0.43 | 0.43 | 0.90 |
| 10.39 | 0.01 | 1.08 | 0.47 | 0.72 | 0.97 | 0.96 | 0.44 | 0.67 | 0.90 |
| 3.17 | 0.00 | 1.01 | 0.65 | 0.92 | 0.90 | 0.95 | 0.66 | 0.90 | 0.92 |
| | 0.02 | 1.18 | 0.54 | 0.55 | 1.17 | 1.03 | 0.47 | 0.51 | 0.98 |
| 47.10 | 0.01 | 1.14 | 0.52 | 0.51 | 0.98 | 1.17 | 0.47 | 0.48 | 0.93 |
| 20.68 | 0.01 | 1.24 | 0.59 | 0.54 | 1.14 | 1.06 | 0.51 | 0.48 | |
| 195.97 | 0.02 | 1.32 | 0.28 | 0.56 | 1.18 | 1.12 | 0.27 | 0.50 | 0.99 |
| | 0.03 | 2.17 | 1.23 | 1.64 | 2.06 | 1.66 | 1.03 | 1.38 | 1.63 |
| | 0.04 | 2.63 | 1.21 | 1.51 | 2.28 | 1.84 | 0.93 | 1.18 | 1.71 |
| | 0.13 | 2.60 | 1.54 | 1.52 | 3.83 | 1.79 | 1.12 | 1.02 | 1.87 |
| | 0.14 | 5.12 | 1.34 | 5.14 | 3.99 | 2.76 | 0.86 | | 2.46 |

**Table 1: Planning time (in seconds) in the Rovers domain.**

# 4. ACKNOWLEDGMENTS

We would like to thank Alejandro Torreño and Naz Rissim. This work has been partially supported by MICINN projects TIN2008-06701-C03-03 and TIN2011-27652-C03-02.

# 5. REFERENCES

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

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

[3] A. Jonsson and M. Rovatsos. Scaling up multiagent planning: A best-response approach. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, pages 114–121, 2011.

[4] A. Torreño, E. Onaindía, and O. Sapena. An approach to multi-agent planning with incomplete information. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'12)*, 2012.

[1]FastDownward code [2]. We used only one run of lazy greedy best first search with actions costs, and FF and landmark count heuristic with preferred operators.

[2]We used a 2.6GHz Intel Core i7 with 4Gb of RAM on MacOS X.