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 $i = 0, 1, \ldots, |F|$ 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. In *Proc. ECAI*, 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.