https://doi.org/10.31256/Ts2Po9M

Using Plan Libraries for Improved Plan Execution

Ionut Moraru
Dept. of Informatics
King's College London
London, UK
ionut.moraru@kcl.ac.uk

Gerard Canal
Dept. of Informatics
King's College London
London, UK
gerard.canal@kcl.ac.uk

Simon Parsons
Lincoln Centre for Autonomous Systems
University of Lincoln
Lincoln, UK
sparsons@lincoln.ac.uk

***Abstract*—The difficulty of task planning for robotic agents arises from the stochastic nature of their environment and the high cost of a failure during execution meaning frequent replanning is required. One way to address this problem is to make use of a pre-defined plan library. In this paper, we present work that combines a plan library with task planning. Initial results show that such an approach alleviates the computational burden of synthesising plans, while providing the same level of autonomy as using a planner that starts from scratch.**

***Index Terms*—Task Planning, BDI, Case-Based Reasoning, Plan Library, ROSPlan**

I. Introduction

In order for robots to become helpful in dynamic and stochastic environments, their reasoning about their actions must combine two qualities: autonomy and speed. Autonomy to decide for themselves how to achieve their goals, regardless of the situation they are placed in [1] and speedy reasoning so that they can perform in dynamic environments where plans can become unusable if a robot takes too long to synthesise them. One way to ensure that the robot has a high degree of autonomy is by reasoning directly about the state of the environment. Most systems that do this are based on STRIPS [2], and help the robot to come up with a sequence of actions (i.e. plans), from a set of available operators that would satisfy a set of explicit goals given in a planning task. The downside of this approach is that it is PSPACE-complete in the simplest versions (propositional planning), and becomes more difficult the more expressive the models become [3].

One approach to deal with this computational complexity is to make use of pre-defined plans that a robot then looks up rather than having to plan from scratch. Several examples within this paradigm have been based on the Belief-Desire-Intention (BDI) model [4], such as Jason [5], which has a predefined Plan Library. The downside to this approach is that the robot is limited to the prescribed behaviours, trading some autonomy for computational efficiency. Meneguzzi [6] describes the approaches that try to combine BDI with state based planning, but in a limited manner as plans are not added into the Library.

Here we describe a complementary approach. It starts with no plan library, carries out task planning to achieve goals, stores the plans that are generated, and reuses them where possible. We implemented this idea by introducing a Plan Library node in ROSPlan [7], a middle-ware layer between task planners and the Robot Operating System (ROS). This Plan Library node checks if the current planning task has already been solved. If it has, rather than invoking the *planner*, the previous plan is sent to the *acting component*.

II. Plan Library for ROSPlan

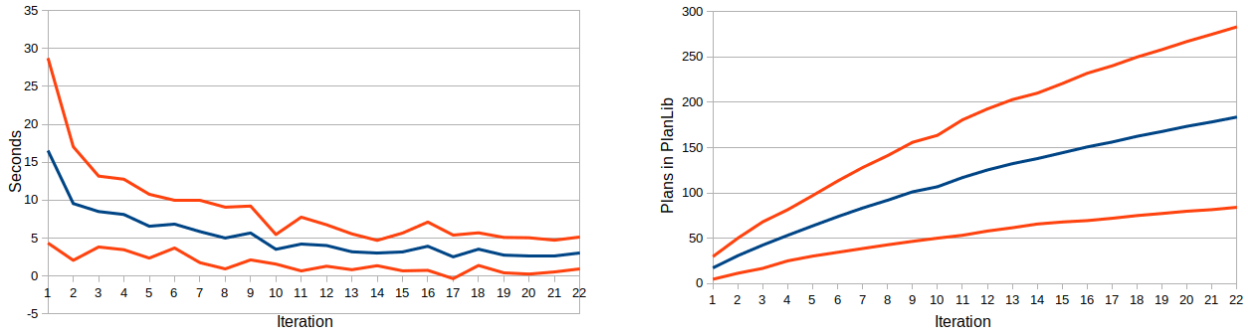
The Plan Library is a proxy for the Planner Interface from the default ROSPlan framework. Previously solved problems together with their plans (stored on in YAML format), are loaded as a dictionary during initialisation.

When the node receives a planning task as a PDDL file from ROSPlan's Problem Interface, it parses it into three parts: *types* indicating the types of instances involved in the problem, *init* defining the predicates in the initial state, and *goal* integrating the variables from the goal state. Next, the node iterates over the Plan Library, matching its initial state and goal elements with those of the problem it needs to solve. If there is a match, the iteration is interrupted and the plan from that Plan Library element is sent to the Parsing Interface.

If no problem from the Plan Library is found to match, then the problem is sent to the planner via the Planner Interface node. If it returns a solution to the problem, then it will be added as a new entry to the Plan Library along with the tasks *types*, *init*, and *goal*. The proposed methodology is PDDL agnostic, accepting all planning languages available in ROSPlan.

III. Empirical Evaluation

We used the temporal domain Office, consisting of a robot-assistant operating in a dynamic office setting. The robot is tasked with navigating the environment and bringing different office resources (e.g.: mugs, post or papers) to the people in it, asking humans for help when needed. We created 10 problems, increasing in planning difficulty — taking from 3 to 20 seconds to compute, and varying in length between 40 and 140 actions. Each of these problems was then solved and its robot execution simulated. Each action had varying probabilities of failing during execution (between between 0.5 and 0.9). When an action failed, a new plan was computed from that state. We ran each problem 40 times sequentially, meaning that the plan library was not cleared between these iterations, allowing the robot to learn through additions to the plan library. We compare our method with a standard version of ROSPlan without a Plan Library. We used the POPF planner



(a) (b)

Fig. 1: Summary plots averaged across all action failure probabilities on (a) Total planning time in seconds; (b) Number of plan in the plan library. Average values in blue, with ± 1 std deviation (red).

[8] with a timeout of 30 seconds to compute the plans. The overall system was given 500 seconds for each problem to be solved and reach the goal.

Over the course of our experiments, the modified version of ROSPlan reached the goal 1403 times out of the 2000 problem instances it faced (timing out the remaining times). The system spent an average of 0.022 seconds (7.2% of the total planning time) searching for plans in the Plan Library. The problem requested from the system was found in the Plan Library 64.6% of the times in total. In comparison, the standard version of ROSPlan managed to reach the goal 1645 times, performing better than the Plan Library version in the problems with more actions (100+). This is due to the fact that the Plan Library was being used blindly, exploiting only the plans it had solved already, with no exploration, meaning that if it got a poor plan in a prior run, it had a higher chance of getting stuck in it. This, together with higher action probability, would lead the agent into states that would need more than the allocated planning time (30 seconds) to solve, causing it to fail to reach the goal.

In Figure 1, we can see how the plan library performed overall. In short, across all the problems and probabilities of action failure, the plan library works effectively. Figure 1a shows that, over successive runs, the cost of planning falls, while Figure 1b exhibits that the plan library grows, but showing signs that the size of the library will plateau. The second of these is exactly what we would expect, and the first is exactly what we would hope.

IV. Conclusion & Future Work

Our results show that for a dynamic environment and medium length tasks, our approach manages, after a short number of runs, to gain enough experience for a considerable speed-up in deliberation to emerge.

Our experiments make the big assumption that all actions fail with the same probability. Once Covid allows, we will run experiments in real world environments, where action failure is a property of the world instead of it being defined in our simulation. This will give us a better idea of how often a Plan Library can be used, and how fast it can accumulate knowledge about the environment. Seeing if we can balance exploitation of past plans with exploration to discover new plans would be complementary to this work.

Knowing that the time spent searching for a plan is short (7.2% of all planning time), we will investigate if it is possible to add planners [9] that search for better quality plans. Comparisons between different types of heuristics [10] will tell us if using a Plan Library, makes it possible to use classical optimal planners in planning for robotics.

Finally, we are investigating if having a library of the robot's abilities would increase the explainability of its reasoning process. Given such a library, the robot would be able to keep track of its executions, giving a more in-depth explanation for its decision based on its *experiences*.

Acknowledgements

This research was partly funded by a PhD studentship from the Faculty of Natural and Mathematical Sciences at King's College London, and by EPSRC EP/R033722/1 Trust in Human Machine Partnerships.

References

[1] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artificial Intelligence*, vol. 247, pp. 10–44, 2017.
[2] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.
[3] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *JAIR*, vol. 20, pp. 61–124, 2003.
[4] M. Bratman *et al.*, *Intention, plans, and practical reason*. Harvard University Press Cambridge, MA, 1987, vol. 10.
[5] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
[6] F. Meneguzzi and L. De Silva, "Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning," *The Knowledge Engineering Review*, vol. 30, no. 1, pp. 1–44, 2015.
[7] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras, "ROSPlan: Planning in the robot operating system." in *ICAPS*, 2015, pp. 333–341.
[8] A. J. Coles, A. I. Coles, M. Fox, and D. Long, "Forward-chaining partial-order planning," in *ICAPS*, 2010.
[9] M. Martinez, I. Moraru, S. Edelkamp, and S. Franco, "Planning-PDBs planner in the IPC 2018," *IPC-9 Planner Abstracts*, pp. 63–66, 2018.
[10] I. Moraru, S. Edelkamp, S. Franco, and M. Martinez, "Simplifying automated pattern selection for planning with symbolic pattern databases," in *Künstliche Intelligenz*. Springer, 2019, pp. 249–263.