Deep Reinforcement Learning for Plan Execution

Gonzalo Montesino Valle[1] **and Michael Cashmore**[2]
University of Strathclyde, Glasgow, Scotland
[1]*gonzalo.montesino-valle@strath.ac.uk*, [2]*michael.cashmore@strath.ac.uk*

Abstract

There are many different methods for the deliberative control of autonomous systems in stochastic environments, each with different strengths and limitations. Reinforcement Learning can provide robust performance in unpredictable environments, but its decisions are often not predictable. In contrast, Automated Planning can provide explicable and transparent behaviour but its performance drops when the environment is uncertain.

In this paper we discuss an approach to plan execution through reinforcement learning by training an agent to follow predetermined plans. The implementation of the approach leads to the complex task of defining evaluation metrics that describe the desired behaviour. We describe the implementation of this approach as a set of agents, which differ in their reward function, and were trained and evaluated in three scenarios in which plan execution can deviate and be recovered.

1 Introduction

Reinforcement Learning (RL) and AI Planning (AIP) are approaches to anticipatory thinking that focus on similar sequential decision-making problems. Their integration into a cohesive approach has been well-studied, and the area has recently gathered more attention. For example, Illanes et al. (2020) use partial order plans to guide a RL agent and to structure the reward function. Kõvári, Hegedũs, and Bécsi. (2020) use RL to provide a policy for action selection in Monte Carlo Tree Search. Agostinelli et al. (2021) use learning to obtain a heuristic function that can then be used to guide AIP search. These approaches either use AIP to determine the hyperparameters of the RL and improve its performance, or to use RL within AIP to enhance the performance of search.

We propose an alternative integration, in which plans are generated offline using search and flexibly executed online using a trained agent. The core idea is that both the current state and plan can be passed as input to the RL agent, and the agent will attempt to execute that plan. Plan execution can be seen as a control task in which the autonomous agent should remain on the planned path, and be able to correct any deviation caused by uncertainty in the environment. Reinforcement learning has proven to be a robust and high performance tool for control problems in these types of environment.

Training such an agent requires the definition of what it means to follow a plan, and to design a reward function that encourages this behaviour. In this paper we define two evaluation metrics – to minimise time spent deviating from the plan, and to maximise the number of planned states visited. We motivate each of these metrics and explain how they are often in conflict with one another. Consequently, we believe that the defining a desired execution behaviour is a domain-specific decision.

We describe the setup used to train agents according to both metrics, including definitions of reward functions, agent architecture, and state representations. We evaluate these agents by executing plans in three scenarios where plan execution might deviate and be recovered. The evaluation provides clear evidence that it is feasible to achieve a RL agent that is able to recover and execute plans, by executing on average above 60% of the overall plan in scenarios with a high degree of uncertainty.

We present related work in robust plan execution and plan recovery in Section 2. In Section 3 we provide some background on the RL techniques used to construct our flexible plan execution agents, which are described fully in Section 4. The Evaluation is described in Section 5.

2 Related Work

While RL is a natural choice for dealing with stochastic environments, there are also a variety of approaches in AIP for dealing with non-deterministic and stochastic models. For example: replanning when execution fails (Fox et al. 2006a); contingent and conformant planning (Ghallab, Nau, and Traverso 2004); or probabilistic planning, which models the non-determinism in the problem used by the planner to compute a plan (Mausam and Kolobov 2012). Jimenez et al. (2013) approach the problem by learning the probability of action success, which is then feed back to the planner to compute an improved plan. Ribeiro et al. (2021) instead develop the OLISIPO algorithm which represents a partially ordered plan as a Dynamic Bayes Net and at each stage of execution selects the action from the partial order that maximises the plan's probability of success. Saint-guillain et al. (2021) propose the LILA algorithm to solve the problem of plan execution for PSTN based on Monte Carlo Tree search. LILA uses the idea of "playing a game against nature" in which the MCTS nodes are divided into decision

nodes and contingency nodes. The contingency nodes simulate the randomness of the PSTN schedule, whereas the decision nodes set the starting time of each action.

In contrast to these approaches, we use standard AIP approaches to synthesize the plan offline, and then apply RL in the flexible execution of that plan. In this way, the plan can be generated and adjusted offline, for instance to adhere to a set of preferences or standard operating procedures, and be executed as closely as possible to what was agreed. This is similar to the problem tackled by Iocchi et al. (Iocchi et al. 2016) who introduce a formalism for plan representation as Petri-Net plans, which can then be extended by *execution rules* to explicitly include recovery behaviours. These rules are able to return to the prescribed plan execution for some set of known non-deterministic outcomes. Our approach instead relies upon RL to execute the plan while responding to known and unknown outcomes.

Flexible plan execution includes plan execution that is able to recover from unexpected outcomes such as action failure. Another concept in flexible plan execution is to execute a plan while exploiting unexpected opportunities. Borrajo and Veloso (2021) define opportunities as the possibility of unplanned shortcuts that appear during execution and not taken into account during planning; they develop an algorithm to exploit these opportunities. Replanning can also be used to adapt to incoming goals; Cushing et al. (2008) describe a method of continually improving the current plan in an anytime fashion, replanning when the main plan deviates too far from current set of known objectives. Cashmore et al (2018) exploit the difference between conservatively estimated and real duration of actions to produce a net time within which soft goals can be achieved by generating separate plans for opportunities and merging them into the main thread of execution.

In this paper we focus on flexible plan execution for failure recovery but the same principles applied to failure recovery could be used to develop an agent which focuses on exploiting opportunities. This would be done by defining an alternative evaluation metric, but is out of the scope for this paper.

3 Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning that learns through successive interaction with the environment to achieve an optimal policy (Sutton and Barto 2018). The agent converges to the optimal policy by updating it through a trial and error process until it converges to the optimal policy. A RL agent is divided into a policy function which maps states to actions and a value function which maps states to numerical estimate of how rewarding that state is. This division leads to two choices while learning: *Value-based* optimises the value function, while *Policy-based* optimises the policy function. Many algorithms use value functions called *Q-functions* which give a numerical estimate of the current state action pair rather than only the state. The choice of RL algorithm in general depends on the simulation cost, needed versatility, and computational simplicity of the problem domain. Below we discuss three model-free RL methods – we exclude model-based as the main focus is on the policy not the model itself, and model-free leads to higher versatility. We describe the applicability of these methods to flexible plan execution in terms of: sample efficiency (which needs to be high for more computationally demanding domains, such as planning problems), versatility of the agent, type of action/state space, and possible variance and bias.

Advantage Actor Critic (A2C) (Mnih et al. 2016) is an upgrade of the actor critic structure that adds to the value function an advantage function that compares each action to the "general" action. The actor-critic method combines the policy-based (Actor) and the value-based method (Critic) to exploit the benefits of both methods. Combining both methods leads to the possibility of using both continuous and discrete action spaces, as well as having a lower bias and being more versatile. This benefit comes with an increase in variance compared to the DDQN and an decrease in sample efficiency.

Proximal Policy Optimizer (PPO) (Schulman et al. 2017) is an improved combination of A2C and Trust Region Policy Optimization (TRPO). The main idea behind TRPO is that the updated policy is not too far from the previous one, so clipping is needed. The introduction of TRPO to A2C leads to decreasing the variance and increasing the sample efficiency, with a slight increase in bias, as well as maintaining the bigger part of the benefits of A2C.

Flexible plan execution involves both discrete and continuous actions, and the method must have very high sample efficiency. Variance and bias problems that come with model-free RL can be solved by hyperparameter tuning. Given these requirements, we use PPO in our evaluation.

4 Flexible Plan Execution Agent

In this section we first discuss what it means to follow a plan, defining two different categories of evaluation metric. Following this we define reward functions that encourage behaviour according to these metrics. Finally we briefly describe observations made by RL agents that include planned states. Through RL we train a policy $\pi : \Pi \times S \rightarrow A$, which is in charge of executing plan Π.

4.1 Evaluation Metrics

RL learns to optimize its policy (behaviour) to maximize reward, making the construction of the reward function a critical design decision. The reward should encourage behaviour that maximises some metric that evaluates how well the plan is executed. Defining an evaluation metric that measures how well a plan was followed during execution is not a trivial task. We propose the two such definitions and for each case further define both a hard (Γ) and soft ($\hat{\Gamma}$) metric. The hard metrics use a Boolean condition that considers states to be part of the plan or not. In contrast the soft metric considers states that are similar to, but not exactly the same as, planned states - this requires a measure of distance between states, which we discuss at the end of this section.

Maximising Plan States Visited How well the plan was executed could be measured by considering how many of the planned states were visited during execution. For instance,

in surveillance scenarios such as fire detection or search and rescue it might make sense to ensure that each planned observation is made and in roughly the same timeline, even if this requires additional actions to recover from unexpected deviations.

This measure is defined as $\max(|S_e \cup S_p|/|S_p|)$ where S_e is the set of all executed states and, S_p is the set of all states that are part of the plan. From this interpretation we can derive the following metrics:

$$\Gamma_p = \frac{|S_p \cap S_e|}{|S_p|} \in [0,1] \tag{1}$$

$$\hat{\Gamma}_p = \frac{1}{|S_p|} \sum_{s_p \in S_p} \frac{1}{1 + \min_{s_e \in S_e}(\gamma(s_e, s_p))} \in [0,1] \tag{2}$$

where $s_e \in S_e$ and $s_p \in S_p$ are the executed states and planned states respectively, and $\gamma(s_e, s_p) \in \mathcal{R}^+$ is a measure of distance between states. In both metrics metric the maximum value 1 means that every state was visited during execution. Equation 2 is a function bounded between 0 and 1 that computes the average of the inverse of the minimum distance to each plan state.

Minimizing Plan Difference Flexible plan execution can also be interpreted as the execution that minimizes the amount of executed states that do not belong to the plan. This interpretation makes sense when the plan that was generated adheres to a set of preferences which are not or cannot be modelled, and keeping close to the planned behaviour meets these preferences. It also makes sense in the context of multi-agent systems, and adhering as close as possible to pre-agreed plans minimises the risk of interference. From this interpretation we can derive the following metrics:

$$\Gamma_n = \frac{|(S_e \setminus S_p)|}{|S_e|} \in [0,1] \tag{3}$$

$$\hat{\Gamma}_n = \frac{1}{|S_e|} \sum_{s_e \in S_e} \min_{s_e \in S_e}(\gamma(s_e, s_p)) \tag{4}$$

In contrast to metrics (1) and (2), lower is better with $\Gamma_n = 0$ (and $\hat{\Gamma}_n = 0$) denoting zero deviance from the planned trajectory of states.

These metrics (1-4) lead to an identical behaviour in deterministic and certain environments, but introducing uncertainty leads to a behaviour difference, as figure 1 illustrates.

In the top path metrics (1) and (2) are optimised, ensuring that all of the planned states are visited, although additional activity is required to return to states after unexpected events. In the bottom representation metrics (3) and (4) are optimised; the agent quickly returns to the planned trajectory of states when some uncertain event causes it to deviate. However, there are some states which are not visited at all. With respect to maximizing metric (1), the top path ($\Gamma_p = 1$) scores better than the bottom path ($\Gamma_p = 0.6$). Conversely, with respect to minimizing metric (2), the bottom path ($\Gamma_n = 0.4$) scores better than the top path ($\Gamma_n = 0.5$).

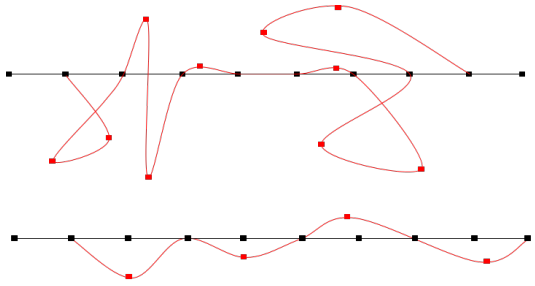

Figure 1: Illustration of plan execution possibilities. The plan is represented as a black dotted line and in red is represented the executed path. The top path maximizes planned states visited, the bottom path minimizes plan difference.

State Distance Measure The definition of a distance function between plans is not an easy task. Fox et al. (2006b) define this distance as the total number of different actions that appear in each plan. Both soft metrics ($\hat{\Gamma}$) rely on a distance function $\gamma(s_e, s_p)$ - in our context we are interested in the distance between states rather than plans. In this direction, Bryce (2014) introduces a distance metric based on landmarks. Below we we outline an initial idea that is used in our implementation.

We describe the logical part of a state s as a set of facts, and divide this set into two distinct subsets: the *dynamic state* (D_s), the set of facts and numeric fluents that can be altered by an action; and the *static state*. Given this we take a similar approach to Bryce (2014), where we define the distance as:

$$\gamma(s_e, s_p) \equiv d(D_{s_e}, D_{s_p}) \tag{5}$$

where $d(D_{s_e}, D_{s_p})$ is a domain-specific distance measure between states s_e and s_p, such as the one which we describe in Section 5.

4.2 Reward Functions

In this paper we distinguish between two types of reward: *final reward* and *per-step reward*. For each type we constructed a separate reward function for training metrics (1-4). Building the reward function for metrics (1) and (2) is straightforward, as RL typically already maximizes a given metric. To convert metrics (3) and (4) into a reward, we can simply consider maximizing the complement of the set.

Final reward functions (R_p^f, R_n^f) only give a positive reward to the agent once the final plan state is achieved. This incentivizes the agent to always achieve the final state, but could lead the agent skipping planned steps to reach the goal. From the two metrics the constructed reward functions are:

$$R_p^f = \frac{|S_p \cap S_e|}{|S_p|} = \Gamma_p$$

and

$$R_n^f = 1 - \frac{|(S_e \setminus S_p)|}{|S_e|} = 1 - \Gamma_n$$

Per step reward functions (R_p^s, R_n^s) give reward to the agent every step. This gives the same importance to all states of the plan, but could lead the agent to repeating plan states

to achieve higher reward, and terminating execution without achieving the goal. From the two metrics the constructed reward functions are:

$$R_p^s = \begin{cases} \frac{1}{|S_p|} & \text{if current state is a plan state} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

and

$$R_n^s = \frac{1}{(\min_{s_p \in S_{vp}} \gamma(s_p, s) + \epsilon)} \quad (7)$$

for some small $\epsilon \in \mathcal{R}^+$ and $S_{vp} \subset S_p$ is a subset of the planned states that is observable by the agent. We describe S_{vp} in more detail in Section 4.3.

4.3 Observation

We use PPO training method throughout this section. Below we discuss the other factor that influences the behaviour of the agent: the representation observed by the agent.

RL agents that are trained to solve planning problems by achieving a goal typically make observations of the state to choose which action to apply. In contrast our agent must include both the current state and the plan within the observation. The nature of reinforcement learning require fixed size observation, placing a limit on the number of planned states include in the observation. This subset of planned states is called the *visible plan states* S_{vp}. The visible plan states includes a set of consecutive planned states, beginning with the state following the most recently achieved planned state. The number of visible plan states in the observation (n_{vps}) can be configured, and the most effective number will depend upon domain-specific factors, such as the expected length of deviation from the plan. We suggest the following guide:

$$n_{vps} \geq \mathbb{E}_{\max}(B_s) \quad (8)$$

where $\mathbb{E}_{\max}(B_s)$ is the expected maximum number of consecutive planned states that cannot be achieved and must be skipped. Intuitively, $\mathbb{E}_{\max}(B_s)$ is how "far ahead" in the plan that the agent will need to observe in order to rejoin its trajectory.

Given that more than one state is included in the observation (the current state and planned states) encoding the entire state described in the planning model will result in unnecessary complexity (i.e. an increase in the number of neurons), which leads to an increase in training time as the learning process abstracts the important features. This problem can be overcome by passing in only the *dynamic state* (D_s) for planned states.

5 Experimentation

We evaluate the agents against the hard evaluation metrics of each definition of "plan following", i.e. equations (1) and (3). We add to these metrics a third, which represents whether the final state is achieved.

$$\Gamma_g = \begin{cases} +1 & \text{if final state achieved} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

The agents were trained using PPO. We chose $n_{vps} = 5$ which is half of the mean plan length (rounded down). Four agents were trained using each of the the four different reward functions (R_p^f, R_n^f, R_p^s, R_n^s), for 10000 episodes (plan executions) in each of four scenarios. To evaluate sensitivity with respect to n_{vps} an additional two agents where trained using the best preforming reward function on the *unachievable plan state* scenario. This scenario was chosen as it is the most highly influenced by this hyper-parameter. Every agent was trained for a total of 5 million steps (actions taken) on each of the scenarios. The number of steps, and other hyper-parameters of PPO could be further tuned to improve performance, which we discuss in the conclusion. In total we trained 18 agents. We scored each agent on each uncertain scenario - excluding the combined training environment - using the evaluation metrics (1), (3), and (7).

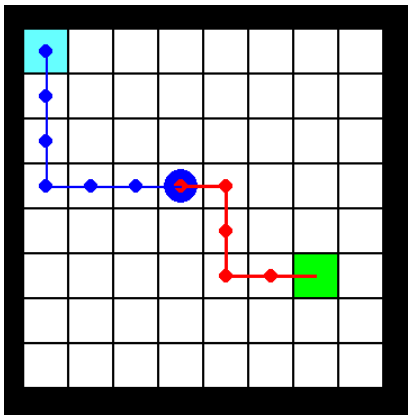

Figure 2: Grid-World example.

In Section 5.1 we describe the four scenarios; in Section 5.2 we discuss the performance of these agents and suggest improvements; in Section 5.3 we analyse the sensitivity with respect to the number of visible plan states[1].

5.1 Evaluation Scenarios

Our evaluation environment is a grid world 8×8 where the start position is in the top left corner and the goal position is randomly generated in the right bottom 4×4 corner area. In figure 2, we show an example of an empty grid world where the green square marks the final goal, the light blue square marks the initial position and the large blue circle is the current position. The blue dotted lines represent the executed path and the red line represents the remaining plan steps. The only possible actions are to move one square up, down, left or right. Grid world environment was selected due to its simple interpretation that lets us clearly explain the suggested concepts. Three plan execution scenarios are derived from main the environment:

- **Non-deterministic action outcomes** include a 10% chance that any action results in a move to a random position within a 3×3 area centered on the current position. The main goal of this scenario is to simulate an agent that,

[1]RL environments, PDDL domains, and source code is available online: https://anonymous.4open.science/r/Intex2022_Codes-97E5

due to uncertainty in the environment, is currently on a state that does not match the states of the plan.

- **Unachievable plan state** modifies the problem instance to include a set of random obstacles on the map. The goal of this scenario is to explore the possibility that some plan states are not achievable due to unexpected events or differences in planning model versus reality. The scenario evaluates the agent's ability to overcome these obstacles and rejoin a later state in the plan.
- **Missing action** is simulated in our environment by removing one random action from the plan before sending it to the agent to be executed (the action can not be the first or last action of the plan). This scenario is based on the idea that the plan could have missing actions due to small model errors. As an example, we have a robot that has to move from one room to another, in the model the door is open but during execution the door is closed. The agent has to perform additional actions to continue with the plan.
- **Combined Environment** combines all of the above. Two minor changes are reducing the probability of moving to a random position 10% to 5% and the introduction of a probability of 10% of any action being missing from the plan. We constructed this environment in such a way that the three tests are specific cases of this more general training environment. Our idea is that the agent is trained on this scenario and then capable of successfully executing plans on any of the previous three.

Unachievable plan states and missing actions are similar in that each must deviate from the plan for a short time in order to continue. The scenarios differ in whether the original actions of the plan are still valid. This difference is significant when evaluating agents that have been trained to specifically follow those actions.

To generate an initial plan between 4 and 16 random obstacles are generated at random, which we refer to as *imaginary obstacles*. Imaginary obstacles cause the plan to differ from the optimal goal-achieving behaviour. They are not seen during plan execution and simulate differences between the planning model, the real environment, and the preferences of the user who generated the plan.

An observation consists of the current state and planned states. The current state is encoded as a 10×10 integer array in which the position of the agent is value 1 and the goal value 2. The 8×8 area is surrounded by a boundary wall of value 3. Obstacles inside boundary have value 4. For the planned states, we only send the dynamic state, which in this case is the position of the agent. This is represented as coordinates $x, y \in [0, 9]$.

For the soft metrics, we define a distance function γ as:

$$\gamma(s_e, s_p) = \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} \qquad (10)$$

where x_i, y_i are the coordinates of the agent's position in state s_i.

5.2 Performance Results

The results over 10000 evaluation episodes are shown as violin plots, where each third of the plot presents one of the evaluation functions over the different test scenarios. For Γ_p and Γ_g higher is better, with the optimal value being 1. For Γ_n lower is better, with the optimal value as 0.

In figure 3 we present the results for each agent trained and evaluated on the non-deterministic scenario. Surprisingly, the final step reward function R_p^f outperforms the other agents, not only with respect to its metric Γ_p, but both other metrics. This could be due to the nature of our evaluation domain, in that actions are reversible - this means achieving every plan state can be done most efficiently by returning as quickly as possible to the planned trajectory. With the exception of R_n^s, all agents achieve a mean 75% of planned states. The results also show that the amount of time spent outside of the plan is less than 40% in all cases, except R_n^s. Finally, the results clearly show that the final state of the plan is achieved more frequently by the final reward functions - this is expected as the reward is only given if the final state is achieved.

In figure 4 we present the results for each agent trained and evaluated on the scenario with unachievable plan states. All agents trained on this environment perform more poorly, with only R_p^s achieving more than 50% of the plan states and agents spending more than 50% of their time off-plan. This environment can be considered the hardest environment, as the agent is given observations of planned states that cannot be achieved. This could potentially be improved with hyper-parameter tuning and more training time, as discussed in the conclusion. Reward R_p^s shows some potential in this environment, achieving the best performance across all three metrics. In figure 4 a bi-modal distribution of Γ_g is more noticeable, although occurs in all of the results. This is due to the nature of the metric, which either awards maximum score or nothing.

In figure 5 we present the results for each agent trained and evaluated on the scenario with missing actions. The bimodal distribution of the evaluation function of R_p^f suggests that the plan is correctly followed, but that there is a roughly 50% chance of the agent to get lost by either repeating states or moving randomly throughout the map without ending the episode. This problem is likely due to bias in training and can be fixed by tuning hyper-parameters. Unlike both previous scenarios, the reward functions based on minimizing time spent off the planned trajectory perform the best across all of the evaluation metrics.

In figure 6 we present the results of the agents trained on the combined environment and evaluated on the other scenarios with the metric Γ_p. This scenario adds complexity that affects training, reducing performance in all agents. R_p^s showed good performance across all previous scenarios. This is translated into the combined environment as it outperforms the other reward functions.

In summary these results imply two important takeaways: First, the final step reward functions perform better than expected, despite the sparse reward. This could be dependent on the length of the plans. A future evaluation should determine how this performance changes with the duration of plans. Second, learning domain-specific behaviours leads to a need for *domain-specific* reward functions that maximise *domain-independent* metrics. This manifests in

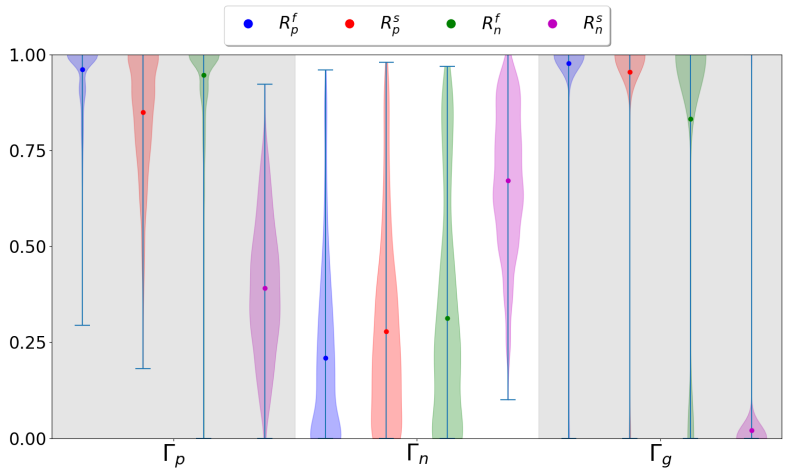

Figure 3: Performance of agents trained and evaluated on **non-deterministic action outcomes** with metrics Γ_n, Γ_p, and Γ_g.

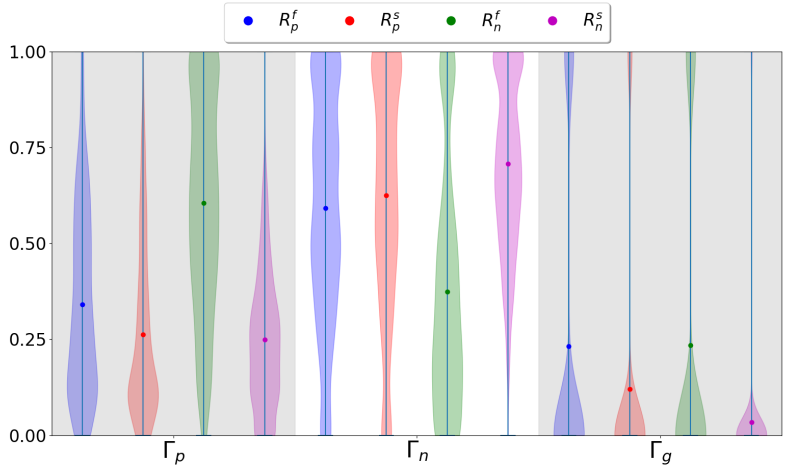

Figure 4: Performance of agents trained and evaluated on **unachievable plan states** with metrics Γ_n, Γ_p, and Γ_g.

R_p^* out-preforming R_n^* functions with respect to Γ_n, even though they are designed to achieve a different metric. Future work should include understanding the general mapping between domain-independent metrics and domain-specific rewards.

5.3 Sensitivity to Visible Plan States

Figure 7 illustrates the performance of the agents trained using different values of n_{vps}. All are trained on the *unachievable plan state* scenario. The reward function R_p^s was used as it has the best performance in this scenario. To carry out the test we chose values $n_{vps} \in \{1, 5, 11\}$, as the minimum, half mean plan length (rounded down), and mean plan length, respectively. The results clearly show that $n_{vps} = 5$ performed best. The expected maximum number of consecutive blocked states $\mathbb{E}_{\max}(B_s) = 2$, and so observing fewer planned states than this provides little information to the agent as to how it might rejoin the plan. A much larger observation (up to the mean plan length) intuitively should provide better performance, but in practice it does not. The more complex observation requires more training time to obtain the same performance. The best value for n_{vps} will depend upon the complexity of the domain, the expected maximum deviation from the planned trajectory, and also the length of training time.

The results obtain point to fact that a plan recovery agent based on reinforcement learning is feasible

6 Conclusion

In this paper we discussed evaluation metrics for plan following and evaluated reinforcement learning rewards for a flexible plan execution agent.

The proposed agents were tested in a set of environments that simulate different types of plan failure. The results show that it is feasible to achieve a flexible plan execution agent based on RL. Improvement in performance can be achieved

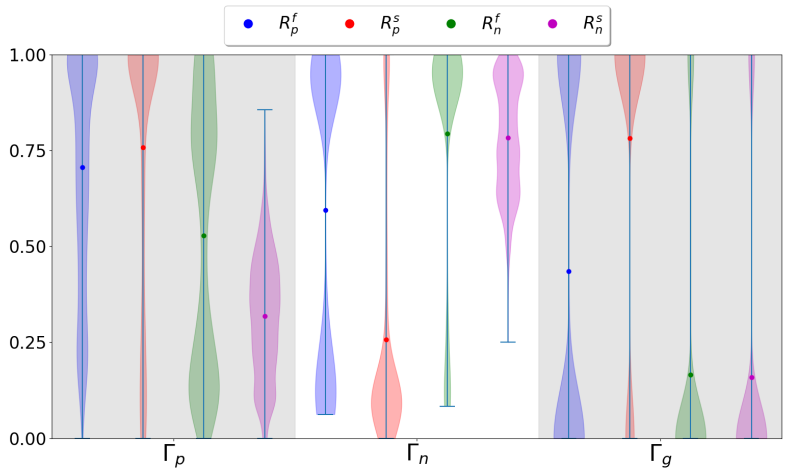

Figure 5: Performance of agents trained and evaluated with **missing actions** with metrics Γ_n, Γ_p, and Γ_g.

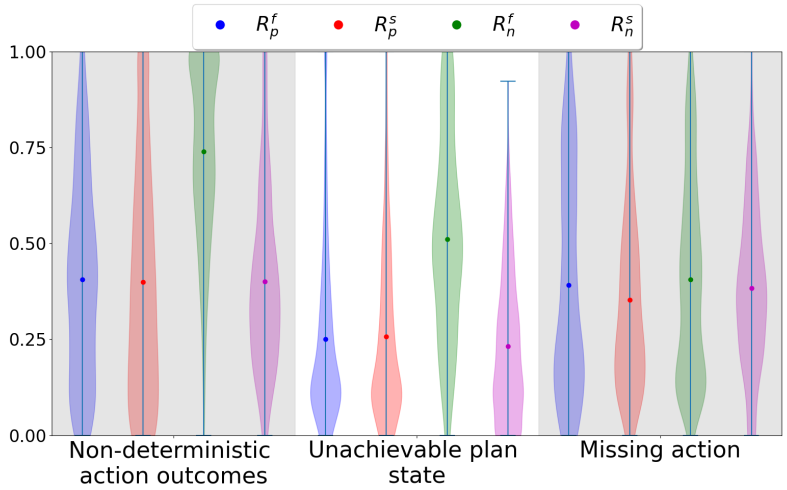

Figure 6: Performance of agents trained on the **combined scenario** and tested on each scenario with metric Γ_p^f.

by a more in-depth investigation in hyper-parameter tuning to reduce the bias in learning and improve variation without harming the sample efficiency.

This paper presents a first step for which there are several exciting directions for future work, including: (i) defining a general form for the state distance measure; (ii) estimating maximum expected blocked states or maximum deviation in general planning problems; (iii) mapping between general plan execution metrics to domain-specific rewards; and (iv) applying the current implementation to more complex planning domains.

References

Agostinelli, F.; Mcaleer, S.; Shmakov, A.; Fox, R.; Valtorta, M.; Srivastava, B.; and Baldi, P. 2021. Obtaining Approximately Admissible Heuristic Functions through Deep Reinforcement Learning and A * Search. In *Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*.

Borrajo, D.; and Veloso, M. 2021. Computing Opportunities to Augment Plans for Novel Replanning during Execution. In Biundo, S.; Do, M.; Goldman, R.; Katz, M.; Yang, Q.; and Zhuo, H. H., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*, 51–55. AAAI Press. URL https://ojs.aaai.org/index.php/ICAPS/article/view/15946.

Bryce, D. 2014. Landmark-based plan distance measures for diverse planning. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS* 2014-January(January): 56–64. ISSN 23340843.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; and Ridder, B. 2018. Opportunistic Planning in Autonomous Underwater Missions. *IEEE Transactions on Automation Science and Engineering* 15(2): 519–530.

Cushing, W.; Benton, J.; and Kambhampati, S. 2008. Re-

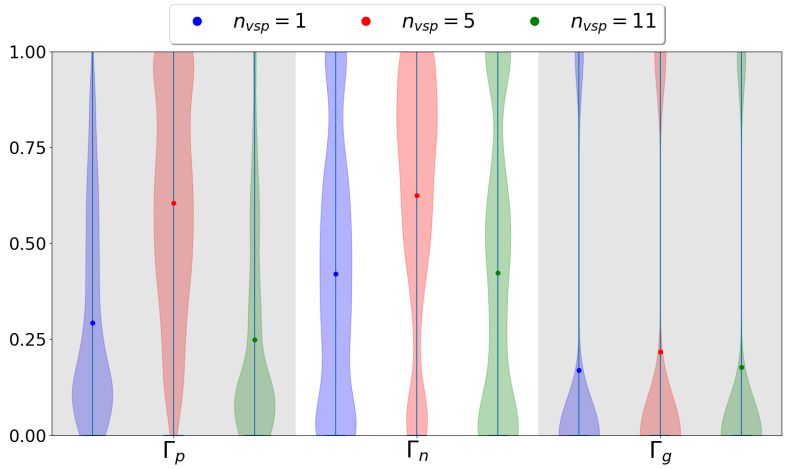

Figure 7: Performance of agent trained with R_p^s and evaluated on **unachievable plan states** with different values of n_{vps}.

planning as a Deliberative Re-selection of Objectives. Technical report, Arizona State University CSE Department.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006a. Plan stability: replanning versus plan repair. In *Proceedings of International Conference on AI Planning and Scheduling (ICAPS)*.

Fox, M.; Gerevini, G.; Long, D.; and Serina, I. 2006b. Plan stability: Replanning versus plan repair. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 212–221.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier. ISBN 978-1-55860-856-6.

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 540–550. AAAI Press.

Iocchi, L.; Jeanpierre, L.; Lazaro, M.; and Mouaddib, A.-I. 2016. A Practical Framework for Robust Decision-Theoretic Planning and Execution for Service Robots. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 486–494.

Jiménez, S.; Fernández, F.; and Borrajo, D. 2013. Integrating planning, execution, and learning to improve plan execution. *Computational Intelligence* 29(1): 1–36. ISSN 08247935.

Kővári, B.; Hegedüs, F.; and Bécsi, T. 2020. Design of a reinforcement learning-based lane keeping planning agent for automated vehicles. *Applied Sciences (Switzerland)* 10(20): 1–24. doi:10.3390/app10207171.

Mausam; and Kolobov, A. 2012. Planning with Markov Decision Processes: An AI Perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6(1): 1–210. doi:10.2200/S00426ED1V01Y201206AIM017.

Mnih, V.; Badia, A. P.; Mirza, L.; Graves, A.; Harley, T.; Lillicrap, T. P.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016* 4: 2850–2869.

Ribeiro, T.; Cashmore, M.; Micheli, A.; and Ventura, R. 2021. Olisipo : A Probabilistic Approach to the Adaptable Execution of Deterministic Temporal Plans. In *TIME International Symposium on Temporal Representation and Reasoning*, 1–11.

Saint-guillain, M.; Vaquero, T. S.; and Chien, S. A. 2021. LILA : Optimal Dispatching in Probabilistic Temporal Networks using Monte Carlo Tree Search. In *Fifth ICAPS Workshop on Integrated Planning, Acting, and Execution (IntEx)*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *arXiv* 1–12.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning : an introduction*. Westchester Publishing Services, second edi edition. ISBN 9780262039246.