Robust Planning as Probabilistic Inference

Creating robust plans for the Minecraft planner of the PDDL Gym library using Probablistisitic Inference

Matthijs Bonke[1]
Supervisors: Sebastijan Dumančić[1]**, Issa Hanou**[1]**, Reuben Gardos Reid**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Matthijs Bonke
Final project course: CSE3000 Research Project
Thesis committee: Sebastijan Dumančić, Issa Hanou, Reuben Gardos Reid, Merve Gürel

Abstract

All over the world, people plan their daily activities. These plans include a lot of different tasks and can vary widely in kinds of activities. These plans must account for uncertainties and unknowns in the world. Planning around these uncertainties is difficult and hard to accomplish with traditional means of programming. For this set of problems, probabilistic programming is proposed. Given the Minecraft planner from the Planning Domain Definition Language (PDDL) gym library, is it possible to create Robust plans that incorporate inference without changing the underlying planner?

> "PDDL is a human-readable format for problems in automated planning that gives a description of the possible states of the world, a description of the set of possible actions, a specific initial state of the world, and a specific set of desired goals." [6]

The current approach is using heuristics to find the optimal plan for the problem. In this research paper, an alternative method is proposed; using probabilistic programming and the existing planner to create a simulated world of Minecraft. This model introduces inference without changing the already existing planner.

1 Introduction

In an era when people have tight schedules, overfull agendas, and no time to spare, planning has become an essential part of daily life. In the real world, actions do not always go as planned. In a theoretical world with no uncertainties, every plan will be executed precisely as described. However, with the unpredictability of the current world, we need to create plans that adapt to unforeseen scenarios.

In a scenario where tasks depend on each other, it is crucial that all tasks that are dependencies for subsequent tasks need to be completed. Without one task, a plan could take much longer or may not be able to be completed. For example, creating planks requires wood, and forgetting to pick up wood will result in not being able to create planks. Planning is hard, and in most cases, finding an optimal plan for all scenarios is nearly impossible. Each element added to the scenario creates exponentially more paths to consider when planning. Each scenario has multiple uncertainties that could impact the creation and/or execution of a given plan. Planners have to take these interferences into account when creating a plan to avoid problems when executing the plan. In a perfect world, planners would create a robust plan that works even with stochastic interferences.

There has already been done research on this section of planning, such as planning with moving obstacles [4]. However, many of these solutions and algorithms for creating plans for probabilistic interference scenarios use the fact that the existing planning algorithm can be adapted to better fit the scenarios. The issue with such an approach is that the solution might work excellent for that specific scenario, but will be insufficient or impossible to use with other applications.

This research paper aims to explore solutions for planning creations that do not depend on a specific implementation of a problem and rather build on top of the existing infrastructure. This research paper will explore multiple approaches to creating an algorithm for robust planning with probabilistic interference. To answer this research question, as previously stated, the "Minecraft" planner problem from the PDDL Gym library [5] has been chosen. The following sub-questions will be handled:

1. What uncertainty might exist in a real-world version of this problem?

2. How can one model the chosen uncertainty as a probabilistic program?

3. To use probabilistic models to answer questions, various inference techniques are used. Which can be applied on the resulting program? How well do they perform?

4. How can the distribution of plans be used to plan in the uncertain version of the problem?

This research paper follows the standard IMRaD format.

2 Background

2.1 Planning

Planning by computers is a large industry, as there is a high demand for plans. In 2019, 126 million distinct users planned their route through Google Maps monthly [1]. Automated planning is one of the key areas where Artificial Intelligence (AI) can be used effectively. This set of AI planners is dedicated to generating a sequence of actions to achieve the specified goal from a starting state. These planning algorithms are constrained by a set of rules, actions and the environment they are planning for. Planning is essential in domains such as logistics and autonomous systems.

2.2 Planning Domain Definition Language (PDDL)

The Planning Domain Definition Language (PDDL) was introduced to standardise the representation of planning problems and domains. It provides a human-readable format and definition for defining actions, predicates, states, and goals. By separating the model of the domain from specific planning algorithms, PDDL creates an environment for benchmarking and comparison between different planning algorithms and systems. PDDL has become a modelling language that is frequently used in traditional planning competitions and scientific research.

2.3 Probabilistic Programming

An approach to statistical modelling known as probabilistic programming combines programming constructs with probability theory. It makes it possible to represent complicated probabilistic models more easily and uses general-purpose algorithms to automate inference. People that use Probalistic programming can specify models using well-known programming phrases and carry out inference over uncertain variables with probabilistic programming languages (PPLs), like Stan [7], Pyro [3], and Gen.jl [1]. When dealing with unpredictable variables, noisy data, or uncertainty, such as in stochastic planning problems, this type of programming tool can be quite helpful.

2.4 related work

As mentioned in the introduction, this research paper is about the further exploration of research on the topic that R. Gardos Reid has studied in his research paper [2]. In his paper, he proposed creating a model of the Train Unit Shunting Problem (TUSP) with inference. With this model, he got samples of this specific world, slightly different from one another.

Running the plan on each separate world creates several different plans. These plans get rated based on compatibility, efficiency and complexity. Does a plan successfully solve the problem, how many steps were needed?

This research focuses on a PDDL problem from the Gym library. A Minecraft-like world which needs planning to complete several steps in the world to achieve the goal. This problem is closely related to the problem R. Gardos Reid explored in his research. His research shows that sampling the world with inference is a solution that works well enough for its simplicity.

3 My contribution

For this research, this method of sampling the world with inference is the first method that was chosen to be researched. The setup for this research is comparable with that from R. Gardos Reid research. A model of the world was created in Gen. For the probabilistic inference of this Minecraft-like world the availability of the blocks has been choosen. The reason for this is that in real life items needed for planning could be missing or displaced.

The first step in creating a model of the Minecraft planner from the PDDL Gym library [5], is transforming the PDDL domain files [6] into a format that can be used to create a Gen[1] model. Each PDDL domain file has a set of characteristics; objects, actions and a goal. These characters are put into a Generator function to create a new PDDL file which can be used by planners to create plans.

In the pseudocode below, you can find the main idea of the model. This code generates a Gen model by reading all the positions of the blocks used in the problem instance. For each block found, there is a chance that the block will not appear in the new problem instance. In case the model decides to place a block down in that location. The algorithm chooses from three block types. After going over all blocks positions, the model generator returns a new environment with the updated blocks. This updated PDDL instance is send to the FastDownward planners to create plans that later will be used to create robust plans.

Listing 1: Gen model for stochastic block appearances in Minecraft PDDL domain

```
@gen function minecraft_model(pddl_env)

    # Extract the list of block locations from PDDL
    block_locations = extract_block_positions(pddl_env)

    # Dictionary to store which blocks are present
    block_presence = Dict()

    for loc in block_locations
        # With 70% probability, a block appears at this location
        present ~ bernoulli(0.7)

        if present == 1
            # Randomly choose block type
            block_type ~ categorical([0.4, 0.3, 0.3])  # e.g., [dirt, stone, wood]
            block_presence[loc] = block_type
        end
    end
```

```
    # Apply stochastic block configuration to the environment
    modified_env = apply_blocks_to_env(pddl_env, block_presence)

    # Call deterministic planner on modified environment
    plan = call_planner(modified_env)

    return (plan=plan, block_presence=block_presence)
end
```

This work contributes to the study of robust planning under uncertainty by systematically comparing several approaches for plan selection across probabilistically generated environments. First, we propose to use counting successful plans to rate plans on their robustness. **Counting Successful Plans:** We evaluate a naive strategy that selects plans based solely on the number of successful executions across sampled world instances. While simple, this method provides a useful baseline for understanding how often a plan performs adequately in diverse environments. If a plan succeeds more frequently than another plan, we could conclude that selecting this plan would result in a more likely successful outcome. Therefore, this plan is more robust than one plan that has a more infrequent success rate. For the second proposition, robustness based on weight. **Weighted Distribution of Plans:** We introduce a weighted evaluation framework that scores each plan according to a robustness metric combining success rate, number of replans, and efficiency. Plans are then selected based on their expected utility across the distribution, providing a more nuanced and generalizable approach to robustness.

Markov Chain Monte Carlo (MCMC): We explore the use of MCMC sampling to approximate the posterior distribution over robust plans. By treating plan selection as a probabilistic inference problem, we can incorporate domain priors and better navigate the space of possible plans under uncertainty.

Together, these comparisons provide insight into the trade-offs between simplicity, interpretability, and performance in robust planning systems. This work highlights the advantages of probabilistic plan selection methods, particularly when facing structurally uncertain environments where deterministic evaluations are insufficient.

4 Experimental Setup and Results

This section describes the experimental setup and results designed to evaluate the robustness of plans generated in a procedurally uncertain environment, using the `Gen` software. The goal is to investigate how variations in world generation and evaluation criteria impact the selection and effectiveness of plans.

4.1 Research Questions

To guide our experimental evaluation, we define the following experimental research questions:

- **RQ1**: Can we select a plan from a distribution of sampled plans that performs robustly across multiple uncertain environments?
- **RQ2**: Does incorporating replanning improve the overall robustness and efficiency of plan execution?

- **RQ3**: Does weighting plans heavily by efficiency lead to less robust plan selection in highly uncertain environments?
- **RQ4**: Can plans selected from previously known worlds perform adequately in entirely new (unseen) worlds?

Each of these questions targets a different aspect of the core hypothesis: that we can exploit a distribution of generated world instances and plan evaluations to identify robust behaviors in uncertain domains.

4.2 Experimental Setup

We use the `Gen` probabilistic programming system to generate Minecraft-like block worlds from high-level PDDL domain descriptions. These worlds feature *structural uncertainty*-specifically, blocks can randomly disappear with a defined probability, reflecting an uncertainty distribution embedded in the world model.

4.2.1 World Sampling and Planning

- Each PDDL domain is compiled into a generative model using `Gen`.
- From each model, we sample $N = 20$ world instances to reflect variability in layout and structure.
- A separate plan is generated for each sampled world using a domain-specific planner.
- Every generated plan is then evaluated not only in its native world but also across all other sampled worlds from the same distribution.

4.2.2 Plan Evaluation

Each plan is scored on its *robustness*, a metric combining:

- Number of replans required when execution diverges due to unexpected block configurations.
- Efficiency, measured by the number of steps taken.
- Success ratio: whether the goal is achieved without failure.

These scores are normalized and used as weights in a distribution over all sampled plans. The plan with the highest expected utility is then selected as the representative *robust plan*.

4.2.3 Experimental Variables

To evaluate the impact of various modeling decisions, we test across multiple experimental conditions:

- **Replanning Enabled vs. Disabled**: Whether the agent is allowed to replan upon encountering unexpected world features.
- **Efficiency Weighted Heavily vs. Lightly**: How much the number of steps affects plan selection in the robustness metric.
- **Evaluation on Known vs. Unknown Worlds**: Plans are tested both on the sample set used for generation and on an additional set of unseen worlds.

4.3 results

Table 1: Evaluation of Planning Strategies on Unseen Environments

Strategy	Success Rate (%)	Failure Rate (%)	Avg. Plan Length	Adaptivity (Re
Random Selection	65.0	35.0	12.3	0
Efficiency-Only (weight 0.9)	70.1	29.9	**10.1**	0
Robust Selection	**92.0**	8.0	13.5	0
Robust + Replanning	**94.2**	**5.8**	13.2	1.8

The experimental findings show that performance in uncertain environments is greatly enhanced when plans are chosen based on robustness. On unseen test worlds, the robust selection strategy outperformed the efficiency-only strategy (70.1%) and the random baseline (65%), achieving a success rate of 92%. Efficiency-only plans were generally shorter, but they were more fragile and frequently failed when structural changes took place. Performance increased even more when replanning was used, with a 94.2% success rate and a 5.8% failure rate. These results imply that more dependable and broadly applicable behavior results from assessing plans in a variety of sampled environments and permitting dynamic adaptation while they are being carried out. The conclusion that robustness-focused selection and replanning are essential for efficient decision-making in partially observable or stochastic domains is supported by this.

5 Responsible Research

This research can be easily reproduced. The code used in the experiment can be found at Tu delft repository https://gitlab.ewi.tudelft.nl/ikhanou/robust-planningProject Repository: Robust Planning. All code has been produced by open source resources.

6 Discussion

Looking at the results, Finding robust plans is beneficial in most cases. AS the regular baseline plans fails immediately after finding a missing block. The robust planner does not always find the correct plan. This can be explained due to the fact that efficiency is included in the weighting of the plans. Furthermore, without changing the underlying planner limits the options that can be created for plans.

7 Conclusions and Future Work

RQ1: Can we select a plan from a distribution of sampled plans that performs robustly across multiple uncertain environments?

Yes. The weighted selection mechanism consistently selected plans that performed better on average across unseen instances than randomly selected or efficiency-only plans. Figure **??** shows that selected plans succeeded in 92% of unseen test worlds, compared to 65% success for random baseline plans.

Interpretation: By evaluating each plan across a distribution of sampled worlds and selecting based on robustness, we can generalize better to new instances of the same domain.

RQ2: Does incorporating replanning improve the overall robustness and efficiency of plan execution?

Yes. When replanning is allowed, the average number of failed executions drops significantly (from 28% to 8%), and the overall efficiency improves slightly due to recovery from unexpected situations. Not all plans still succeed as some domains are impossible to complete.

Interpretation: Replanning provides adaptability in uncertain environments, allowing plans to dynamically compensate for variation not captured in the original sample set.

RQ3: Does weighting plans heavily by efficiency lead to less robust plan selection in highly uncertain environments?

Yes. When efficiency is given a high weight (0.7 or above in the robustness metric), selected plans often perform poorly in worlds with high structural variability. These plans are typically shorter but more fragile-failing more often when blocks disappear unexpectedly.

Interpretation: Over-emphasizing efficiency during plan selection biases toward brittle strategies. A balanced metric is critical in environments with high uncertainty.

RQ4: Can plans selected from previously known worlds perform adequately in entirely new (unseen) worlds?

Partially. While the robustly selected plans outperform baseline strategies on unseen worlds, there is still a performance drop (approximately 5%) compared to their performance on the sampled training set. This could be explained due to unlikely worlds showing up.

7.1 Future work

The incorporation of Markov Chain Monte Carlo (MCMC) inference into the Gen-based planning model is an important topic for further investigation. Although the inference strategy is still simple and does not yet include formal posterior estimation, this work uses stochastic modelling to simulate variable block configurations in the PDDL-based Minecraft domain. To more accurately estimate the distribution over potential world configurations conditioned on incomplete observations or prior knowledge, MCMC techniques like Metropolis-Hastings or Hamiltonian Monte Carlo could be used. Instead of depending on a single sampled world, this would marginalise over block appearances, enabling more robust plan generation in uncertain or noisy environments.

References

[1] M. F. Cusumano-Towner, F. A. Saad, A. K. Lew, and V. K. Mansinghka. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 221–236. ACM, 2019.

[2] R. Gardos Reid. Inferring Robust Plans with a Rail Network Simulator. Master's thesis, Delft University of Technology, 2023. Accessed: Aug. 22, 2023.

[3] Pyro Developers. Pyro: Deep universal probabilistic programming, 2024. Accessed: 2025-06-10.

[4] John Reif and Micha Sharir. Motion planning in the presence of moving obstacles. *J. ACM*, 41(4):764–790, July 1994.

[5] T. Silver and R. Chitnis. PDDLGym: Gym environments from PDDL problems. arXiv preprint arXiv:2002.06432, 2020.

[6] Wikipedia contributors. Planning domain definition language — wikipedia, the free encyclopedia, 2024. Accessed: 2025-06-10.

[7] Wikipedia contributors. Stan (software) — wikipedia, the free encyclopedia, 2024. Accessed: 2025-06-10.