Robust Planning for Sokoban with Probabilistic Inference

Daniël 't Mannetje[1]
Supervisor(s): 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: Daniël 't Mannetje
Final project course: CSE3000 Research Project
Thesis committee: Sebastijan Dumančić, Issa Hanou, Reuben Gardos Reid, Merve Gürel

Abstract

Planning problems are a set of problems in which an objective must be reached by a sequence of actions. Planning problems traditionally do not consider uncertainty, however for most real-world planning problems uncertainty must be considered to create effective plans. The objective of this paper is to use an existing deterministic planning algorithm in order to create robust plans that solve an uncertain version of Sokoban called *Uncertain Move Sokoban*. To this end a probabilistic programming language, Gen.jl, is used, which enables creating probabilistic models and inferring its parameters using code. A probabilistic model is created in Gen.jl, that generates plans for the problem as well as robustness scores using a simulator embedded in the model. Probabilistic inference techniques are then used to obtain a robust plan for the uncertain problem, namely: importance sampling and Metropolis-Hastings. We find that the technique is able to create robust plans for small to medium-sized problems and that Metropolis-Hastings is the better-performing inference technique.

1 Introduction

Planning for the future is a difficult task as the future is often shrouded in an air of uncertainty. A planning problem is a computational problem, whose solution is a sequence of actions to reach a desired goal state from some initial state. Classically planning problems are formulated without uncertainty and planning algorithms for them therefore do not consider uncertainty either. However, for problems in the real world this is often not an accurate representation as they often have many uncertain parameters, for example, due to measurement error. This paper therefore aims to demonstrate and evaluate a method for forming robust plans for these uncertain problems by studying how to accomplish robustness in an uncertain version of the Sokoban problem. Sokoban is a game in which the player's goal is to push all the boxes in the level to a goal square. Studying how to make robust plans for the theoretical Sokoban problem will aid in understanding how to do so for uncertain problems that are of interest to the real world as well.

The work done in this paper builds upon a previous master's thesis [1]. In it the author proposes a framework for using an existing planner of a planning problem to create a robust plan for the uncertain version, by making use of probabilistic programming. The framework is then applied to the train unit shunting problem (TUSP). Baydin et al. [2] have done similar work where they use existing scientific simulators and apply probabilistic programming to it.

Multiple questions about the the framework proposed in [1] have not yet been investigated. First, it has only been used on the TUSP, so its suitability for other problems such as an uncertain version of Sokoban has not yet been studied. Additionally, as noted by the author of [1] the problem instances studied in the thesis were small for performance reasons. As a result the usefulness of providing multiple alternative plans was not studied. Applying the technique on other problems, which have higher performing planners, could be useful to study outputting multiple robust plans.

The main research question of this paper is how an existing planning algorithm can be used to robustly solve uncertain Sokoban problems. In doing so we demonstrate that the framework proposed in [1] can indeed be applied to other planning problems as well. In order to answer the research question we divide the problem into sub-questions: how can an uncertain Sokoban problem be modeled with probabilistic programming; how can a distribution of plans be used to solve an uncertain Sokoban problem instance; which probabilistic inference technique gives the highest robustness?

The rest of the paper is structured as follows. Section 2 will give relevant background. With this background the exact uncertain Sokoban problem will formally be defined in section 3. Given this problem section 4 will describe the method used to solve it. The method's robustness is evaluated using experiments, whose description and results will be discussed in section 5. A further discussion of the results and limitations of the research can be found in section 6. Section 7 will reflect on the ethical aspects of the research and discuss its reproducibility. Finally, conclusions of the research will be discussed in section 8.

2 Background

2.1 Automated Planning

According to Jiménez et al. [3] automated planning is the field of study that aims to computationally find a plan, represented as a sequence of actions, to achieve a goal in a certain environment. An environment consists of two elements: a domain, and a problem. The domain describes the set of states of this environment and the actions that can be taken to transition between them. The problem describes the initial state and the goal that needs to be achieved.

Sokoban can be viewed as an automated planning problem. The Sokoban domain describes the rules governing how the player can move and push boxes. The problems for this domain are Sokoban levels, in which the goal is to push all boxes to a goal square.

Planning algorithms are used in order to solve planning problems. These algorithms search a path in the state space from the initial state to a goal state [3]. In general, finding a solution to a planning problem is known to be PSPACE-Complete [4], meaning that large problem instances are computationally infeasible as the size of state space explodes for these large problems. This is also the case for Sokoban which has been proven to be PSPACE-Complete [5] as well as NP-Hard [6]. Therefore, planning algorithms make use of heuristics in order to prune the state space and provide a solution in a reasonable time span [3].

The planning algorithm used in this paper is the Fast Downward planning system [7]. This planning algorithm is a domain-independent planning algorithm, since it is capable of solving problems for any domain. In order to achieve this, Fast Downward takes as input a file describing the domain as well as a file describing the problem. Both of these files are written in the Planning Domain Definition Language (PDDL) [8]. Any domain and its problems can be written in PDDL, which can then be interpreted by Fast Downward in order to create a plan.

2.2 Probabilistic Programming

As described by Gordon et al. [9] probabilistic programming languages (PPLs) are programming languages with two additional features: the ability to draw values from a probability distribution, and the ability to constrain these random samples to a certain value. Using the former feature in a function allows the programmer to define a probabilistic model using code. The latter feature can be used together with observations for some of the random variables in the model to infer the value of the other variables.

The particular PPL used in this paper is the Gen.jl library for the Julia programming language [10]. In Gen.jl the programmer can define probabilistic models in *generative functions*. Each random choices made in the model is assigned an *address*. Upon execution of the generative function the values for each random variable are stored at the addresses in

an object known as the *trace*. Finally, conditioning variables is done by assigning a value to the address of the variable in a structure called a *choice map*. Listing 1 shows an example of a generative function `foo`, modeling the probability that at least one of two coin flips lands on heads. The example contains two random variables which have a Bernoulli distribution. The addresses of these variables are `"flip-1"` and `"flip-2"`. The generative function `foo` can be called to generate a random value. Alternatively, inference techniques can be applied to it, for example, the distribution with `"flip-1"` constrained to false can be inferred.

```
1    @gen function foo()
2        flip1 = {"flip-1"} ~ bernoulli(0.5)
3        flip2 = {"flip-2"} ~ bernoulli(0.5)
4        return (flip1 || flip2)
5    end
```

Listing 1: A generative function of two coin flips.

2.3 Related Works

As stated previously, the research in this paper is based on the framework proposed in [1]. In the paper the framework is applied to an uncertain version of the train unit shunting problem (TUSP), which is a planning problem where trains must be routed to shunting yards in such a way that they depart on their scheduled departure time. The author utilizes a PPL to create a probabilistic model of the TUSP with uncertain arrival times, which uses an existing planning algorithm and simulator as part of the model. Importance sampling and that Metropolis-Hastings algorithm are applied to the model as inference methods to obtain robust plans. The author finds that the approach is able to create more robust plans for the TUSP and that Metropolis-Hastings is the more efficient inference method.

A probabilistic programming framework created by Baydin et al. [2] similarly allows performing inference tasks on existing stochastic simulators. Their work specifically focuses on scientific simulators. These simulators often consist of thousands of lines of code, which means significant work would need to be performed in order to port them to an existing PPL. The authors specifically apply the framework to a particle physics simulator and achieve "efficient inference on LHC collision events" [2].

3 Formal Problem Description

The goal of this paper is to describe and evaluate a method to create robust plans for an uncertain version of Sokoban. In this section we precisely describe the problem with a number of formal definitions. Before each definition an intuitive description of the definition is provided.

An instance of the Sokoban problem consists of multiple elements. A Sokoban level is a grid of squares, each of which is either a wall, clear square or a goal square. Some of these squares have boxes on them, which need to be pushed to a goal square by the player. The player starts at one of the squares and is able to move to an adjacent square in one of the four directions: up, down, left or right. The player is able to move to a square if it is not a wall. To push a box the player moves to a square that has a box on it. The box on that square then moves one square in the same direction as the player has moved. If that square is a wall or another box, neither the player nor the box moves.

Formally a Sokoban problem is described as a tuple as given by the following definition:

Definition 3.1 (Sokoban Problem)**.** A Sokoban Problem is a 5-tuple $\langle S, A, T, g, s_0 \rangle$ where:

- S is the set of states for the Sokoban level.
- $A = \{\text{UP}, \text{DOWN}, \text{LEFT}, \text{RIGHT}\}$ is the set of actions that the player can take.
- $T : S \times A \to S$ is the transition function, which gives the next state $s_{k+1} \in S$ after a given action $a \in A$ is performed in state $s_k \in S$, so $s_{k+1} = T(s_k, a)$.
- $g : S \to \{0, 1\}$ is the goal function, which indicates whether a given state is a goal state i.e. all goal squares have a box on them. If a state s is a goal state $g(s) = 1$, else $g(s) = 0$.
- s_0 is the starting state of the Sokoban level.

The uncertain Sokoban problem discussed in this paper is the *Uncertain Move Sokoban problem*. The instances and rules for this problem are the same as the Sokoban problem described above, however each action has a probability α of failing. If an action fails then the player does not move and the state does not change, so $T(s_k, a) = s_k$. Additionally, if an action fails the next action is guaranteed to succeed.

Definition 3.2 (Uncertain Move Sokoban Problem)**.** An Uncertain Move Sokoban problem is a Sokoban problem where $T(s_k, a)$ is a discrete random variable with probability mass function:

$$p(s_{k+1}|s_k, a, f) = \begin{cases} 1 & \text{if } f = 1 \text{ and } s_{k+1} = T(s_k, a) \\ \alpha & \text{if } f = 0 \text{ and } s_{k+1} = s_k \\ 1 - \alpha & \text{if } f = 0 \text{ and } s_{k+1} = T(s_k, a) \\ 0 & \text{else} \end{cases}$$

In this function $s_k \in S$ is the current state, $a \in A$ is the action taken, $f \in \{0, 1\}$ indicates whether the previous action failed, α is the probability of failure and $s_{k+1} \in S$ is the next state.

The Uncertain Move Sokoban problem is a planning problem, so to solve it one must give a plan. A plan is a sequence of actions a player must take in order to reach the goal state. This plan can then be executed by sequentially applying the actions starting at the start state and simulating whether the action fails or not at each step.

Since the transition function is uncertain a given plan is not guaranteed to reach the goal state. Therefore, it is helpful to give each plan a score, indicating how robust it is. This *robustness score* is defined as the probability that the final state after executing the plan is the goal state.

The optimal solution of the Uncertain Move Sokoban problem is called the robust plan. The robust plan should be able to reliably reach the goal state even under the condition that the success of the moves is uncertain. Therefore, the robust plan is the plan with the highest robustness score.

Definition 3.3 (Plan)**.** A plan is a sequence of n actions $a_0, a_1, \ldots, a_{n-1}$. The plan can be executed, giving the sequence of states $s_0, s_1, \ldots, s_n$, where $s_{k+1} = T(s_k, a_k)$ for each $0 \leq k \leq n - 1$.

Definition 3.4 (Robustness)**.** The robustness of a plan of length n is given as the probability that state s_n of the execution of the plan is a goal state $P(g(s_n) = 1)$.

Definition 3.5 (Robust plan)**.** A robust plan is a plan that maximizes the robustness for a certain instance of the Uncertain Move Sokoban problem.

4 Creation of Robust Plans

4.1 Probabilistic Model of Plans

The framework described in [1] uses a generative model of plans in order to infer a robust plan. The generative model for the Uncertain Move Sokoban problem is shown in Listing 2. It consists of two parts: the creation of a plan, and the evaluation of that plan.

First, lines 2-9 contain the generation of a plan. The generative model takes a plan for the deterministic version of Sokoban as input and uses it to create a plan for a specific execution of the Uncertain Move Sokoban problem. For each action in the original plan the model has to make a choice: either assume that this move will fail, or that it will succeed. Since it is not known which assumption is made by the robust plan, the model will assume with probability α that the move fails. Thus, the decision is sampled from a Bernoulli distribution with $p = \alpha$. If the assumption is made that the action fails, the action will be repeated in the generated uncertain version of the plan, otherwise the action will only be performed once.

Second, the evaluation of the generated plan is performed in lines 11-13. The generated plan is executed 100 times on the Uncertain Move Sokoban problem by the `evaluate` function, which is also a generative function since it makes random choices when executing the plan. The `evaluate` function keeps track of how many executions finished in a goal state and in the end returns the percentage of runs that achieved a goal state. In this way a robustness score is created for the plan, and it is stored in the `"score"`-address so that it can be used to perform inference. Note that the score random variable is actually sampled from a narrow normal distribution centered at the robustness score. This is done to work around Gen.jl's restriction of only allowing random variables to be stored at trace addresses. The same trick was performed by [1] in order to store the delay of the shunting plan in the address of a random variable.

```
1   @gen function uncertain_move_sokoban(grid, plan, alpha)
2       uncertain_plan = []
3       for (i, action) in enumerate(plan)
4           push!(uncertain_plan, action)
5           action_failed = {"sampled-failure-$i"} ~ bernoulli(alpha)
6           if action_failed
7               push!(uncertain_plan, action)
8           end
9       end
10
11      score = {"evaluation"} ~ evaluate(grid, uncertain_plan,
12          alpha, 100)
13      {"score"} ~ normal(score, 0.01)
14
15      return uncertain_plan
16  end
```

Listing 2: The generative model of plans.

4.2 Inference

The model defined above can be used to infer a distribution of robust plans. The goal of this inference is to find the most likely assignment of each random variable `"sampled-failure-i"`

under the constraint that the `"score"` random variable is 1. Constraining the `"score"` random variable to 1 has the effect of making assignments with high robustness scores more probable, thus making the model more likely to output robust plans.

Two inference techniques are used in order to obtain robust plans: importance sampling [11] and Markov chain Monte Carlo (MCMC) [12].

First, importance sampling generates n possible plans under the constraint that `"score"` is 1. These plans are generated by resampling all `"sampled-failure"` values at once. Additionally, for each plan the likelihood that it conforms to the constraint that `"score"` is equal to 1 is computed. Plans that have a low robustness score will be further away from 1 and will therefore have a lower likelihood. In the end the plan with the highest likelihood will be considered the robust plan.

Second, the specific MCMC method used is Metropolis-Hastings sampling. This method starts out by generating a plan under the condition that `"score"` is 1. The plan generated might not be a very probable plan and therefore not very robust, but it serves as a starting point which will be iterated upon by Metropolis-Hastings. After the initial sample is generated the plan is updated in n iterations. In each iteration every variable `"sampled-failure-i"` will be resampled individually to get a proposal sample. This proposed change to the plan is evaluated to get a robustness score for the change. If the robustness score is higher than the current plan, the proposed change more likely conforms to the constraint that `"score"` is 1 and Metropolis-Hastings is more likely to accept it. If the robustness score is lower the change is less likely to be accepted. Note that even proposed changes to the plan that lower its robustness score can be accepted. This is the case because it might sometimes be necessary to temporarily make the plan less robust so that other changes to the plan later on can make it more robust. In the end this method will create n plans forming a distribution of plans that are biased to have a high robustness score. The final robust plan can be acquired by taking the plan with the highest robustness.

Metropolis-Hastings can be viewed as more efficient than importance sampling, since its resampling is more fine-grained. Instead of resampling all variables at once it tweaks each variable one by one. This should in theory lead it to be more likely to sample robust plans as it can build upon previous plans.

5 Experimental Setup and Results

5.1 Experimental Setup

The performance of the two methods for inferring robust plans mentioned in the previous section will be tested using test cases provided by the PDDLGym project [13] and test cases we created ourself. These two sets together allow evaluating the methods on levels varying in structure and length.

We have assigned each test case their own move fail probability α based on the length of the plan for the deterministic version of the problem. Longer levels will have smaller α than shorter levels. This is done in order to make sure that the deterministic plan does not have a robustness score close to 0. A robustness score above 0 is required for the evaluation of the uncertain plan in the model to be able to determine the difference between two plans. If the base robustness score is close to zero the limited number of evaluation iterations in the model cannot quantify the difference between two plans, since a plan is very likely to get an evaluation score of 0 in this case. This difficulty of comparing two plans can significantly

decrease the robustness of the inferred plans and therefore it is important to set α to a value that will allow the creation of plans that perform well.

The performance of the two methods will be evaluated for each test case according to the following procedure. First, importance sampling and Metropolis-Hastings will be run for 1000 iterations in order to obtain the robust plans for which the performance will be measured. The number of iterations was chosen as a balance between robustness and inference time. Second, the robustness score of both plans is computed by running the plans 1000 times on the Uncertain Move Sokoban problems being evaluated. Finally, the robustness score of the deterministic plan is also measured to serve as a baseline for comparing the effectiveness of the methods.

5.2 Results

Figure 1 shows the results of this experiment for each of the evaluated test cases. The robustness scores for the baseline, importance sampling, and Metro-Hastings are grouped by test case. Test cases labeled with *"Custom"* were created by us, while test cases labeled with *"Gym"* were provided by PDDLGym. Furthermore, for each test case the value of α used is provided.

The first conclusion that can be drawn from the results presented in Figure 1 is that in general Metropolis-Hastings performs better than importance sampling. In almost every test case Metropolis-Hastings has an equal or higher robustness score than importance sampling. The only exceptions to this are *Gym 0* and *Gym 2*, however even in these cases the difference in scores is not very significant. These results align with the theoretical intuition posited in the previous section that Metropolis-Hastings should perform better because of its more fine-grained nature of inference.

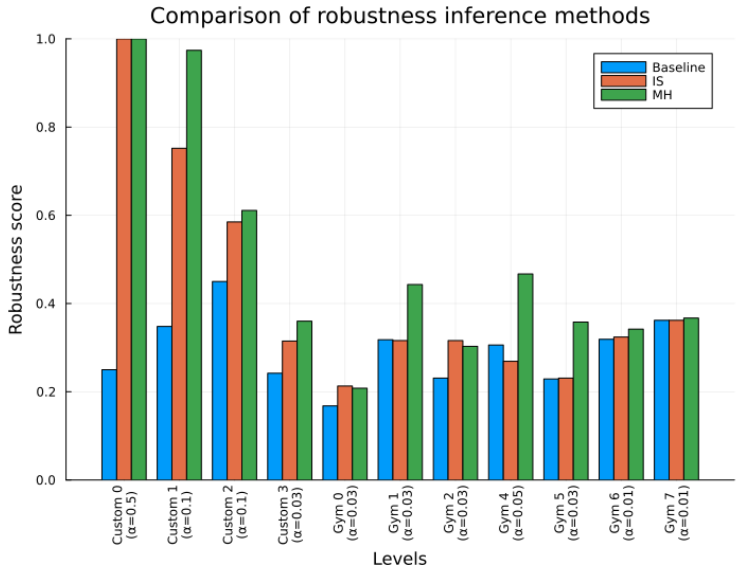

Figure 1: The measured robustness scores of the inference methods for each test case.

The superiority of Metropolis-Hastings is further supported by looking at the distribution of proposed plans for both methods. Figure 2 shows a histogram of the robustness scores of

the plans proposed by importance sampling and Metropolis-Hastings for the *Custom 0* test case. Note that these are the scores computed within the probabilistic model to evaluate a given plan. The scores are therefore less accurate because they are only computed using 100 executions instead of 1000. However, they still give an indication on the behavior of the two approaches.

As can be seen in the figure the distribution for Metropolis-Hastings is much more biased towards plans with high robustness compared to that of importance sampling. This is also reflected by the median score of the plans, which is indicated by the dashed line in the figure. The bias towards higher scores is a result of the fact that Metropolis-Hastings iterates on the previous sampled plan. If the previous plan had high robustness, subsequent plans will be similar and will therefore also have high scores, leading to a distribution that is biased towards high robustness scores. Importance sampling, on the other hand, does not have this iterative characteristic and its distribution is much less biased to high scores because of it.

Metropolis-Hastings being more biased to higher scores than importance sampling holds for the other test cases as well. This is demonstrated in Table 1, which shows the median robustness score of the plans generated by both inference techniques. As can be seen the median for Metropolis-Hastings is significantly higher in every case.

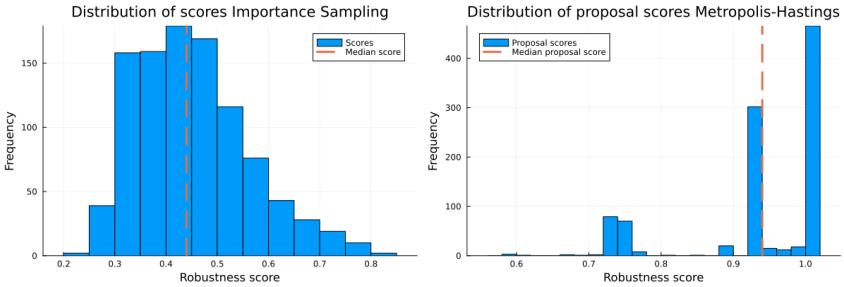

Figure 2: Distribution of plans for *Custom 1* generated by importance sampling and Metropolis-Hastings.

Table 1: Median robustness for Importance sampling's and Metropolis-Hastings' plans.

Level	**Importance sampling**	**Metropolis-Hastings**
Custom 0	0.561	1.000
Custom 1	0.438	0.941
Custom 2	0.387	0.646
Custom 3	0.027	0.430
Gym 0	0.157	0.278
Gym 1	0.268	0.418
Gym 2	0.208	0.378
Gym 4	0.070	0.418
Gym 5	0.200	0.307
Gym 6	0.038	0.338
Gym 7	0.327	0.369

A second conclusion supported by Figure 1's results is that success of generating a robust plan using these probabilistic inference techniques varies significantly depending on the problem instance. In some instances the method manages to increase the robustness significantly, while in other instances the improvement is marginal or even non-existent.

One explanation for this variation is the length of the plan needed to solve the problem instance. Figure 3 shows the relation between the length of the plan in the deterministic version of the instance and the improvement factor the robust plan offers. The improvement factor is calculated as the ratio of the robustness score of the best performing inference method for that instance and the baseline robustness score. In the graph there is a clear trend that the improvement decreases as the problem instance takes more steps to solve. This trend could be explained by the fact that longer problems increase the search space for robust plans making it harder for the inference methods to find plans that significantly improve the robustness.

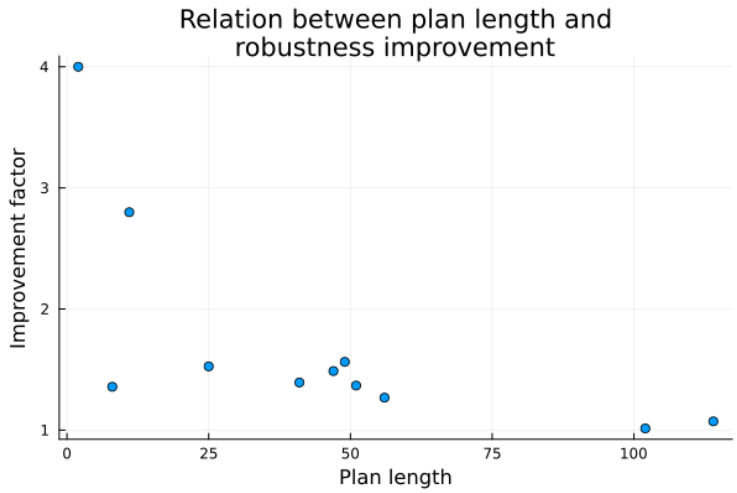

Figure 3: Relation between the length of the deterministic plan and the improvement of the robust plan offers.

5.3 Example Robust Plan

An example of a robust plan gives insight into how the inference makes the plans robust. Figure 4 shows the starting state of the test case *Custom 1*. The robust plan for it is: 5 times down, twice right, left, 4 times up, right 3 times, and down twice. The movements in a straight line in this plan have more moves than the minimum required. These extra moves account for failed moves that could happen. For example, the down movements in the beginning assume that two moves will fail in this sequence. If this happens then the plan will still succeed since it was accounted for, and if less than two moves fail the extra moves will be absorbed by the player walking into the wall a few times. This strategy of making more moves than necessary and using walls to block excess moves is a general theme in the robust plans generated.

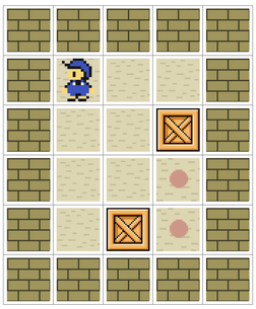

Figure 4: Start state of *Custom 1*.

6 Discussion

The results presented in section 5 indicate that it is possible to create robust plans for the *Uncertain Move Sokoban problem* by using an existing planner and probabilistic inference on a probabilistic model using it. The technique studied in this paper is able to create more robust plans for short to medium-length problem instances, however it struggles with longer ones.

Furthermore, the results show that the approach from [1] can indeed be generalized to other problems. Similarly to [1] we found that Metropolis-Hastings has better performance and can infer more robust plans. This finding is significant, because this means that the approach might also be applicable to uncertain problems of more real-world value than the *Uncertain Move Sokoban problem*.

The approach to create robust plans in this paper uses an unmodified existing planner, however the downside of this is that the approach might never be able to find the robust plan. Consider the example in Figure 5 in which the player must move to the square down right of them. The robust plan on the right moves twice down and then twice right using the walls to stop the player if the double move does not fail. This plan is guaranteed to move to the target square. However, the plan on the left, in which we move right first, does not have this guarantee and is therefore not robust. The problem is that the deterministic planner has no preference for one or the other as they are both the same number of steps and it does not take into account the uncertainty of the problem. So, if the base plan provided to the probabilistic model is the plan on the left the probabilistic inference has no possibility to achieve the optimal robust plan.

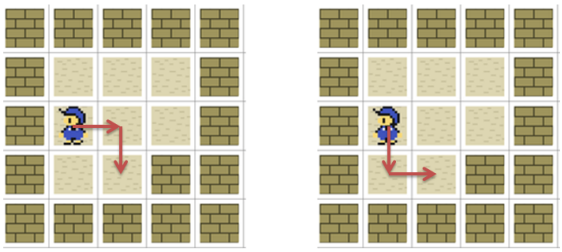

Figure 5: A scenario where the move order does not matter for the deterministic problem, but does for the uncertain problem.

Another limitation of the technique studied is that it is quite slow. In order to generate robust plans the program has to execute thousands of plans, which takes a significant amount of time. Even for relatively small instances of around 50 moves the inference took several minutes. As a consequence two test cases from the PDDLGym set, which had more than 140 moves, were not used for testing as the inference simply did not finish in a reasonable time span. Therefore, it is not known how the technique performs for these very large instances. However, if the trend in Figure 3 can be extrapolated, then the performance will likely not be very different from the instances of length 100.

7 Responsible Research

7.1 Research Ethics and Integrity

In this research we have attempted to uphold a high standard of ethics and integrity. First, the data used in the research was handled in a responsible manner. The research was theoretical in nature and therefore there were no privacy concerns with the data as it did not relate to human subjects. PDDLGym and its data is licensed under the permissive MIT open-source license and we were therefore allowed to use it to evaluate our methods. The complete dataset of levels, containing both PDDLGym and our custom levels, is publicly available in a repository[1], so that other researchers can make use of it.

Second, utmost attention has been paid to crediting the sources used to conduct this research. All scientific literature, software, and data sources used in the research are properly referenced. This attribution is of great importance to uphold academic integrity and provide transparency.

Finally, the reported results have been represented as accurately as possible. The figures directly follow from the raw data of the experiment. Additionally, the levels, that were excluded from the data set for performance reasons, were reported as to not misrepresent the results.

7.2 Reproducibility

Ensuring that the results of the research are reproducible was accomplished in three ways. First, the research method and the experimental setup were described in detail. The description was written such that other researchers would be able to reproduce our setup. This included giving a thorough explanation of the generative model and inference methods used, as well as stating the experimental parameters. Noting these parameters is very important for reproducibility, as they have a significant impact on the results. The α-value of each level is particularly important, as varying it changes the baseline robustness and therefore also the robustness of the inferred plans.

Second, the raw data from the experiments is publicly available in a repository[1]. This data consists of the generated robust plan, its robustness score, and the histogram distribution of plans for both inference methods evaluated. Access to this raw data allows other researchers to reproduce the results in this paper using the exact same data.

Finally, all source code used to conduct experiments and process the experiment data is also included in the previously mentioned repository[1]. A *README file* in the repository gives instructions on how to run the scripts. This enables other researchers to run the exact same experiments on their machine to verify the methods used in this paper.

[1] `https://doi.org/10.5281/zenodo.15715783`

8 Conclusions and Future Work

The aim of this paper was to study how an existing planning algorithm can be used to robustly solve the *Uncertain Move Sokoban problem*. This was achieved by creating a probabilistic model in a probabilistic programming language called Gen.jl. The model uses the plan from an existing planner and generates a plan for the *Uncertain Move Sokoban problem* out of it together with a robustness score for the plan using a simulator embedded in the model. The probabilistic inference techniques importance sampling and Metropolis-Hastings were used in order to infer robust plans from this model. The description of the model and the use of inference techniques answer the sub-questions about how the problem can be modeled, and how a distribution of plans can be used to solve the problem, respectively.

The technique was evaluated on instances provided by PDDLGym and custom-made instances. The results of this evaluation showed that the technique could create robust plans for short to medium-length problem instances, but it was not able to do so for long problems.

Furthermore, the results indicated that Metropolis-Hastings was the preferable inference technique as it was able to generate plans that achieve higher robustness scores than the ones generated by importance sampling. This result answers the last sub-question about which inference technique produces the plans with the highest robustness. Metropolis-Hastings achieved the greater robustness score, because its more fine-grained form of resampling was able to bias the plans it proposed more towards high robustness scores compared to importance sampling.

An avenue for further research is to develop a planner for the *Uncertain Move Sokoban problem*, that takes into account the uncertainty. In this paper we used an existing planner to create robust plans, however this could lead to scenarios in which it is impossible to find the most robust plan. To avoid these scenarios and improve the robustness, a dedicated planner could be built which is aware of the uncertainty in the problem. Such a planner could compute the probability of success for routes to locations in the level and try to solve the problem using only routes with high probability of success. Besides an increase in robustness such an approach could also be faster than our approach as it does not need to simulate thousands of plans.

References

[1] R. Gardos Reid, "Inferring Robust Plans with a Rail Network Simulator", M.S. thesis, Delft University of Technology, 2023.

[2] A. G. Baydin, L. Shao, W. Bhimji, *et al.*, "Etalumis: Bringing probabilistic programming to scientific simulators at scale", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, event-place: Denver, Colorado, New York, NY, USA: Association for Computing Machinery, 2019, ISBN: 978-1-4503-6229-0. DOI: 10.1145/3295500.3356180. [Online]. Available: https://doi.org/10.1145/3295500.3356180.

[3] S. Jiménez, T. De La Rosa, S. Fernández, F. Fernández, and D. Borrajo, "A review of machine learning for automated planning", *The Knowledge Engineering Review*, vol. 27, no. 4, pp. 433–467, 2012. DOI: 10.1017/S026988891200001X.

[4] T. Bylander, "The computational complexity of propositional strips planning", *Artificial Intelligence*, vol. 69, no. 1, pp. 165–204, 1994, ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(94)90081-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370294900817.

[5] J. Culberson, "Sokoban is PSPACE-complete", University of Alberta, Tech. Rep. TR97-02, 1997. [Online]. Available: https://doi.org/10.7939/R3JM23K33.

[6] D. Dor and U. Zwick, "SOKOBAN and other motion planning problems", *Computational Geometry*, vol. 13, no. 4, pp. 215–228, Oct. 1999, ISSN: 0925-7721. DOI: 10.1016/S0925-7721(99)00017-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925772199000176.

[7] M. Helmert, "The fast downward planning system", *J. Artif. Int. Res.*, vol. 26, no. 1, pp. 191–246, Jul. 2006, Place: El Segundo, CA, USA Publisher: AI Access Foundation, ISSN: 1076-9757.

[8] M. Ghallab, Adele Howe, Craig Knoblock, *et al.*, "Pddl - The Planning Domain Definition Language", Yale Center for Computational Vision and Control, Tech. Rep. CVC TR-98-003/DCS TR-1165, Oct. 1998.

[9] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani, "Probabilistic programming", in *Future of Software Engineering Proceedings*, ser. FOSE 2014, event-place: Hyderabad, India, New York, NY, USA: Association for Computing Machinery, 2014, pp. 167–181, ISBN: 978-1-4503-2865-4. DOI: 10.1145/2593882.2593900. [Online]. Available: https://doi.org/10.1145/2593882.2593900.

[10] 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 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019, event-place: Phoenix, AZ, USA, New York, NY, USA: Association for Computing Machinery, 2019, pp. 221–236, ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314642. [Online]. Available: https://doi.org/10.1145/3314221.3314642.

[11] S. T. Tokdar and R. E. Kass, "Importance sampling: A review", *WIREs Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010. DOI: https://doi.org/10.1002/wics.56. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wics.56. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.56.

[12] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications", *Biometrika*, vol. 57, no. 1, pp. 97–109, Apr. 1970, ISSN: 0006-3444. DOI: 10.1093/biomet/57.1.97. eprint: https://academic.oup.com/biomet/article-pdf/57/1/97/23940249/57-1-97.pdf. [Online]. Available: https://doi.org/10.1093/biomet/57.1.97.

[13] T. Silver and R. Chitnis. "PDDLGym: Gym Environments from PDDL Problems". arXiv:2002.06432 [cs.AI]. (2020), [Online]. Available: https://arxiv.org/abs/2002.06432.