

---

# [Re] Utilising Uncertainty for Efficient Learning of Likely-Admissible Heuristics

---

**Jason M. Wille**

School of Computer Science and Applied Mathematics  
University of the Witwatersrand  
Johannesburg, South Africa  
[1352200@wits.ac.za](mailto:1352200@wits.ac.za)

## Reproducibility Summary

*This summary **must fit** in the first page, no exception will be allowed.*

### Scope of Reproducibility

State the main claim(s) of the original paper you are trying to reproduce (typically the main claim(s) of the paper). This is meant to place the work in context, and to tell a reader the objective of the reproduction.

### Methodology

Briefly describe what you did and which resources you used. For example, did refer to the author's code? Or did you implement the method only from the paper's description? Did you recreate the environments, or use existing implementations? You can also use this space to list the hardware used, and the total budget (e.g. GPU hours) for the experiments.

### Results

Start with your overall conclusion — where did your results reproduce the original paper, and where did your results differ? Be specific and use precise language, e.g. "we reproduced the accuracy to within 1% of reported value, which supports the paper's conclusion that it outperforms the baselines". Getting exactly the same number is in most cases infeasible, so you'll need to use your judgement to decide if your results support the original claim of the paper.


### What was easy

Describe which parts of your reproduction study were easy. For example, was it easy to re-implement their method based on the description in the paper? The goal of this section is to summarize to a reader which parts of the original paper they could easily apply to their problem.


### What was difficult

Describe which parts of your reproduction study were difficult or took much more time than you expected. Perhaps the environment description was insufficient and you couldn't verify some experiments, or the author's code was broken and had to be debugged first. Or, perhaps some experiments just take too much time/resources to run and you couldn't verify them. The purpose of this section is to indicate to the reader which parts of the original paper are either difficult to re-use, or require a significant amount of work and resources to verify.

# 1 Introduction

Creating admissible heuristics is desirable as they lead to optimal plans. The challenge faced is that creating strong admissible heuristics often requires expert domain knowledge or high memory resources. In order to work around these issues machine learning techniques are used to develop likely-admissible heuristics. This means that they are  admissible with some probability and therefore the plans produced are also likely-optimal plans with some probability.

The development of likely-admissible heuristics originally required training data that consisted of optimal plans which were prohibitive to produce. This then led to a method that did not require optimal plans, however, it had two main shortcomings. One was that it took very long to train the heuristics and the other was that learning on non-optimal plans meant that errors compounded which lead to heuristics that had high suboptimality.

The paper being reproduced [MR20] aims to overcome both of the shortcoming mentioned in the previous paragraph by  utilising different measures of uncertainty. The paper was able to produce optimality statistics that are competitive with previous approaches trained on optimal plans.

## 2 Scope of reproducibility

The paper being reproduced [MR20] had the main aim of being able to train likely-admissible heuristics on plans that are not always optimal and have the likely-admissible heuristics still perform well compared to heuristics trained on optimal plans. The other shortcoming it tried to alleviate was efficiency, the generation of tasks should perform better than random task generation.

- By modelling uncertainty the heuristics trained on non-optimal plans will be able to solve the 15-puzzle optimally at a percentage of time similar to previous work that trained on optimal plans.
- The approach used in `GenerateTaskPrac` should be more efficient at generating tasks than random task generation.

## 3 Background


### 3.1 Planning

A planning domain is defined as a tuple  $\langle S, O, \varepsilon, C, S_0, S_g \rangle$ . The values are representative of the following:

- $S$  - The state space.
- $O$  -  $O(s)$  is an operator that returns legal operators that can be executed in state  $s \in S$ .
- $\varepsilon$  -  $\varepsilon(s, o)$  is an effect function that returns the next state  $s' \in S$  when operator  $o \in O(s)$  is applied in state  $s$ .
- $C$  -  $C(s, s')$  is a strictly positive and bounded function for the cost of moving from state  $s$  to state  $s'$ .
- $S_0$  -  $S_0 \in S$  is the set of possible start states.
- $S_g$  -  $S_g \in S$  is the set of possible goal states.

For a planning task, the start state and goal state are set to one state each and the start state and goal state may not equal each other ( $s_0 \neq s_g$ ). In [MR20] each domain has a unique goal state and therefore tasks in a domain only differ in start states.

Planning algorithms require a heuristic function  $h(s)$  that estimates the optimal cost-to-goal from state  $s$ . We denote the optimal and unknown cost-to-goal from  $s$  with  $h^*(s)$ . Given a planning task  $T$ , a planning algorithm aims to find a plan  $\pi = (s_0, s_1, s_2, \dots, s_n = s_g)$ . If  $h$  is an admissible heuristic so that  $h(s) \leq h^*(s)$  for all  $s \in S$  then certain planning algorithms guarantee that  $\pi$  is optimal.  $\pi$  is optimal if it is a plan with minimal cost. We denote optimal plans with  $\pi^*$ .

 Guaranteeing admissibility is difficult and a weaker property is for a heuristic to be likely-admissible. A likely-admissible heuristic, denoted  $h^\alpha$ , is admissible with probability  $\alpha \in (0, 1)$ . i.e.  $P(h^\alpha(s) \leq h^*(s)) = \alpha$  for all  $s \in S$ .

### 3.2 Learning Heuristics from Data

Given a plan  $\pi$  we can compute for each  $s_j \in \pi$  where  $j < n$  the cost to goal from  $s_j$  to  $s_n = s_g$ ,  $y_j = \sum_{i=j}^{n-1} C(s_i, s_{i+1})$ . If we are given a dataset consisting of  $M$  plans from the planning domain we can construct a training dataset  $D = \{(x_i, y_i)\}_{i=1}^N$  of size  $N = \sum_{i=1}^M |\pi_i| - M$ . This would be done by calculating the

cost-to-goal for the relevant states in each plan where  $x_i = F(s_i)$  and  $F$  is a function that converts a state to a feature representation that can be used as input to a supervised machine learning algorithm.


We can then learn a heuristic from the dataset  $D$ . In general, such a heuristic is not admissible and therefore will not lead to optimal plans. The quality of the learned heuristic depends on the quality of the plans used for training. Training on optimal or near optimal-plans will produce a heuristic that is a closer approximation to  $h^*$  and therefore have lower sub-optimality.

### 3.3 Bayesian Neural Networks

Neural networks can be viewed as a probabilistic model  $P(y|x, \mathbf{w})$  where  $x$  is the input to the network and  $\mathbf{w}$  are the weights. For regression we generally assume that  $y|x, \mathbf{w} \in \mathbb{R}$  follows a Gaussian distribution such that  $y|x, \mathbf{w} \sim N(\hat{y}(x, \mathbf{w}), \sigma_a^2(x, \mathbf{w}))$ .

Typically when doing regression with neural networks only a single output neuron is used to learn  $\hat{y}$  while  $\sigma_a^2$  is assumed to be a known constant, and this corresponds to the well-known squared loss objective  $(\hat{y} - y)^2$ . However, it is straightforward to adjust the network to have two output neurons and learn  $\sigma_a^2$  as well. This leads to the following minimisation objective function:

$$L(D, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \frac{(\hat{y}(x_i, \mathbf{w}) - y_i)^2}{2\sigma_a^2(x_i, \mathbf{w})} + \frac{1}{2} \log \sigma_a^2(x_i, \mathbf{w}) \quad (1)$$

 In practice, we learn a second output neuron  $p \in \mathbb{R}$  and then apply a transformation to get  $\sigma_a \in \mathbb{R}^+$  such as  $\sigma_a = \log(1 + \exp(p))$ . Then  $\sigma_a^2$  captures the noise in the data and is thus a measure of aleatoric uncertainty. This is the uncertainty that the model cannot describe.

Epistemic uncertainty (which accounts for uncertainty in the model that can be explained away given enough data) is more difficult to model as it requires the posterior distribution  $P(\mathbf{w}|D)$  which is generally intractable to compute.

One effective and computationally efficient method to approximate the posterior distribution is to use a weight uncertainty neural network (WUNN) and choose to model the weights not as fixed point values but as independent Gaussians,  $w_i|D \sim N(\mu_{w_i|D}, \sigma_{w_i|D}^2)$ .

Using a Gaussian prior for the weights with mean  $\mu_0$  and variance  $\sigma_0^2$  so that  $w_i \sim N(\mu_0, \sigma_0^2)$ , the variational approximation to the posterior can be shown to be the following minimisation objective:

$$L(D, \theta) = \beta KL[P(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{P(\mathbf{w}|\theta)}[\log P(D|\mathbf{w})] \quad (2)$$

In this equation  $KL$  is the Kullback-Leibler divergence,  $\theta = \{(\mu_{w_i|D}, \sigma_{w_i|D}^2)\}_i$  for each weight connection in the network and  $\beta > 0$  controls how much weight to give the prior relative to the likelihood. A common strategy is to decay  $\beta$  over time so as to initially give more emphasis to the prior and gradually reduce this as the network is trained on more data.

In practice, the objective function is approximated with Monte Carlo during training by sampling  $z_i^s \sim N(0, 1)$  for each weight connection in the network and then applying the transformations  $w_i^s = z_i^s \sigma_{w_i|D} + \mu_{w_i|D}$  where  $s \in [1, S]$  and  $S$  is the number of Monte Carlo samples averaged over.


Given a WUNN trained to minimise the objective, we can get a measure of epistemic uncertainty for an input  $x$  by sampling weights from the network and computing the variance of the network output

$$\sigma_e^2(x) = \frac{1}{K} \sum_{k=1}^K \hat{y}^2(x, \mathbf{w}_k) - \hat{y}^2(x) \quad (3)$$

where  $K$  is the number of samples from the network and  $\hat{y}(x) = \frac{1}{K} \sum_{k=1}^K \hat{y}(x, \mathbf{w}_k)$ .

Finally, given an input  $x$  we can approximate the posterior predictive distribution as

$$y|x \sim N(\hat{y}(x), \sigma_t^2(x)) \quad (4)$$

 here  $\sigma_t^2(x) = \sigma_a^2(x) + \sigma_e^2(x)$  and  $\sigma_a^2(x) = \frac{1}{K} \sum_{k=1}^K \sigma_a^2(x, \mathbf{w}_k)$ .

## 4 Methodology

Explain your approach - did you make use of the author's code, or did you aim to re-implement the approach from the description in the paper? Summarise the resources (code, documentation, libraries, GPUs) that you used.

### 4.1 Model descriptions

Include a description of each model or algorithm used. Be sure to list the type of model, the number of parameters, and other relevant info (e.g. if it's pretrained).

### 4.2 Environments

For each domain that the method is tested on, provide a description of how it operates, the state space representations, available actions, cost function, etc. Describe whether you implemented them from scratch, or whether you used or modified an existing one.

### 4.3 Hyperparameters

Describe how the hyperparameter values were set. If there was a hyperparameter search done, be sure to include the range of hyperparameters searched over, the method used to search (e.g. manual search, random search, Bayesian optimization, etc.), and the best hyperparameters found. Include the number of total experiments (e.g. hyperparameter trials). You can also include all results from that search (not just the best-found results).

### 4.4 Experimental setup and code

Include a description of how the experiments were set up that's clear enough a reader could replicate the setup. Include a description of the specific measure used to evaluate the experiments (e.g. accuracy, etc). Provide a link to your code.

### 4.5 Computational requirements

Include a description of the hardware used, such as the GPU or CPU the experiments were run on. For each experiment, include the total computational requirements (e.g. the total CPU/GPU hours spent). (Note: you'll likely have to record this as you run your experiments, so it's better to think about it ahead of time). Generally, consider the perspective of a reader who wants to use the approach described in the paper — list what they would find useful.

## 5 Results

Start with a high-level overview of your results. Do your results support the main claims of the original paper? Keep this section as factual and precise as possible, reserve your judgement and discussion points for the next "Discussion" section.

### 5.1 Results reproducing original paper

For each experiment, say 1) which claim in Section 2 it supports, and 2) if it successfully reproduced the associated experiment in the original paper. For example, an experiment training and evaluating a model on a dataset may support a claim that that model outperforms some baseline. Logically group related results into sections.

#### 5.1.1 Result 1

#### 5.1.2 Result 2

### 5.2 Results beyond original paper

Often papers don't include enough information to fully specify their experiments, so some additional experimentation may be necessary. For example, it might be the case that some hyperparameter was not specified, and so different values need to be evaluated to reproduce the original results. Include the results of any additional experiments here, if needed.

### 5.2.1 Additional Result 1

### 5.2.2 Additional Result 2

## 6 Discussion

Give your judgement on if your experimental results support the claims of the paper. Discuss the strengths and weaknesses of your approach - perhaps you didn't have time to run all the experiments, or perhaps you did additional experiments that further strengthened the claims in the paper.

### 6.1 What was easy

Give your judgement of what was easy to reproduce. Perhaps the author's code is clearly written and easy to run, so it was easy to verify the majority of original claims. Or, the explanation in the paper was really easy to follow and put into code.

Be careful not to give sweeping generalizations. Something that is easy for you might be difficult to others. Put what was easy in context and explain why it was easy (e.g. code had extensive API documentation and a lot of examples that matched experiments in papers).

### 6.2 What was difficult

List part of the reproduction study that took more time than you anticipated or you felt were difficult.

Be careful to put your discussion in context. For example, don't say "the maths was difficult to follow", say "the math requires advanced knowledge of calculus to follow".

## References

- [MR20] Ofir Marom and Benjamin Rosman. Utilising uncertainty for efficient learning of likely-admissible heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 560–568, 2020.