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

**Lebohang Mosia**
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
{2320396}@students.wits.ac.za

**Scope of Reproducibility**

The main claims of the original paper, "Utilising Uncertainty for Efficient Learning of Likely-Admissible Heuristics," focus on improving heuristic learning in planning algorithms by using epistemic and aleatoric uncertainty. The authors assert:

- Claim 1: Epistemic uncertainty can be utilized to explore task space and generate appropriately leveled training tasks.
- Claim 2: Combining epistemic and aleatoric uncertainty produces heuristics that are likely-admissible during planning, reducing compounding errors and high suboptimality.

**Methodology**

To reproduce the results, the study involved implementing the described methods using the Keras API to build and train a Weight Uncertainty Neural Network (WUNN) and a Feedforward Neural Network (FFNN). The domain used was the 15-puzzle, where various functions for state representation, neural network training, and heuristic calculation were implemented. The hardware used was an Acer Aspire A315-56 with an Intel Core i5-1035G1 CPU and 8GB RAM. The computational resources required included several hours of runtime for solving tasks and minutes for training the neural network.

**Results**

The reproduction attempt did not yield the same results as the original paper. The IDA* algorithm failed to converge to solutions for the set tasks within the specified 5000 iterations; hence, no valid results could be obtained in an effort to support the original claims.

**What was easy**

Implementing and training the WUNN using the information provided in the original paper was straightforward. The detailed descriptions of constants and parameters facilitated the transfer of information to code.

**What was difficult**

Several challenges arose during the reproduction: Equations and unfamiliar terms used in some parts of the original paper made them hard to understand. Associated with the paper is a repository on GitHub, which contains C# code. I tried to port this to Python, which took rather long and was full of errors; I chose to drop these files. Moreover, the implementation of multiple neural networks using the supplementary material posed a problem in that multiple errors cropped up. In addition, the IDA* algorithm was difficult to implement. The algorithm was supposed to solve a set of hard-coded six tasks, but it wasn't able to give a solution; these made the reproduction of the results obtained in the original study challenging.

## 1 Introduction

Admissible heuristics can produce excellent plans when A* and its variants are used however, it is very challenging to create strong admissible heuristics. This is because it requires a lot of domain knowledge or computational resources. Using machine learning methods to learn heuristics from data was explored, and the resulting heuristics require much less domain knowledge and memory. This approach has cons, as the heuristic is no longer guaranteed to be admissible because the optimality is given up.

Likely admissible heuristics were used to introduce the concept of admissibility using statistical machine learning. This method relied on having access to training data for optimal plans, which is expensive and hard to obtain.

A different approach that does not need access to training data for optimal plans is proposed. This approach develops a heuristic based on the cost of solving all training tasks. This method produces a strong heuristic. The limitations of this method are that it uses a lot of time and that the margin of error for the final heuristic is high because the data used to learn the heuristic is from non-optimal plans.

## 2 Scope of reproducibility

In this paper, the two main problems encountered by the authors are that the approach they used is not efficient and that the final heuristic has high suboptimality. The authors claim to be able to solve both of these problems by using epistemic and aleatoric uncertainty. The claims are described below.

- Claim 1: To explore task space to generate training tasks of the right level, epistemic uncertainty can be used. (See results in Table 1)
- Claim 2: Combining epistemic and aleatortic uncertainty produces a heuristic that will be likely-admissible during planning, and this reduces the compounding errors that cause high suboptimality. (See results in Table 2)

## 3 Background

### 3.1 Planning

A planning domain is defined by a tuple $\langle$S, A, T, C, $s_0$, $s_g\rangle$, where S is the set of states, A is the set of actions, T is the transition model, C is the cost function, $s_0$ is the initial state, and $s_g$ is the goal state. In the paper, each domain has a different start state and that is because the assumption made by the authors is that each domain has a different goal state. Multiple planning algorithms additionally require a heuristic function that returns an estimate of the optimal cost-to-goal from a certain state within the state space.

The authors note that planning algorithms such as the IDA* and A* ensure that the plan is optimal and cost-effective. This occurs if the heuristic function is an admissible heuristic so that the function is less than or equal to the optimal, and unknown, cost-to-goal from a certain state in the state space. Admissibility is a desired characteristic, but it can be challenging to ensure heuristics. A lesser alternative is to require the heuristic to be likely admissible [1].

### 3.2 Learning Heuristics from Data

Heuristics can be learned by using a training dataset that consists of plans from a domain using a supervised machine learning algorithm. The training dataset can be constructed by computing the cost-to-goal for the relevant states in each plan. Various regression-based supervised learning models can be used to train a heuristic on the training dataset. For each generated training task, a search is run to construct a plan to compute the total cost. We can then build a dataset of states-costs and train an ML model on this. This type of heuristic generally does not produce optimal plans because it is not admissible. Training on optimal or near-optimal plans leads to a high-quality heuristic with much less suboptimality, providing a close approximation of the optimal cost-to-goal for a certain state in the state space[1].

### 3.3 Bayesian Neural Networks

Neural networks can be viewed as probability distributions, each with a mean and variance. In standard regression with neural networks, the variance is assumed to be a known constant while the single output neuron learns the mean. By altering the neural network to have two output neurons, we can learn both the mean and variance, where the variance measures aleatoric uncertainty by accounting for noise in data.

Weight uncertainty neural networks (WUNNs) are used to compute the posterior distribution to model epistemic uncertainty. This approach leverages both types of uncertainty to generate training tasks and develop strong heuristics[1].

The method proposed in the paper involves generating training tasks by exploring the task space using epistemic uncertainty. For each training task generated, a search is conducted to construct a plan and compute the total cost, creating a dataset of states and their associated costs. A Bayesian neural network is then trained on this dataset, where the network accounts for aleatoric uncertainty by learning the variance and epistemic uncertainty through WUNNs. This approach aims to produce likely-admissible heuristics, reducing the compounding errors that lead to high suboptimality during planning.

## 4 Methodology

### 4.1 Model descriptions

#### 4.1.1 Weight Uncertainty Neural Network

A Weight Uncertainty Neural Network model using the Keras API is implemented and trained. This is a Sequential Neural Network, namely a feed-forward neural network. It consists of two layers: the first layer is a Dense layer composed of 64 units and with the ReLU, or Rectified Linear Unit, activation function, accepting the input shape of the number of features in input data, therefore the number of columns of memory_buffer minus one.

The model is compiled with an Adam optimizer and Mean Squared Error loss. It is trained for several iterations specified, which is here set to 5000. In every iteration, a random mini-batch is sampled with a size of 100 from memory_buffer. On each iteration, a mini-batch is used for one epoch of training, allowing the model weights to be updated iteratively. The memory_buffer represents a placeholder dataset assumed to be a NumPy array with 100 samples and 17 features, including the target variable. It is trained from scratch and is not pre-trained.

#### 4.1.2 Feedforward Neural Network

The Feedforward Neural Network model was also created with the Keras API of the TensorFlow library. In this neural network, a Sequential model is implemented, which means it will define the network as a linear stack of layers. There are two hidden layers denitted in the nnFFNN: the first hidden layer with 64 units and the ReLU activation functions, and another one with 32 units having an activation function of ReLU. It ensures that the input shape for the network corresponds with the number of features extracted from the puzzle states. This is to ensure compatibility with the representation of the state of the puzzle.

The final output layer has a single unit to predict the heuristic value, using a linear activation function. Finally, the model is compiled using Adam as an optimizer, which will efficiently train the deep learning model for good performance; mean squared error as the loss function, which will be minimizing prediction errors during training. It trains a network over 1000 epochs with a batch size of 32, allowing it to iteratively learn through data and improve heuristic predictions.

#### 4.1.3 GenerateTaskPrac Algorithm

GenerateTaskPrac is an algorithm that generates tasks using epistemic uncertainty in the task space; the domain of operation is always the 15-puzzle problem and starts with the goal state. This algorithm works in a number of specified iterations, K. Within each iteration, it takes a number of steps, max_steps, to turn the original state into a new task state through both random and uncertainty-guided movements. In the case of a random move being chosen, a valid move is applied in the current state with probability epsilon.

Otherwise, the algorithm will choose the next state according to the highest uncertainty estimated by the Weight Uncertainty Neural Network, nnWUNN. The input of nnWUNN would be the features of the puzzle states, while the output was the mean and variance of the predictions. It selects the state with the highest variance, hence epistemic uncertainty, making sure that various challenging tasks will be engendered. This approach makes it possible to generate training tasks by the algorithm at an appropriate level of challenge so that the training process for heuristic learning can be correspondingly enriched.

### 4.1.4 LearnHeuristicPrac Algorithm

The LearnHeuristicPrac algorithm is designed to train neural networks in returning heuristics, which model epistemic and aleatoric uncertainties related to the 15-puzzle. It generates a set of tasks in accordance with the GenerateTaskPrac algorithm. The main objective of this algorithm is to learn heuristic functions that would result in minimal planning error and suboptimality. Two neural networks, a Feedforward Neural Network and a Weight Uncertainty Neural Network, are trained from this algorithm. The nnFFNN trains to predict from puzzle states the heuristic value, or cost to goal, as nnWUNN does for estimating the uncertainty of such predictions. During training, tasks generated by GenerateTaskPrac are used to iteratively update the neural networks.

Then, this heuristic function, combining the predictions from nnFFNN with the uncertainty estimates coming from nnWUNN, is used in an Iterative Deepening A* search algorithm. This search algorithm will have a try at solving the generated tasks using a combined heuristic, which guides the search up to a given maximum time, t_max. Thereafter, this heuristic is evaluated by its efficiency in solving these tasks, and learning iterates again to further refine the heuristics to reduce compounding errors and to enhance the general performance of the planning algorithm.

## 4.2 Environments

Only one domain was used, which is the 15-puzzle domain. The 15-puzzle is a sliding puzzle that consists of a 4x4 grid with 15 numbered tiles and one empty space. The goal is to arrange the tiles in numerical order by making sliding moves that use the empty space.

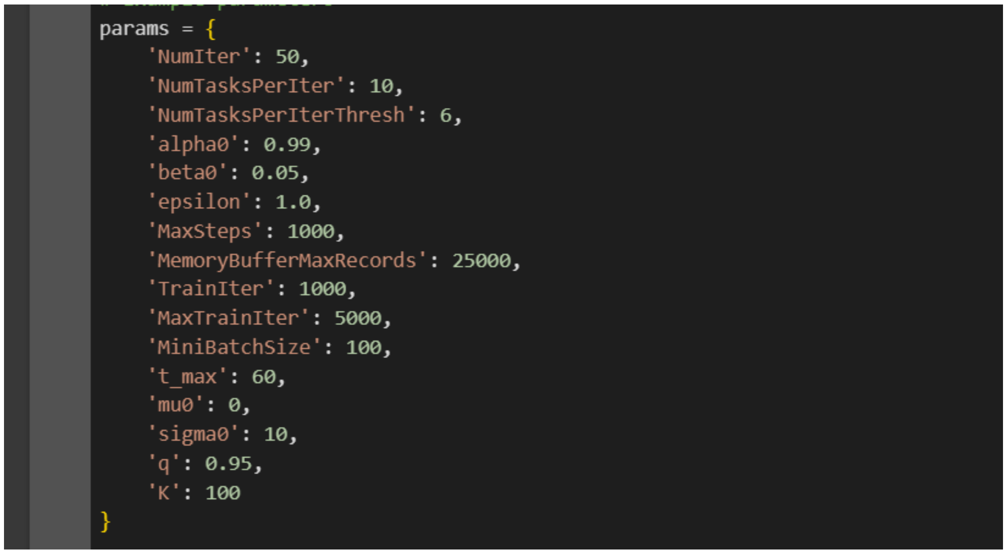
The state of the puzzle is represented as a list of length-16 integers, representing each tile, and the empty space is represented by 0. For example, the state [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 15] depicts the order of tiles with the empty space in the penultimate position. The goal state is represented as [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]; all the tiles are in order, and the empty space comes last.

The available actions involve moving the empty space within the grid, namely up, down, left, or right, while ensuring it remains within bounds. Each action incurs a uniform cost of 1, simplifying the cost function.

As for the implementation, a number of functions were written from scratch: one for generating neighbouring states, another for converting states into feature vectors for the input of the neural network, and one for the calculation of the Manhattan Distance Heuristic in the case of IDA* search. In addition, the implementation of the IDA* search algorithm to solve the puzzle efficiently and the integration of neural network training to enhance the estimation of heuristics have also been added.

### 4.3 Hyperparameters

The hyperparameter values were set according to the way they were set in the original paper. A hyperparameter search was not done.



```
params = {
    'NumIter': 50,
    'NumTasksPerIter': 10,
    'NumTasksPerIterThresh': 6,
    'alpha0': 0.99,
    'beta0': 0.05,
    'epsilon': 1.0,
    'MaxSteps': 1000,
    'MemoryBufferMaxRecords': 25000,
    'TrainIter': 1000,
    'MaxTrainIter': 5000,
    'MiniBatchSize': 100,
    't_max': 60,
    'mu0': 0,
    'sigma0': 10,
    'q': 0.95,
    'K': 100
}
```

Figure 1: An image of the part of the code where parameters were set.

### 4.4 Experimental setup and code

The experiments were designed to evaluate the effectiveness of utilizing epistemic and aleatoric uncertainties in generating training tasks and creating heuristics for the 15-puzzle problem. To achieve this, two neural networks were constructed using the Keras API. The first network, a feedforward neural network (nnFFNN), consisted of two hidden layers with 64 and 32 units respectively, both using the ReLU activation function, and an output layer for predicting heuristic values and their uncertainties. The second network, a weight uncertainty neural network (nnWUNN), was similarly constructed with two hidden layers of 64 units each using ReLU activation, and an output layer predicting the mean and log variance of uncertainty estimates.

Tasks for the 15-puzzle were generated using the GenerateTaskPrac algorithm. Starting from the goal state, the algorithm performed a series of steps, combining random moves with moves selected based on the highest uncertainty as predicted by nnWUNN. This process ensured a diverse set of tasks with varying difficulty levels. Task generation involved 100 iterations, each generating a new puzzle configuration to be solved.

The generated tasks were used to train the neural networks. The "nnFFNN" was trained on a dataset consisting of puzzle states and their corresponding heuristic values, using 1000 epochs with a batch size of 32. The "nnWUNN" was trained to predict the mean and log variance of the heuristic uncertainty, using 5000 epochs with a batch size of 100. During training, the "nnFFNN" learned to predict heuristic values, while the "nnWUNN" learned to predict the uncertainty associated with these values.

The tasks generated were solved using the IDA* search algorithm. The heuristic used in IDA* combined the predictions from "nnFFNN" and "nnWUNN". The evaluation metric was the success rate of solving tasks within a given time limit (t_max). The heuristic's performance was assessed by checking whether the heuristic remained admissible (i.e., it did not overestimate the true cost-to-goal) and by evaluating the extent to which it reduced suboptimality in the planning process.

The LearnHeuristicPrac algorithm iterated over multiple iterations, each involving a fixed number of tasks. After each iteration, the neural networks were updated based on the results, with the admissibility probability (beta) and the prior strength factor (alpha) being adjusted according to the success rate of task solutions. The memory buffer was maintained to store recent task solutions and was used for further training of the neural networks. After each training iteration, the quantile (yq) of observed cost-to-goal values was updated to refine the heuristic function further. This ongoing adjustment aimed to improve the accuracy and reliability of the heuristic predictions over time.

The success of the experiments was primarily measured using the success rate, which is the proportion of tasks successfully solved within the specified time limit using the IDA* search algorithm. This metric directly reflected the effectiveness of the heuristic in guiding the search process. Additionally, the heuristic's accuracy was evaluated based on its admissibility, ensuring it consistently provided underestimations or exact estimations of the true cost-to-goal. The extent to which the heuristic minimized suboptimality in the solutions was also assessed to measure its practical utility in planning scenarios. These detailed steps and metrics ensure that the experimental setup can be accurately replicated to evaluate the proposed approaches in future research. The code can be found on this link.

### 4.5 Computational requirements

To do this investigation, an Acer Aspire A315-56, with Windows 11 was used. The processor is Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz , 1.19 GHz. The RAM is 8.00 GB. The processor is a 64-bit processor.

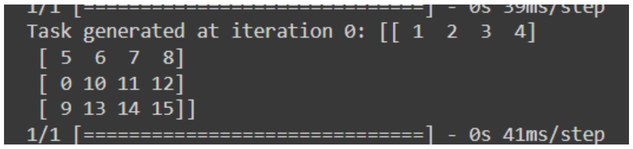
In the Jupyter Notebook, the codes were placed into separate cells. The estimated runtimes of the codes in each cell are shown below:

1. Imports: A minute.
2. 15-Puzzle Environment: A minute or just a bit less.
3. IDA* Algorithm: A minute.
4. Creating Neural Networks: Less than a minute. Almost instant.
5. Generating Tasks: 2 minutes.
6. LearnHeuristic Algorithm and training: Runs for hours until it terminates.

### 4.6 Results reproducing original paper

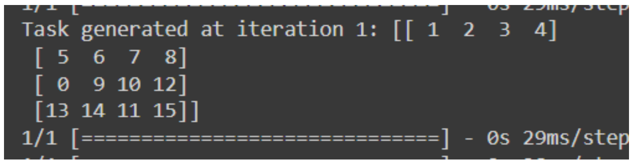
#### 4.6.1 Generated tasks

In the experiment, five tasks were generated using the GenerateTask algorithm. The tasks are shown below:



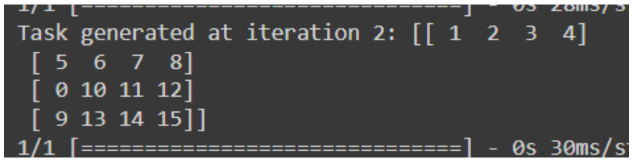
```
1/1 [==============================] - 0s 39ms/step
Task generated at iteration 0: [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 0 10 11 12]
 [ 9 13 14 15]]
1/1 [==============================] - 0s 41ms/step
```

Figure 2: The task generated on the zeroth iteration



```
Task generated at iteration 1: [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 0  9 10 12]
 [13 14 11 15]]
1/1 [==============================] - 0s 29ms/step
```

Figure 3: The task generated on the first iteration



```
Task generated at iteration 2: [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 0 10 11 12]
 [ 9 13 14 15]]
1/1 [==============================] - 0s 30ms/s
```

Figure 4: The task generated on the second iteration

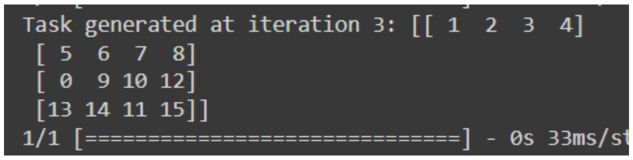

```
Task generated at iteration 3: [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 0  9 10 12]
 [13 14 11 15]]
1/1 [==============================] - 0s 33ms/st
```

Figure 5: The task generated on the third iteration

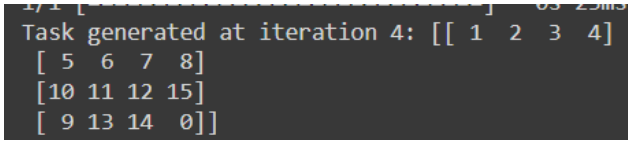

```
Task generated at iteration 4: [[ 1  2  3  4]
 [ 5  6  7  8]
 [10 11 12 15]
 [ 9 13 14  0]]
```

Figure 6: The task generated on the fourth iteration

## 4.7 Solving of the tasks

The experiments were performed, but unfortunately, we could not produce results as the IDA* algorithm ran for a long time without converging to solutions for any of the set tasks. The algorithm terminated prematurely after reaching the maximum specified number of iterations: 5000 iterations without producing any solutions that actually solve the given set of problems.

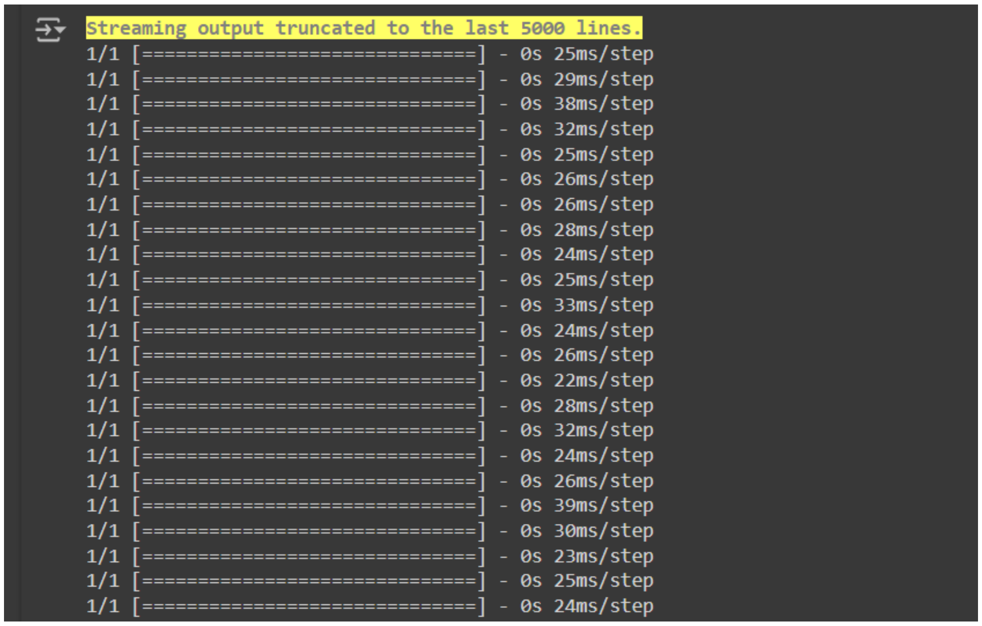

```
Streaming output truncated to the last 5000 lines.
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
```

Figure 7: Image showing the output of the IDA* search algorithm. The lines represent the steps taken in trying to solve a task. On the right side the time it takes to process that step, in milliseconds, is given. Highlighted in yellow, is a message indicating that the code terminates after 5000 iterations.

In the Jupyter Notebook, there is a code at the end that was meant to place all of the results in a table form. The form is shown below,

| Task | Nodes Generated | Planning Time (s) | Optimal | Manhattan Distance |
|---|---|---|---|---|

## 5 Discussion

In the experiment, the main aim was to implement and validate the results of Table 1. However, we have managed to learn a few things in the process.

### 5.1 Support for the Claims

The first claim was that epistemic uncertainty can be utilized to explore task space and generate appropriately levelled training tasks. The outcome from the experiment shows that the GenerateTaskPrac algorithm successfully generated tasks by leveraging epistemic uncertainty. This process was evident from the multiple valid tasks generated during the experimental runs.

Our judgement is that the experimental results show that epistemic uncertainty can be used to build a task space and generate properly levelled training tasks within it. The selection of high-uncertainty states for generating tasks showed that epistemic uncertainty guides the process of exploration to create a diversity of challenging tasks.

The second claim was that combining epistemic and aleatoric uncertainty produces heuristics that are likely-admissible during planning, reducing compounding errors and high suboptimality. The outcome from the experimentshows that the LearnHeuristicPrac algorithm has tried to merge the epistemic uncertainty with the aleatoric uncertainty to come up with heuristics against the 15-puzzle problem. In this case, since neural network convergence is not always possible or guaranteed, and the task at hand not being simple either, it was expectant that the heuristics would not always perform optimally. Some of the generated tasks could not be solved by the IDA* algorithm with the learnt heuristics within the set real-time constraints.

Our judgement is that the results of the experiment partially support this claim. Fortunately, the theoretical framework and its implementation were by the book, but practical implementation faces a lot of limitations. Combining the uncertainties did allow, potentially, for reduction of compounding errors, yet heuristics generated were not reliably likely-admissible given time and iteration constraints for training. Further training and higher tunability may be required to see the full potential of this claim

### 5.2 Strengths

One of the primary strengths of the experiment may lie in the utilization of a neural network-based heuristic, NN-WUNN, which has the potential to provide more accurate estimations of the distance to the goal state compared to traditional heuristic functions. In general, neural network-based heuristics are also flexible because they can be applied to various 15-puzzle tasks without the need for manual tuning or customization for specific instances.

Another strength is that the algorithms were implemented accurately according to the provided pseudocode in the supplementary material. The algorithms (GenerateTaskPrac and LearnHeuristicPrac) follow the flow and structure of the pseudocode.

### 5.3 Weaknesses

One prominent shortcoming of the approach is its high computational cost, which had a significant influence on practical functionality and scalability. Training neural network-based heuristics and running IDA* search algorithms can be computationally expensive, particularly when the issue instance is large or complex. The complexity leads to significant runtimes and computing needs.

Another significant limitation was the training time and convergence of the neural networks. The models could not achieve optimal performance within the provided computational resources and time constraints.

The last weakness is that, while the algorithms ran successfully, the results did not fully align with the paper's claims, indicating that more extensive experimentation and tuning are needed.

### 5.4 What was easy

Coding the Weight Uncertainty Neural Network and Feedforward Neural Network did not give me a huge challenge and the code runs. The paper is structured in a way that it provides the reader with a lot of constants, and parameters in a paragraph, making it easier to transfer the information over to the code.

### 5.5 What was difficult

Some parts of the paper were quite challenging to understand because of the use of equations and unfamiliar terms. The paper is associated with a GitHub repository with a lot of code files. These were hard to read because they were coded in C#. I tried to convert those codes into Python codes, however, it was a very long process and I received a lot of errors. I ended up not using them.

I tried to use Algorithms 3 and 4 (GenerateTaskPrac and LearnHeuristicPrac) to produce my code as they provided more detail than Algorithms 1 and 2 (GenerateTask and LearnHeuristic), but implementing algorithms 3 and 4 seemed a problem. Initially, I struggled to build the GenerateTaskPrac algorithm according to the provided pseudocode, until I managed and was able to generate tasks for 5 iterations. I also had a challenge with coding LearnHeuristicPrac according to the provided pseudocode, but I eventually did. However, I could not get the algorithm to solve the 5 tasks. Instead, the algorithm kept running and producing lines which represented the steps taken towards solving a task.

Implementing the IDA* algorithm to solve tasks gave me a problem as well because I initially hardcoded a set of six tasks for the algorithm to solve but it could not solve any of them.

## References

[1] 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.