Combining heuristics with neural networks*

†

Herschel Darko
Earlham College
Richmond, Indiana

ABSTRACT

Heuristic search algorithms are a common technique seen in AI, especially for path-finding and automated planning. Such algorithms often use an admissible heuristic to guarantee optimal solutions to real-world problems, but memory and time limitations can render this method unfeasible for a number of these problems. A useful alternative is to let these algorithms use inadmissible heuristics instead, resulting in sub-optimal yet practical solutions. This alternative poses the question of which heuristic would produce the solution closest to being optimal while remaining practical for a specific problem. This paper proposes an answer: Use multiple inadmissible heuristics as features of a neural network. This combines the strengths of the individual heuristics via the neural network and results in a single heuristic . This procedure will be tested with the 11-sliding tile puzzle problem, a variant of the more common 15-sliding tile puzzle problem seen in heuristic research.

CCS CONCEPTS

• **Neural Networks;**; • **Problem-solving**; • **A* Search**; • **Heuristic Functions**; • **Sub-optimal Heuristics**;

ACM Reference Format:
Herschel Darko. 2019. Combining heuristics with neural networks: . In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

An admissible heuristic is a function that never overestimates the cost to move from a single problem state to the problem's goal state. When A* uses an admissible heuristic for a problem, it is guaranteed to find the cheapest path to the goal, the problem's optimal solution. Searching for a problem's optimal solution can be extensive, depending on the complexity of the problem and the number of possible states [7]. Such complex problems include real-time pathfinding and pathfinding in large maps in commercial video games.[2] These problems require a practical solution, and can be solved with a sub-optimal search [3]. One method for A* to perform a sub-optimal search is for A* to use an inadmissible heuristic. We believe that research into building stronger inadmissible heuristics is beneficial for obtaining cheaper but still sub-optimal solutions. This research topic we believe is less common as most studies for building heuristics, especially for A*, focus on solving problems optimally [10]. This paper proposes the use of a neural network, that has been trained using multiple inadmissible heuristics, as part of a new heuristic for the 11-sliding-tile puzzle problem. We seek to introduce a new inadmissible neural network heuristic that, when paired with A*, produces practical solutions that are more optimal and require the same amount of resources as A* using the individual inadmissible heuristics. These solutions should also have a greater cost but require less resources than the optimal solutions obtained by A* using an admissible heuristic.

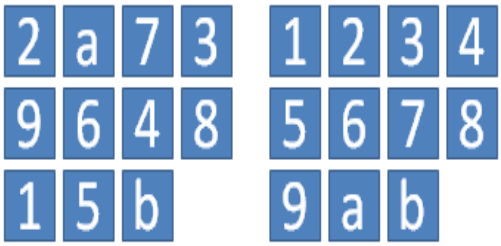

Figure 1: A random solvable configuration (left), and a goal configuration (right) of an 11-puzzle

2 BACKGROUND

2.1 11 Sliding-Tile Puzzle

The sliding puzzle is a classical problem for modelling algorithms that use heuristics [8]. This makes it a suitable choice to experiment with the neural network heuristic and observe its performance. The 11 Sliding Tile puzzle, or 11-puzzle is a 3 x 4 grid with 11 numbered square tiles and one empty (blank) position. Any tile horizontally or vertically adjacent to the blank can be moved into the blank's position. The goal is to shift all the tiles into an ordered goal configuration from a randomized, solvable initial configuration.

2.2 A* Search Algorithm

The A* algorithm is a best-first heuristic search algorithm. A* starts with a root node, which contains the initial state of the problem that A* is trying to solve. A* then performs every possible action in that initial state and the new problem states generated are stored in child nodes. As an example,the root node shown in Figure 2 has three possible actions ("Down", "Right" and "Left") in it's state and generates three child nodes from the result of those actions.

A*, if using an admissible heuristic, finds a guaranteed optimal path to a specific goal node from the root node [6]. The goal node

*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

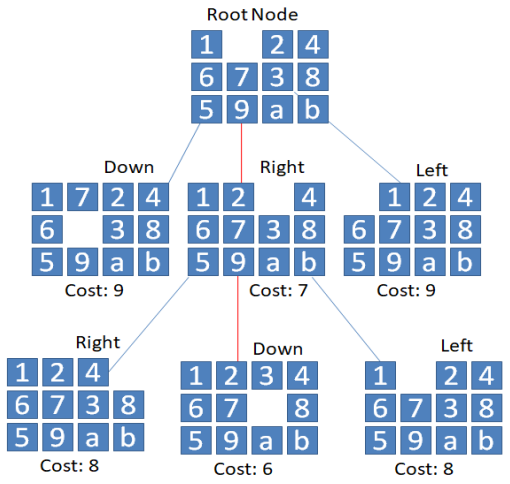

Figure 2: A* expanding nodes using heuristic costs

is any node containing a goal state for the problem, which is the state with the tiles in the correct order for the sliding tile puzzle. Each path to the goal node has a cost, with the optimal path being the cheapest path. In order to continue a path, A* assigns as cost to each child node of the current problem node and then expands the child node with the lowest cost. The cost A* assigns is the result of the formula, f(n) = g(n) + h(n). In this case, n is the child node, g(n) is the cost of the path from the root node to this node and h(n) is the value return by a heuristic for the node's state. Using the Linear Conflict heuristic and ignoring g(n) for all nodes for Figure 2 as an example, A* picks the "Right" then "Down" nodes sequentially because they have the lowest heuristic cost of their neighbors. This example shows how A* chooses the next node to expand, but g(n) is another cost that adds to the overall cost or f(n) of a node.

A* stores every node in one of two lists. The first list is the Open List and contains every child node that A* has generates but has not expanded yet. The second list, the Closed List contains every node A* has already expanded. In Figure 2, A* stores the root node, the first "Right" node, and the "Down" node in the Closed List, and stores the remaining nodes in the Open List. A* picks the node with lowest f(n) out of the Open List and expands it by generating child nodes for every possible action in that node's state. Since each child node contains a reference to it's parent node, this allows A* to extend the current path or even different paths if the node with the lowest cost expands from a earlier node in the current path. The drawback of A* is its ability to expand an exponential number of nodes, depending on the number of states for a problem. Using A* for the more complex 24-sliding tile, for example, results in about 10 ^ 25 possible nodes [9] which is not feasible for our paper due to the memory and time costs.

2.3 NEURAL NETWORK

An artificial neural network, or neural network, is a machine learning model that learns to perform a task through training. This training consists of giving the neural network examples of input data along with the expected output or label. The neural network also requires a way to measure the difference between the output it predicts and the expected output. The neural network uses this measure to assess its performance so it can update and improve it's accuracy. A neural network is part of a specific subset of machine learning known as deep learning. This means that the neural network is made up of successive layers with each layer receiving its input from the preceding layer and sends its output to the succeeding layer. It is the final layer that determines what the neural network's measure will be.

Neural networks can learn non-linear relationships between the input and label data. They can also generalize and predict the output of new input data after learning these relationships, making neural networks useful for data grouping and classifying. The use of a neural network in this paper means that there is no need to create a specific formula or calculation to make an inadmissible heuristic from the other heuristics, as the neural network learns from example and these examples can differ depending on the problem. However, the neural network must be carefully designed, both the number of layers along with the individual layers, so that it can predict the output with an acceptable degree of accuracy.

2.4 RELATED WORK

Few studies have been done on inadmissible heuristics, but a number of studies have been done on neural networks.

Wilt, C., & Ruml, W., (2016) [10] examined how and why the techniques for constructing heuristics for optimal search weren't effective for sub-optimal search. They used greedy best-first search to prove established wisdom for creating heuristics for A* resulted in poor results for a sub-optimal heuristic search algorithm. They used the 11 sliding-tile puzzle over the 15 sliding-tile puzzle because their available memory was not enough for a 15 sliding-tile puzzle. This demonstrates the need to research new methods for sub-optimal heuristic search as using established wisdom gives poor results.

Arfaee et al [1] implemented a bootstrapping procedure proposed in another paper. They wished to validate the earlier paper by adding the necessary details to make the procedure practical, then they tested the procedure via experiments. Bootstrapping used a weak heuristic, h0, and a heuristic search algorithm to solve a set of problems within a time limit. The set of solved problems, along with their solution lengths, were then fed into a neural network, which outputted a new heuristic h1. Bootstrapping would use h1 to solve any remaining unsolved problems from the initial set and continued the cycle of feeding input to the neural network and building stronger heuristics. This method was proven to be a success and shows that weak heuristics can be used to build a stronger one via machine learning.

Jyh-Da Wei et al (2011) [4] proposed a method by which a neural network could be trained to form an efficient heuristic function via an adaptive heuristic search procedure. They used an inadmissible heuristic and the A* search algorithm. Their procedure extracted a vector of features from a node along with it's heuristic value

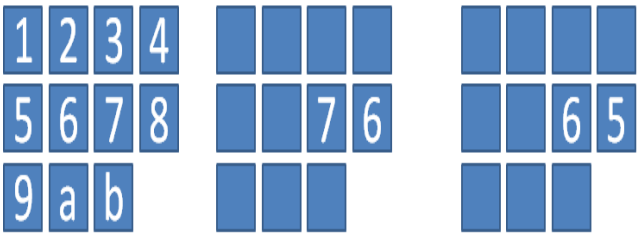

Figure 3: Solved puzzle(left) and two states with a linear conflict(right)

and fed them to the neural network as the input vector and label respectively. The output of the neural network, called A(N), was then used in a series of functions called the Adaptive Heuristic Search to change the heuristic values of the A* search's child nodes. The nodes were then put into the Open list. Their results showed that the solutions obtained by this method were close to the optimal path and use of the method reduced the size of the Closed list. Thus means the method reduced the number of resources needed while returning cheaper sub-optimal solutions.

3 METHODOLOGY

3.1 Neural Network Architecture

We build our neural network from four successive layers. The first layer is an Embedding layer; this layer accepts integer vectors of a fixed size. After processing the input fed to it, the Embedding layer sends the output to the next layer, the LSTM layer. The LSTM layer recognizes patterns in sequences of input data; this layer also has a form of memory as it uses the previous input it has received, along with the current input to produce its output. The third layer is a Dense layer which takes all the input data it receives from the previous layer, processes it and sends the generated output from each input to the next layer.

The last layer is a Dense SoftMax layer that returns a vector of four probability scores whose sum is one. These scores each represent the probability that the current input data maps to one of the four output moves. Since our output is four exclusive moves for a puzzle state, this layer is ideal for outputing the results of the neural network. The measure used for our paper is the categorical crossentropy loss function which finds the distance between the probability scores produced by the neural network and the one in the expected label for that specific input. The smaller the distance the closer the neural network gets to outputting the expected output or label.

Each layer is chosen as a result of experimentation with multiple layers, with every layer tested to see how it affects the accuracy of the current architecture made up of previous layers. The current design for the neural network is the most effective one we produced but can be further expanded on or redesigned in future work to improve the accuracy further.

3.2 Heuristic Functions

By using the heuristics as the input of the neural network, the neural network heuristic becomes a general use heuristic since a problem can have it's own set of heuristics to estimate costs. The output of the neural network is an action from the set of possible actions for the given problem.

As shown in Figure 2, it is possible for different puzzle states to have the same heuristic cost. Using multiple heuristics per problem state alleviates this problem as it differentiates these states. Different costs will have different meanings depending on which heuristic they came form. For example, a cost of two for the Misplaced Tile heuristic means one tile is misplaced, but the Tiles Out of Rows and Columns heuristic may output a cost of two for the same puzzle state if the misplaced tile is either in in the wrong row or wrong column or a cost of four, if the tile is in the wrong row and the wrong column.

A sufficient number of heuristics needs to be chosen for the neural network's input. This number can not be too large else the neural network heuristic is impractical for problem solving compared to using individual heuristics. For our preliminary experiment, three inadmissible heuristics are the neural network's input per 11-puzzle state. These heuristics each estimate a cost to reach a specific goal node from a current node and the cost values are grouped together in an input vector for the neural network. The heuristics we use for the input vector are as follows:

- Sequence Score - This heuristic function adds a sequence cost to the Manhattan Distance. The Manhattan Distance is the sum of the horizontal and vertical distances of each tile from their goal positions and is also an admissible heuristic. The Sequence cost adds an extra cost of two for each tile not followed by its proper successor tile as well as a cost of one if the blank is not in it's goal position.
  As an example is the sequence "1234". If numbered tiles of "1" to "4" follow this sequence then there is no extra cost. However, if the sequence of tiles is "1", "3","2" then "4" that would result in a cost of 6 being added as shown in Figure 3. This results in a inadmissible heuristic that gives more information on the puzzle state than the admissible Manhattan Distance alone.
- Misplaced Tiles - This heuristic is the total number of numbered tiles that are misplaced from their correct positions, including the blank .
- Tiles Out Of Rows and Columns - This heuristic is the sum of the number of tiles, including the blank, that are not in their correct rows and the number of tiles not in their correct columns.

The output of the neural network is a problem action, for the 11-puzzle, a move. This focus on moves instead of cost is to narrow the possible outputs of the neural network and focus on improving the accuracy of those predictions. Ideally, this move the neural network predicts, for the current node, create the child node with the lowest cost when compared to the other possible child nodes. A*, with an admissible heuristic, can determine which child node has the lowest cost and thus, which move is the expected output per state. By using this method to generate the output move per state, the neural network heuristic, when used by A*, is more likely to produce near-optimal solutions. At the same time, the neural network heuristic can't guarantee admissibility therefore the heuristic

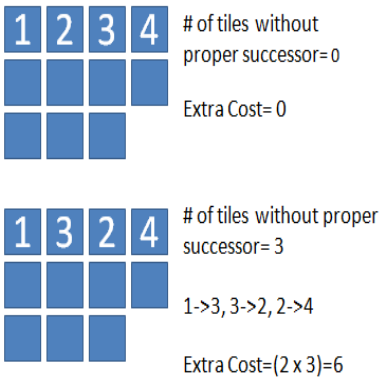

Figure 4: Sequence Score adding extra cost

is still inadmissible. The admissible heuristic we use to generate the moves for this paper is:

- Linear Conflict- This heuristic adds a cost of 2 to the Manhattan Distance for each pair of conflicting tiles in the board configuration. Two tiles x and y are in a linear conflict if x and y are in the same row , the goal positions of both tiles are also in the row, x is to the right of y and goal position of x is to the left of the goal position of y.

The Linear Conflict heuristic has been proven to be more efficient than just using the Manhattan Distance as it explores less nodes to find a optimal solution and is closer to the optimal cost.[5]. While we chose the previous inadmissible heuristics for their ease of implementation and availability, we chose Linear Conflict as the admissible heuristic over Manhattan Distance, despite being more complex to implement, because of its effectiveness.

4 EXPERIMENT DESIGN

The paper compares the results of A*, using the three inadmissible heuristics, against our heuristic search algorithm that uses the neural network heuristic. These results are the length of the solutions if any are found along with the size of the Open List and Closed lists. The size of the lists provides an estimate of the resources needed by the 2 algorithms.

4.1 Our Heuristic Search Algorithm

This algorithm takes a root node and stores the output costs of the three inadmissible heuristics, for the node's problem state, in a vector for the neural network to receive as input.

The neural network outputs an array containing the probability for the blank to move one tile Down from its original position in the node's problem state, move one tile Left from the original position, moved one tile Up from the original position and move one tile Right from the original position. Each move is represented by index 0, 1, 2 and 3 respectively and the sum of the probabilities is one. The array output is [Down, Left, Up, Right]. An output of [0.12, 0.26, 0.20, 0.42], for example, states that the neural network predicts moving the blank one tile right has the highest possibility to be the output move, with Left being the second highest possibility. The heuristic assigns a cost to the possible child nodes created from

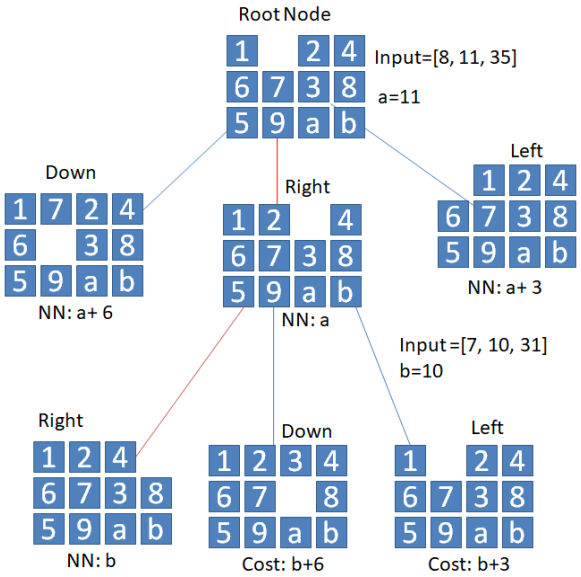

Figure 5: The Neural Network heuristic

these moves. This cost is based on the probability score of the move that generated the child node. The child nodes are then stored in an Open List and the node with the lowest cost is taken out for expansion and stored in the Closed List.

The cost assigned to a child node is based on the formula, a + 3(p-1). For this paper, 'a' is the average of the three heuristic costs that are in the input vector and weight for (p-1) is 3. These are the preliminary values for 'a' and the weight. Future work should involve different calculations for 'a', and the weight to determine the optimal values. 'p' represents the move for the child node having the p highest probability in the neural network's output array. For example, using Figure 5, neural network takes an input vector of [8, 11, 35] and outputs [0.20, 0.27, 0.17, 0.38]. Moving the blank one tile right or Right has a probability of 0.38. Right has the 1st highest probability so the formula is a + 0 or a. Left has the 2nd highest probability so the formula is a + 3. Right has a cost of a + 6.

This formula is chosen because the neural network heuristic should not assign a fixed cost based on the p highest probability or else multiple nodes in the Open List will have the same cost. The heuristic search algorithm should be able to expand nodes that were stored in the Open List but were not previously chosen for expansion. Using a fixed cost means that only the nodes whose moves have the highest probability will be chosen for expansion at any point. Depending on the heuristic's performance, this may be a method to be explored in future work but for this preliminary experiment the heuristic will assign dynamic costs. As shown in Figure 5, the formula of a + 3(p-1) allows 'a' to change per problem state. Initially a = 11 but for the next expansion a= 10 which we rename from 'a' to 'b' to highlight the difference. Regardless of the the cost of 'a', the node with the highest probability will have the lowest cost, with second highest having the second lowest cost and so on. Thus, the neural network's predictions are responsible for

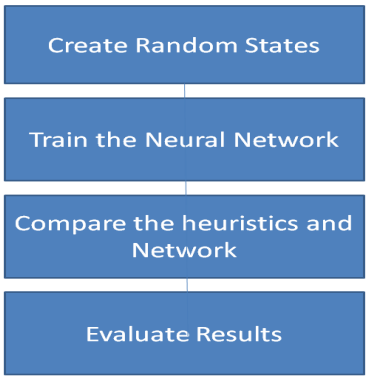

Figure 6: A chart of the flow between the parts of the program

the paths that the heuristic algorithm takes, instead of the value used for 'a'.

4.2 Experimentation

The first step of the experiment is to have a function generate random initial puzzle states for the puzzle. These board states are all solvable as the function creates them by starting with a solved puzzle and making a number of random moves. For this experiment we will test with 5, 10 and 20 and 40 random moves, to see how scaling up the puzzle's complexity affects the neural network heuristic.

The second step is to train the neural network. The training uses solvable puzzle states created from 45 random moves. The maximum number of moves required to solve an 11 puzzle was estimated by our calculations to be 45 so we picked it for the puzzle states.

The third step is compare the A* and the new heuristic search algorithm. We run A* using each of heuristics mentioned earlier in the paper, (Sequence Score, Misplaced Tiles, Tiles out of Rows and Columns, Linear Conflict), on the states stored in the file and record the length of their solutions and the size of the two node lists. We perform the same with the new algorithm.

The final step is evaluate the results and see how the new algorithm's solutions compares to A*'s. Ideally the size of the lists are similar and the length of the solution is smaller to those obtained from the inadmissible heuristics

5 EXPERIMENT RESULTS

The neural network this paper uses is able to achieve a 50% accuracy. On individual puzzle states, the neural network predicts the expected move 50% of the time. The neural network was found to be a less efficient heuristic when compared to the other heuristics, outputting solutions with greater cost and requiring more resources regardless of how complex the initial puzzle state is.

6 DISCUSSION

The neural network proved to be a less reliable heuristic than expected. With a accuracy of only 50%, the neural network, when used as a heuristic, was observed to rarely tie with inadmissible heuristics on simpler puzzles and be worse on more complex problems. The new heuristic search algorithm is still able to solve the puzzles so we believe that improving the neural network's accuracy to at least 68-70/

7 FUTURE WORK

A neural network that uses the heuristics for its input and is a part of an inadmissible heuristic is a concept that can be explored in a variety of ways. Future work should try to increase the accuracy of the neural network, either possibly by increasing the number of inadmissible heuristics used in the neural network's input, using more distinct heuristics to distinguish between different states, using a different neural network architecture or by other means. Another method is to use different means to calculate the 'a' value used as the base cost for the neural network heuristic and the weight used for the dynamic cost of the neural network heuristic. A different possibility to use a problem domain that requires the neural network to output a move from a different set of actions, or have the neural network output a value that can be used directly as a heuristic output instead of a move or action.

More research on inadmissible heuristics and ways to make them more effective will let us explore an rarely-studied topic and discover new ways to approach problems and solve them.

REFERENCES

[1] Shahab Jabbari Arfaee, Sandra Zilles, and Robert C.Holte. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175(16-17):2075–2098, 2011.
[2] Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1(1):7–28, 2004.
[3] Ethan Burns, Wheeler Ruml, and Minh Binh Do. Heuristic hearch when time matters. *Journal of Artificial Intelligence Research*, 47:697–740, 2013.
[4] Hung-Che Chen and Jyh-Da Wei. Using neural networks for evaluation in heuristic search algorithm. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.
[5] Othar Hansson, Andrew E. Mayer, and Mordechai M. Yung. Generating admissible heuristics by criticizing solutions to relaxed models. 1985.
[6] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
[7] Malte Helmert and Gabriele Röger. How good is almost perfect? In *AAAI*, volume 8, pages 944–949, 2008.
[8] Richard E. Korf. Recent progress in the design and analysis of admissible heuristic functions. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 45–55. Springer, 2000.
[9] Richard E. Korf and Larry A Taylor. Finding optimal solutions to the twenty-four puzzle. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1202–1207, 1996.
[10] Christopher Wilt and Wheeler Ruml. Effective heuristics for suboptimal best-first search. *Journal of Artificial Intelligence Research*, 57:273–306, 2016.