Obtaining Approximately Admissible Heuristic Functions through Deep Reinforcement Learning and A* Search

Forest Agostinelli,[1,2] **Stephen McAleer,** [3] **Alexander Shmakov,** [3] **Roy Fox,** [3] **Marco Valtorta,** [2] **Biplav Srivastava,** [1,2] **Pierre Baldi** [3]

[1] AI Institute, University of South Carolina
[2] Department of Computer Science and Engineering, University of South Carolina
[3] Department of Computer Science, University of California, Irvine
foresta@cse.sc.edu, smcaleer@uci.edu, ashmakov@uci.edu, royf@uci.edu, mgv@cse.sc.edu, biplav.s@sc.edu, pfbaldi@uci.edu

Abstract

Deep reinforcement learning has been shown to be able to train deep neural networks to implement effective heuristic functions that can be used with A* search to solve problems with large state spaces. However, these learned heuristic functions are not guaranteed to be admissible. We introduce *approximately admissible conversion*, an algorithm that can convert any inadmissible heuristic function into a heuristic function that is admissible in the vast majority of cases with no domain-specific heuristic information. We apply approximately admissible conversion to heuristic functions parameterized by deep neural networks and show that these heuristic functions can be used to find optimal solutions, or bounded suboptimal solutions, even when doing a batched version of A* search. We test our method on the 15-puzzle and 24-puzzle and obtain a heuristic function that is empirically admissible over 99.99% of the time and that finds optimal solutions for 100% of all test configurations. To the best of our knowledge, this is the first demonstration that approximately admissible heuristics can be obtained using deep neural networks in a domain independent fashion.

Introduction

Path finding algorithms such as A* search (Hart, Nilsson, and Raphael 1968) are used to find a path to a goal state from any given starting state. These path finding algorithms use heuristic functions to give an estimate of the cost required to reach the goal state from any given state. Using heuristic functions, the amount of computation required to find a path to the goal state can be drastically reduced. One important aspect of heuristic functions is *admissibility*. A heuristic function is said to be *admissible* if it never overestimates the cost of a shortest path. Given an admissible heuristic function, A* search is guaranteed to find a shortest path to the goal, also referred to as an optimal solution, from any given state, provided that a path exists (Hart, Nilsson, and Raphael 1968). Additionally, given an admissible heuristic function, modified versions of A* search allow one to find a path to the goal with bounded suboptimality (Pohl 1970, 1973; Harris 1974; Ghallab and Allard 1983; Valenzano et al. 2013). Having these theoretical guarantees for real world applications would ensure that artificial intelligence agents can solve problems in the most efficient way possible, or close to the most efficient way possible, which could significantly reduce the resources consumed by such agents.

Obtaining an admissible heuristic function often requires domain-specific knowledge. For example, pattern databases (PDBs) (Culberson and Schaeffer 1998) have been successful at finding optimal solutions to puzzles such as the Rubik's cube (Korf 1997), 15-puzzle, and 24-puzzle (Korf and Felner 2002; Felner, Korf, and Hanan 2004). However, ensuring that these PDBs produce admissible heuristics requires knowledge about how the puzzle pieces interact. There has been previous research on using deep neural networks to learn heuristic functions (Chen and Wei 2011; Wang et al. 2019; Ferber, Helmert, and Hoffmann 2020) including the DeepCubeA algorithm (McAleer et al. 2019; Agostinelli et al. 2019) which used deep reinforcement learning and weighted A* search (Pohl 1970) to solve the aforementioned puzzles. However, the heuristic functions produced by DeepCubeA are not admissible.

In this paper, we define an approximately admissible heuristic function as one whose overestimation is bounded by a reasonably small e where overestimation occurs in $100 - \delta$ percent of cases, where δ is small. To convert an inadmissible heuristic function to an approximately admissible heuristic function, we first obtain a set of states that is representative of the state space where each state in the representative set is assigned an approximately admissible heuristic value which is initially a trivially admissible value of zero. We then iterate over two steps: (1) adjust the inadmissible heuristic function using the approximately admissible heuristic values on the representative set; (2) update the approximately admissible heuristic values on the representative set by performing A* search, without having to search all the way to the goal, with the adjusted heuristic function. These two steps are repeated until the average value of the adjusted heuristic function stops increasing.

Since we are using deep neural networks, which are complicated non-linear functions of the state, and a limited size representative set, we cannot guarantee the approximately admissible conditions how for all states. However, we empirically show that e and δ are small and that we find a shortest path for 100% of all test states. We also show that we can relax the adjustment model to obtain an approximate additive bound (Harris 1974; Valenzano et al. 2013) on the cost of a solution. This algorithm builds on the DeepCubeA algorithm; therefore, we call the resulting algorithm DeepCubeA-Admissible (DeepCubeA2).

Related Work

Harris (1974) showed that heuristics that are approximately admissible can still find optimal solutions, often much faster than strictly admissible heuristics, while also being easier to construct. Valenzano et al. (2013) built on this work by investigating heuristic functions bounded by an additive bound. Pearl and Kim (1982) investigated search algorithms that explicitly take into account the risk of terminating search with an approximately admissible heuristic function. Given an admissible heuristic function, weighted A* search (Pohl 1970) guarantees bounded suboptimal solutions while potentially finding solutions much faster. These methods, and others, are explored further in Pearl (1984).

Neural networks have been shown to be able to learn informative heuristic functions. Ernandes and Gori (2004) use supervised learning to train a neural network to learn a heuristic function. Using an uneven error-function, they encourage the neural network to be admissible and were able to find optimal solutions to the 15-puzzle in 50% of cases. Samadi, Felner, and Schaeffer (2008) use a neural network to combine multiple given heuristics into a single heuristic. Arfaee, Zilles, and Holte (2011) and Chen and Wei (2011) bootstrap from given heuristics to learn an improved heuristic using a neural network and solve many of the test instances optimally.

Preliminaries

Deep Approximate Value Iteration

Value iteration (Puterman and Shin 1978) is a dynamic programming algorithm and a core reinforcement learning algorithm (Bellman 1957; Bertsekas and Tsitsiklis 1996; Sutton and Barto 1998) that iteratively improves a cost-to-go function J, that estimates the cost required to reach the goal state. This cost-to-go function J can be readily used as a heuristic for path finding algorithms.

In traditional value iteration, J takes the form of a lookup table where the cost-to-go $J(x)$ is stored in a table for all possible states x. Value iteration loops through each state x and computes an updated $J'(x)$ using the following equation:

$$J'(x) = \min_a \sum_{x'} P^a(x,x')(g^a(x,x') + \gamma J(x')) \tag{1}$$

Here $P^a(x,x')$ is the *transition matrix* representing the probability of transitioning from state x to state x' by taking action a; $g^a(x,x')$ is the *transition cost*, the cost associated with transitioning from state x to x' by taking action a; and γ is the *discount factor*.

However, representing J as a lookup table is too memory intensive for problems with large state spaces. Therefore, we turn to approximate value iteration (Bertsekas and Tsitsiklis 1996) where J is represented as a parameterized function. J is trained by minimizing the mean squared error between $J(x)$ and an updated $J'(x)$ obtained from Equation 1. We choose to use a deep neural network (DNN) (Schmidhuber 2015) to represent J. This combination of deep neural networks and approximate value iteration is referred to as deep approximate value iteration (DAVI). In the case of the 15-puzzle and 24-puzzle, all transitions are deterministic and all costs are 1. We set γ to 1 for all experiments.

A* Search

A* search (Hart, Nilsson, and Raphael 1968) is a search algorithm that finds a path between the node associated with the starting state x_s and the node associated with the goal state x_g. A* search maintains a set, OPEN, from which it iteratively removes and expands the node with the lowest cost and a set, CLOSED, that contains nodes that have already been expanded. The cost of each node is $f(x) = g(x) + h(x)$, where $g(x)$ is the path cost, the distance between x_s and x, and $h(x)$ is the heuristic, the estimated distance between x and x_g. After a node is expanded, that node is then added to CLOSED and its children that are not already in CLOSED are added to OPEN. If a child node x is already in CLOSED, but a less costly path from x_s to x has been encountered, then x is removed from CLOSED and added to OPEN with the updated path. The algorithm starts with only the node associated with the starting state in OPEN and terminates when the node associated with the goal state is removed from OPEN.

Given an admissible heuristic, A* search is guaranteed to find a shortest path to the goal. A heuristic function h is admissible if $h(x) \leq h^*(x)$ for all possible states x, where $h^*(x)$ is the cost of a shortest path from x to the goal (Hart, Nilsson, and Raphael 1968).

Methods

Approximately Admissible Conversion

While there are successful applications of approximate value iteration using DNN approximators (Bertsekas and Tsitsiklis 1996; Agostinelli et al. 2019; Ferber, Helmert, and Hoffmann 2020), they do not address the admissibility of J, which will serve as the heuristic function. As shown in previous work (Agostinelli et al. 2019), the learned heuristic overestimates the cost of a shortest path over 30% of the time. We address this problem using a set of states that is representative of the state space and that has an approximately admissible heuristic value assigned to each state. We can compute the maximum amount that the learned heuristic function will overestimate the approximately admissible heuristic values on the representative set and subsequently adjust the heuristic function so that it does not overestimate any of these heuristic values. As we will show in the Results section, as the size of the representative set increases, the empirical values for e and δ decrease. Each state in the representative set is initially assigned a trivially admissible heuristic value of zero. We then iterate over the following

Algorithm 1: Approximately Admissible Conversion

```
Input:
    $h$: Inadmissible heuristic function
    $\mathcal{X}$: Representative set
    $\eta$: target increment for $h^a$
Output:
    $h'$: Converted approximately admissible heuristic function
$h^a(x) \leftarrow 0, \forall x \in \mathcal{X}$
$is\_solved(x) \leftarrow False, \forall x \in \mathcal{X}$
while $\exists x \in \mathcal{X} \mid is\_solved(x) == False$ do
    $h' \leftarrow adjust(h^a, h, \mathcal{X})$
    for $x \in \mathcal{X}$ do
        $h^a(x), is\_solved(x) \leftarrow A^*(x, h', h^a(x) + \eta)$
$h' \leftarrow adjust(h^a, h, \mathcal{X})$
Return $h'$
```

two steps to obtain an approximately admissible heuristic function from an inadmissible one:

1. Adjust the inadmissible heuristic function so that it does not overestimate any of the approximately admissible heuristic values on the representative set.
2. Update the approximately admissible heuristic values of the representative set using A* search with the adjusted heuristic function.

This process is outlined in Algorithm 1. As we will show, we do not have to wait for A* search to terminate in order to update the approximately admissible heuristics on the representative sets. We instead aim for these heuristics to increase by a fixed value η.

Adjusting an Inadmissible Heuristic Assume we have an inadmissible heuristic h and an approximately admissible heuristic h^a where h^a is only defined over a set of states in a representative set $\mathcal{X}$. We can then compute an adjusted heuristic:

$$h'(x) = h(x) - o_{max} \tag{2}$$

where o_{max} is the maximum overestimation of the approximately admissible heuristic h^a:

$$o_{max} = \max_{x \in \mathcal{X}} (h(x) - h^a(x)) \tag{3}$$

Assuming all transition costs are positive, we know that all path costs are greater than or equal to zero; therefore, the upper bound of overestimation increases as $h(x)$ becomes larger. As a result, we can compute the maximum overestimation based on the value of $h(x)$. In practice, we subtract $h(x)$ by o^c_{max}, where c is the smallest value in a user-defined set of cutoffs $\mathcal{C}$, such that c is greater than or equal to $h(x)$. The adjusted now becomes:

$$h'(x) = h(x) - o^c_{max} \tag{4}$$

where

$$o^c_{max} = \max_{x \in \mathcal{X} | h(x) \leq c} (h(x) - h^a(x)) \tag{5}$$

In our experiments:

$$\mathcal{C} = \{0, k, 2k, ..., \max_{\forall x \in \mathcal{X}} h(x)\} \tag{6}$$

where k is a user-defined value. For all of our experiments we set k to 1.

In some cases, using an admissible heuristic, or approximately admissible heuristic, to find an optimal solution may to take too much time or use too much memory. This has led to research on bounded suboptimal heuristic search (Pohl 1970, 1973; Ghallab and Allard 1983; Pearl and Kim 1982). (Harris 1974) showed that one could bound the cost of a solution found to be no greater than the cost of an optimal path plus b provided that the heuristic function does not overestimate the cost of an optimal path by more than b. We can modify our approximately admissible heuristic to be bounded $100-\delta$ percent of the time by simply subtracting b from o^c_{max} for all c in $\mathcal{C}$:

$$o^c_{max} = \max(o^c_{max} - b, 0) \tag{7}$$

This modification is performed only on the final adjusted model obtained by approximately admissible conversion.

A* Search Update The approximately admissible heuristic h^a for the representative set is initially zero for all states. Ideally, we would like h^a to equal h^* for every state in the representative set. To approximate this, for each state in the representative set, we perform A* search using the adjusted heuristic h'. When A* search finds a solution for some state x using a heuristic function that does not overestimate by more than e, the cost of the path found is between $h^*(x)$ and $h^*(x) + e$ (Harris 1974; Valenzano et al. 2013). However, since h' is being adjusted based on h^a, h' will initially be zero for all states. Therefore, we cannot expect A* search to find a solution for all states in the representative set in a reasonable amount of time. Instead, for each state x, in $\mathcal{X}$, we set a termination criteria based on how much $h^a(x)$ has increased.

We set $h^a(x)$ to be the maximum cost of all nodes that have been expanded. In Theorem 1, we prove that h^a is bounded by $h^* + e$ as long as h' is bounded by $h^* + e$. To prove Theorem 1, we will need Lemma 1 and 2. The proof to Lemma 1 may be found in Hart, Nilsson, and Raphael (1968) as Lemma 1 and Lemma 2 is a modification of the Corollary to that Lemma.

While we cannot guarantee that $h' \leq h^* + e$ will hold for all states, Harris (1974) has shown that theoretical properties can still hold in the majority of cases as long as the risk of overestimation remains low. In the Results section, we show that the percent of inadmissible heuristics δ is empirically as low as 0.0019%.

Lemma 1. *For any node x that has not been expanded and for any optimal path P from x_s to x, there exists a node x' in OPEN on P with a path cost of $g(x')$ equal to the cost of a shortest path from x_s to x'.*

Lemma 2. *Assume positive transition costs, a heuristic function that does not overestimate the cost of an optimal path by more than e, and that A* search has not yet terminated. Then, for any optimal path P from x_s to a goal node, there exists a node x' on P in OPEN with $f(x') \leq h^*(x_s) + e$.*

Proof. By Lemma 1, there exists a node x' in OPEN on P. So, by definition, $f(x') \leq g(x') + h(x') \leq g(x') + h^*(x') + e$. By Lemma 2 $g(x')$ is the cost of a shortest path from x_s to x', therefore, $g(x') + h^*(x') = h^*(x_s)$. Therefore, $f(x') \leq h^*(x_s) + e$ □

Now, we prove that, as long as A* search has not terminated, every node expanded by A* search has a cost no greater than $h^*(x_s) + e$.

Theorem 1. *Assume positive transition costs, a heuristic function that does not overestimate the cost of an optimal path by more than e, and that A* search has not yet terminated. For start node x_s, $f(x) \leq h^*(x_s) + e$ for all expanded nodes x.*

Proof. Assume the contrary: there exists some expanded node x' such that $f(x') > h^*(x_s) + e$ and x' is not the goal. We know that when x' was popped from OPEN, $f(x') \leq f(x)$ for all x in OPEN. Let P be an optimal path from the start node to the goal node. From Lemma 2 we know that when x was obtained from OPEN, there existed a node y in OPEN on path P such that $f(y) \leq h^*(x_s) + e$. Therefore, $f(x') \leq f(y) \leq h^*(x_s) + e$. This leads to a contradiction since we assumed $f(x') > h^*(x_s) + e$. □

Batch A* Search with Shortest Path Guarantees

Graphics processing units (GPUs) are often used for DNNs due to the drastic speedup they provide through parallelism. However, classical A* search does not fully use the parallel computing capabilities of GPUs because classical A* search only expands one node at a time. To address this, a batched version of A* search was proposed (Agostinelli et al. 2019) where N nodes are removed from OPEN every iteration and where batch A* search terminates if one of those N nodes is a goal node. However, this also does not guarantee that an optimal solution is found because the path to the goal node is not guaranteed to be the shortest path.

We modify batch A* search to ensure that it will return a bounded suboptimal solution provided a bounded suboptimal heuristic. The modifications are as follows: if a goal node is encountered, we save this goal node and continue search and only terminate when the saved goal node has a cost less than or equal to the first node removed from OPEN or if there are no more nodes in OPEN. If we have already saved a goal node and encounter a new goal node, we replace the saved goal node with the new goal node if it has a lower cost than the saved goal node. We prove that this will guarantee that a bounded suboptimal path is found in Theorem 2.

Theorem 2. *Assume positive transition costs, a heuristic function that does not overestimate the cost of an optimal path by more than e, and that batch A* search has saved a goal node. There are two cases in which batch A* search terminates: OPEN is empty or when the saved goal node has a cost less than or equal to the first node removed from OPEN. Termination in either of these two cases guarantees that the cost of the path to the saved goal node is suboptimally bounded by e.*

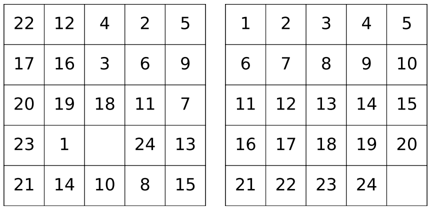
Proof.

Case 1. *OPEN is empty. If OPEN is empty, then there are no more nodes left to expand, meaning we have exhausted all possible paths from the start node. Since we update the saved goal node to be the lowest cost node of all goal nodes encountered, the path to the saved goal must be an optimal path.*

Case 2. *The saved goal node, x_g, has a cost less than or equal to the first node removed from OPEN, x'. That is, $f(x_g) \leq f(x')$. Since we know that x' has the lowest cost of all nodes in OPEN, we know that $f(x_g) \leq f(x)$ for all x in OPEN. If x_g is on a shortest path, then, by definition, its path cost is suboptimally bounded by e. If x_g is not on a shortest path, then, because we save the goal node with the lowest path cost, we must have yet to remove a goal node on a shortest path from OPEN. Therefore, by Lemma 1, there must be a node y in OPEN on a shortest path. Since our heuristic function overestimates the cost of an optimal path by no more than e, then y will have a cost of no more than $h^*(x_s) + e$. Therefore, $f(x_g) \leq f(y) \leq h^*(x_s) + e$. Therefore, the cost of the path to the saved node x_g is suboptimally bounded by e.*

□

Training



22	12	4	2	5
17	16	3	6	9
20	19	18	11	7
23	1		24	13
21	14	10	8	15

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	

Figure 1: Visualizations of a random state (left) and the goal state (right) for the 24-puzzle.

We train DeepCubeA2 on the 15-puzzle and the 24-puzzle (shown in Figure 1), two popular sliding tile puzzles. Moves are made by swapping the blank square with any square horizontally or vertically adjacent to it. The representation given to the network is a one-hot encoding that specifies the position of each square.

We generate training data by scrambling the goal state between 0 and 1,000 times. We attempt to solve each generated state using greedy best-first search with the cost-to-go function J as the heuristic. Each state encountered during greedy best-first search is also added to the training set. We train J using DAVI. J is initialized to be zero for all states and is used to compute an updated value J' using Equation 1. J' is learned by a DNN. J is updated to be J' every 1,000 iterations.

The DNN has two hidden layers followed by four residual blocks (He et al. 2016). The size of the first two hidden layers is 5,000 and 1,000, respectively. Each residual block has two layers of size 1,000. We train with a batch size of 10,000 and optimize the network using ADAM (Kingma and Ba 2015) with a constant learning rate of 0.001. No regularization is used. We train for 200,000 iterations for the 15-puzzle and 280,000 iterations for the 24-puzzle. The network is built using the PyTorch software package. (Paszke et al. 2017).

Results

To compare the results of DeepCubeA2 to the cost of an optimal path, we rely on shortest path solvers built for the 15-puzzle and 24-puzzle. Specifically, we use pattern databases (PDBs) (Culberson and Schaeffer 1998) developed by (Felner, Korf, and Hanan 2004). These PDBs use lookup tables to compute a heuristic and use that heuristic for iterative deepening A* search (IDA*) (Korf 1985). While we do include results obtained by pattern databases in Table 1 for transparency, we would like to note their purpose is mainly for verifying the cost of an optimal path and that a direct comparison cannot be made between DeepCubeA2 and PDBs as their applicability and scope vary significantly. Furthermore, the PDBs we use take advantage of domain specific knowledge to speed up lookup table access.

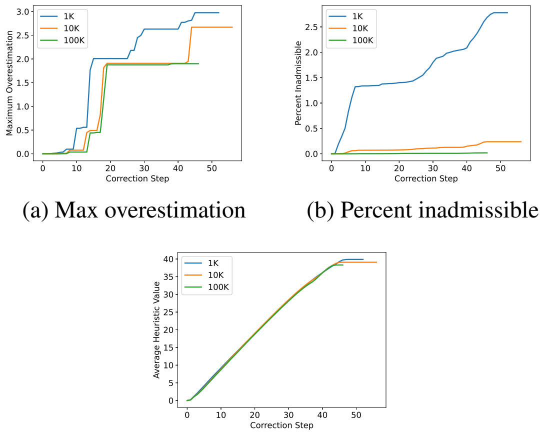
Performance of Approximately Admissible Conversion

We run approximately admissible conversion for the DNN trained on the 15-puzzle and 24-puzzle. We have one million examples in the representative set obtained by scrambling the goal state between 0 and 1,000 times. We set the lower bound increment η in Algorithm 1 to 1.0. To speed up performance, we did not update the lower bound of all examples for every iteration, instead, we used the difference between $h^a(x)$ and the output of $h'(x)$ for all unsolved states x to identify which states were "easiest" to update, where a smaller difference is considered easier. We updated at least 10% of unsolved states in each iteration. We ran A* search on these states simultaneously using four NVIDIA Titan V GPUs to compute h'. To ensure high GPU utilization, we terminated A* search for all states once we have fulfilled the termination criteria for 50% of the states. We stopped conversion when the average value of h' on the representative set stopped increasing. Approximately admissible conversion took 3.4 hours for the 15-puzzle and 15 hours for the 24-puzzle.

In order to determine the effectiveness of approximately admissible conversion, we need to determine how often the heuristic function overestimates the cost of an optimal path before and after conversion. PDBs are very effective at solving the 15-puzzle, however, PDBs are much more time consuming for the 24-puzzle, with some states taking weeks to solve. Therefore, we focus on the 15-puzzle. We create a test set of one million states for the 15-puzzle by randomly scrambling the goal state between 0 and 1,000 times.

We first investigate the effect that the size of the representative set has on the admissibility of the adjusted heuristic function. We create a representative set of size 10^3, 10^4, and 10^5. We then empirically evaluate the converted heuristic function on the test set. Figure 2 shows how the maximum overestimation, e, percent inadmissible, δ, and the average heuristic value changes as a function of each step of approximately admissible conversion. The figure shows that the both e and δ decrease significantly as the size of the representative set is increased, while the average heuristic value decreases only slightly.

We then evaluate the converted heuristic function when using the entire representative set of size 10^6. A comparison of the heuristic function before and after approximately admissible conversion can be seen in Figure 3. Before conversion, the heuristic function is inadmissible 71.37% of the time, with the largest overestimation being 8.28. After conversion, the heuristic function is inadmissible only 0.0019% of the time (19 out of one million examples), with the largest overestimation being 0.62. This is relatively small considering that the diameter of the 15-puzzle is 80 (Brüngger et al. 1999; Korf 2008). Empirically, we can see that approximately admissible conversion creates a heuristic function that is admissible $100 - \delta$ percent of the time where $\delta = 0.0019$.



(a) Max overestimation

(b) Percent inadmissible

(c) Average heuristic value

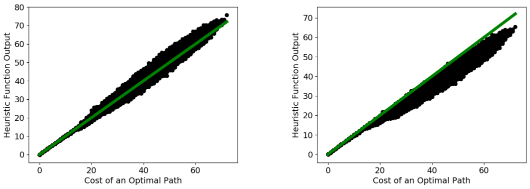
Figure 2: How various aspects of the heuristic function changes during approximately admissible conversion. Each line represents a different size for the representative set.

Solving with Batch A* Search

After converting the heuristic using approximately admissible conversion, we now test how well batch A* search performs when using the converted heuristic. We use a test set of 500 states from (Agostinelli et al. 2019) that have been generated by randomly scrambling the goal state between 1,000 and 10,000 times. We test batch A* search using batch sizes of 1, 10, 100, and 1,000. In addition, we vary the suboptimal bound between zero and the maximum overestima-

Puzzle	Solver	Len	Worst Subopt	% Opt	Nodes	Secs	Nodes/Sec
15-Puzzle	PDBs	52.02	0	100.0%	3.22E+04	0.002	1.45E+07
	DeepCubeA	52.03	2	99.4%	3.85E+06	10.28	3.93E+05
	DeepCubeA2	52.02	0	100.0%	5.76E+05	2.40	2.38E+05
24-Puzzle	PDBs	89.41	0	100.0%	8.19E+10	4,239.54	1.91E+07
	DeepCubeA	89.49	6	96.98%	6.44E+06	19.33	3.34E+05
	DeepCubeA2	89.41	0	100.0%	2.92E+07	145.03	2.21E+05

Table 1: Comparison of DeepCubeA and DeepCubeA2. DeepCubeA was not able to find optimal solutions for all test states, however, DeepCubeA2 is able to find optimal solutions for 100% of the test states.



(a) Before approximately admissible conversion

(b) After approximately admissible conversion

Figure 3: The plots show the output of the learned heuristic function compared to the cost of an optimal path on a test set of one million examples for the 15-puzzle. Any heuristic values above the green line are inadmissible. Before approximately admissible conversion, the heuristic is inadmissible 71.37% of the time. After approximately admissible conversion, the heuristic is inadmissible only 0.0019% of the time (19 out of one million).

tion encountered on the representative set during the final iteration of approximately admissible conversion. This maximum overestimation on the representative set was 9.02 for the 15-puzzle and 19.42 for the 24-puzzle.

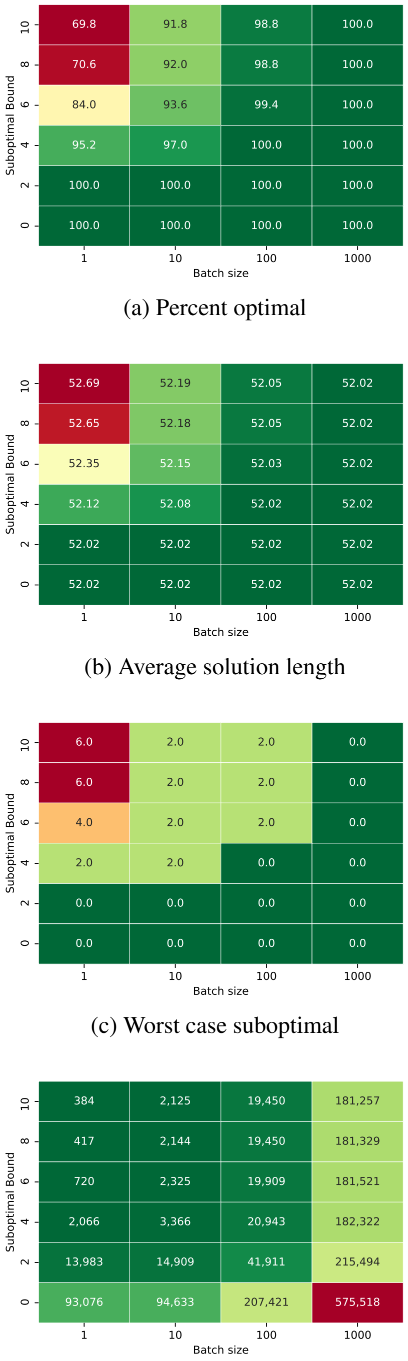
Figures 4 and 5 show the performance of batch A* search on the 15-puzzle and 24-puzzle, respectively. The plots show that batch A* search is able to solve 100% of all test states optimally under multiple conditions. A larger batch size leads to more optimal solutions and expands significantly more nodes. Furthermore, batch A* search remains consistent with the theoretical guarantees provided in Theorem 2, that is, all solutions are suboptimally bounded by b. When keeping the batch size the same, the most extreme difference in optimal solutions between a large b and a small b is over 30% for both the 15-puzzle and the 24-puzzle.

Finally, we compare the results from DeepCubeA2 to the results of DeepCubeA in Table 1. The table contains the 15-puzzle for the case of using a batch size of 1,000 and a suboptimal bound b of 0 and the 24-puzzle we show the case of using a batch size of 1,000 and suboptimal bound b of 2. The table shows that DeepCubeA2 consistently outperforms DeepCubeA by finding shorter solutions. While DeepCubeA did not find optimal paths to all solutions, we see that DeepCubeA2 is able to do so 100% of the time. In Figure 5, we see that even for parameter settings where DeepCubeA2 is not as optimal as DeepCubeA, DeepCubeA2 is still able to achieve a better suboptimality bound.

Discussion

We introduce DeepCubeA2, an algorithm that learns a heuristic function using deep reinforcement learning and then converts that heuristic function to an approximately admissible heuristic function. DeepCubeA2 accomplishes this using approximately admissible conversion, an algorithm that can take any inadmissible heuristic function and convert it to an approximately admissible heuristic function. Furthermore, the resulting approximately admissible heuristic function can easily be relaxed to be a bounded inadmissible heuristic function for bounded suboptimal solutions and potentially less resource usage. The heuristic produced by DeepCubeA2 has been empirically shown to be almost always admissible with a relatively small overestimation in the cases when it is inadmissible. Furthermore, we have shown that one can do batch A* search to take advantage of parallelism provided by GPUs while maintaining theoretical guarantees. Finally, we test this method on the 15-puzzle and 24-puzzle and show that the theoretical guarantees hold in practice and that DeepCubeA2 can solve 100% of test states optimally. It is possible that improvement can be made to the approximately admissible heuristic using lookahead search when computing the heuristic value (Lam et al. 2017). This could be particularly useful in cases where the learned heuristic function is very inaccurate.

Many current real-world problems are posed as path finding tasks. For example, robotic manipulation tasks such as key insertion (Florensa et al. 2017), object manipulation (Finn and Levine 2017; Lanier, McAleer, and Baldi 2019), and object assembly (Gullapalli, Franklin, and Benbrahim 1994; Levine et al. 2016) are often posed as pathfinding tasks. Furthermore, DeepCubeA has been used to solve problems in quantum computing (Zhang et al. 2020) and cryptography (Jin and Kim 2020). Solving real-world problems using an admissible heuristic function would ensure that tasks are completed in the most cost-effective way possible, which could significantly decrease the resources consumed by artificial intelligence agents. In this paper, we addressed the issue of learning an admissible heuristic using deep neural networks (DNNs) in deterministic fully-observable environments in which the goal state is known and a model is provided. Research on model learning (Racanière et al. 2017; Kaiser et al. 2019) suggests that this work can be extended to settings where the model must be approximated using deep learning. This work could also be extended to modified versions of A* search, such as anytime A* search (Likhachev, Gordon, and Thrun 2003), multi-heuristic A* search (Aine et al. 2016), and AQ* search (Agostinelli et al. 2021).



Suboptimal Bound	1	10	100	1000
10	69.8	91.8	98.8	100.0
8	70.6	92.0	98.8	100.0
6	84.0	93.6	99.4	100.0
4	95.2	97.0	100.0	100.0
2	100.0	100.0	100.0	100.0
0	100.0	100.0	100.0	100.0

(a) Percent optimal

Suboptimal Bound	1	10	100	1000
10	52.69	52.19	52.05	52.02
8	52.65	52.18	52.05	52.02
6	52.35	52.15	52.03	52.02
4	52.12	52.08	52.02	52.02
2	52.02	52.02	52.02	52.02
0	52.02	52.02	52.02	52.02

(b) Average solution length

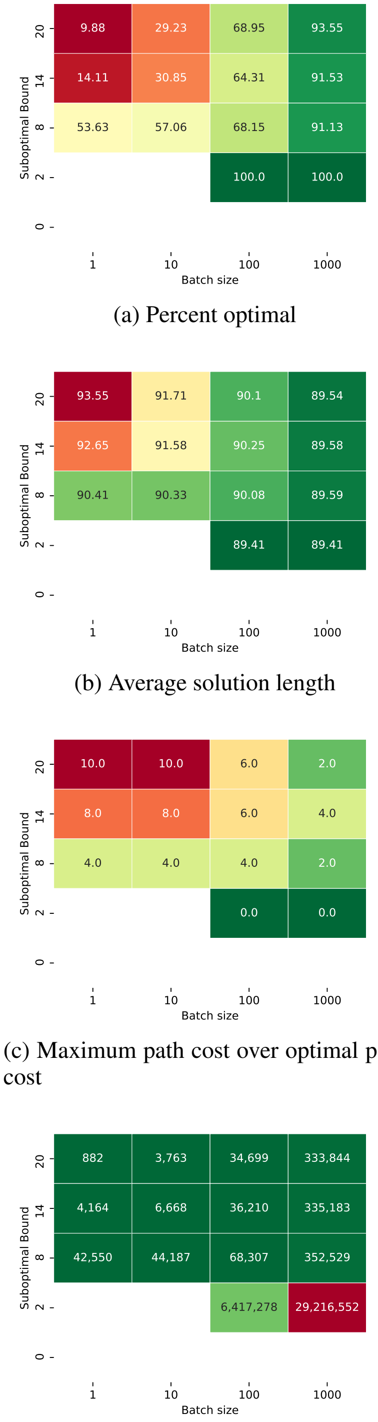
Suboptimal Bound	1	10	100	1000
10	6.0	2.0	2.0	0.0
8	6.0	2.0	2.0	0.0
6	4.0	2.0	2.0	0.0
4	2.0	2.0	0.0	0.0
2	0.0	0.0	0.0	0.0
0	0.0	0.0	0.0	0.0

(c) Worst case suboptimal

Suboptimal Bound	1	10	100	1000
10	384	2,125	19,450	181,257
8	417	2,144	19,450	181,329
6	720	2,325	19,909	181,521
4	2,066	3,366	20,943	182,322
2	13,983	14,909	41,911	215,494
0	93,076	94,633	207,421	575,518

(d) Number of nodes generated

Figure 4: Effect of batch size and suboptimal bound parameter b on solving performance for the 15-puzzle when using batch A* search. The worst case suboptimal paths found by batch A* search are all bounded by the suboptimal bound b. There are multiple parameter settings that result in batch A* search always finding a shortest path.



(a) Percent optimal

(b) Average solution length

(c) Maximum path cost over optimal path cost

(d) Number of nodes generated

Figure 5: Effect of batch size and suboptimality bound b on solving performance for the 24-puzzle when using batch A* search. The worst case suboptimal paths found by batch A* search are all bounded by the suboptimal bound b. There are multiple parameter settings that result in batch A* search always finding a shortest path. The blank squares result from the parameter settings using too much time or too much memory.

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence* 1(8): 356–363.

Agostinelli, F.; Shmakov, A.; McAleer, S.; Fox, R.; and Baldi, P. 2021. A* Search Without Expansions: Learning Heuristic Functions with Deep Q-Networks. *arXiv preprint arXiv:2102.04518* .

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2016. Multi-heuristic a. *The International Journal of Robotics Research* 35(1-3): 224–243.

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17): 2075–2098.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1996. *Neuro-dynamic programming*. Athena Scientific. ISBN 1-886529-10-8.

Brüngger, A.; Marzetta, A.; Fukuda, K.; and Nievergelt, J. 1999. The parallel search bench ZRAM and its applications. *Annals of Operations Research* 90: 45–63.

Chen, H.-C.; and Wei, J.-D. 2011. Using neural networks for evaluation in heuristic search algorithm. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3): 318–334.

Ernandes, M.; and Gori, M. 2004. Likely-admissible and sub-symbolic heuristics. In *Proceedings of the 16th European Conference on Artificial Intelligence*, 613–617. Citeseer.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research* 22: 279–318.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Reinforcement Learning for Planning Heuristics. In *Proceedings of the 1st Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*, 119–126. University_of_Basel.

Finn, C.; and Levine, S. 2017. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2786–2793. IEEE.

Florensa, C.; Held, D.; Wulfmeier, M.; Zhang, M.; and Abbeel, P. 2017. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300* .

Ghallab, M.; and Allard, D. G. 1983. A*: An efficient near admissible heuristic search algorithm. In *Proceedings IJCAI-83*. Citeseer.

Gullapalli, V.; Franklin, J. A.; and Benbrahim, H. 1994. Acquiring robot skills via reinforcement learning. *IEEE Control Systems Magazine* 14(1): 13–24.

Harris, L. R. 1974. The heuristic search under conditions of error. *Artificial Intelligence* 5(3): 217–234.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2): 100–107.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Jin, J.; and Kim, K. 2020. 3D CUBE Algorithm for the Key Generation Method: Applying Deep Neural Network Learning-Based. *IEEE Access* 8: 33689–33702.

Kaiser, L.; Babaeizadeh, M.; Milos, P.; Osinski, B.; Campbell, R. H.; Czechowski, K.; Erhan, D.; Finn, C.; Kozakowski, P.; Levine, S.; et al. 2019. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374* .

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)* .

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence* 27(1): 97–109.

Korf, R. E. 1997. Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, AAAI'97/IAAI'97, 700–705. AAAI Press. ISBN 0-262-51095-2. URL http://dl.acm.org/citation.cfm?id=1867406.1867515.

Korf, R. E. 2008. Linear-time disk-based implicit graph search. *Journal of the ACM (JACM)* 55(6): 26.

Korf, R. E.; and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial intelligence* 134(1-2): 9–22.

Lam, W.; Kask, K.; Larrosa, J.; and Dechter, R. 2017. Residual-guided look-ahead in AND/OR search for graphical models. *Journal of Artificial Intelligence Research* 60: 287–346.

Lanier, J. B.; McAleer, S.; and Baldi, P. 2019. Curiosity-Driven Multi-Criteria Hindsight Experience Replay. *arXiv preprint arXiv:1906.03710* .

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17(1): 1334–1373.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in neural information processing systems* 16: 767–774.

McAleer, S.; Agostinelli, F.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's Cube With Approximate Policy Iteration. *International Conference on Learning Representations (ICLR)* .

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Pearl, J. 1984. Heuristics: intelligent search strategies for computer problem solving .

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4(4): 392–399.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial intelligence* 1(3-4): 193–204.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, 12–17. Morgan Kaufmann Publishers Inc.

Puterman, M. L.; and Shin, M. C. 1978. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science* 24(11): 1127–1137.

Racanière, S.; Weber, T.; Reichert, D.; Buesing, L.; Guez, A.; Rezende, D. J.; Badia, A. P.; Vinyals, O.; Heess, N.; Li, Y.; et al. 2017. Imagination-augmented agents for deep reinforcement learning. In *Advances in neural information processing systems*, 5690–5701.

Samadi, M.; Felner, A.; and Schaeffer, J. 2008. Learning from Multiple Heuristics. In *AAAI*, 357–362.

Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural networks* 61: 85–117.

Sutton, R. S.; and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Valenzano, R. A.; Arfaee, S. J.; Thayer, J.; Stern, R.; and Sturtevant, N. R. 2013. Using alternative suboptimality bounds in heuristic search. In *Twenty-Third International Conference on Automated Planning and Scheduling*.

Wang, J.; Wu, N.; Zhao, W. X.; Peng, F.; and Lin, X. 2019. Empowering A* Search Algorithms with Neural Networks for Personalized Route Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 539–547. ACM.

Zhang, Y.-H.; Zheng, P.-L.; Zhang, Y.; and Deng, D.-L. 2020. Topological Quantum Compiling with Reinforcement Learning. *Physical Review Letters* 125(17): 170501.