Learning Admissible Heuristics with Neural Networks

by

Tianhua Li

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Machine learning has been used to solve single-agent search problems. One of its applications is to guide search algorithms by learning heuristics. However, it is difficult to provide guarantees on the quality of learning from a neural network, since the resulting heuristics can be inadmissible, leading paths to be suboptimal when the heuristic is used as part of a search algorithm. This thesis introduces empirical methods to guarantee the admissibility (non-overestimation) of the heuristics learned by neural networks and discusses their application as a compression technique. As supervised learning alone is hard to guarantee admissibility, admissibility is achieved through the combination of three ideas: learning a classifier, adjusting the classifier quantile, and taking the minimum value of an ensemble of neural networks. The quantile and ensemble methods are able to learn admissible heuristics either on their own or together. These methods are able to compress a pattern database (PDB) heuristic by several orders of magnitude with a lower loss than DIV compression. Although the average heuristic value from neural networks can be no greater than that of the original PDB, it is significantly higher than a comparable size PDB compressed from the original PDB by DIV compression.

Acknowledgements

I give my sincere gratitude towards my supervisor Professor Nathan R. Sturtevant for his years of support. I very much appreciate that he gave me the opportunity to study computer science and explore the new frontiers in AI. He cares about not only my studies, but also my life. He offers the best he can to help me achieve my goals at school and in my career. To me, he is not only my supervisor, but also a friend and mentor.

I am also grateful to Amii for providing years of funding for our research. I thank Professor Ariel Felner and his students for giving feedback to our research. I thank Karim Ali for maitaining Bluesky Server and solving all my connection issues. Finally, I would like to thank all my colleagues at Moving AI Lab for being supportive to one another.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As a traditional research area in artificial intelligence, search is a process of finding solutions to unsolved problems. A search algorithm finds a path from a start state to a goal state. In the modern society we are living in, we are using search algorithms everyday without even noticing. Imagine that you are driving from Edmonton to Regina. Edmonton is your start state and Regina is your goal state. You would probably plan a path before you depart by running a search algorithm on Google Maps. Once the path is planned, you are ready to go. Obviously, efficient search algorithms, which immediately inform you and instantly plan a new route for you when you accidentally make a wrong turn, make our lives more convenient. Search is a research field where we solve path-finding problems, and, more often than not, we are interested in searching for the shortest or optimal path in a timely manner.

Some problems are too difficult to solve efficiently as the search spaces are exponentially large. To overcome this, researchers have developed several enhancements, one of which is to use a heuristic. Heuristic search is the method of solving pathfinding problems with the help of heuristic functions. Single-agent search is a sub-area of heuristic search. Single-agent search algorithms such as A* [12] and IDA* [17] are guided by heuristic functions to find the optimal path from a start state to a goal state. A heuristic function points the search algorithm in the right direction by estimating the distance from a state to a goal state. Figure 1.1 shows four cities and the distances between any two adjacent cities. Suppose we are in Edmonton and would like to go to

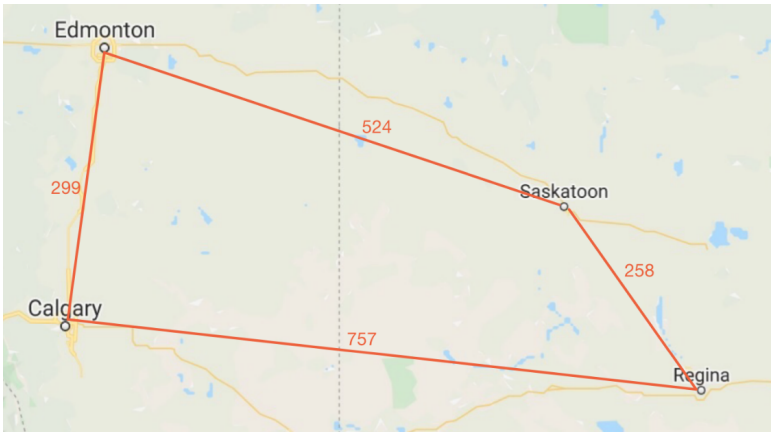


Figure 1.1: Search example

Regina. The shortest path is to go to Saskatoon first, and then go to Regina from Saskatoon. A heuristic function is a mapping from a given state to a number, a distance estimate from a given state to the goal state. A valid heuristic function in this example is

$$h(s) = \begin{cases} 730 & \text{if } s = Edmonton \\ 750 & \text{if } s = Calgary \\ 230 & \text{if } s = Saskatoon \\ 0 & \text{if } s = Regina \end{cases}$$

In addition to the shortest path from Edmonton to Saskatoon to Regina in this example, another path is Edmonton-Calgary-Regina. An important question in search is how to find optimal solutions, i.e., find the shortest path and avoid all other suboptimal paths. An admissible heuristic can guarantee an optimal solution will be found. A heuristic function is said to be admissible if it never overestimate the cost from a given state to a goal state. The heuristic function above is admissible while the below is not

$$h(s) = \begin{cases} 10000 & \text{if } s = Edmonton \\ 750 & \text{if } s = Calgary \\ 3000 & \text{if } s = Saskatoon \\ 0 & \text{if } s = Regina \end{cases}$$

With this heuristic function, a search algorithm using the heuristic as a guidance will return the suboptimal path Edmonton-Calgary-Saskatoon as the

0 (Edmonton-Regina)	782
1 (Saskatoon-Regina)	258
2 (Calgary-Regina)	757
3 (Regina-Regina)	0

Table 1.1: PDB example

estimated cost is too high from Saskatoon to Regina. After a search Edmonton is expanded, the search algorithm is faced with choosing between Saskatoon and Calgary to expand. Since the estimated cost, or heuristic, from Calgary to Regina is much lower than that from Saskatoon to Regina, Calgary seems more promising to the algorithm. As a result, the search algorithm expands Calgary, finds the destination Regina, and returns a suboptimal solution Edmonton-Calgary-Saskatoon.

A perfect heuristic function gives the shortest distance from a given state to a goal state. The perfect heuristic function for this example is

$$h(s) = \begin{cases} 782 & \text{if } s = Edmonton \\ 757 & \text{if } s = Calgary \\ 258 & \text{if } s = Saskatoon \\ 0 & \text{if } s = Regina \end{cases}$$

With a perfect heuristic function on hand, we can find shortest path by moving to states with the lowest costs because the lowest cost corresponds to the shortest distance as the cost estimation is accurate. It is highly efficient as a search algorithm never waste efforts expanding a state not in the optimal solution.

There are various kinds of heuristics. Some heuristics, such as Manhattan Distance, can be computed at runtime, while some are precomputed and loaded into memory when the algorithms are running. An example of precomputed table-based heuristics are pattern databases (PDBs) [5]. In the simplest form, a PDB is a table of numbers. The indices of the table entries corresponds to states in a problem of interest, and the numbers are cost or distance estimates from the current state to the goal state. Table 1.1 is a PDB for the search example.

One problem of using table-based heuristics is that they can be too large to

fit into memory, and thus cannot be effectively used at runtime. The standard way to overcome this difficulty is to compress heuristics to fit into memory. A variety of techniques have been suggested. For instance, DIV and MOD [9] approaches are used to compress PDBs. These compressing techniques strive to maintain admissibility. That is, they try not to overestimate the heuristics as overestimation would lead to suboptimal solutions. Recently, neural networks have been used to learn from PDBs and represented heuristics. Although neural networks as compression methods have shown promising results [23], they either are not be able to maintain admissibility or require additional techniques including using hash tables to maintain admissibility.

This thesis introduces two compression methods to learn admissible heuristics with neural networks for use in finding optimal solutions to single-agent heuristic search problems. The two methods can be applied independently or together to preserve admissibility. The first method is to use a classifier instead of regression to learn a mapping from states to heuristics. With a classifier, we can always find at least one quantile level for which admissible heuristics can be produced. The second method is to use an ensemble of neural networks. By taking the minimum value from multiple (usually two) neural networks, we can guarantee the property that the resulting heuristics are admissible. Our methods have higher average heuristic values than existing compression approaches.

According to Korf's finding [18] [4], the larger the size of a PDB, the more efficient it is for IDA* to solve a problem. With the methods in this thesis, we are able to generate fewer nodes and improve the efficiency of search algorithms while keeping the sizes of PDBs unchanged.

Overall, we make several contributions in this thesis. First, we present a novel quantile tuning and ensemble methods to guarantee admissible heuristic. Second, we treat the learning problem as a classification rather than a regression problem. This significantly improves heuristic values. As we will show later, the heuristic learned with a regression method is very weak by both quantile tuning and ensemble approaches. Finally, we introduce a novel training method, with which PDBs can be further compressed and less mem-

ory is required. We first extract "hard data", which is a fraction of the original PDB, and then train neural networks specifically on those data.

Chapter 2

Background & Related Work

A *heuristic search* problem is defined by $\{\mathcal{G} = \{V, E\}, s, g, c, h\}$, where $\mathcal{G}$ is a graph with vertices V and edges E, s and g are vertices that corresponds to a start state and a goal state, c is a cost function which gives the cost of an edge, and h is a heuristic function which estimates the cost from a given state to the goal state. The problem is to find a shortest path from s to g. Let $h^*(v)$ be the shortest distance from $v \in V$ to g, then h is admissible if $\forall v \in V$, $h(v) \leq h^*(v)$ and consistent if $\forall a, b \in V$, $h(a) \leq c(a, b) + h(b)$. For problems such as the sliding tile puzzle and Rubik's cube, whose state spaces are too large to fit in memory, the state spaces are given implicitly, i.e., a successor operator is used to transition a current state to the next state instead of storing $\mathcal{G}$. A* [12] and IDA* [17] are both admissible algorithms that will find optimal paths if the heuristic is admissible. A* might re-expand some nodes when the heuristic is not admissible, but will not re-expand any nodes when the heuristic is consistent .

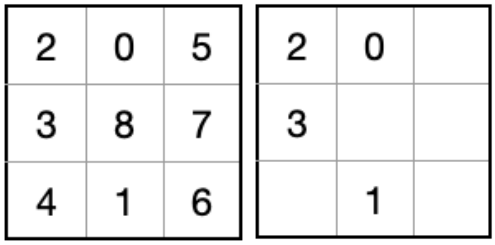


Figure 2.1: 3x3 STP Example

2.1 Sample Domains

2.1.1 The Sliding-Tile Puzzle

The Sliding-Tile puzzle (STP) is a classic problem domain in AI research. The common versions of STP are 4x4 and 5x5 STP. The STP consists of a square frame containing a set of numbered tiles. The legal actions are to swap the 0 numbered tile horizontally or vertically with an adjacent tile within the frame. The goal is to put all those numbered tiles in order.

2.1.2 The TopSpin Puzzle

The (n, r) Topspin puzzle has n tokens arranged in a ring. The start state is a state where n tokens are arranged randomly. A valid action is to reverse the order of any r consecutive tokens. The goal state is a state where all tokens are in ascending order.

2.2 Pattern Databases

Some heuristics are computed at runtime, while some are stored using additional memory. An example of the heuristic computed at runtime is Manhanttan distance. In the STP, the Manhanttan distance is the sum of the least number of actions for each tile in a given state to move to the correct position in the goal state. An example of a heuristic that requires memory is a Pattern Database [5], a table-based heuristic commonly used in exponential domains. A PDB is a lookup table that stores a cost estimate from a state to a goal state. A pattern is a subproblem of the original state representation. During search, a state is abstracted, and then a ranking function [21] is used to convert the abstract state into a unique integer, which indexes the state in a lookup table storing the distance from the abstract state to the abstract goal. In 3x3 sliding tile puzzle, a state is a sequence of numbers. For instance, state (2 0 5 3 8 7 4 1 6) is the left configuration in Figure 2.1. Its corresponding 4-tile pattern that abstracts away tiles 4-8 is (2 0 - 3 - - - 1 -) as can be seen in the right configuration in Figure 2.1.

Rank	0	1	2	3	...	998	999
Heuristics	12	8	10	150	...	41	55

Table 2.1: Example PDB

When the original problem is solved, the abstract problem must have been solved as well. As the number of moves to solve the abstract problem is no greater than that to solve the original problem, we can use the number of moves required to solve the abstract problem as a heuristic function that provides us with the lower bound to solve the original problem. An example of a PDB is shown in Table 2.1. The first row is the rank of a state and the second row is the corresponding heuristic. A rank is a single integer number that represents a state. This is achieved by applying a ranking function to a state.

Numerous PDB enhancements have been developed. Two notable enhancements are additive PDBs [8], which have smaller values in each PDB, but multiple PDBs can be added together while still guaranteeing an admissible heuristic. Another common approach with PDBs is to only store the delta between the computed PDB value and an inexpensive base heuristic. At runtime the original value can be restored by adding the stored delta to the base heuristic [8], [24]. This reduces the total number of unique values in the PDB, which is important in our approach below.

2.3 PDB Compression

One problem of using table-based heuristics such as PDBs is that PDBs can be too large to fit into memory [7] [15], even though the state spaces abstracted by PDBs are much smaller. The problem can be overcome by compressing PDBs. Compression has been widely studied for memory-based heuristics [9] [1] [2] [11] and PDB compression approaches have been developed. DIV compression [9] merges k adjacent entries in a PDB by taking the minimum of those k entries. Similarly, MOD compression [9] merges every $\frac{n}{k}$ entry in a PDB by taking the minimum of those k entries, where n is the total number of entries. DIV and MOD compression [9] are able to compress a PDB to be k times smaller. The

Rank	0	1	...	449
Heuristics	8	10	...	41

Table 2.2: DIV Compression

effectiveness of these approaches depends on action dependencies in the state space [13] and help determine the effectiveness of compressing large PDBs. An example of DIV compression when $k = 2$ is shown in Table 2.2. The heuristic of the state with rank 3 is 150 in the original PDB. After it is compressed with DIV by a factor of 2, the state with rank 3 is merged with the state with rank 2 by taking the minimum of their heuristics. When referring to the compressed PDB, we look for the heuristic for rank 1 (3 divided by 2), which is 10. The effectiveness of these approaches depends on action dependencies in the state space; compressing a significantly larger PDB may be no more effective than building a smaller PDB directly since the loss can be huge, as can be seen from the example (the heuristic decreases from 150 to 10). Moreover, the heuristics in a compressed PDB are no greater than those in the original PDB.

Relatively little work has been done on using artificial neural networks (ANNs) to learn admissible heuristics, with one exception of work by as the primary past approach. This approach is called ADP [23] for ANNs, decision trees, and partitioning. This approach requires significant engineering efforts to produce admissible heuristics. This includes a specialized loss function to penalize overestimates, using a decision tree and smaller PDBs to classify subsets of states, followed by using an ANN at the leaves of the decision tree to train on a small subsets of problems. Finally, any states that still return inadmissible heuristics are placed in a hash table. Overall, ADP is complex, requiring multiple different approaches to achieve admissibility. The work also predates the current advances in ANNs. Thus, there is significant room for improvement.

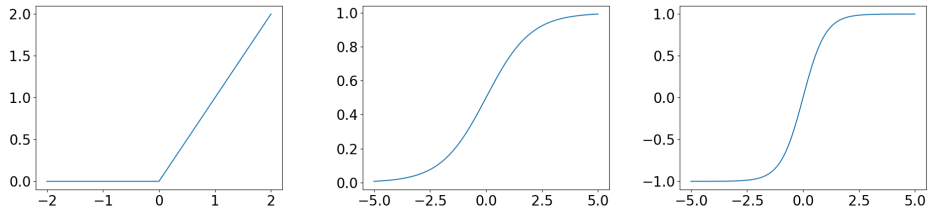
Activation Function	$f(x)$
ReLU	$max(0, x)$
Logistic	$\frac{1}{1+e^{-x}}$
TanH	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Table 2.3: Activation Function

2.4 Neural Networks

ANNs are inspired by biological neurons. In the simplest case a fully-connected neuron is a composition of a linear function followed by a nonlinearity e.g. tanh or ReLU [22], logistic, or softmax. The ReLU activation function is a piecewise function that will output the input value directly when the input is positive, while 0 when the input is non-positive. The logistic (or sigmoid) activation function and tanh are similar. They are both S-shaped real function. However, the logistic function is bounded between 0 and 1, whereas tanh ranges from -1 to 1. The logistic activation function is typically used for binary classification problems. The function expressions of the ReLU, the logistic, and TanH are shown in Table 2.3 and their graphs are shown in Figure 2.2. The softmax activation function, transforming a vector of numbers to a vector of probabilities, is typically used for multi-classifiction problems. In a vector v with length j, the probability of the ith class is $P(v_i) = \frac{e^{v_i}}{\sum_1^j e^{v_j}}$.

Typically, deep ANNs have multiple hidden layers which increases representation power. ANNs typically use fully connected layers, which are general



(a) ReLU (b) Logistic (Sigmoid) (c) TanH

Figure 2.2: Activation Function Graphs

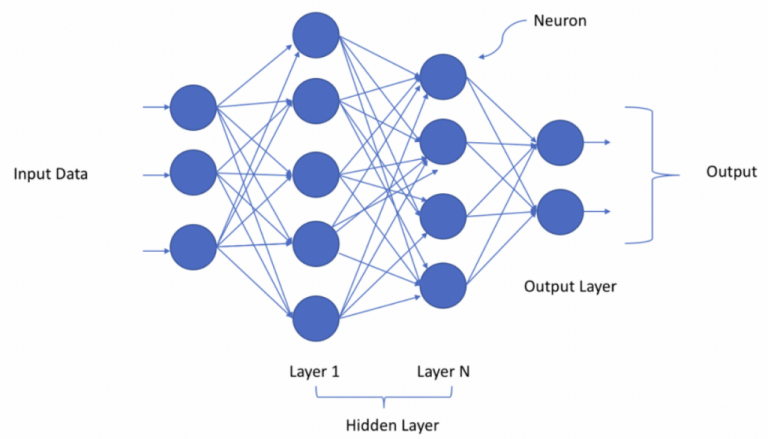


Figure 2.3: ANN Example

and do not assume any specific structure of data. Figure 2.3[1] shows a 3-layer ANN. The input layer has three neurons, the first layer and second layer has five and four neurons, respectively. The output layer has two neurons. By contrast, convolutional layers use a specialized type of architecture that tries to exploit the structure of 2D data such as images or game boards. Convolutional layers consist of multiple 2D filters called kernels. Each kernel learns a translation invariant feature by applying the kernel simultaneously to every $k \times k$ subset of the original image. In general, Convolutional Neural Networks (CNNs) consist of multiple convolutional layers followed by fully connected layers [19]. In each layer, an activation function is applied. Activation functions such as ReLU and logistic are widely used in many settings. ANNs have strong representational power. Theoretically, ANNs can approximate any Lebesgue integrable function defined on a compact set [20]. Furthermore, Chollet [3] draws a parallel connection between ANN and locality-sensitive hash tables. However, increasing the approximating power of an ANN requires exponential growth of the number of neurons [6]. In practice, the size of ANN is bounded by GPU memory. A more practical approach to improve the approximating power of an ANN is to combine predictions of multiple ANNs, i.e. ensembling. In particular, boosting [10] is a well-known approach that creates ensembles by

[1]https://trailhead.salesforce.com/en/content/learn/modules/deep-learning-and-natural-language-processing/understand-dl

training each new model on the data points that were misclassified by previous ANNs.

Chapter 3

Compressing PDBs with Neural Networks

While the larger context of the work in this thesis is to solve heuristic search problems efficiently, the purpose of this thesis is to build admissible heuristic with ANNs. There are three tasks in this problem. The main problem is to build a h such that $\forall v,\ h(v) \leq h^*(v)$. We do this by training ANNs on PDBs and ensuring the inequality holds for the learned ANNs with respect to the PDBs. Since PDBs are admissible, h is admissible. We treat the problem as a supervised learning problem where the input and output are patterns and corresponding heuristics of PDBs. Unlike typical supervised learning problems, overfitting is not an issue in the problem, because the goal is not to generalize well beyond training data, but to learn heuristics as accurate as possible from PDBs [16]. The admissible h is one or more ANNs trained on PDBs.

The second task of this problem is to make the sizes of the ANNs that produce admissible heuristic smaller than those of the original PDBs. Therefore, the ANN approach works as a compression technique. This is useful when table-based PDBs are too large to fit into memory and thus extremely inefficient for search algorithms to use. We compare the ANN compression approach with DIV compression and report in the experiment chapter the effectiveness of these two approaches measured by the average heuristic value. The average heuristic value might not be a good measurement for the quality of heuristic in some cases such as when comparing very different heuristics

or inconsistent heuristics [14]. However, in our problem setting, we use the same PDBs consistently throughout our experiments. Therefore, the average heuristic value should be indicative of the quality of heuristic.

The third task is to use ANN as a guide to search and find optimal solutions efficiently. In this thesis, we address the first two tasks. Our colleagues are working on the third task. They have developed a batch A* algorithm which can use an ANN more efficiently during search.

In this chapter we focus on our approaches which maintain admissibility. We will give full architecture details in the experiment results section.

3.1 Admissible Regression Heuristics

When using regression to learn a PDB heuristic, the input is a multi-channel binary image, while the output is a real number. The range of the output depends on the activation function of the last layer in a neural network. For instance, the range of the output is $[0, \infty)$ when using RELU activation function, while $(0, 1)$ when using the logistic activation function. In our experiments, we use the RELU activation function. When using regression, some portion of the learned heuristic is likely to be inadmissible, especially with a mean squared error loss function. We tackle this by applying an affine transform to build a new heuristic to guarantee admissibility. Let the PDB heuristic be h, heuristic learned through regression $h_{\hat{\mathrm{R}}}$, and the heuristic after applying affine transformation h_{R}. Note that the heuristic of the goal state must be 0. This can be ensured by subtracting $h_{\hat{\mathrm{R}}}(g)$ from the heuristic of each state. Next, to maintain admissibility for a single state, first, we compute the value $\alpha_v = \arg\max_\omega (h_{\hat{\mathrm{R}}}(v) - h_{\hat{\mathrm{R}}}(g))\omega \leq h(v)$ for each state v. Second, for all $v \in V$ we choose $\alpha = \min_{v \in V} \alpha_v$ and let $h_{\mathrm{R}}(v) = (h_{\hat{\mathrm{R}}}(v) - h_{\hat{\mathrm{R}}}(g))\alpha$. h_{R} is admissible since $h_{\mathrm{R}}(v) \leq h(v)$ for any $v \in V$.

Although admissibility can be guaranteed by applying this affine transform, we found that h_{R} is still a very weak heuristic. We obtained a much stronger heuristic by treating the task as a classification problem instead of a regression problem. When using classification, the input is a multi-channel binary image,

while the output is a vector of probabilities on classes that corresponds to heuristic values.

3.2 Admissible Classification Heuristics

We now introduce quantile tuning and ensemble of neural networks to preserve admissibility of classifier based heuristics.

3.2.1 Using Classifier Quantiles

When learning a classifier, the ANN produces one output for each class that is learned. After being passed through a soft-max function, this gives a probability distribution on each class. Typically, the final classification returned is the class with the largest probability. This is illustrated in Figure 3.1, which shows hypothetical classifier output when classifying a single state. After applying softmax activation function in the last layer, a classifier output a vector of probabilities. The class with the highest probability is class 3, which would normally be returned as the classification value for this input. However, when the actual heuristic is less than 3, the classifier overestimates the heuristic and causes inadmissibility. This problem can be overcome by tuning a quantile. As the cumulative probability distribution across heuristic values, shown in Figure 3.2, is non-decreasing for higher class, the quantile can make the classifier output any class by letting the classifier returns the highest class whose corresponding probability is no greater than the quantile. For instance, suppose the quantile is 0.5; then the classifier will return class 2. Thus given a classifier, we can tune the prediction to be more or less aggressive by tuning the quantile used for the final prediction value. Using a smaller quantile will return a smaller heuristic, at the cost of loss of heuristic accuracy, as some states that originally had correct heuristic values will then produce underestimates. Importantly, using a quantile of 0 will return the zero heuristic, which is always admissible.

More precisely, assume that a classifier $h_{\hat{C}}$ has been learned through training on the input heuristic h that has maximum value h_{max}. Let $P_{h_{\hat{C}}}(v, i)$ be

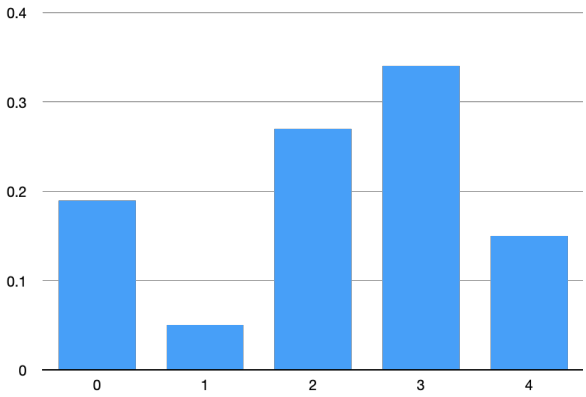


Figure 3.1: Hypothetical output data from a classifier

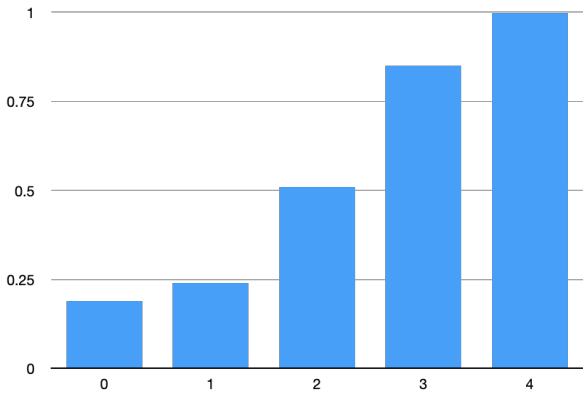


Figure 3.2: Cumulative distribution of hypothetical output data from a classifier

the classifier probability of the ith class on state v. Then, let $C_{h_{\hat{C}}}(v, q)$ return the class (heuristic) that cumulatively, from small to large classes, has at least probability q. That is $C_{h_{\hat{C}}}(v, q) = \arg\min_i(\sum_{j=0}^{i} P_{h_{\hat{C}}}(v, j)) \geq q$.

We can use $C_{h_{\hat{C}}}(v, q)$ to build a new heuristic $h_{\text{QNT}}(v)$ which is guaranteed to be admissible. This is done in two steps. First, we compute the maximum quantile q_v which guarantees an admissible heuristic for state v. For each $v \in V$ let $q_v = \arg\max_{q \in 0...1} C_{h_{\hat{C}}}(v, q) \leq h(v)$. Then, we let q^* be the quantile that guarantees an admissible heuristic for all states; $q^* = \min_{v \in V} q_v$. Putting this together, $h_{\text{QNT}}(v) = C_{h_{\hat{C}}}(v, q^*)$. This is the largest heuristic that can be returned, using this approach, that is guaranteed to be admissible on every state.

One might fear that q^* would be too small to be useful in practice, but

heurisitics	0	1	2	3	4	5	6	7	8	9
P_{left}	0	0	0	0	5.56e-43	1.09e-32	3.98e-24	2.07e-18	1.03e-04	0.99
P_{right}	3.86e-33	1.63e-18	0.99	1.30e-03	5.50e-11	2.94e-21	3.24e-42	0	0	0

Table 3.1: Probability Vectors

even small values of q^* can be effective. Consider an actual example using the sliding-tile puzzle (STP) states in Figure 3.3. The PDB values of the left and right states are 9 and 1 respectively after subtracting Manhattan Distance (MD). The ANN probability output for each class is given in Table 3.1. Using $q = 0.5$, $h(P_{left}) = 9$ because $C(P_{left}, 8) \approx 0.01 < 0.5$ and $C(P_{left}, 9) = 0.99 \geq 0.5$. $h(P_{right}) = 2$ for $q = 0.5$. But, using $q^* = 1.63\text{e}{-18}$ results in $h(P_{right} = 1)$ which is admissible. $v(P_{left}, 1.63\text{e}{-18})$ is then 7 instead of 9, causing a small loss, but the heuristic is admissible. Although the quantile margins are small, the computations are deterministic, so there is no danger of losing admissibility.

Theorem 1. *There must exist some q^* such that h_{QNT} is admissible.*

Proof. $q^* = 0$ results in a heuristic of 0 for every state, which is admissible. Thus, it follows that some $q^* \geq 0$ will result in h_{QNT} being admissible. □

In general, to use the quantile tunning method, we first train a classifier, then for each state, we compute the largest quantile that makes the heuristic admissible. Lastly, we take the minimum of all these quantiles, and use that quantile level for all the states.

3.2.2 Ensemble of Neural Networks

Our second method that preserves admissibility is to use an ensemble of ANNs, where the returned heuristic is the minimum of the heuristic returned by each ANN in the ensemble. Formally, we build a set $H = \{h_0, h_1, \ldots, h_k\}$ of heuristics which are combined for the ensemble heuristic:

$$h_{\text{ENS}(H)}(s) = \min_{i \in 0 \ldots |H|} h_i(s)$$

While the set H could be composed of any set of heuristics, we will build them by training successive ANNs on states that are inadmissible in the current

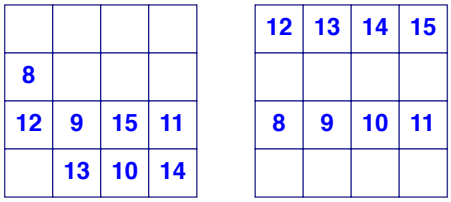


Figure 3.3: Left and right 4x4 Sliding Tile Puzzle (STP) states

heuristic. In particular, given H_t after t training steps, H_{t+1} is built by training h_{i+1} on the states $S = \{s \mid h_{\text{ENS}(H_t)}(s) > h_{\text{PDB}}(s)\}$. The overall procedure is started by training h_0 on h_{PDB}.

Lemma 1. *Adding h_{i+1} to a set of heuristics H_i (creating set H_{i+1}) cannot increase the number of states with inadmissible heuristics in $h_{\text{ENS}(H_i)}$.*

Proof. The heuristic of any state s computed by $h_{\text{ENS}(H_i))}$ is the minimum of the heuristics $h \in H_i$. Thus, adding an additional heuristic can only decrease the heuristic value for any state s, not increase it. □

Given this, it is easy to show that this process can learn an admissible heuristic.

Theorem 2. *Assuming that (1) ANN heuristics are deterministic and (2) that an ANN can also be trained to learn at least one state in the input set, then there exists some value k for which $h_{\text{ENS}(H_k)}$ will be admissible.*

Proof. Since each new ANN heuristic added to the ensemble makes at least one new state admissible and no new states inadmissible, the set H will only require a finite number of heuristics before $h_{\text{ENS}(H)}$ is admissible. □

As with the quantile method, the ensemble method is guaranteed to produce an admissible heuristic. In our experiments two ANNs were sufficient in an ensemble to achieve admissibility, even on a heuristic with 500 million unique states.

3.3 Training Improvements

3.3.1 Training with Underestimated Entries

The two approaches described above are sufficient to preserve admissibility of heuristics and can be generally applied to learn almost any PDBs. This section discusses ways to improve the quality of heuristics and further compress PDBs based on the two approaches.

The approach of taking the minimum of multiple actively trained ANNs focuses on dealing with overestimated entries to achieve 100% admissible rate. Underestimated entries do not affect admissibility, nonetheless, they do affect the quality of heuristics. One way to improve the quality of heuristics is to include part or all of the underestimated entries in the dataset that a succeeding ANN will be trained on. The underestimated entries can be added as a new class or grouped into the highest class in the original training set. If the succeeding ANNs together with preceding ones are sufficient to preserve admissibility, adding underestimated entries will improve the average heuristic value predicted by ANNs.

3.3.2 Quantile Tuning and Ensemble Combined

Another way to improve the heuristic from the ANN is to combine the quantile and ensemble methods together. Using some $q > q^*$ increases the average heuristic value in an ANN while still reducing the number of overestimated entries. This then reduces the training set required for the next ANN in the ensemble.

3.4 Training on Hard Data

Training on hard data is a slightly different way to train neural networks that will be later ensembled for use. Hard data is defined as a subset of data that an ANN, after having been trained on the whole dataset, tends to overestimate. An ANN overestimates some heuristics to minimize the overall mean square error. Such data points are problematic, as they cause ANNs

to produce inadmissible heuristics. Training an ANN on the hard data is one way to resolve the issue. Since the number of hard data points are much fewer than the whole dataset, it is much easier for an ANN to preserve the admissibility on the hard data. In our tests, the heuristic values for hard data are relatively small because entries usually overestimated by ANNs tend to have small heuristic values. Therefore, once an ANN has been trained on the hard data, it already achieves relatively high admissible rate (the number of admissible entries over the total number of entries) for the whole dataset. Then we may apply the ensemble method to preserve admissibility for the whole dataset. Training specifically on hard data not only achieves our primary goal of preserving admissible heuristics, but also requires much less training time. But, we must find the hard data first.

The process of finding hard data is similar to the process of training multiple models in an ensemble. First, we train an ANN on all data. Then, we train a second ANN on entries overestimated by the first ANN. The first group of hard data is the entries overestimated by the second ANN on the entire dataset. Instead of training the third ANN on the entries overestimated by both the first and second ANN, we train it on the first group of hard data to obtain the second group of hard data by gathering the overestimated entries by the third ANN on the entire dataset. The process repeats until a learning ANN outputs no inadmissible heuristic. As the learning process goes, the number of hard data points gets fewer and fewer, and thus the learning task becomes easier and easier. However, when only two models are trained and enough to ensemble an admissible heuristic, the hard data training is the same as the normal ensemble training.

Chapter 4

Learning Results

The aim of this thesis is to build admissible heuristic from PDBs with ANNs and show that this method can achieve better compression than standard approaches. In this chapter, we describe the training process. We train all the ANN models sequentially with Pytorch on CUDA 10.1 using one 2080ti GPU. Experiments are conducted on the 4x4 Sliding Tile Puzzle (STP), the 5x5 STP, and 16-tile TopSpin puzzle. Each of these domains can be represented as a permutation of numbers. The STP is a grid of numbers that must be sorted while the TopSpin puzzle is a continuous loop of numbers that must be sorted by rotating 4 adjacent tiles. PDBs were built using publicly available open-source software [1]. Domain abstractions we use for the experiments include tiles 1-7 and tiles 8-15 in additive PDBs taken as a delta over Manhattan distance (MD) in 4x4 STP, 4 6-tile additive PDBs delta over MD in 5x5 STP, and a 0-7 tile non-additive PDB in TopSpin. In a PDB, we use one byte to store a heuristic value. Therefore, the number of entries in a PDB is the size of the PDB in byte. The sizes of the PDBs in our experiments are shown in Table 4.2 and 4.3.

[1] https://github.com/nathansttt/hog2/tree/PDB-refactor

Domain	PDB Pattern	Accuracy (%)	Accu. after applying a floor func. (%)	No. of Overestimated Entries
4x4 STP	1-7	54.4713	99.9999	5,851
	8-15	61.2074	99.9861	72,115
5x5 STP	1, 5-6, 10-12	12.1656	99.9997	39
	2-4, 7-9	9.2832	99.9995	670
	13-14, 18-19, 23-24	96.9799	99.9998	195
	15-17, 20-22	2.6401	99.9998	285
TS	0-7	50.6427	95.9072	21,238,078

Table 4.1: Regression Accuracy

We use mean squared error (MSE) loss function for all regression models whereas cross-entropy loss function for all classification models. We use ReLU activation function for all the layers in all the models except the last layer of classification models, where we use softmax activation function instead. In addition, we use Adam optimizer with a learning rate of 0.01 as well as 3×3 kernels and a stride of 1 for all the convolutional layers in our models. When linear layers follow convolutional layers, reshaping is needed to ensure the dimensions agree. Patterns with heuristics i are labeled as the i-th class in TopSpin and the $\frac{i}{2}$-th class in STP since there are no odd heuristics in additive STP heuristics after taking the delta over MD. Quantile and ensemble combined models use the same structure as ensemble models. The quantile level for the combined quantile and ensemble models are set to 0.02. It is important to note that we do not tune the parameters, though parameter tuning might improve the heuristic results. The input representation, structures of our models, and the details of training can be found in Appendix A.

4.1 Regression Accuracy

We define accuracy as $accurary = 1 - \frac{the\ number\ of\ overestimated\ entries}{the\ number\ of\ total\ entries}$. From Table 4.1 we find that applying a floor function to the prediction can improve the accuracy to a great extent. This means that for approaches such as ADP using a hash table to store overestimated entries, the required amount of memory can be significantly reduced by merely applying a floor function. A small amount of memory is needed to store overestimated entries in the 5x5 STP problem domain since there are only 39, 670, 195, and 285 overestimated entries, respectively. However, it requires a large amount of memory to store overestimated entries with a hash table in Topspin puzzle.

4.2 Classification vs. Regression

When we compare the classification with regression, it is clear (Table 4.3 and Table 4.2) that our novel approaches (quantile, ensemble, and quantile and ensemble combines) with classification outperforms those (affine transform and

Domain	PDB Pattern	Heuristic Size (MB)				Avg. Heuristic Value			
		h_{PDB}	h_{DIV}	h_{AFF}	h_{ENS}	h_{PDB}	h_{DIV}	h_{AFF}	h_{ENS}
5x5 STP	1, 5-6, 10-12	127.51	1.27	1.27	1.27	2.0773	1.7219	0.0079	0.5389
	2-4, 7-9					0.9639	0.3050	0.0006	0.0293
	13-14, 18-19, 23-24					0.9639	0.2673	0.0079	0.0108
	15-17, 20-22					0.9639	0.3361	0.0519	0.0608
4x4 STP	1-7	57.66	0.58	0.57	0.54	3.9122	2.0825	0.3011	1.4802
	8-15	518.92	5.19	5.12	5.12	3.9728	1.6521	0.3363	1.0549
TS	0-7	518.92	5.18	5.18	5.18	9.1350	6.9862	5.5465	4.2460

Table 4.2: Summary results comparing all techniques (regression)

Domain	PDB Pattern	Heuristic Size (MB)					Avg. Heuristic Value				
		h_{PDB}	h_{DIV}	h_{QNT}	h_{ENS}	h_{Q+E}	h_{PDB}	h_{DIV}	h_{QNT}	h_{ENS}	h_{Q+E}
5x5 STP	1, 5-6, 10-12	127.51	1.27	1.27	1.27	1.27	2.0773	1.7219	**1.9154**	1.5600	1.0772
	2-4, 7-9						0.9639	0.3050	**0.7353**	0.4075	0.3767
	13-14, 18-19, 23-24						0.9639	0.2673	**0.7856**	0.4379	0.3751
	15-17, 20-22						0.9639	0.3361	**0.7972**	0.4237	0.3275
4x4 STP	1-7	57.66	0.58	0.57	0.54	0.62	3.9122	2.0825	1.7868	**2.8422**	2.4562
	8-15	518.92	5.19	5.12	5.12	6.33	3.9728	1.6521	2.3362	1.8535	**2.5094**
TS	0-7	518.92	5.18	5.18	5.18	5.18	9.1350	**6.9862**	6.8593	6.0478	5.8854
TS	0-7	518.92	0.52	0.52	0.52	0.52	9.1350	5.6442	4.6089	**5.7716**	5.5841

Table 4.3: Summary results comparing all techniques (classification)

ensembled) with regression. In the 5x5 STP with classification, the quantile plus ensemble performs the worst. The average heuristic values are 1.0772, 0.3767, 0.3751, and 0.3275, respectively. However, quantile plus ensemble with classification still delivers much better average heuristic value than affine transform or ensemble with regression. The average heuristic values for affine transform with regression are 0.0079, 0.0006, 0.0079, and 0.0519, respectively, while those for ensemble with regression are 0.5389, 0.0293, 0.0108, and 0.0608, respectively. In the 4x4 STP, with classification quantile method performs the worst on 1-7 PDB and ensemble performs the worst on 8-15 PDB, but any of these outperforms affine transform or ensemble method with regression. In the TopSpin puzzle, with classification the quantile and ensemble combined method performs the worst, delivering an average of heuristic of 5.8854. Nonetheless, the quantile and ensemble combined with classification still generate greater average heuristic than affine transform or ensemble with regression. In general, quantile, ensemble, or quantile plus ensemble with classification outperforms affine transform or ensemble with regression. In the following we discuss the learning results with classification in detail.

4.3 The 4x4 STP

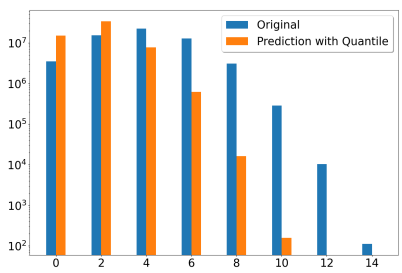
For the 1-7 PDB, the average of uncompressed heuristic is 3.911. With an average heuristic of 2.8422, ensemble methods performs the best, while DIV compression gives an average heuristic of 2.0825. For the 8-15 PDB, the average uncompressed heuristic is 3.9728. With an average heuristic of 2.5094, the quantile and ensemble combined method performs the best in this domain, while DIV compression performs the worst. Overall, the quantile and ensemble combined method achieves the best average heuristic of 4.9656, while DIV compression performs the worst.

4.4 The 5x5 STP

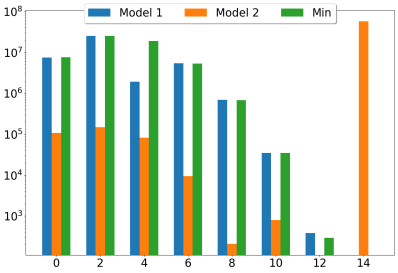
For all the 5x5 STP PDBs, quantile tuning method performs the best. Quantile tunning method gives an average heuristic of 1.9154, 0.7353, 0.7856, and 0.7972, respectively. A 100 times smaller CNN can learn a PDB really well compared with the learning results in other problem domains as the loss from compression in heuristics are the smallest in the 5x5 STP. For the second to the fourth PDBs, the ensemble method performs the second best, whereas DIV compression method performs the worst. The quantile plus ensemble method does not perform very well in 5x5 STP problem domain. This is because the 5x5 STP problem domain is relatively easy. One 100 times smaller CNN model can learn the heuristic quite well with a small portion of entries overestimated. With only a few overestimated entries, the second model cannot learn the overall heuristic of those PDBs.

4.5 Topspin

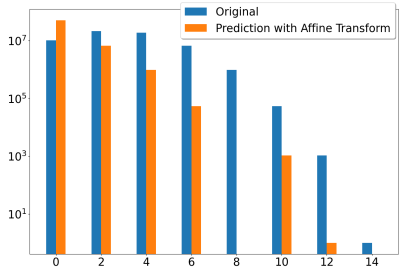
In Topspin, DIV compression has relatively strong performance given a compression factor of 100, and is able to outperform the quantile and ensemble methods. However, we discover that as we further compressed the heuristics, the performance of the ANNs grew relative to the DIV compressed method, and the ensemble ANN is able to outperform the DIV compression by a small



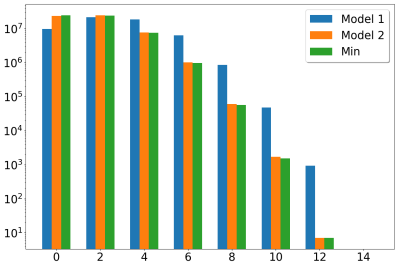
(a) Quantile with Classification



(b) Ensemble with Classification



(c) Affine Transform with Regression



(d) Ensemble with Regression

Figure 4.1: Prediction Distribution for 1-7 PDB in 4x4 STP

margin. Given a compression factor of 1000, the ensemble method delivers an average heuristic of 5.7716, higher than DIV compression by 0.1274.

4.6 Distribution by ANN Models

We now look at the distribution learned by the ANN models in our tests. Since all distributions are very similar, we are going to discuss distributions for 1-7 PDB in the 4x4 STP in detail. All other distributions can be found on Appendix B. As shown in 4.1a and 4.1c, the prediction with quantile and the prediction with affine transform distributions can be obtained by shifting original prediction values towards small ones. Likewise, the prediction distribution by taking the minimum of two models can be obtained by shifting the prediction values by the first model towards small ones. It shows that our methods guarantee admissibility of heuristics by changing overestimated heuristic values to small ones. In Figure 4.1a, there are more states with a heuristic of 0 and 2, but fewer with a heuristic greater than 2. In Figure 4.1b, there are fewer states with a heuristic of 12. In Figure 4.1c, there are more

states with a heuristic of 0, but fewer with all other heuristics. In Figure 4.1d, there are states with a heuristic of 0, but fewer with heuristic values greater than 0.

When we compare the real distribution with the model distribution in quantile or affine transform, or the first model in ensemble as shown in Figure 4.1, we find that they are very similar. When we compare the original distribution by those ANN models with the distribution after setting a quantile level, applying affine transform, or ensembling, we find that those methods shift the distribution towards small values.

4.7 Other Experiments

There are additional experiments through which we gain more understanding about the properties of our approaches. The sizes of the models in those experiments are not necessarily 100 times smaller than the original PDBs.

4.7.1 Quantile

We can now look in more detail on the performance of the quantile approach. We show the result of learning a single 4x4 STP heuristic with different quantile values in Table 4.4. With $q = 0.1$ over 99% of the heuristic values are learned, although given that the 8–15 PDB has 519 million entries, there is still a larger number of overestimated entries. These values give a sense of the loss incurred using different quantiles during learning. We observe a trade-off between the admissibility rate and the quality of heuristics measured by the average heuristic value.

	1-7		8-15	
q	% Adm.	$\overline{h}_q$	% Adm.	$\overline{h}_q$
0.30	97.4656	3.7663	96.7981	3.8425
0.20	98.4687	3.6889	98.0109	3.7568
0.15	98.9197	3.6396	98.5742	3.7019
0.10	99.3402	3.5746	99.1101	3.6294
(h_{PDB})	100.0000	3.9122	100.0000	3.9728

Table 4.4: Different quantiles for CNN 1-7 and 8-15

4.7.2 Ensembles

In the 4x4 STP 8-15 tile PDB, ensemble method achieve an average heuristic of 2.9118 with 3.21 MB and 2.13 MB CNNs, while the same PDB compressed to 5.41 MB with DIV compression has an average heuristic of only 1.6686. A DIV compressed PDB with an average heuristic of 2.8417 is 74.13 MB in size. The size of two CNNs combined is more than 14 times smaller than the DIV compressed PDB with the same average heuristic value. In addition, ensemble methods can compress the 518.92 MB TS PDB by a factor of more than 1,800 with slightly better average heuristics than a comparable size PDB with DIV compression.

4.7.3 Ensembles and Quantiles

We now look specifically at the 1-7 tile PDB in the 4x4 STP, which was difficult to learn with both ensemble and quantile approaches individually. Using q = 0.1 for the first CNN, the training set is reduced to approximately 2.5 million data points, which makes learning task much easier. By taking the minimum of a 0.42 MB CNN with q = 0.1 and a 0.81 MB CNN without using quantile, an average heuristic of 2.6501 is obtained. A comparably sized (1.31 MB) PDB compressed with DIV from the 1-7 PDB has an average heuristic of 2.3483, and a DIV compressed PDB with an average heuristic of 2.5882 requires 2.62 MB of storage.

4.7.4 Node Expansions

Now we use the 1-7 NN models described in section 5.7.3 and 8-15 models described in 5.7.2 to solve the 100 4x4 STP problems in Richard E. Korf's paper [17]. We use the quantile and ensemble combined method for the 1-7

		CNN Size		
Domain	PDB	1st	2nd	$\overline{h}_{\text{CNN}}$
4x4 STP	8-15	3.21MB	2.13MB	2.9118
TS	0-7	0.20MB	0.06MB	5.7716

Table 4.5: Min of ensembles

NN models with a quantile level of 0.1 for the first model, and the ensemble method for the 8-15 NN models. We compare the average and the median number of node expansions by IDA* using those ANN models with using comparable size (6.57 MB in total) PDBs compressed with DIV. The average number of node expansions is 2,190,237 for using NN models and 6,728,043 for using comparable size PDBs. The median number of node expansions is 709,062 for using NN models and 1,245,480 for using comparable size PDBs. Using the NN heuristics during IDA* search expands fewer nodes than using the comparable size PDBs. The results are consistent with our findings that the average heuristic values of the NN heuristics are higher than those of the comparable size PDBs.

4.8 Summary

Our experimental results show that our approaches are able to guarantee admissible heuristic values by learning from PDBs, either with regression or classification. Nonetheless, the average heuristic value with classification is much greater than that with regression method. Used as a compression method, our approaches have achieved better average heuristic values than DIV compression in most of PDBs in 4x4 STP, 5x5 STP, and Topspin problem domains. The only exception is the 100 times smaller model to learn 0-7 Topspin PDB. In addition, fewer nodes are expanded using NNs heuristics than using comparable size PDBs during IDA* search. We also found that the hard data training method can reduce the number of CNN models needed to maintain the admissibility of heuristics. For all the PDBs in our experiments, merely two CNN models are finally used to ensemble admissible heuristics. As we have shown in distribution figures, the quantile, affine transform, and ensemble method change the distribution of the initial model by changing large heuristic values to small ones. Overall, the resulting distribution is similar to the initial one, but the resulting distribution is more right skewed.

Chapter 5

Conclusion

In this thesis, we have shown how convolutional neural networks can be used in heuristic search. We introduced two approaches, i.e., the quantile and the ensemble, to maintain the admissibility of heuristic. We have conducted various experiments in three problem domains (the 4x4 STP, the 5x5 STP, and the TopSpin Puzzle) to show that those approaches are able to preserve admissibility with both classification and regression. Having examined the distributions generated by the CNN models, we found that the quantile and the ensemble method change their original prediction distributions by changing the high values to small ones. Although we suffer some losses, the results in the experiments indicate that, given a certain size, our approaches deliver greater average heuristic values in most problems compared to DIV compression techniques. Experimental results show that our approaches with classification performs significantly better than with regression in any problem domain. The following findings are based on our approaches with classification. Based on our experiments, we found that the quantile method performs the best in the 5x5 STP. A 1.27 MB CNN model can learn any one of the 4 6-tile PDBs really well. The ensemble method performs the best in 1-7 PDB in the 4x4 STP, while the quantile and the ensemble combined method performs the best in 8-15 PDB, the largest PDB in the experiment, in the 4x4 STP. In the TopSpin, when the size limit is set to be 5.18 MB (around 100 times smaller than the 0-7 PDB in the TopSpin), DIV compression does the best. When the size limit is to be 0.52MB (around 1000 times smaller than the 0-7 PDB in the TopSpin),

the quantile method does the best. In addition, we found that IDA* expands much fewer nodes using our ANN models than using comparable size PDBs with DIV compression in the 4x4 STP. Besides the quantile and the ensemble methods, this thesis introduces the hard data training approach, which can simplify the training process. With the hard data approach, only two models are needed to ensemble an admissible CNN model for all the PDBs in our experiments.

For training with regression, we found that applying a floor function to a model significantly reduce the number of overestimated entries. Therefore, in method such as ADP [23], applying a floor function can significantly reduce the memory required to store overestimated entries with a hash table.

As for the training time, we found that the time needed to train a 100 times smaller CNN model 100 epochs is linearly correlated with the size of PDBs. In the 4x4 STP, it takes around 4 minutes to train a 0.57 MB (100 times smaller than the 1-7 PDB) CNN model one epoch on 1-7 the PDB, while 39 minutes to train a 5.8 MB (100 times smaller than the 8-15 PDB) CNN model on the 8-15 PDB. According to our rough estimates, it would take over 3000 hours to train a 30 GB CNN model one epoch on a 3 TB PDB. Unfortunately, we were not able to complete the scaling tasks due to the limited resources, but we have provided a foundation for it to be done in the future. In addition to the scaling task, another challenge is to develop high-performing search algorithms based on efficient parallel GPU heuristic lookups.

References

[1] M. Ball and R. C. Holte, "The compression power of symbolic pattern databases.," in *ICAPS*, 2008, pp. 2–11. 8

[2] T. M. Breyer and R. Korf, "1.6-bit pattern databases," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. 8

[3] F. Chollet, "The measure of intelligence," *arXiv preprint arXiv:1911.01547*, 2019. 11

[4] R. K. Clausecker and F. Schintke, "A measure of quality for IDA* heuristics," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 12, 2021, pp. 55–63. 4

[5] J. C. Culberson and J. Schaeffer, "Pattern databases," *Computational Intelligence*, vol. 14, no. 3, pp. 318–334, 1998. 3, 7

[6] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989. 11

[7] R. Döbbelin, T. Schütt, and A. Reinefeld, "Building large compressed pdbs for the sliding tile puzzle," in *IJCAI Workshop on Computer Games*, 2013, pp. 16–27. 8

[8] A. Felner, R. E. Korf, and S. Hanan, "Additive pattern database heuristics," *Journal of Artificial Intelligence Research*, vol. 22, pp. 279–318, 2004. 8

[9] A. Felner, R. E. Korf, R. Meshulam, and R. C. Holte, "Compressed pattern databases," *Journal of Artificial Intelligence Research*, vol. 30, pp. 213–247, 2007. 4, 8

[10] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997. 11

[11] M. Goldenberg, N. R. Sturtevant, A. Felner, and J. Schaeffer, "The compressed differential heuristic," in *AAAI Conference on Artificial Intelligence*, 2011, pp. 24–29. 8

[12] P. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 2 1968. 1, 6

[13] M. Helmert, N. R. Sturtevant, and A. Felner, "On variable dependencies and compressed pattern databases," *Symposium on Combinatorial Search (SoCS)*, 2017. 9

[14] R. C. Holte, J. Newton, A. Felner, R. Meshulam, and D. Furcy, "Multiple pattern databases.," in *ICAPS*, 2004, pp. 122–131. 14

[15] S. Hu and N. R. Sturtevant, "Direction-optimizing breadth-first search with external memory storage," *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019. 8

[16] S. Judd, "On the complexity of loading shallow neural networks," *Journal of Complexity*, vol. 4, no. 3, pp. 177–192, 1988. 13

[17] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial Intelligence*, vol. 27, no. 1, pp. 97–109, 1985. 1, 6, 27

[18] R. E. Korf, M. Reid, and S. Edelkamp, "Time complexity of iterative-deepening-A*," *Artificial Intelligence*, vol. 129, no. 1–2, pp. 199–218, 2001. 4

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105. 11

[20] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in neural information processing systems*, 2017, pp. 6231–6239. 11

[21] W. Myrvold and F. Ruskey, "Ranking and unranking permutations in linear time," *Information Processing Letters*, vol. 79, pp. 281–284, 2001. 7

[22] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814. 10

[23] M. Samadi, M. Siabani, A. Felner, and R. Holte, "Compressing pattern databases with learning.," in *ECAI*, 2008, pp. 495–499. 4, 9, 30

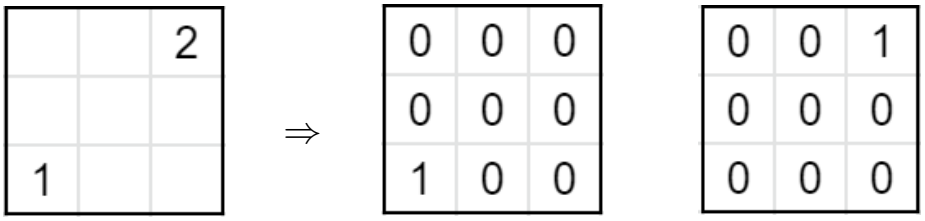
[24] N. R. Sturtevant, A. Felner, and M. Helmert, "Value compression of pattern databases," in *AAAI Conference on Artificial Intelligence*, 2017. 8

Appendix A

Training

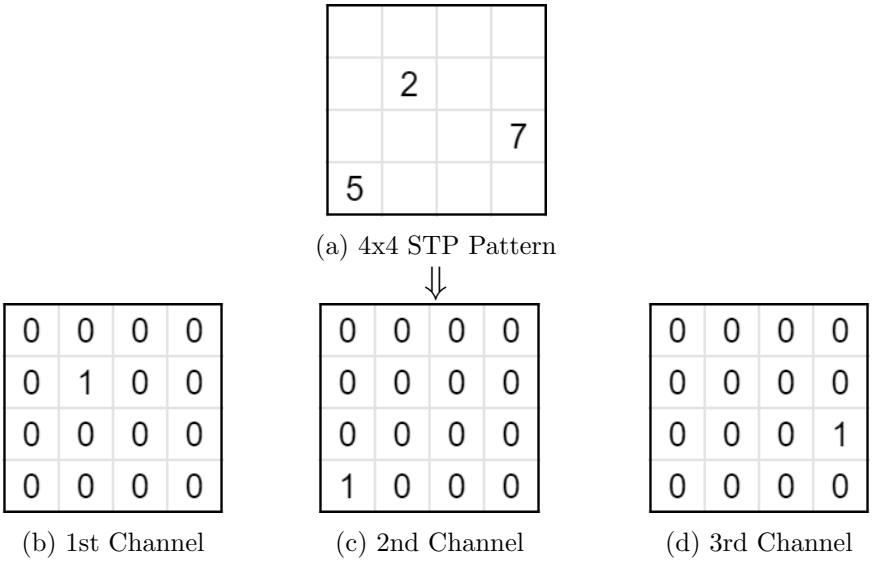
A.1 State Representation

In an ANN, patterns containing m numbers are represented as a sequence of m-channel binary images. Elements corresponding to the position of the i-th element in the pattern are ones and all other elements are zeros. For example, Figure A.1 shows how a 3x3 sliding-tile puzzle pattern containing two numbers (1 and 2) is represented. In Figure A.1a, a pattern with 1 on the bottom-left corner and 2 on the upper-right corner is represented as a two-channel image. Inside the first channel of the image shown in Figure A.1b, the bottom-left element corresponding to the position of 1 in the original pattern is one, and all other elements are zeros. Similarly, inside the second channel of the image shown in Figure A.1c, the upper-right element corresponding to the position of 2 in the original pattern is one, and all other elements are zeros. In Figure A.2 a 4x4 Sliding-tile puzzle pattern (Figure A.2a) that contains three numbers (2, 5, 7) is represented as a three-channel image (Figure A.2b A.2c A.2d), and the element corresponding to the i number in the pattern in the i-th channel is



(a) 3x3 STP Pattern (b) 1st Channel (c) 2nd Channel

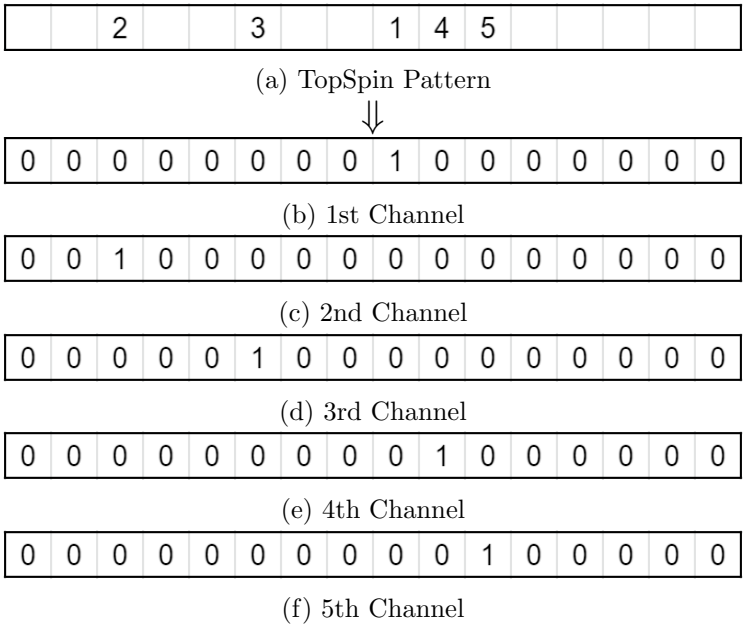
Figure A.1: 3x3 STP Representation



(a) 4x4 STP Pattern

(b) 1st Channel (c) 2nd Channel (d) 3rd Channel

Figure A.2: 4x4 STP Representation



(a) TopSpin Pattern

(b) 1st Channel

(c) 2nd Channel

(d) 3rd Channel

(e) 4th Channel

(f) 5th Channel

Figure A.3: Topspin Representation

one with all else being zeros. For the Topspin problem, we represent the states and patterns as 1-dimensional images with multiple channels. In Figure A.3, a Topspin pattern with numbers 1 to 5 (A.3a) is represented as a five channel image. The corresponding position of the i-th element are ones and all other elements are zeros.

A.2 Regression Models

A.2.1 The 4x4 STP

There are two delta additive PDBs for the 4x4 STP: 1-7 and 8-15. In the 0.55 MB model, there is 1 convolutional layer and 3 linear layers. The first layer is a convolutional layer with an input channel size of 7 and output channel size of 32. Following the convolutional layer are three linear layers. The first linear layer has an input channel size of 128 and an output channel size of 256, the second linear layer have an input channel size of 256 and an output channel size of 400, and the third layer (the last layer) has an input channel size of 400 and an output channel size of 1. It takes around 5 hours to train the model for 100 epochs. After the training completed, we applied an affine transform to obtain an admissible heuristic from the model. The value that we subtract to make the goal state prediction 0 is 2, and the fraction we use to scale the prediction after the subtraction is 1.25.

In the two ensemble models with a total size of 0.56 MB (0.28 MB each), there is 1 convolutional layer and 3 linear layers as well. The first layer is a convolutional layer with an input channel size of 7 and output channel size of 32. Following the convolutional layer are three linear layers. The first linear layer has an input channel size of 128 and an output channel size of 256, the second linear layer have an input channel size of 256 and an output channel size of 138, and the third layer (the last layer) has an input channel size of 138 and an output channel size of 1. It takes around 3 hours to train the first model for 100 epochs and 1 hour to train the second model until the second model does not overestimate any entries.

In the 5.7 MB model for 8-15 PDB in the 4x4 STP, there are 7 layers in

total. The first layer is a convolutional layer with an input channel size of 8 and output channel size of 32. Following the convolutional layer are six linear layers. The first linear layer has an input channel size of 128 and an output channel size of 256, the second linear layer has an input channel size of 256 and an output channel size of 512, and the third layer has an input channel size of 512 and an output channel size of 1024, the fourth layer has an input channel size of 1024 and an output channel size of 512, the fifth layer has an input channel size of 512 and an output channel size of 400, and the sixth (the last) layer has an input channel size of 400 and an output channel size of 1. It takes around 60 hours to train the model for 100 epochs. After the training completed, we applied affine transform to obtain admissible heuristic from the model. The value that we subtract to make the goal state prediction 0 is 1.0470, and the fraction we use to scale the prediction after the subtraction is 1.0094.

In the two ensemble models with a total size of 5.7 MB (2.85 MB each), there is 1 convolutional layer and 6 linear layers as well. The first layer is a convolutional layer with an input channel size of 8 and output channel size of 32. Following the convolutional layer are three linear layers. The first linear layer has an input channel size of 128 and an output channel size of 256, the second linear layer have an input channel size of 256 and an output channel size of 512, the third layer has an input channel size of 512 and an output channel size of 1024, the fourth layer has an input size of 1024 and an output size of 512, the fifth layer has an input size of 512 and an output size of 400, the sixth layer (the last layer) has an input size of 400 and an output size of 1. It takes around 50 hours to train the first model for 100 epochs and 1 hour to train the second model until the second model does not overestimate any entries.

A.2.2 The 5x5 STP

There are four delta additive PDBs for the 5x5 STP. For the 4 affine transform models, there is 1 convolutional layer, and 3 linear layers. The input size of the convolutional layer is 6, while the output size is 32. The first linear layer

has an input channel size of 288 and an output channel size of 400, the second linear layer have an input channel size of 400 and an output channel size of 500, and the third layer (the last layer) has an input channel size of 500 and an output channel size of 1. It takes around 6 hours to train the model for 100 epochs. After the training completed, we applied affine transform to obtain admissible heuristic from the model. The values that we subtract to make the goal state prediction 0 are 1.0062, 0.0079, -0.0038, and 0.0111, respectively. The fractions we use to scale the prediction after the subtraction are 1.0031096, 0.5019828, 0.4990518, and 0.5027905, respectively.

In the 2 ensemble models with a total size of 1.27 MB (0.63 MB each), there is 1 convolutional layer and 3 linear layers as well. The first layer is a convolutional layer with an input channel size of 6 and output channel size of 32. Following the convolutional layer are three linear layers. The first linear layer has an input channel size of 288 and an output channel size of 288, the second linear layer have an input channel size of 288 and an output channel size of 252, and the third layer (the last layer) has an input channel size of 252 and an output channel size of 1. We use the hard data training approach to train ensemble models. It takes around 5 hours to train the first model for 100 epochs. We train the second model for 10 hours with entries overestimated by the first model. Then we check the prediction accuracy of the second model on the entire PDB. Those entries overestimated by the second model are used as the training data for the third model. Lastly, we train the third model until it does not overestimate any of its training data. We use the second and the third models to ensemble an admissible heuristic.

A.2.3 The TopSpin

In our 5.17 MB affine transform model, there are 6 layers. The first layer is a convolutional layer with an input channel size of 8 and an output channel size of 32. Following the convolutional layer are five linear layers. The first linear layer has both input and output channel sizes of 488, the second linear layer has an input size of 488 and an output size of 512, the third layer has an input channel size of 512 and an output channel of 836, the fourth layer has

an input channel size of 836 and an output channel of 515, and the fifth layer (the last layer) has an input channel size of 515 and an output channel of 1. It takes around 50 hours to train the model for 100 epochs. The value that we subtract to make the goal state prediction 0 is 1.2392, and the fraction we use to scale the prediction after the subtraction is 0.7731162.

In the two ensemble models with a total size of 5.18 MB (2.59 MB each), there are 5 layers. The first layer is a convolutional layer with an input channel size of 8 and output channel size of 32. Following the convolutional layers are 4 linear layers. The first linear layer has an input channel size of 488 and an output channel size of 488, the second has an input channel size of 488 and an output channel size of 512, the third linear layer has an input channel size of 512 and an output channel size of 418, and the last layer has an input channel size of 418 and an output channel size of 1. It takes around 50 hours to train the first model for 100 epochs. We train the second model for 10 hours with entries overestimated by the first model. Then we check the prediction accuracy of the second model on the entire PDB. Those entries overestimated by the second model are used as the training data for the third model. Lastly, we train the third model until it does not overestimate any of its training data. We use the second and the third models to ensemble an admissible heuristic.

A.3 Classification Models

A.3.1 The 4x4 STP

In the 0.57 MB quantile model, there are 4 layers. The first layer is a convolutional layer with an input channel size of 7 and an output channel size of 32. Following the convolutional layer are three linear layers. The first linear layer has an input channel size of 128 and an output channel size of 312, the second linear layer have both input and output sizes of 312, and the third layer (the last layer) has an input channel size of 312 and an output channel size of 8. It takes around 5 hours to train the model for 100 epochs. After the training is finished, we find the largest possible quantile level for each state that can make the heuristic for that state admissible. Finally we take the minimum

from all those quantile levels.

In our 2 ensemble models and 2 quantile and ensemble combined models with a total size of 0.57 MB (0.28MB each), there are 4 layers. The first layer is a convolutional layer with an input channel size of 7 and an output channel size of 32. Following the convolutional layer are three linear layers. The first linear layer has an input channel size of 128 and an output channel size of 164, the second linear layer has an input channel size of 164 and an output sizes of 256, and the second layer (the last layer) has an input channel size of 256 and an output channel of 8. We use the hard data approach to train the ensemble models. There are three models in total. We train the first model for 10 hours. We train the second model for 5 hours with entries overestimated by the first model. Then we check its prediction accuracy of the second model on the whole PDB. Those entries overestimated by the second model are used as the training data for the third model. Lastly, we train the third model until it does not overestimate any of its training data. We use the second and the third models to ensemble an admissible heuristic.

We also use the hard data approach to train our 1-7 models with ensemble and quantile tuning combined method. We train the first model for 20 hours. Then we check the prediction accurary of that model with a preset quantile level. We set all quantiles to 0.02 for all the ensemble and quantile tuning combined models. The smaller the quantile, the fewer entries overestimated. Then we train the second model on the entries overestimated by the first model with the 0.02 quantile level until it does not overestimate any of its training data.

In our 5.12 MB 8-15 quantile model, there are 6 layers. The first layer is a convolutional layer with an input channel size of 8 and an output channel size of 32. Following the convolutional layer are five linear layers. The first linear layer has an input size of 128 and an output channel size of 256, the second linear layer has an input size of 256 and an output size of 512, the third layer has an input channel size of 512 and an output channel of 1024, the fourth layer has an input channel size of 1024 and an output channel of 568, and the fifth layer (the last layer) has an input channel size of 568 and an output

channel of 10. It takes around 60 hours to train the model for 100 epochs. After the training is finished, we find the largest possible quantile level for each state that can make the heuristic for that state admissible. Finally we take the minimum from all those quantile levels.

In our two 8-15 ensemble models and two 8-15 quantile and ensemble combined models with a total size of 5.12MB (2.06MB each), there are 4 layers. The first layer is a convolutional layer with an input channel size of 8 and an output channel size of 32. Following the convolutional layer are three linear layers. The first linear layer has an input channel size of 128 and an output channel size of 512, the second linear layer has an input channel size of 512 and an output sizes of 854, and the second layer (the last layer) has an input channel size of 854 and an output channel of 10. We use the hard data training approach to train our models. We train the first model for 20 hours. We train the second model for 5 hours with entries overestimated by the first model. Then we check the prediction accuracy of the second model on the entire PDB. Those entries overestimated by the second model are used as the training data for the third model. Lastly, we train the third model until it does not overestimate any of its training data. We use the second and the third models to ensemble an admissible heuristic.

For the ensemble and quantile tuning combined method. We train the first model for 20 hours. Then we check the prediction accurary of that model with a preset quantile level. We set all quantiles to 0.02 for all the ensemble and quantile tuning combined models. The smaller the quantile, the fewer entries overestimated. Then we train the second model on the entries overestimated by the first model with the 0.02 quantile level until it does not overestimate any of its training data.

A.3.2 The 5x5 STP

All models for quantile method share the same architecture. Those 1.27MB models have 4 layers. The first layer is a convolutional layer with an input channel size of 6 and output channel size of 32. There are 3 linear layers following the convolutional layer. The first linear layer has an input channel

size of 288 and an output channel size of 396, the second has an input channel size of 396 and an output channel size of 496, and the last linear layer has an input channel size of 496 and an output channel size of 6. We train the model for 10 hours and then use the same method as we use for PDBs in the 4x4 STP to find the quantile level.

All models for ensemble method also share the same architecture. They are 0.63MB in size. Two models are ensembled for each PDB. Each model have 4 layer. The first layer is a convolutional layer with an input channel size of 6 and output channel size of 32. There are 3 linear layers following the convolutional layer. The first linear layer has an input channel size of 288 and an output channel size of 288, the second has an input channel size of 288 and an output channel size of 248, and the last linear layer has an input channel size of 248 and an output channel size of 6. As can be seen from Table **??**, the architectures for quantile plus ensemble method are exactly the same as those models for ensemble method. For ensemble and quantile plus ensemble method, we train the first model for 10 hours, and then we train the second model with entries overestimated by the first model until it does not overestimate any.

A.3.3 The TopSpin

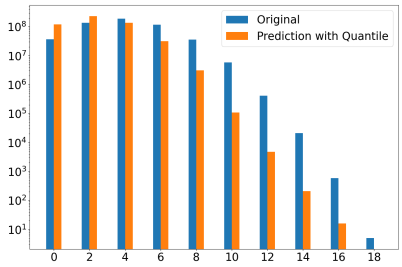
In our 5.18 MB quantile model, there are 6 layers. The first layer is a convolutional layer with an input channel size of 8 and an output channel size of 32. Following the convolutional layer are five linear layers. The first linear layer has both input and output channel sizes of 488, the second linear layer has an input size of 488 and an output size of 512, the third layer has an input channel size of 512 and an output channel of 836, the fourth layer has an input channel size of 836 and an output channel of 512, and the fifth layer (the last layer) has an input channel size of 512 and an output channel of 13. It takes around 50 hours to train the model for 100 epochs. After the training is finished, we use the same method as we use for PDBs in the 4x4 STP to find the quantile level.

Two 2.55 MB 5-layer model are used to ensemble an admissible heuristic.

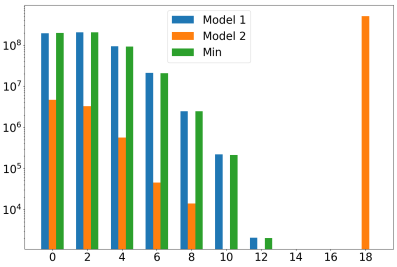
The first layer is a convolutional layer with an input channel size of 8 and output channel size of 32. There are 4 linear layers following the convolutional layer. The first linear layer has an input channel size of 488 and an output channel size of 488, the second has an input channel size of 488 and an output channel size of 512, the third linear layer has an input channel size of 512 and an output channel size of 408, and the last layer has an input channel size of 408 and an output channel size of 13. In this problem domain, we also experiment with 1000 times compression. Two 0.26MB models have 5 layers. The first layer is a convolutional layer with an input channel size of 8 and output channel size of 32. There are 4 linear layers following the convolutional layer. The first linear layer has an input channel size of 448 and an output channel size of 112, the second has an input channel size of 112 and an output channel size of 64, the third linear layer has an input channel size of 64 and an output channel size of 64, and the last layer has an input channel size of 64 and an output channel size of 13. The architectures for quantile plus ensemble method are exactly the same as those models for ensemble method. For ensemble method and quantile plus ensemble method, we use hard data approach to train our model. There is a total of 3 models. The training procedure is exactly the same as how we train models on the 1-7 4x4 STP PDB.

Appendix B

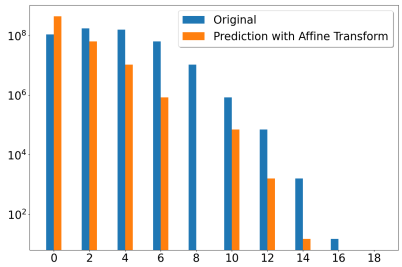
Heuristic Distributions by ANN Models



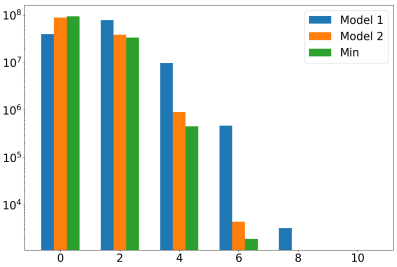
(a) Quantile with Classification



(b) Ensemble with Classification

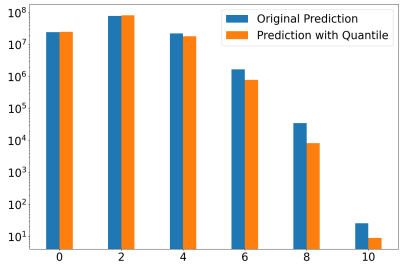


(c) Affine Transform with Regression

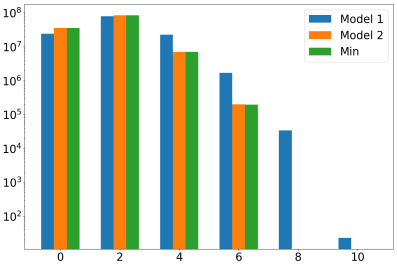


(d) Ensemble with Regression

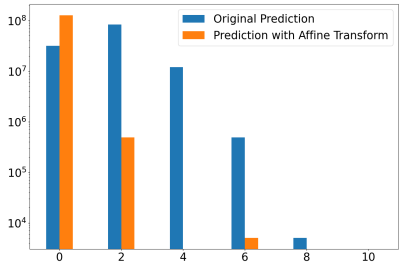
Figure B.1: Prediction Distribution for 8-15 PDB in 4x4 STP



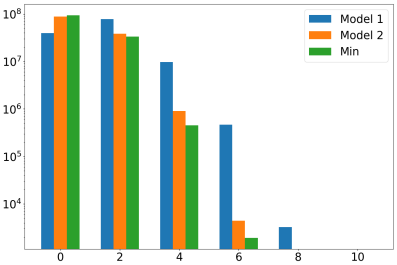
(a) Quantile with Classification



(b) Ensemble with Classification

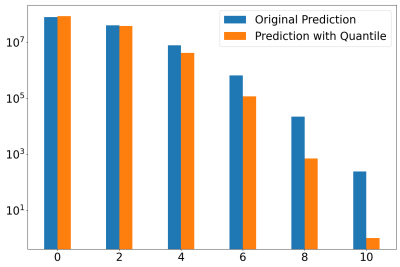


(c) Affine Transform with Regression

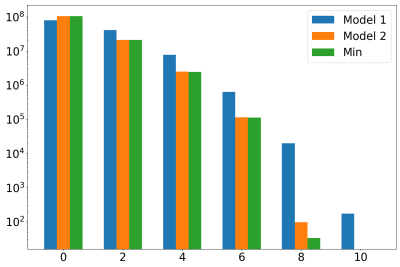


(d) Ensemble with Regression

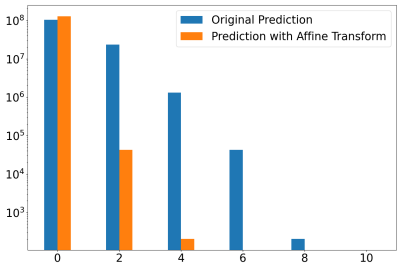
Figure B.2: Prediction Distribution for the 1st PDB in 5x5 STP



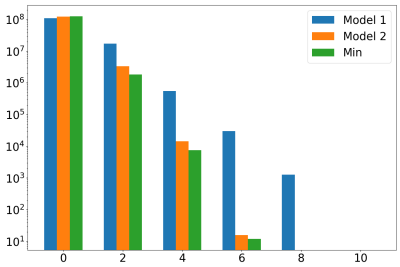
(a) Quantile with Classification



(b) Ensemble with Classification

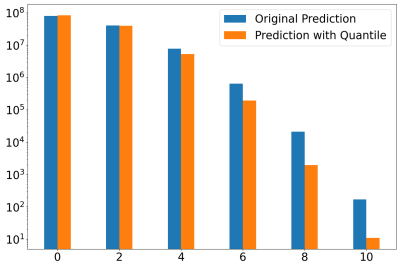


(c) Affine Transform with Regression

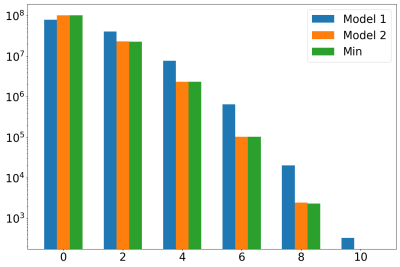


(d) Ensemble with Regression

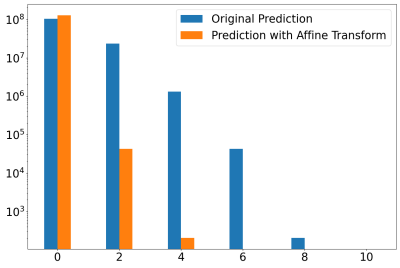
Figure B.3: Prediction Distribution for the 2nd PDB in 5x5 STP



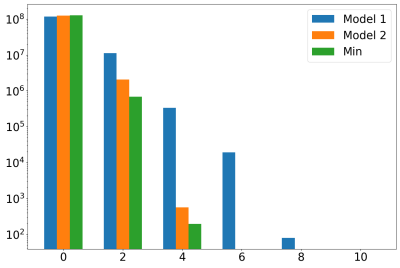
(a) Quantile with Classification



(b) Ensemble with Classification

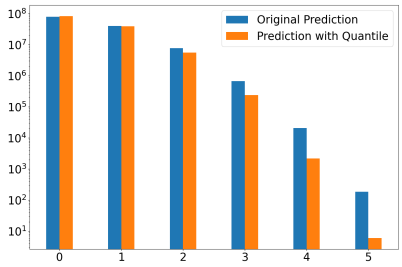


(c) Affine Transform with Regression

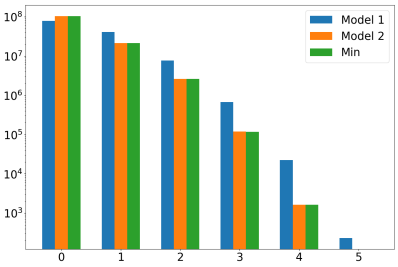


(d) Ensemble with Regression

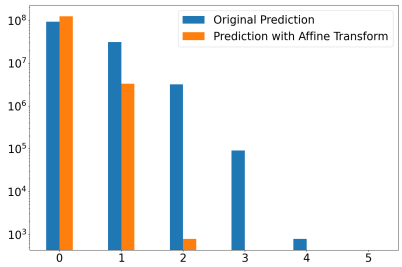
Figure B.4: Prediction Distribution for the 3rd PDB in 5x5 STP



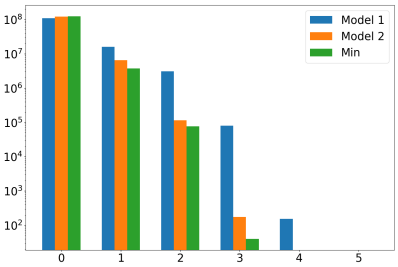
(a) Quantile with Classification



(b) Ensemble with Classification

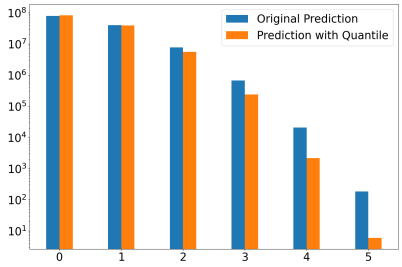


(c) Affine Transform with Regression

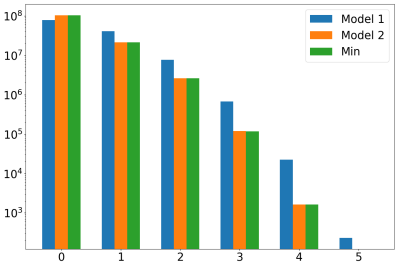


(d) Ensemble with Regression

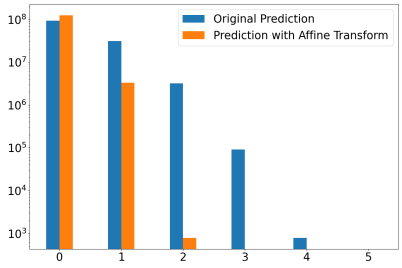
Figure B.5: Prediction Distribution for the 4th PDB in 5x5 STP



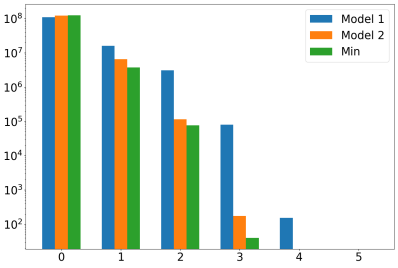
(a) Quantile with Classification



(b) Ensemble with Classification

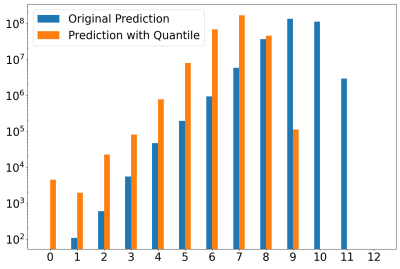


(c) Affine Transform with Regression

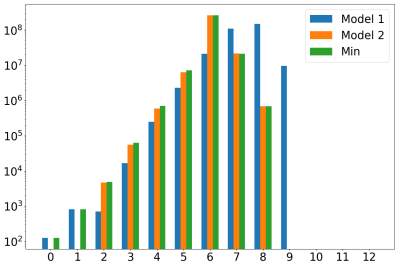


(d) Ensemble with Regression

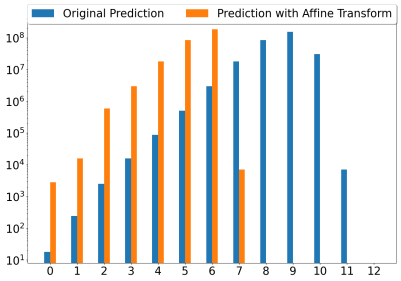
Figure B.6: Prediction Distribution for the 0-7 PDB in TopSpin



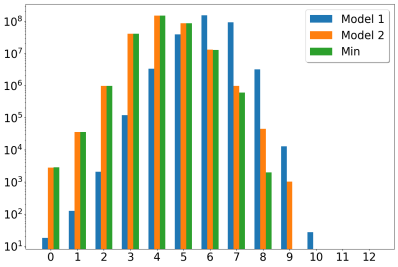
(a) Quantile with Classification



(b) Ensemble with Classification



(c) Affine Transform with Regression



(d) Ensemble with Regression

Figure B.7: Prediction Distribution for the 0-7 PDB in TopSpin