Problem	|State Space|	Approach	Num. Parameters	Solution Quality	Expanded Nodes
48 Tile puzzle	3.00×10^{62}	DeepCubeA	3.00×10^7	253.4	$*5.73 \times 10^6$
24 Tile puzzle	7.70×10^{24}	DeepCubeA	$\uparrow 2.10 \times 10^7$	$\downarrow 89.5$	$\uparrow *2.01 \times 10^6$
		PHS*	1.05×10^6	$\uparrow 224.0$	$\downarrow 2.87 \times 10^3$
		PHS_h	1.05×10^6	119.5	5.86×10^4
		BLH	$\downarrow *3.00 \times 10^4$	-	5.22×10^6
15 Tile puzzle	1.00×10^{13}	DeepCubeA	$\uparrow 1.82 \times 10^7$	52.0	$\uparrow 1.28 * \times 10^6$
		BLH	$\downarrow *3.00 \times 10^4$	-	$\downarrow 1.01 \times 10^4$
Sokoban	$*1.53 \times 10^{15}$	DeepCubeA	$\uparrow 1.50 \times 10^7$	$\downarrow 32.9$	$\downarrow 1.05 \times 10^3$
		PHS*	$\downarrow 3.71 \times 10^6$	37.6	1.52×10^3
		PHS_h	$\downarrow 3.71 \times 10^6$	$\uparrow 39.1$	$\uparrow 2.13 \times 10^3$

Table 1: Comparison of previous approaches: DeepCubeA Agostinelli et al. (2019), Policy Guided Heuristic (PHS_h, PHS*) Orseau & Lelis (2021), Bootstrap Learning Heuristic (BLH) (Arfaee et al., 2010). '*' in front of a value denotes, some approximation based on additional assumptions. '-' denotes unreported values in the original paper. '↑' in front of the value denotes the worst value, and '↓' denotes the best value for the domain. '|State Space|' denotes the size of state space.

A Summary of Previous Results

Table 1 shows a rough comparison between the results of previous approaches. For each domain ('problem') we report the size of its state space and for each approach, the number of parameters used to fit the underlying heuristic estimator, the resulting solution quality (cost), and its running complexity (number of expanded states).

For the number of parameters of BLH, we calculate a rough estimate assuming a 3 layer neural network with 1000 neurons in each layer. DeepCubeA, reports the number of generated nodes instead of the number of expanded nodes, which we approximate as *generated_nodes / branching_factor*. PHS* and PHS_h are two variants as used in Orseau & Lelis (2021). For the 10×10 Sokoban grid with 4 boxes, the size of state space is approximately calculated as $100 \times \binom{100}{4} \times \binom{100}{4}$ where 100 are possible player locations $\binom{100}{4}$ are box locations as well as box targets. We disregard parameters added due to any batch normalization layer. We note that the change in the number of parameters for a given approach results from varying the number of input features for the instance size, and not from changes to the hidden layers' layout. An exception to this is DeepCubeA, which uses 6 residual blocks He et al. (2016) for *Tile puzzle* and 4 for *Rubik's Cube* and *Sokoban*. The results suggest a trend where scaling the instance size necessitates a larger approximator (w.r.t the number of parameters). If the approximator's size is not increased, then we observe reduced accuracy ('Solution Quality') and/or increased computational complexity ('Expanded Nodes').

B Domain Description

The domains used in our experiments are as follows.

Pancake: (1) *Description:* The pancake sorting problem is an NP-H problem Bulteau et al. (2015), where the task is to sort pancakes stacked one on top of another in the minimum number of steps. A step allows inserting a spatula at any position in the stack and flipping (inverting) all the pancakes above it. (2) *Dataset Generation:* Following Agostinelli et al. (2019), we generate the training data by taking a random walk from the goal state. To calculate the h^* values (or labels), we solve the problem with A* search and an admissible heuristic function known as the gap heuristic Helmert (2010). (3) *Encoding:* The state encoding, $\phi(s)$ was defined through a one-hot encoding of each pancake location in the stack. An example of a pancake problem along with state representation can be seen in Figure 2 (a). (5) *Instance size:* Instance size of n is a pancake problem with n pancakes.

Travelling Salesman Problem (TSP): (1) *Description:* TSP is an NP-H problem Cormen et al. (2009) where, given a graph of cities, the task is to find the shortest route (sum of the cost of edges) that visits every city (vertex in the graph) exactly once while returning to the start city. (2) *Dataset Generation:* We

generate complete weighted directed graphs with edges uniformly sampled from numbers in the range [0.1, 5] with 0.1 granularity. The start node is randomly picked. Note that, for the experiments, we consider only initial states, that is, no city is traveled, but use different graphs with different start cities. To calculate h^*, we use the Held–Karp algorithm Held & Karp (1962) as the solver. (3) *Encoding:* The state encoding, $\phi(s)$ was defined by a 1-dimensional representation of the adjacency matrix of the graph concatenated with an array of length n denoting which nodes are visited and the start node. An example of TSP problem along with state representation can be seen in Figure 2 (b). (5) *Instance Size:* Instance size of n is a TSP problem with n distinct cities.

Blocks World: (1) *Description:* The blocks world problem is an NP-H problem Gupta & Nau (1992) that consists of a number of blocks stacked into towers Slaney & Thiébaux (2001), where the task is to turn a given start state to a given goal state with a minimum number of steps. We suspect that it remains NP-H with the fixed goal configuration that we use. A step allows moving one block from the top of a tower onto the top of another one or to the table. (2) *Dataset Generation*: We fix a goal state and generate instances with uniform random start states. To calculate the h^* values, we solve the instances with A* using the number of blocks that are in the wrong positions as heuristic. We use a goal state where all blocks are stacked on each other in order with the lowest numbered block at the bottom. (3) *Encoding*: We use the same state encoding, $\phi(s)$, as used in Slaney & Thiébaux (2001). An example of blocks world problem along with the state encoding can be seen in Figure 2 (c). (5) *Instance Size:* Instance size of n is a blocks world problem with n blocks.

C Varying the Dataset Size

In this section, we study the number of parameters required to fit the dataset when the size of the dataset grows with the instance size for the pancake problem. Specifically, we collect a dataset of size equals to 0.02% of the entire state space for the instance size. These sets of experiments are limited only to the pancake problem for two reasons: (1) the size of the dataset grows exponentially (2) optimal solvers take longer to solve larger instances. The time required to attain the dataset gets worse (in months), especially with the increasing size of the dataset. The gap heuristic for the pancake problem is relatively good (results in better performance) (Helmert, 2010), thus we present a scalability study for the pancake problem. Similar to the previous results, the number of parameters required to fit the dataset with increasing instance size grows quickly.

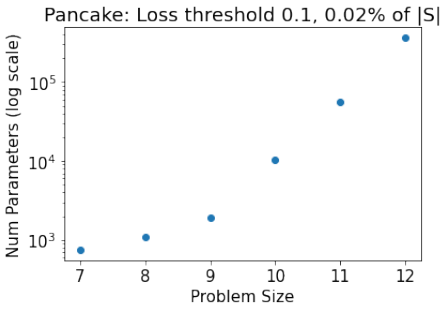

Figure 8: Increase in the minimum number of parameters (log scale) required to fit problems with increasing instance sizes and varying dataset sizes for the Pancake problem. On the x-axis, we have the instance sizes. On the y-axis, we have the number of parameters on a log scale. The size of the dataset used is 0.02% of the entire state space for each instance size.