Near Optimal Solving of the (N^2–1)-Puzzle Using Heuristics Based on Artificial Neural Networks

Vojtěch Cahlík and Pavel Surynek[0000−0001−7200−0542]

Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00 Praha 6, Czechia
mail@cahlikvojtech.com, pavel.surynek@fit.cvut.cz

Abstract. We address the design of heuristics for near-optimal solving of the (N^2–1)-puzzle using the A* search algorithm in this paper. The A* search algorithm explores configurations of the puzzle in the order determined by a heuristic that tries to estimate the minimum number of moves needed to reach the goal from the given configuration. To guarantee finding an optimal solution, the A* algorithm requires heuristics that estimate the number of moves from below. Common heuristics for the (N^2–1)-puzzle often underestimate the true number of moves greatly in order to meet the admissibility requirement. The worse the estimation is the more configurations the search algorithm needs to explore. We therefore relax from the admissibility requirement and design a novel heuristic that tries estimating the minimum number of moves remaining as precisely as possible while overestimation of the true distance is permitted. Our heuristic called *ANN-distance* is based on a deep *artificial neural network* (ANN). We experimentally show that with a well trained ANN-distance heuristic, whose inputs are just the positions of the tiles, we are able to achieve better accuracy of estimation than with conventional heuristics such as those derived from the Manhattan distance or pattern database heuristics. Though we cannot guarantee admissibility of ANN-distance due to possible overestimation of the true number of moves, an experimental evaluation on random 15-puzzles shows that in most cases the ANN-distance calculates the true minimum distance from the goal or an estimation that is very close to the true distance. Consequently, A* search with the ANN-distance heuristic usually finds an optimal solution or a solution that is very close to the optimum. Moreover, the underlying neural network in ANN-distance consumes much less memory than a comparable pattern database. We also show that a deep artificial neural network can be more powerful than a shallow artificial neural network, and also trained our heuristic to prefer underestimating the optimal solution cost, which pushed the solutions towards better optimality.

Keywords: $(N^2 - 1)$-puzzle, heuristic design, artificial neural networks, deep learning, near optimal solutions

1 Introduction

The (N^2–1)-puzzle [40, 33] due to its challenging difficulty and yet simple formulation became an important benchmark for a variety of problem solving methods and heuristic

search algorithms [4, 15]. The task in the (N^2–1)-puzzle is to rearrange $N^2 - 1$ square tiles on a square board of size $N \times N$ by shifting them horizontally or vertically into a configuration where tiles are ordered from 1 to $N^2 - 1$ (see Figure 1 for an illustration of the 8-puzzle). Tiles can never overlap so there is one blank position on the board that enables one tile at a time to move; that is, a tile can be moved horizontally of vertically to the neighboring blank position in one step.

It is known that finding an optimal solution of the (N^2–1)-puzzle, that is, finding the shortest possible sequence of moves that reaches the goal configuration from the initial one, is an NP-hard problem [25, 26, 5]. Hence the (N^2–1)-puzzle is considered to be a challenging benchmark problem for testing variety of algorithms. On the other hand, the difficulty of the puzzle is not absolutely prohibitive as it belongs to the NP-class which can be seen due to existence of algorithms that can generate feasible sub-optimal solutions of polynomial length ($O(N^6)$ to be precise for the (N^2–1)-puzzle case).

Various approaches have been adopted to address the problem using search-based and other techniques. They include heuristics built on top of pattern databases [8, 10] which are applicable as part of the A* algorithm to obtain optimal solutions. The disadvantage of optimal search-based algorithms is their limited scalability for larger N even with advanced heuristics. So far the best known result is an optimal A*-based algorithm for the 24-puzzle by Korf [16] built on top of sophisticated pattern database heuristics. The 35-puzzle is still waiting for an optimal algorithm that can finish in reasonable time on commodity hardware.

Different approaches are represented by solving the puzzle (N^2–1)-puzzle sub-optimally using rule-based algorithms such as those of Parberry [22] or those using permutation group reasoning [17]. The advantage of these algorithms is that they are fast and can be used in an online mode. However, solutions produced by these algorithms are usually very far from the optimum, which precludes their application in practice. These algorithms are commonly used to test existence of a solution before an optimal algorithm is started.

Various attempts have been made to combine good quality of solutions with fast solving by unifying above streams of research. For example improvements of sub-optimal solutions by using *macro operations*, where instead of moving single tile to its goal position, multiple tiles are taken together to form a so called *snake* that moves as a single entity, were introduced in [39]. Moving tiles together consumes fewer moves than if tiles are moved individually. Another approach is to design better tile rearrangement rules that by themselves lead to shorter solutions as suggested in [23]. In the average case, these rule-based algorithms can generate solutions that are much better that those produced by plain rule-based algorithms [24].

1.1 Contributions

In this paper, we present an attempt to solve the (N^2–1)-puzzle near-optimally or optimally using a heuristic based on *artificial neural networks* (ANNs) [14] called *ANN-distance*. Our heuristic is intended to be integrated into the A* algorithm [13]. Particularly our contributions are as follows:

- We design a near-admissible heuristic to be used in the A* algorithm. Unlike conventional heuristics used in A* that focus on finding optimal solutions and must

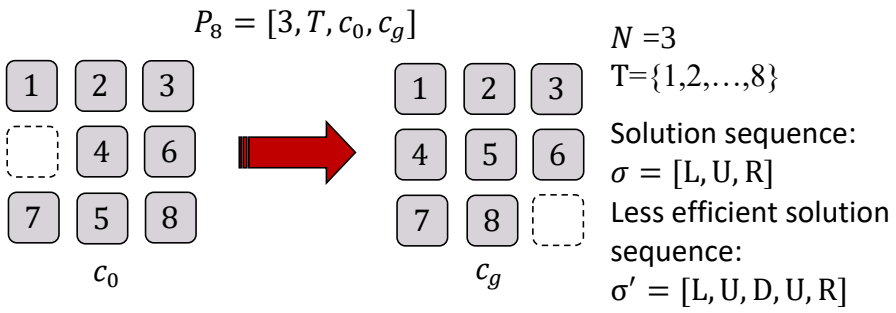

Fig. 1. An illustration of an initial and a goal configuration of the *8-puzzle*. [2]

therefore strictly satisfy the admissibility requirement, our heuristic only tries to make the best possible estimation while admissibility can be occasionally violated (that is, the heuristic overestimates). Specifically we try to directly calculate the estimation of the number of moves remaining to reach the goal configuration using an ANN. This is in contrast to admissible heuristics that try to make estimations from below which often leads to significant underestimations.

- We test the hypothesis that our ANN-distance heuristic does not violate the admissibility requirement often and provides better estimations than heuristics based on pattern databases. That is, we have shown in our experimentation with the 15-puzzle that the A* algorithm with *ANN-distance* produces sub-optimal solutions only rarely. Moreover, estimations made with ANN-distance are closer to the true minimum number of moves required than those obtained from the comparable 7-8 pattern database heuristic [9].

This paper extends our previous work on the topic of ANNs in (N^2–1)-puzzle solving presented in [2]. Specifically we improved the training process of the ANN-distance heuristic so that it generates better results than those presented in [2]. Additionally we present here a significantly extended set of experiments that did not fit in the original conference format.

The paper is organized as follows: we first introduce the (N^2–1)-puzzle formally and put it in the context of related works. Then, we describe the design of the ANN-distance heuristic and the process of its training. Finally, we experimentally evaluate ANN-distance as part of the A* algorithm and compare it with A* using the 7-8 pattern database heuristic. All tests are done on random instances of the 15-puzzle.

2 Background

In this section we will formally define the (N^2–1)-puzzle and introduce artificial neural networks.

2.1 The (N^2–1)-puzzle

The (N^2–1)-puzzle consists of a set of tiles $T = \{1,2,...,N^2-1\}$ of uniform size 1×1 placed in a non-overlapping way on a square board of the size $N \times N$. There are $N \times N$ positions on the board numbered from 1 to N. One position on the board remains empty. A *configuration* (placement) of tiles can be expressed as assignment $c : T \rightarrow \{1,2,...,N^2\}$ for which it holds that $c(i) \neq c(j)$ whenever $i \neq j$.

Definition 1. *The (N^2–1)-puzzle is a quadruple $P_{N^2-1} = [N,T,c_0,c_g]$, where $c_0 : T \rightarrow \{1,2,...,N^2\}$ is an initial configuration of tiles and $c_g : T \rightarrow \{1,2,...,N^2\}$ is a goal configuration (usually an identity with $c_g(t) = t$ for $t = 1,2,...,N^2-1$).*

The movements of tiles are always possible into blank neighboring positions; that is, one move at a time. The task in the (N^2–1)-puzzle is to rearrange tiles from initial configuration c_0 into desired goal configuration c_g using allowed moves. Since there is only one blank position, movements of tiles naturally correspond to: Left, Right, Up, Down movements while it is unambiguous where the movement takes place. The solution sequence can be expressed as $\sigma = [m_1,m_2,...,m_l]$ where $m_i \in \{L,R,U,D\}$ with natural meaning L=Left, R=Right, U=Up, and D=Down for $i = 1,2,...,l$; l denotes the length of solution. We call solution σ optimal if l is as small as possible, otherwise we say that the solution is sub-optimal (illustration of a solution is shown in Figure 1).

2.2 Artificial Neural Networks

In our attempt to design a heuristic that gives very precise estimations we made use of a *feed-forward artificial neural network* (ANN). The ANN consists of multiple computational units called *artificial neurons* that perform simple real-valued computations on their input vectors $x \in \mathbb{R}^n$ as follows: $y(x) = \xi(\sum_{i=0}^{n} w_i x_i)$ where $w \in \mathbb{R}^n$ is vector of weights, $w_0 \in \mathbb{R}$ is a bias and ξ is an *activation function*, for example a sigmoid as follows, where λ determines the shape of the sigmoid:

$$\xi(z) = \frac{1}{1+e^{-\lambda z}} \tag{1}$$

Neurons in an ANN are arranged in layers. Neurons in the first layer represent the input vector. Neurons in the second layer get the outputs of neurons in the first layer as their input, and so on. At every layer neurons are fully interconnected with neurons from the previous layer. Outputs of neurons in the last layer form the output vector. Usually we want an ANN to respond to given inputs in a particular way.

So mathematically the network implements a function $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$ where n is the dimension of input and m is the dimension of output. Our assumption is that the heuristic function determining the minimum number of moves required to reach c_g from some given configuration c can be approximated using the feed forward neural network. Hence after encoding c_g and c as a real-valued vector and giving it at the input of the network we want the network to respond with the minimum number of moves at the output layer.

After deciding a topology of the network, that is how many layers and how many neurons per layer it should consist of, this is achieved through the process of learning

[27, 29]. Learning usually does not alter the topology but only sets the weight vectors w and biases w_0 of individual neurons so that for a given input x^i the network responds with a desired output y^i in the last layer.

The network is trained for a data-set which contains pairs of input x^i and desired output y^i. If the ANN is designed well, that is if it has the proper number of neurons per layer and if the data-set is representative enough, then the ANN can appropriately respond even to inputs that are outside the training data-set. We then say that the ANN generalizes well: this is the goal in our design as well.

However, we are aware of the challenging aspect of the task as it is unrealistic to train the network for all possible configurations c of the (N^2–1)-puzzle, of which there are as many as $N!$ ($N!/2$ solvable ones).

3 Related Work

MAPF Algorithms. The (N^2–1)-puzzle represents a special case of a problem known as *multi-agent path finding* (MAPF) on graphs, where instead of the $N \times N$-board we are given an undirected graph $G = (V, E)$ with so called agents occupying its vertices. The graph models a more general environment in which agents can move across edges. In MAPF, there is at most one agent per vertex, and similarly to the $N \times N$-board we can move an agent into the empty neighboring position (vertex). The task is to move agents so that each agent reaches its unique goal position (vertex). The (N^2–1)-puzzle can be represented as MAPF if we replace each position on the board with a vertex and connect the vertex with vertices corresponding to neighboring positions on the board. That is, we obtain a 4-connected grid of size $N \times N$ with one empty vertex where initial positions and goals of agents are set according to the initial and goal configuration of the puzzle.

Generally all MAPF algorithms rely on the fact that the environment is typically not fully occupied by agents in MAPF - they are often adaptive with respect to the density of agents. In sparsely occupied cases MAPF algorithms typically do not consider all possible interactions between agents and tend to plan individual paths as independently as possible. Hence these algorithms cannot benefit from their advantages in case of the (N^2–1)-puzzle as there is only one empty position.

Search Algorithms in Transformed Spaces. ICTS [31, 32] and CBS [30, 1] on the other hand view the search space differently, not as a graph of states. ICTS searches through various distributions of costs among multiple individual agents while the high level mechanism iterates the overall cost from the lower bound upwards until a solvable cost is reached. The CBS algorithm searches the tree of conflicts between agents and resolutions of these conflicts. Initially CBS searches for individual paths as if other agents are not present in the instance. Paths found in this way are then verified with respect to collisions between agents. If there are collisions constraints to avoid them are introduced into the individual path search process and the high level is branched as each collision can be usually resolved by two cases - the first or the second agent avoids the place and time of the collision.

Efficient Rule-based Algorithms. Besides search-based algorithms there are also polynomial-time rule-based algorithms like BIBOX [35, 36] and Push-and-Swap [19].

They use predefined macro-operations to reach the goal configuration. Solutions generated by rule-based algorithms are sub-optimal. As these algorithms are designed for practical use, solutions generated by them are closer to the optimum than in the case of permutation group-based algorithms.

Compilations-based Algorithms. Important group of optimal solving techniques for MAPF and consequently for the (N^2–1)-puzzle is represented by compilation-based algorithms. These algorithms reduce the task of finding solution to MAPF to some of the existing formalism such as *propositional satisfiability* (SAT) [37, 38], *constraint satisfaction* (CSP) [18] or *answer set programming* (ASP) [7]. This approach is characterized by employing efficient off-the-shelf solvers for the target formalism, an efficiency one can hardly achieve by own dedicated solver. Moreover, any progress of the target formalism solver is immediately translated into the progress of solving the original problem.

A*-based Search Algorithms. There are currently many solving algorithms for MAPF and consequently for the (N^2–1)-puzzle. Optimal A*-based algorithms include *independence detection - operator decomposition* ID-OD [34] that tries to make use of not fully occupied graph by dividing the set of agents into multiple independent groups that are solved separately. This is in the case of algorithms of exponential time complexity useful technique as it divides the exponent by the number of independent groups.

As for the combination of state-space search and neural networks, Samadi et al. [21] successfully used an ANN-based heuristic function to estimate optimal solution costs of the 15-puzzle. They used the estimates of several pattern database heuristics as inputs to their neural network. They used a custom error function in order to penalize overestimation, biasing the heuristic's estimates towards admissibility. They used the RBFS search algorithm. The unbiased ANN heuristic resulted in search trees which were smaller by a factor of 10 than those generated with the 7-8 pattern database heuristic, and the generated solutions were inadmissible by about 4 %. Using a biased ANN heuristic further pushed inadmissibility to only 0.1 %, but increased the size of the generated search trees. In [20] by Ernandes and Gori, the authors used an artificial neural network to estimate the optimal solution costs of the 15-puzzle, using only the positions of individual tiles as input features. They used a very small network with a single hidden layer of only 15 neurons. IDA* with their heuristic then achieved an optimal solution in about 50 % of cases, and compared to the Manhattan distance heuristic, time cost was reduced by a factor of about 500.

Arfaee et al. [28] used a *bootstrapping* procedure which allowed them to eventually solve random 24-puzzle instances even without starting with a sufficiently strong heuristic. They started with an untrained artificial neural network, which they used as a heuristic to solve a number of 24-puzzle instances. Even though this procedure failed to solve most of the instances within time limit, they used the handful of solved instances to train the next iteration of the neural heuristic. Repeating this procedure several times resulted in obtaining a very powerful ANN-based heuristic, which was able to solve the majority of the input set of random 24-puzzle instances. Suboptimality of the solutions was about 7 %.

4 Designing a New Heuristic

The ANN-distance heuristic will be integrated as the underlying heuristic of the A* graph search algorithm. Let us first describe basics of the A* algorithm. A* explores the space of possible configurations of the puzzle. It maintains the OPEN list in which it stores candidate configurations for further exploration. The algorithm always chooses configuration c from OPEN with the minimum $g(c)+h(c)$ for the next expansion, where $g(c)$ is the number of steps taken to reach c from c_0, and $h(c)$ is the estimation of the number of remaining steps from c to c_g. It is generally true that with an admissible heuristic function, the closer h is (from below) to the true number of steps remaining, the fewer configurations the algorithm will expand. With an inadmissible heuristic function, the situation is more complicated. Experiments have shown that the weighted A* algorithm, in which the heuristic estimates are scaled by a weight $W > 1$ (and are thus inadmissible), reaches the solution faster with greater values of W, but the obtained solutions are typically further and further from the optimum. However, it is reasonable to expect that if the estimates produced by the ANN-distance heuristic are relatively accurate and do not overestimate the optimal solution cost too often, the solutions found by A* will not be too far from the optimum. Some of the ANN-distance heuristics will therefore be designed to produce as accurate estimates of the optimal solution costs as possible. Other ANN-distance heuristics will be trained to underestimate the optimal solution costs, which can be expected to result in solutions whose cost will be closer to the optimum.

4.1 Encoding the Input and Output

The input to the underlying fully-connected feed-forward ANN of the ANN-distance heuristic are the one-hot encoded positions of the tiles, including the blank. In order to one-hot encode the tile positions of a 15-puzzle instance, a 256-dimensional vector is required. This vector is the input to ANN. The output of the network is a single neuron outputting the estimated optimal solution cost.

4.2 Design of the Neural Networks

The hyperparameters of the ANN, most importantly the layer sizes, were calculated using *grid search* [12]. The most powerful architecture turned out to be a deep ANN with 5 hidden layers composed of 1024, 1024, 512, 128 and 64 hidden neurons. This is a relatively large network, but the 15-puzzle is a complex problem domain, with some of the configurations being deceptive — there are for example states with only a few tiles being out of their place, yet with a high optimal solution cost. The ANN must be flexible enough to learn to recognize these quirks.

Several state-of-the-art techniques for training deep ANNs were used. These techniques include the Adam optimizer, ELU activation function, dropout regularization, and batch normalization [11]. Layer weights were initialized with He initialization [11]. The output neuron used no activation function. Learning rate was set experimentally.

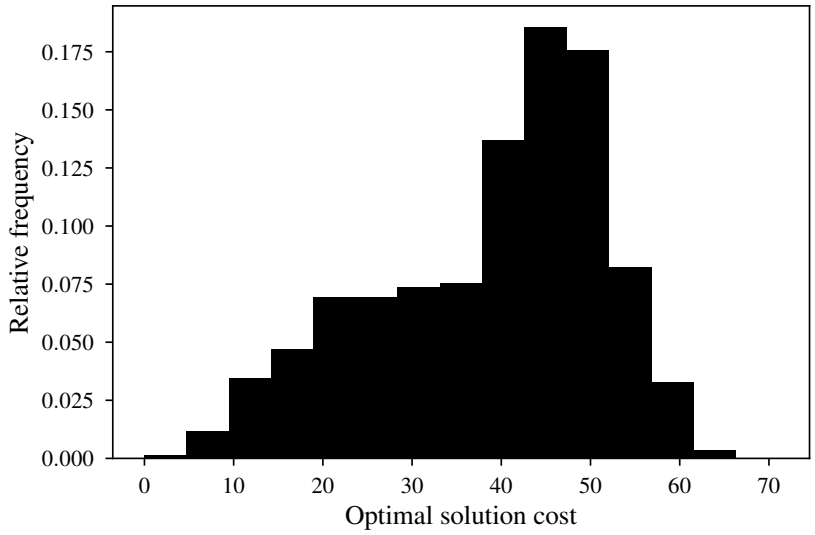

Fig. 2. The distribution of the optimal solution lengths of boards in the training dataset

4.3 Training Data and Training

The training dataset was obtained by randomly shuffling the boards, using both high and low shuffle counts. This resulted in some of the boards having high optimal solution cost of about 50, but with some of the boards still having a low optimal solution cost. This is preferable, as the ANN-distance heuristic must correctly estimate optimal solution costs when the search algorithm is both far away and close to the solution. However, preliminary tests with the pattern database (PDB) heuristic revealed that the A* algorithm spends most of the time far away from the solution. Hence we took the assumption that queries to the ANN-distance heuristic will follow a similar distribution of distances from the optimum, and we selected instances into the training dataset that are mostly hard. In total, there are 4.8 million instances in the training dataset, which is a relatively large number, but still just a $\frac{1}{10^7}$ of the total size of the state-space of the 15-puzzle. The optimal solution costs were calculated with the IDA* algorithm with the underlying PDB 7-8 heuristic. The distribution of boards in the training dataset can be seen in figure 2.

The cost function used while training some networks was mean squared error (MSE). However, for some networks, a novel cost function, which we named *asymmetric mean squared error* (AMSE), was used. This function is basically the MSE cost function skewed to one side. The AMSE function is defined as

$$\mathrm{AMSE}(\hat{Y}, Y) = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2(\mathrm{sgn}(\hat{Y}_i - Y_i) + \alpha)^2, \tag{2}$$

where n is the number of instances, $\hat{Y}$ is a vector of the estimated values, Y is a vector of the target values, and α is a parameter between 0 and 1, which controls the skew of

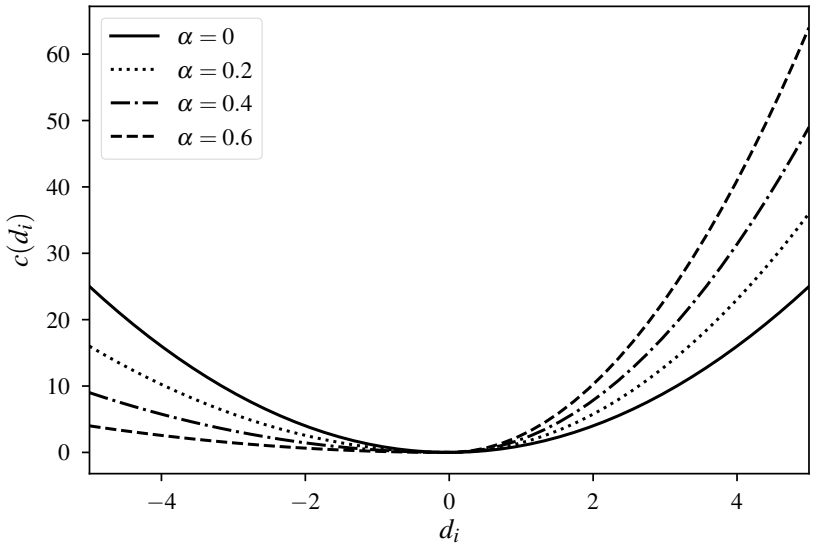

Fig. 3. The AMSE cost of a single instance for different deviations from the optimal solution cost and various values of α

the function. The closer α is to 1, the more the overestimations are penalized. AMSE with $\alpha = 0$ is equal to the mean squared error cost function. The behavior of AMSE cost denoted as $c(d_i)$ for different estimation errors d_i and various values of α is shown in figure 3.

Training was performed by *gradient descent using reverse-mode autodiff*, a process historically known as *backpropagation* [11, 6]. The Adam optimizer was used. Training was run for 40 epochs, which took about 3 hours [1]. During training, the training cost kept decreasing steadily, but the cost on the validation set kept fluctuating, so the network's weights were saved after each epoch and at the end of training, and only the weights corresponding to the lowest validation error were restored and saved as the final model.

4.4 Resulting ANN-distance heuristics

Several versions of the ANN-distance heuristic were chosen for further experimental evaluation. This included the ANN-distance heuristic with a deep underlying ANN trained with the MSE cost function (with the layer sizes stated in section 4.2), then another heuristics with the underlying deep ANN trained with the asymmetric AMSE cost function with α of 0.4 and 0.8, and a final heuristic with a shallow ANN composed

[1] All experiments were run on an eight-core Intel Xeon E5-2623 v4 CPU with 30 GB of RAM and a NVIDIA Quadro P4000 graphics card under Debian Linux. The ANN was implemented and trained using the Keras library [3] with the TensorFlow backend.

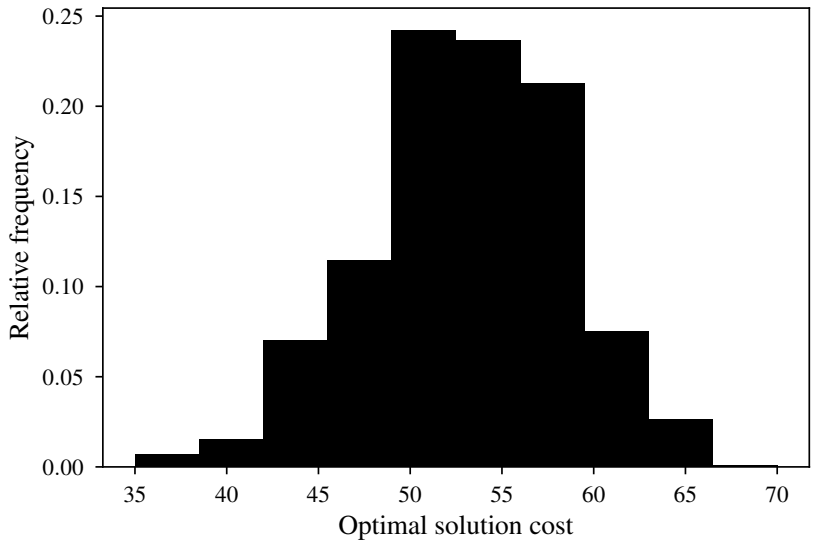

Fig. 4. Histogram of optimal solution costs of boards in the evaluation dataset

of a single hidden layer of 2752 neurons, which is the same total number of neurons as that of the deep ANNs.

5 Experimental Evaluation

Our experimental evaluation was focused on competitive comparison of the ANN-distance heuristics against the admissible PDB 7-8 heuristic (the pattern database heuristic with pattern sizes 7 and 8) and its weighted variants, which are inadmissible.

The tests were run on a set of 1172 instances obtained as random permutations. The distribution is shown in figure 4, the mean optimal solution cost of boards in the evaluation dataset is 52.8. We focused on measuring the performance of the heuristics in two scenarios:

1. when they are used to obtain single estimations
2. when they are used as the underlying heuristic of the A* algorithm

5.1 Evaluation on single estimations

The first experiment tested how closely the true distance from the goal configuration has been estimated by the PDB heuristics and by the ANN-distance heuristics. The resulting distributions of estimations are shown in figures 5 and 6. It can be seen that the shapes of the distributions are similar to the shape of the distribution of the true optimal solution costs of boards in the evaluation dataset. The biggest difference over the evaluation dataset is that the distributions are shifted to more positive or negative values.

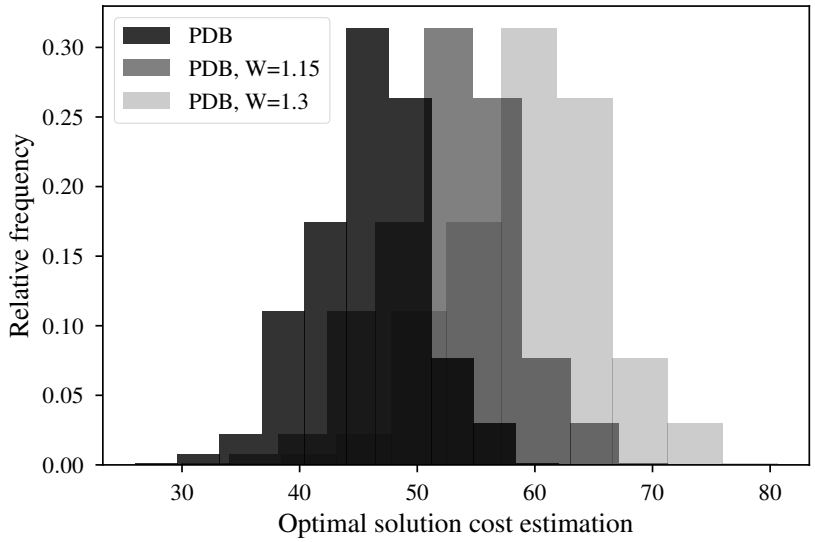

Fig. 5. PDB 7-8 heuristics: histogram of single estimations

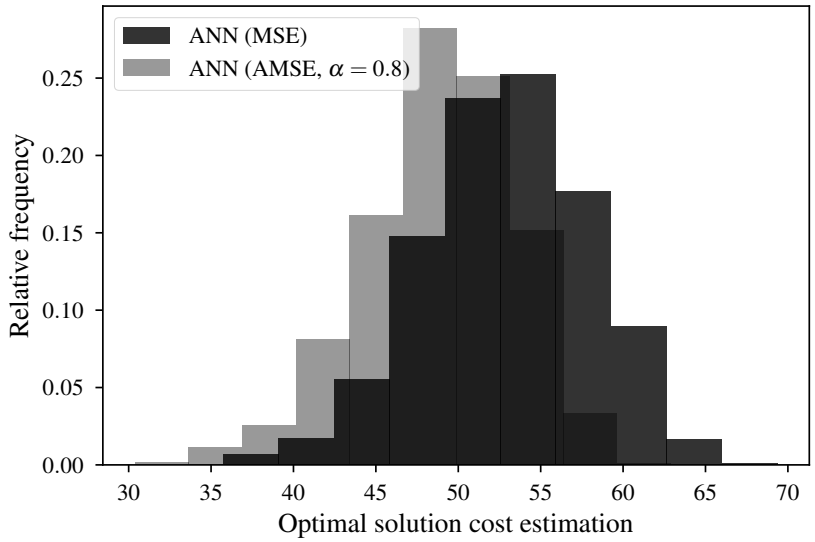

Fig. 6. ANN-distance heuristics: histogram of single estimations

The PDB 7-8 heuristic produces smaller estimates than its weighted counterparts, and the distributions of the estimations of the weighted PDB 7-8 heuristics are also spread more widely than the distribution of the optimal solution cost estimates produced by

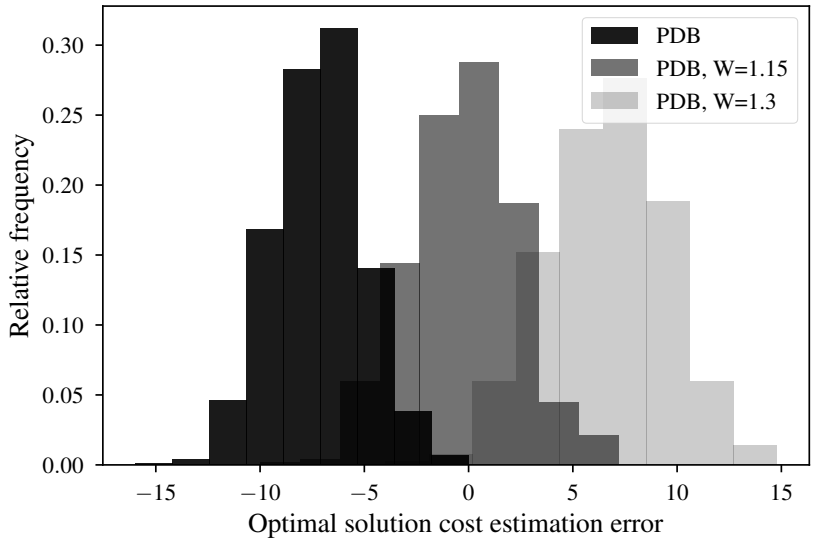

Fig. 7. PDB 7-8 heuristics: histogram of errors on single estimations

the original PDB 7-8 heuristic. The ANN-distance heuristic produces optimal solution cost estimates whose distribution is quite similar to the true distribution of the optimal solution costs, with the mean of 52.9. The ANN-distance heuristic that was trained to underestimate its estimations using an asymmetric cost function with $\alpha = 0.8$ produces estimates which are somewhat lower than those produced by the original ANN-distance heuristic trained with the mean squared error cost function, with the mean of 48.9.

The distributions of estimation errors with respect to the true optimal solution costs are shown in figures 7 and 8. Clearly, both of the ANN-distance heuristics are inadmissible according to the test. The ANN-distance heuristic trained with the MSE cost function has the error's mean of 0.06 and the error's standard deviation of 1.42, which means that its estimations are relatively accurate. The ANN-distance heuristic trained with the AMSE cost function with $\alpha = 0.8$ tends to underestimate the cost for almost all estimations, with the mean error of -3.9, but is still inadmissible. On the other hand, the PDB 7-8 heuristic tends to strongly underestimate the optimal solution costs by default, and is more accurate only after being weighted by $W = 1.15$.

5.2 Evaluation on A* searches

In the second experiment, the heuristics were evaluated as the underlying heuristics of the A* algorithm. No time limit was imposed on the runs of the A* algorithm, and thus the solutions to all boards with all heuristics were found. The results are presented in table 1. The meaning of the columns is as follows: the first column states the heuristic used, the second column represents the average cost of the solutions found, the third column states the average number of nodes expanded by the searches, the fourth column

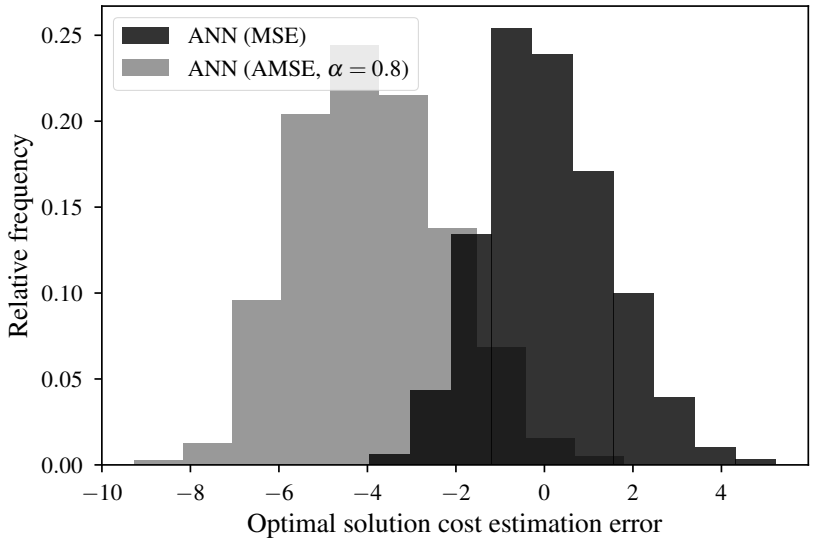

Fig. 8. ANN-distance heuristics: histogram of errors on single estimations

represents the average runtime of the search, the fifth column states what percentage of solutions found were optimal. The sixth column represents the average suboptimality of solutions, measured in percentages. Average suboptimality is defined as the average cost of the solutions, divided by the average optimal solution cost (and after this computation, 1 is subtracted and the result is multiplied by 100 to obtain percentages). The last column shows the average number of expanded nodes per second. The rows simply represent the results of the heuristics: the first four rows represent the pattern database heuristic and its weighted variants, the fifth row represents the deep ANN-distance heuristic trained with the MSE cost function, the sixth row is the shallow ANN trained with the MSE cost function, and the last two rows are the deep ANN-distance heuristics trained with the AMSE cost function.

Heuristic	Cost	Expanded	Time	%Opt	%Avg. subopt.	Expanded/s.
PDB 7-8	52.8	62,222	6.946	100	0	8958
PDB 7-8, W=1.15	53.4	3,296	0.346	71	1.19	9526
PDB 7-8, W=1.3	55.2	1,017	0.107	38	4.46	9505
PDB 7-8, W=1.45	57.5	540	0.056	25	8.86	9643
ANN	53.7	355	2.405	61	1.62	148
ANN, 1 h.l.	53.7	1,577	5.988	60	1.69	263
ANN, AMSE (α=0.4)	53.5	532	3.543	67	1.34	150
ANN, AMSE (α=0.8)	53.5	2,576	16.812	69	1.23	153

Table 1. Performance analysis of A* running with various heuristics

Heuristic	Opt.	Opt.+2	Opt.+4	Opt.+6
PDB 7-8	1172	0	0	0
PDB 7-8, W=1.15	834	309	29	0
ANN	718	409	44	1
ANN, 1 h.l.	701	421	49	1
ANN, AMSE ($\alpha = 0.4$)	786	357	29	0
ANN, AMSE ($\alpha = 0.8$)	814	336	21	1

Table 2. Suboptimality of the solutions found by A*

From the table, it is clear that in terms of the number of expanded nodes, the deep ANN-distance heuristic trained with the MSE cost function performs significantly better than all of the heuristics based on pattern databases. The number of expanded nodes has been decreased by two orders of magnitude compared to the unweighted PDB heuristic. However the success rate of finding the optimal solutions is also relatively low, with the deep ANN-distance heuristic leading to an optimal solution only in about 60 percent of cases. Exact number of unsuccessful searches for the optimum depending on the distance from the optimum can be found in table 2.

The suboptimality of the solutions was further decreased by the ANN-distance heuristics trained with the AMSE cost function. However, even though the heuristic trained with the AMSE cost function with $\alpha = 0.8$ underestimates almost all of its estimations, it still lead to suboptimal solutions being found by the A* search algorithm in about 30 % of cases. It is possible that this suboptimality mostly comes from the inconsistency of the heuristic, as the graph version of the A* algorithm only guarantees optimality when the heuristic function used is consistent.

An interesting result is that the ANN-distance function based on a shallow ANN lead to a significantly higher amount of expanded nodes than the deep ANN-distance heuristic. Even during training, the prediction cost on the training set was larger with the shallow ANN than with the deep ANN. This suggests that the use of deep learning can be beneficial even in the field of heuristic design.

The biggest weakness of the ANN-distance heuristic is the speed of inference, as the pattern database heuristics are faster at making estimations by almost two orders of magnitude. The shallow ANN's estimations are slightly faster than those of the deep ANNs. On the other hand, the underlying neural networks in the ANN-distance heuristics only take up about 20 MBs of memory, which is a very low number compared to the PDB 7-8 heuristics, which take up about 4.5 GBs of memory.

The distributions of the number of nodes expanded by the A* searches with the deep unbiased ANN-distance heuristic and the unweighted PDB 7-8 heuristic can be seen in figure 9.

5.3 Competitive comparison against heuristics presented in other studies

The ANN-distance heuristic functions presented in this thesis can be directly compared to neural heuritics presented in two earlier studies, both of which were mention in section 3. The first paper by Ernandes and Gori from 2004 [20] created a heuristic function

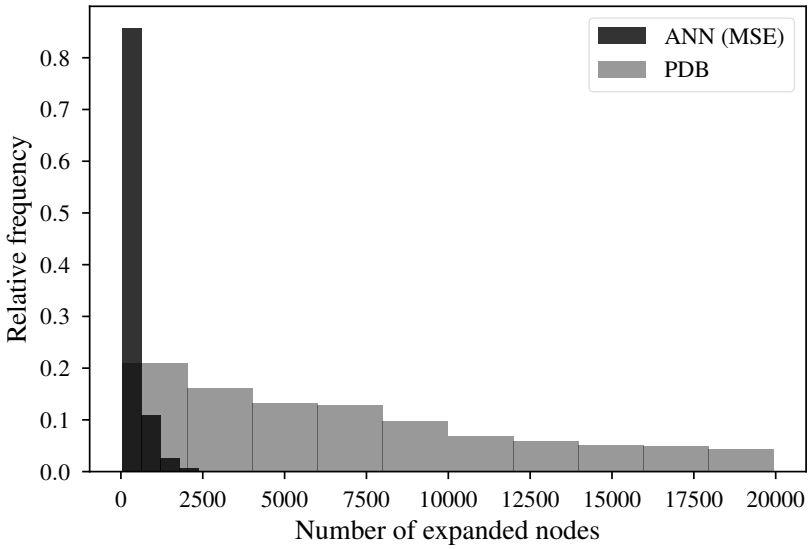

Fig. 9. Histogram of number of nodes expanded during A* search (trimmed)

based on an artificial neural network, which was then evaluated using the IDA* algorithm. The second paper by Samadi et al. from 2008 [21] also designed a heuristic function based on an ANN, but trained it using outputs of other heuristic functions as features, namely the outputs of the PDB 7-8 heuristic and the Manhattan distance heuristic. Samadi et al. evaluated their heuristic using the RBFS search algorithm. Neither of these studies presented the performance of their network on single estimations, and the algorithms used differ, so comparison against these heuristics is difficult. Nevertheless, the results are presented in table 3.

It is evident that the number of expanded nodes is significantly smaller with the unbiased ANN-distance heuristic presented in this paper than with the heuristics of Samadi et al. and Ernandes and Gori. Our ANN distance expands by two order of magnitudes fewer nodes than previous ANN-based heuristics. It is however likely that at least the heuristic presented by Ernandes and Gori would perform better in number of expanded nodes if ran as part of the A* algorithm, since IDA*, which they used, tends to run the low-level multiple times and the low-level revisits large subtrees, and thus the IDA* algorithm expands more nodes than necessary.

As for optimality, Ernandes and Gori stated that 28.7 % of their solutions were optimal, which is far worse than roughly 60 % solutions obtained by the unbiased ANN-distance heuristic presented in this paper. However, Samadi et al. stated that suboptimality of their biased heuristic is less than 0.1 %, which is significantly lower than the suboptimality of 1.23 % of the biased ANN-distance heuristic trained with the AMSE cost function with $\alpha = 0.8$. Unfortunately, it is not clear whether Samadi et al. used exactly the same definition of suboptimality.

Study	Algorithm	Heuristic	Avg. cost	Avg. nodes	Avg. time (s)
This	A*	ANN	53.7	355	2.405
This	A*	ANN, AMSE ($\alpha = 0.4$)	53.5	532	3.543
This	A*	ANN, AMSE ($\alpha = 0.8$)	53.5	2,576	16.812
[20]	IDA*	ANN	54.5	24,711	7.38
[21]	RBFS	ANN (MD+PDB)	54.3	2,241	0.001
[21]	RBFS	ANN (MD+PDB, asym. cost)	52.6	16,654	0.021

Table 3. Comparison of search algorithms against heuristics presented in other papers

The runtime comparison is included for completeness, but is not very informative when compared over different studies, as for example Samadi et al., whose algorithm only takes a millisecond to find a solution, had likely well optimized their code.

5.4 Analysis of the behavior of A* search with the underlying ANN-distance heuristic

When evaluating a heuristic function, it is important to know what parts of the state-space the algorithm spends the most time in. This analysis can be performed by calculating the optimal solution cost belonging to each expanded state. The distribution of the optimal solution costs belonging to nodes expanded by the A* algorithm can be seen in figure 10. The data were obtained by running an IDA* search with the PDB

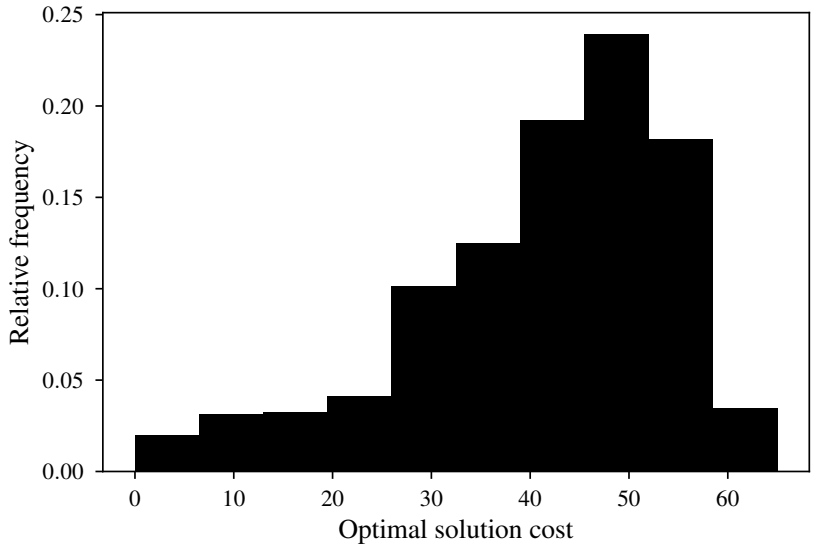

Fig. 10. Histogram of optimal solution costs of nodes expanded during A* search with the ANN-distance heuristic trained with the MSE cost function

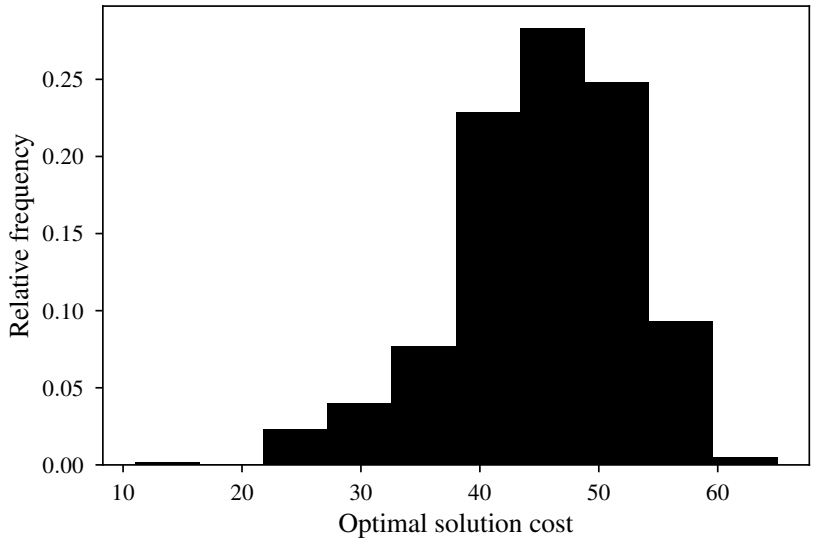

Fig. 11. Histogram of optimal solution costs of nodes expanded during A* search with the PDB 7-8 heuristic

7-8 heuristic to get the optimal solution cost, for each state the A* search with the deep ANN-distance heuristic (trained with the MSE cost function) expanded, for boards obtained as random permutations.

It can be also observed that the distribution is much more skewed to the left than the distribution of optimal solution costs of states expanded by the A* algorithm with the underlying PDB 7-8 heuristic, which is presented in figure 11. This could mean that the heuristic estimations are very accurate for instances of high optimal solution costs, so the search algorithm with the ANN-distance gets faster from the initial state than with the PDB 7-8 heuristic.

6 Discussion and Conclusion

This paper refers on the use of artificial neural networks as the underlying paradigm for the design of heuristics for the (N^2–1)-puzzle. We designed a heuristic called ANN-distance with an underlying deep artificial neural network that estimates for a given configuration the distance in terms of the minimal number of moves required to reach the goal configuration. Although the ANN-distance heuristic is not admissible, it is relatively accurate and does not significantly overestimate the true distance. As a result, the ANN-distance usually yields an optimal solution when used as a part of A*. Moreover, since ANN-distance is relatively precise in its estimations, A* with the ANN-distance heuristic expands much fewer nodes than with other heuristics like 7-8 pattern database.

When compared to a shallow artificial neural network with a single hidden layer, the ANN-distance with an underlying deep ANN gives more accurate estimations and

therefore leads to lower number of nodes expanded by the A* search. This demonstrates that deep learning can be successfully applied to heuristic design in symbolic domains like the (N^2–1)-puzzle. We also tried to bias the ANN-distance heuristic to prefer underestimating its estimations, which resulted in a solutions that are closer to the optimum.

When compared to the results of other scientific papers, the ANN-distance heuristic performs significantly better in the (N^2–1)-puzzle domain than any other similar previously designed heuristic. Our ANN-distance improved the number of expanded nodes over the best published heuristic functions based on artificial neural networks by an order of magnitude. It seems that the cause of the performance gain was a combination of a much larger artificial neural network and training dataset, as well as the use of deep learning and other modern techniques for ANN training.

An important advantage of the ANN-distance is that it consumes much less memory than a comparable pattern database heuristic. This feature enables the use of similar ANN-based heuristics even in cheap embedded systems.

ANN-based heuristics similar to our ANN-distance could be used in other problem domains similar to the (N^2–1)-puzzle. Our hypothesis is that it could be possible to use these heuristics even in problem domains which are mostly *opaque*, that is, problem domains whose inner structure is not well understood by humans.

As we continue our work, we plan to move to the domain of the 24-puzzle. As it will be much more challenging to obtain a sufficient number of training instances, transfer learning could be used to utilize the ANN-distance heuristic trained on the 15-puzzle domain. Such a heuristic function could then be improved with the bootstrapping algorithm [28]. We also plan to perform further experiments with the distribution of the boards in the training dataset.

Acknowledgements

This research has been supported by GAČR - the Czech Science Foundation, grant registration number 19-17966S. We would like to thank anonymous reviewers for their valuable comments.

References

1. Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., Shimony, S.: ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In: IJCAI. pp. 740–746 (2015)
2. Cahlík, V., Surynek, P.: On the design of a heuristic based on artificial neural networks for the near optimal solving of the (n2-1)-puzzle. In: Proceedings of the 11th International Joint Conference on Computational Intelligence, IJCCI 2019, Vienna, Austria, September 17-19, 2019. pp. 473–478 (2019)
3. Chollet, F., et al.: Keras. `https://keras.io` (2015)
4. Culberson, J.C., Schaeffer, J.: Efficiently searching the 15-puzzle. Tech. rep., Department of Computer Science, University of Alberta (1994)
5. Demaine, E.D., Rudoy, M.: A simple proof that the (n2-1)-puzzle is hard. Theor. Comput. Sci. **732**, 80–84 (2018)

6. E. Rumelhart, G. Hinton, R.W.: Learning internal representations by error propagation (1985)
7. Erdem, E., Kisa, D.G., Öztok, U., Schüller, P.: A general formal framework for pathfinding problems with multiple agents. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA (2013)
8. Felner, A., Adler, A.: Solving the 24-puzzle with instance dependent pattern databases. In: SARA-05. pp. 248–260 (2005)
9. Felner, A., Korf, R.E., Hanan, S.: Additive pattern database heuristics. Journal of Artificial Intelligence Research **22**, 279–318 (2004)
10. Felner, A., Korf, R.E., Meshulam, R., Holte, R.C.: Compressed pattern databases. J. Artif. Intell. Res. **30**, 213–247 (2007)
11. Geron, A.: Hands-On Machine Learning with Scikit-Learn and TensorFlow, pp. 275–312. First edn. (2017)
12. Ghawi, R., Pfeffer, J.: Efficient hyperparameter tuning with grid search for text categorization using knn approach with BM25 similarity. Open Computer Science **9**(1), 160–180 (2019)
13. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **SSC-4(2)**, 100–107 (1968)
14. Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice Hall (1999)
15. Korf, R.E.: Sliding-tile puzzles and Rubik's Cube in AI research. IEEE Intelligent Systems **14**, 8–12 (November 1999)
16. Korf, R.E., Taylor, L.A.: Finding optimal solutions to the twenty-four puzzle. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2. pp. 1202–1207 (1996)
17. Kornhauser, D., Miller, G.L., Spirakis, P.G.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: FOCS, 1984. pp. 241–250 (1984)
18. Li, J., Harabor, D., Stuckey, P.J., Ma, H., Koenig, S.: Symmetry-breaking constraints for grid-based multi-agent path finding. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. pp. 6087–6095 (2019)
19. Luna, R., Bekris, K.: Efficient and complete centralized multi-robot path planning. In: IROS. pp. 3268–3275 (2011)
20. M. Ernandes, M.G.: Likely-admissible and sub-symbolic heuristics (2004)
21. M. Samadi, A. Felner, J.S.: Learning from multiple heuristics (2008)
22. Parberry, I.: A real-time algorithm for the (n^2-1)-puzzle. Inf. Process. Lett. **56**(1), 23–28 (1995)
23. Parberry, I.: Memory-efficient method for fast computation of short 15-puzzle solutions. IEEE Trans. Comput. Intellig. and AI in Games **7**(2), 200–203 (2015)
24. Parberry, I.: Solving the (n2 - 1)-puzzle with 8/3 n3 expected moves. Algorithms **8**(3), 459–465 (2015)
25. Ratner, D., Warmuth, M.K.: Finding a shortest solution for the N x N extension of the 15-puzzle is intractable. In: AAAI. pp. 168–172 (1986)
26. Ratner, D., Warmuth, M.K.: Nxn puzzle and related relocation problem. J. Symb. Comput. **10**(2), 111–138 (1990)
27. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**, 533– (Oct 1986)
28. S. Arfaee, S. Zilles, R.H.: Learning heuristic functions for large state spaces (2011)

29. Schmidhuber, J.: Deep learning in neural networks: An overview. CoRR **abs/1404.7828** (2014), http://arxiv.org/abs/1404.7828
30. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. Artif. Intell. **219**, 40–66 (2015)
31. Sharon, G., Stern, R., Goldenberg, M., Felner, A.: The increasing cost tree search for optimal multi-agent pathfinding. Artif. Intell. **195**, 470–495 (2013)
32. Sharon, G., Stern, R., Goldenberg, M., Felner, A.: Pruning techniques for the increasing cost tree search for optimal multi-agent pathfinding. In: Symposium on Combinatorial Search (SOCS) (2011)
33. Slocum, J., Sonneveld, D.: The 15 Puzzle. Slocum Puzzle Foundation (2006)
34. Standley, T.: Finding optimal solutions to cooperative pathfinding problems. In: AAAI. pp. 173–178 (2010)
35. Surynek, P.: A novel approach to path planning for multiple robots in bi-connected graphs. In: ICRA 2009. pp. 3613–3619 (2009)
36. Surynek, P.: Solving abstract cooperative path-finding in densely populated environments. Computational Intelligence **30**(2), 402–450 (2014)
37. Surynek, P.: Time-expanded graph-based propositional encodings for makespan-optimal solving of cooperative path finding problems. Ann. Math. Artif. Intell. **81**(3-4), 329–375 (2017)
38. Surynek, P.: Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 1177–1183 (2019)
39. Surynek, P., Michalík, P.: The joint movement of pebbles in solving the (n^2 - 1)-puzzle suboptimally and its applications in rule-based cooperative path-finding. Autonomous Agents and Multi-Agent Systems **31**(3), 715–763 (2017)
40. Wilson, R.M.: Graph puzzles, homotopy, and the alternating group. Journal of Combinatorial Theory, Series B **16**(1), 86 – 96 (1974)