CS365

Artificial Intelligence

Project Report

Learning Heuristic Functions for 24 Puzzle

Instructor : Dr. Amitabha Mukerjee

Vinit Kataria (vinitk@iitk.ac.in)
N.V.Subba Rao (subbarao@iitk.ac.in)

April 11, 2012

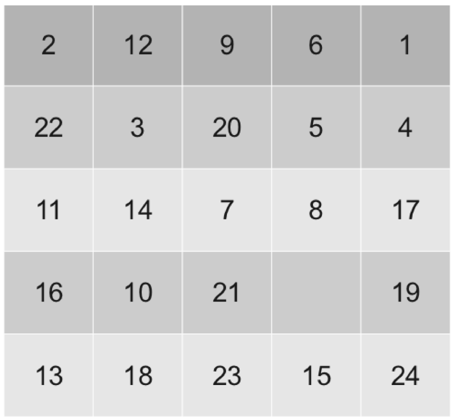
Department of Computer Science and Engineering

ABSTRACT

Solving large state space problems within the given computational limits is an active area of research in the recent years and a lot of work is being done on finding better methods. In our project we study one such method, the Bootstrapping Procedure[1]*, used to solve 24 sliding tile puzzle and investigate the results associated with solving this large state space problem and also the interleaving procedure for solving single problem instances. Experimental results indicate that the time taken for solving is reduced significantly with some reasonable sub-optimality in the solution cost.*

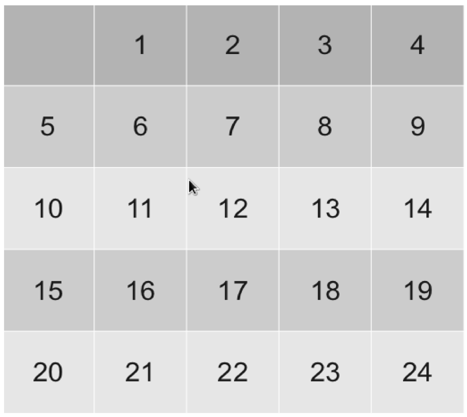
1. Introduction

The 24 tile puzzle (known as “24 puzzle”) is a sliding tile puzzle that consists of 24 numbered square tiles (in a 5x5 board) in random order with one tile missing. It is a single player puzzle. The objective of the player is to move the tiles adjacent to the blank position successively so that the final configuration resembles the goal (which has two possibilities).



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

An Initial State



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

Goal State

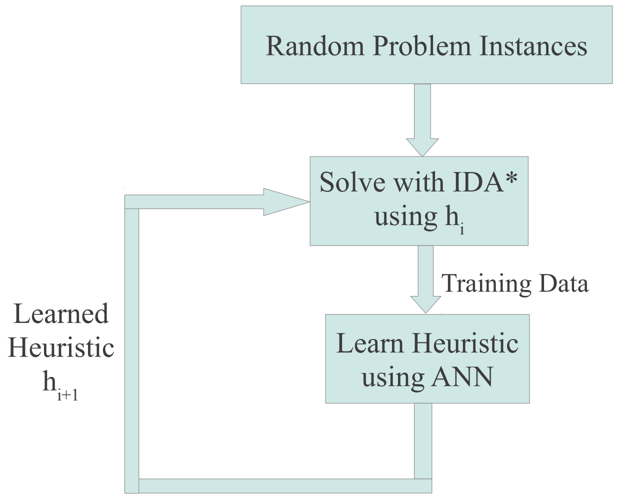
It has $25!/2$ (~ 10^{25}) solvable states. Solving it using brute force search takes time of the order of billion years whereas using an efficient search algorithm like IDA* with Manhattan Distance heuristic also takes few thousand years on an average. Using stronger heuristics with IDA* significantly reduces the number of nodes generated and the time required to solve problem instances. The bootstrapping procedure[1] aims at finding stronger heuristics(which may be inadmissible) which would help in solving problem instances in reasonable time using machine learning.

2. Related Work

Machine learning approach to create heuristic functions was first applied successfully to 15-puzzle and other similar puzzles (Ernandes and Gori [2] and Samadi, Felner, and Schaeffer [3]), but could not be applied to larger spaces, e.g., the 24-puzzle, due to difficulties in the creation of a sufficiently large training set with various possible distances to goal. Ernandes and Gori [2] also proposed a way of extending the machine learning approach for larger problems, called “bootstrap learning of heuristic functions"(i.e.bootstrapping) but could not implement it. Later, automatic bootstrapping was practically implemented for the 24-puzzle by Jabbari, Holte and Zilles [1] and their results showed that this method is successful in creating strong heuristics (though inadmissible) outperforming Weighted IDA* and other previous methods. They also implemented a method to solve single problem instances quickly by interleaving learning and problem-solving phases which we have also tried to implement similarly.

3. Bootstrapping Procedure

In this iterative method, we start with a set of problems called 'bootstrap instances' as a training set. We try to solve these problems using IDA* with the initial weak heuristic which is the maximum of 13 heuristics used namely: manhattan distance, misplaced tiles, blank tile position and four 6-tile disjoint PDBs(pattern databases),their sum and their reflections about diagonal and their sum. The set of solved problem instances together with all the states along their solution paths are passed on to the learning phase along with their solution lengths. These solved problems are removed from the bootstrap instances. In the learning phase, single hidden layer feed-forward back-propagation neural networks are used to learn a new heuristic function (h_1) which can predict the solution length for any state. This process is repeated for the next iteration with h_1 in lieu of h_0, then h_2 in place of h_1 and so on. Finally, when there are only a few bootstrap instances left, we discontinue this process and the obtained heuristic h_n is strong enough to solve random test instances reasonably quickly.


Random Problem Instances
Solve with IDA* using h_i
Training Data
Learn Heuristic using ANN
Learned Heuristic h_{i+1}

Schematic Diagram showing the Bootstrapping Procedure

The total time needed for learning a sufficiently strong heuristic is rather large for large problem domains(like 24-puzzle). That makes bootstrapping useful only when a large number of test problem instances need to be solved. In case a single test instance is to be solved, we decrease total time substantially so that Bootstrap becomes practical by Interleaving method.

3.1 Artificial Neural Network (ANN)

A Feed Forward Neural Network[1] with back-propagation learning algorithm is used in the learning phase of the Bootstrapping procedure. The new heuristic is trained as a function of the 13 heuristics. Its architecture can be described as follows

1. An input layer consisting of 13 input neurons which represent the 13 heuristics as stated earlier.
2. A single hidden layer with three hidden neurons with tangent sigmoid (tansig) as the activation function/transfer function.
3. An output layer comprising of a single neuron with linear transfer function (purelin).

The input is a 13-dimensional heuristic vector which is transformed finally into an output i.e. the solution length (need not be optimal). Gradient descent is used as the weight update function. The performance/evaluation function used is MSE (mean square error). The neural network is trained for 100 epochs or until MSE falls below 0.005.

After this process is complete, we obtain the weights and biases of the network and to compute the value of the new heuristic for a state 's', we pass the feature vector H(s) into the ANN. The output of the ANN (using the weights and biases) is used as the new heuristic value for that state 's'.

4. Interleaving Procedure

Interleaving that we used in our project is an extension of the bootstrapping procedure for solving single test instances. In this we start with no training set. We first try to solve a given problem using IDA* with the initial heuristic within a certain manually set time-limit t_{max}. In case it is unsuccessful, we perform random-walks backwards from the goal, using a Random Walk method as described below, to generate problems which can be solved using the present heuristic within manually set time limits. Then using this as the training data, we learn a new heuristic function. The test problem is again tried to be solved within given t_{max} using the new heuristic and this procedure is used until the test problem is solved.

4.1 Random Walk Procedure

The random walk method generates problem instances of varying lengths which are subsequently incremented after each iteration. Starting from the goal state, random walks are taken ensuring that the previous step is not followed back. This gives a collection of problem instances which can be supposedly be solved using the current heuristic to generate training data over which learning of better heuristic may take place during the interleaving procedure. The choice of time limits for solving each of these random walk instances that we did were experimental and their effects haven't been looked upon in detail but have worked so far.

5. Implementation

- The code for the Bootstrapping procedure was in two parts as follows:
    1. C++ code which solves using IDA* and generates training data of 13-feature vector and solution length.
    2. MATLAB code which uses the training data to generate stronger heuristic function using feed-forward back-propagation neural network through a built-in function newff available in Neural Network Toolbox.
- The code for the Random walk procedure was written in C language.
- The interleaving procedure was implemented using bash scripts which made use of the above codes.

6. Results

Bootstrapping for 5000 instances

Iteration	Number Solved	Remaining Unsolved	Nodes Generated	Solving Time(sec)	Learning Time(sec)
0	3301	1699	659,791,749	2,167.42	2,139.64
1	1290	409	529,834,012	2,886.16	1,327.79
2	288	121	201,783,421	959.83	686.29
3	68	53	58,824,222	350.52	304.28

Comparison of Initial Heuristic & Final Heuristic (Bootstrapping) for 50 random easy problem instances

	Initial Heuristic	Final Heuristic
Time taken (sec)	94.39	3.31
Total nodes generated	40,770,726	1,378,488

We observe that the total nodes generated above substantially decrease in case of final heuristic hence the search by IDA* speeds up thus it is a stronger heuristic. A stronger heuristic function can hence be achieved by applying the bootstrapping process on weak heuristics to reduce the solving time significantly. The Bootstrapping we did was on easier problem sets mostly and then the final heuristic obtained was tested on easier problem instances.

The Interleaving procedure when we applied to some single problem instances that we took from a set of problems from 50 problems given in [4] for which we set solving timelimit : learning timelimit =100:200 (in which 200 random walk instances were generated for each iteration with a length increment which we set to be 20 starting from random-walk length=10) solved the problems with average optimal costs 88.8 in about 24 minutes on an average with a reasonable sub-optimality less than 10% on an average. Thus, the interleaving procedure seems to work effectively on the problems that we tested on and saves total time. The machine that we used throughout had a 2.33 GHz processor.

7. Acknowledgements

We would like to thank Mr. Shahab Jabbari Arfaee for sharing his code of bootstrapping procedure with us and also making the pattern databases available to us. We would also like to thank Dr. Amitabha Mukerjee for his guidance and insight.

8. References

[1] Shahab Jabbari Arfaee, Sandra Zilles, Robert C. Holte. Learning Heuristic Functions for Large State Spaces. In *Journal of Artificial Intelligence*,175:pages 2075–2098, 2011.

[2] Marco Ernandes and Marco Gori. Likely-admissible and sub-symbolic heuristics. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, pages 613–617, 2004.

[3] Jonathan Schaeffer, Ariel Felner, Mehdi Samadi. Learning from multiple heuristics. In *Proceedings of 23rd AAAI Conference on Artificial Intelligence(2008)*, pages 357-362, 2008.

[4] Richard E. Korf and Ariel Felner. Disjoint pattern database heuristics. *Artificial Intelligence*, 134:9–22, 2002.

[5] Larry A. Rendell. A new basis for state-space learning systems and a successful implementation. *Artificial Intelligence*, 20:369–392, 1983.