Using Neural Networks for Evaluation in Heuristic Search Algorithm

Hung-Che Chen   Jyh-Da Wei

Department of Computer Science and Information Engineering, Chang Gung University
259, Wenhwa 1st Road, Kweishan, Taoyuan, Taiwan 333

Introduction

A major difficulty in a search-based problem-solving process is the task of searching the potentially huge search space resulting from the exponential growth of states. State explosion rapidly occupies memory and increases computation time. Although various heuristic search algorithms have been developed to solve problems in a reasonable time, there is no efficient method to construct heuristic functions (B. Coppin 2004; S. Russell and P. Norvig 2010). In this work, we propose a method by which a neural network can be iteratively trained to form an efficient heuristic function. An adaptive heuristic search procedure is involved in the training iterations. This procedure reduces the evaluation values of the states that are involved in the currently known best solution paths. By doing so, the promising states are continuously moved forward. The adapted heuristic values are fed back to neural networks; thus, a well-trained network function can find the near-best solutions quickly. To demonstrate this method, we solved the fifteen-puzzle problem (Parberry 1995). Experimental results showed that the solutions obtained by our method were very close to the shortest path, and both the number of explored nodes and the search time were significantly reduced.

A^* Search Algorithm

The A^* algorithm was proposed in 1968 (Hart, Nilsson, and Raphael 1968), and it has been the most famous heuristic search algorithm ever since. To implement the A^* algorithm, a programmer has to declare two linked lists, i.e., OPEN and CLOSE, to save the nodes whose successors have not been explored and the nodes whose successors have been explored, respectively. For easy computation, the OPEN list is sorted by ordering the evaluation values (eqn(1)) in ascending order. At the start of a search, a node with the initial state and the evaluation value thereof is generated. Then, we add this node to the OPEN list and repeat the following two steps:

Step 1. Generate the successors of X. For each successive node Y, derive the evaluation value f(Y) and then do the following:

Step 2. Generate the successors of X. For each successive node Y, derive the evaluation value f(Y) and then do the following:

(a) Examine whether the state of Y is identical to that of another existing node Z in the OPEN list or the CLOSE list. If Z exists, either delete Z or discard Y, depending on whether $f(Z) > f(Y)$.

(b) If Y is not discarded, let the ancestral node of Y be X and add Y to the OPEN list.

(c) Return to Step 1.

The above-mentioned evaluation values in the A^* algorithm are calculated by the following path-based evaluation function:

$$f(n) = g(n) + h(n), \tag{1}$$

where $g(n)$ is the cost of the path from the initial state to the concerned node n, and $h(n)$ is an estimation of $h^*(n)$, i.e., the cost along an optimal path from n to the goal. The performance of this algorithm depends greatly on the heuristic function $h(n)$. If $h(n) \leq h^*(n)$, the A^* algorithm is guaranteed to find an optimal solution. Notably, a higher value of $h(n)$ implies a more accurate estimate that drives the search process faster.

Training Neural Networks for Evaluation

Figure 1(a) presents the flow chart of the overall training process. The parts separated by the connector "A" are the preprocessing and the iterative training parts. Three tasks are associated with the preprocessing part: (1) Design a reasonable heuristic function that NN can simulate in the initial stage. The heuristics are not restricted admissible; thus, we denote this sub-optimal function as h_{sub}. (2) Generate instances at random. For each instance N, extract the feature vector $v(N)$ and the heuristic value $h_{sub}(N)$ separately from the state of N. (3) Normalize all the feature and the heuristic values; then, add them to the set of training data.

In the iterative training process, we set a parameter, $max\_iteration$, as the upper bound of the training loops. Provided that we have several distinct initial states as the sample problems to be solved, the training process repeats the following steps: (1) Train the network using the training data and then clear the set of training data. (2) For each sample problem to be solved, run the Adaptive Heuristic Search

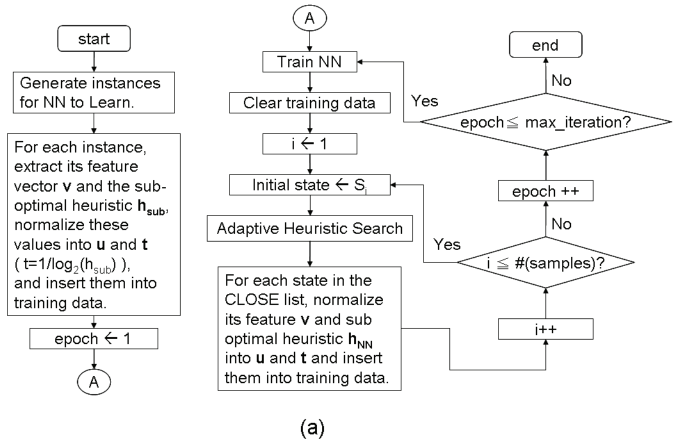
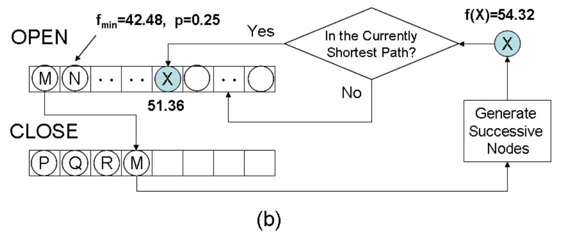

start
Generate instances for NN to Learn.
For each instance, extract its feature vector v and the sub-optimal heuristic h_sub, normalize these values into u and t (t=1/log2(h_sub)), and insert them into training data.
epoch ← 1
A
A
Train NN
Clear training data
i ← 1
Initial state ← S_i
Adaptive Heuristic Search
For each state in the CLOSE list, normalize its feature v and sub optimal heuristic h_NN into u and t and insert them into training data.
end
No
Yes
epoch ≤ max_iteration?
epoch ++
No
Yes
i ≤ #(samples)?
i++
(a)


Figure 1: *The proposed method.* (a) Flow chart of the overall training process. (b) Adaptive heuristic search (AHS).

(AHS) procedure. This procedure uses the neural network designed by us to generate the heuristic value h_{NN}; nevertheless, heuristic values can be further adjusted during the search process. (3) Once a sample problem has been solved, record all the feature vectors and the heuristic values (could be adjusted) from the nodes in the CLOSE list. Add these values to the set of training data after normalization and then repeat the steps listed above.

Adaptive Heuristic Search

Figure 1(b) presents the kernel of our training method, i.e., the adaptive heuristic search (AHS) procedure. This procedure is modified from the A^* algorithm and also involves the use of the OPEN and the CLOSE lists. The heuristic function of AHS, denoted as h_{NN}, is built upon a feedforward neural network. For each search node N, we extract the feature vector $v(N)$ from the state of the node, normalize the feature, and send it to the network. Assuming that the output is $A(N)$, the heuristics are given by

$$h_{NN}(N) = 2^{(1/A(N))} \tag{2}$$

Following eqn(1), we derive a temporary evaluation function

$$\hat{f}(N) = g(N) + h_{NN}(N). \tag{3}$$

In the normal case, we assume

$$f(N) = \hat{f}(N); \tag{4}$$

and

$$h(N) = h_{NN}(N). \tag{5}$$

Adaptation occurs when we explore a successive node, X, and the state of this node is verified to be included in the currently known shortest path. In this case, we reduce the evaluation value of X as

$$f(X) = p\,\hat{f}(X) + (1-p)\,f_{min} \tag{6}$$

and replace the heuristic value of X with

$$h(X) = f(X) - g(X), \tag{7}$$

where f_{min} is the minimum of the evaluation values among all OPEN nodes, and p $(0 \leq p \leq 1)$ is the adaptation ratio, referred to as the "promotion rate." After adaptation, we return to step 2(a) of the A^* algorithm, i.e., we insert X in the OPEN list.

Experimental Results and Conclusion

Using the fifteen-puzzle problem as an example,the solutions obtained by our method are very close to the shortest path and both the explored nodes and the searching time are significantly reduced. According to a literature survey, similar methods to train a neural network for heuristic search have been presented (Boyan and Moore 2001; Chellapilla and Fogel 1999). However, these methods have to find the best solutions first, such that the cost of the optimal path from the concerned state to the goal can be considered as the target value for training the neural network. The requirement of finding the best solutions implies the necessity of constructing an admissible heuristic function for performing a search in the first stage. If we can find only sub-optimal solutions by using non-admissible heuristics, the capability of the neural network is restricted to approximating the cost leading to sub-optimal solutions.

Acknowledgment

This work was supported by the National Science Council under the Grant NSC99-2628-E-182-051, advised by Jyh-Da Wei, who is the advisor of Hung-Che Chen and also the corresponding author of this work.

References

B. Coppin. 2004. *Artificial Intelligence Illuminated.* Sudbury, MA: Jones and Bartlett.

Boyan, J., and Moore, A. W. 2001. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research* 1:77–112.

Chellapilla, K., and Fogel, D. 1999. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE* 87(9):1471–1496.

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

Parberry, I. 1995. A real-time algorithm for the (n^2-1)-puzzle. *Infromation Processing Letters* 56:23–28.

S. Russell and P. Norvig. 2010. *Artificial Intelligence – A Modern Approach.* New Jersey: Pearson, 3rd edition.