Multi Neuron Heuristic Search

Dr. Anupam Shukla and Rahul Kala
anupamshukla2000@yahoo.com rahulkalaiiitm@yahoo.co.in

Indian Institute of Information Technology and Management Gwalior, Gwalior, Madhya Pradesh, INDIA

Citation: A. Shukla, R. Kala (2008) Multi Neuron Heuristic Search, *International Journal of Computer Science and Network Security* 8(6), 344-350.

Final Version Available At: http://paper.ijcsns.org/07_book/200806/20080648.pdf

Summary
We know the various searching algorithms available today. Searching has become one of the most essential parts of the artificial intelligence algorithms these days. We have so many algorithms like A*, Heuristic Search, Breadth-First Search, Depth First Search, etc. All these are applied to various problems in their own way. We need to predict the most appropriate search technique as the input data is not known. In this paper we present a new searching algorithm. This algorithm works on the principle of applying many neurons (elementary searching units) for working on different data one after the other. Hence as in the case of A* and heuristic search, we do not only select the best current node, but we select a range of nodes from the best to worst. At each iteration various nodes are seen and expanded which have varying heuristic costs. This algorithm would work very well on data in which heuristics change suddenly from very good to bad or vice-versa. We implemented this algorithm and put it on the maze-solving problem, where the heuristic cost was the distance between the nodes to goal point. We saw that the algorithm worked better than any existing algorithm and visited the least number of nodes. This proves the efficiency of the algorithm. We have also shown that this algorithm lies between A* Algorithm and Breadth First Search. Both these algorithms can be reached using this algorithm.

Key Words
Searching, artificial intelligence, A*, heuristics, multi-processing, multi-neurons

1. Introduction

We have a huge number of search algorithms available for us today. Any programmer tries to select the best out of the available alternatives by making a guess of the input data. All the search algorithms work well on a set of data. There may have a set of data for which the performance is bad. Hence for most of the problems, input drives the performance. The choice of the search algorithm is driven by practical scenarios that are encountered. People are spending huge times to improvise on the searching algorithms as it is the most basic operation. The normal sequence followed is to implement an algorithm, find out the weaknesses when applied on practical data, and then if required change the algorithm.

We can see that all algorithms of searching have some or the other weaknesses for most of the problems. People mostly try to shift to the modern algorithms, A* and heuristic search. But for these algorithms, the choice of the heuristic function should be very good. A bad choice of heuristic can lead to a reduced performance from the Breadth-First Search or the Depth First Search.

In this paper we propose an algorithm that optimizes the performance of searching in cases where the heuristics are not very strong and can't be depended upon very much. This algorithm uses a kind of multi-processing or multi-neuron model which optimizes the performance.

In section 2 we have a look at the present algorithms and their strengths and weaknesses. In Section 3 we would discuss the conditions in which this algorithm may be used. The algorithm is discussed in Section 4. In Section 4.2 we will discuss the various factors affecting the algorithm. In Section 5 we give a comparative analysis of various algorithms using a particular problem. Section 6 gives some problems where this algorithm may be applied. Section 7 gives the conclusion and scope for future work.

2. Present Algorithms

Presently the following are the major algorithms being used. We discuss their strengths and weaknesses in brief, one by one[8].

2.1 Breadth First Search

The algorithm iterates from one level to other, taking the breadth of the graph/tree first. Various improvements in the past like introduction of parallels[1][5][10][12].

Strengths: Very good for small input size (gives the node closest to the starting point)

Very good if the node is very close to the start point.

Weaknesses: Practically data may be very large and nodes may be very far away from the start. Very poor performance in such cases.

2.2 Depth First Search

The algorithm iterates and tries to go to the innermost levels, taking the depth of the graph/tree first[3][4][9][11].

Strengths: Very good for highly connected graphs where one node may be connected in many ways to many nodes.

Very good if the node is very far from start point but can be reached in many ways, so that the probability of reaching the goal from any node is high.

Weaknesses: Does not work well if the nodes go on expanding indefinitely. Algorithm can not guess how close it is from goal.

2.3 Iterative Deepening Search

The algorithm works quite similar to the Breadth first Search. It iterates from one level to other, at each level. Inside a level the function is similar to depth first search[11].

Strengths: Very good for small input size

Very good if the node is very close to the start point.

Weaknesses: the algorithm faces problems if the goal is very deep, or when the input size is very large.

2.4 A* Algorithm

The algorithm tries to minimize the sum of heuristic cost and the cost from the start point till the node. Hence it takes into account both the major costs and tries to find the most optimal solution[6][7][13][14][15].

Strengths: Very good if a decent heuristic function is available. It tries to get closer and closer to goal keeping the distance from source shortest.

Very commonly used in practical applications for its efficiency.

Weaknesses: If the heuristic function is bound to sudden changes, on moving a unit node, the algorithm would reduce its efficiency.

2.5 Heuristic Search Algorithm

The algorithm tries to minimize the heuristic cost. Hence it is a kind of futuristic algorithm which tries to minimize the distance from goal[2][13][14][15].

Strengths: Very good if a decent heuristic function is available.

Always tries to get closer and closer to the goal.

Weaknesses: If the heuristic function is bound to sudden changes, on moving a unit node, the algorithm would reduce its efficiency.

3 Conditions for Implementation of New Algorithm

The algorithm can be taken as a betterment of all the above algorithms where the heuristic function exists, but is bound to change suddenly. The heuristic and A* approach use the heuristic function in order to get the search closer and closer to the goal, but when it changes suddenly, the strategy is destroyed. Hence these algorithms suffer. Also if the heuristic function is available, it is always better to use it rather than not to use it altogether as was the case with other algorithms.

The algorithm can be applied to the cases where he following problems occurs in heuristic function:

- The heuristic function reaches near goal, but suddenly shows that no way is possible to reach goal.
- The heuristic function keeps fluctuating from the good values to bad values making it hard to predict the goal.
- The heuristic function drops suddenly from very high value to low value.

These conditions can easily be understood from the problem of maze solving, if the heuristic function of any point (x,y) on the maze denotes its squared distance from the goal. We can see that if the search algorithm reaches last but one position and then finds itself surrounded by walls, the heuristics increase suddenly. Similarly if the solution is a series of bad moves followed by another series of good moves, the heuristics decrease from high to low.

Hence in such problems though we may take the heuristic function, its performance would be low. The solution is to use the new algorithm which respects all the good, bad and moderate values of heuristics, so that no value suffers.

Readers may kindly note that such an algorithm, due to its parallel nature will take huge benefits from modern concepts like multi-processor, grid computing etc.

4. Algorithm

The basic idea of this algorithm is the use of many neurons working one after the other. Each of these take care of high to low values of the heuristic

functions. The algorithm hence gives respect to all values of the heuristics. It may be seen as the way of employing different neurons for different types of works and whichever finds the target, is rated successful. If you were to find a treasure, it would be justified to divide your team at various places, some at high probability places, some at low.

In all we take α neurons. We have a list of heuristic costs each corresponding to node seen but waiting to be processed. We divide the cost range into α ranges equally among them. Each of these neurons is given a particular range. Each neuron selects the minimum most element of the cost range allotted to it and starts searching. At one step of each neuron processes its element by searching and expanding the element. This process is repeated.

4.1 Algorithm

Step 1: open ← empty priority queue
Step 2: closed ← empty list
Step 3: add a node n in open such that position(n) = CurrentPosition, previous(n) = null and f(n), g(n), h(n) are as calculated by respective formulas with priority f(n)
Step 4: while open is not empty
Begin
Step 5: extract the node n_1, n_2, n_3, n_4….. n_α from open with the priority of n_1 as highest and the others equally distributed between other α-1 nodes.
Step 6: if n_i = final position for i=1,2,3,4,5…..α then break
Step 7: else
Step 8: nodes ← nodes from the expanding of node n_i
Step 9: for each node m in nodes
Begin
Step 10: if m is already in open list and is equally good or better then discard this move
Step 11: if m is already in closed list and is equally good or better then discard this move
Step 12 delete m from open and closed lists
Step 13: make m as new node with parent n
Step 14: calculate f(m), h(m), g(m)
Step 15: Add node m to open with priority f(m)
Step 16: Add n to closed
Step 17: Remove n from open

Here for any node n,
h(n) = heuristic cost
g(n) = cost from source
f(n) = is the total cost.
F(n) = g(n)+h(n)

4.2 Factors affecting the algorithm

The various factors which may affect the algorithm are:

- The number of neurons, α: It has a huge impact on the algorithm. If α=1, the search is an A* algorithm. If α=infinity, the search is equivalent to breadth first search.
- The fluctuation of the heuristic function: The more the fluctuation, the better will be its performance from other algorithms.

5 Implementation on a Problem

Consider the problem of solving a maze. The problem is that we have to move from the initial position to the final position in the maze without colliding from walls.

Refer Table 1 for the problem input. Here 0 represents the region we cannot move (wall) and 1 represents the region we can move (path). Top left is the start point. Bottom right is the finish point. The heuristic function is taken as the square of the distance of the current point to the final point. As mentioned in table 1, the numbers in results show the order in which they were discovered. The number of bottom right corner is the number of nodes explored. The results recorded are as given in table 1.

Table1: Inputs and Results to the maze solving problem

Input #1

1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	0	1
1	0	0	0	0	0	0	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	0	0	0	1	0	1	0	1
1	1	1	1	0	1	0	1	0	1
1	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1

A*

01	02	03	04	05	06	07	08	09	10
17	00	00	00	00	00	00	00	00	11
18	19	20	21	22	23	24	25	00	12
30	00	00	00	00	00	00	26	00	13
31	32	33	34	35	36	00	27	00	14
39	00	00	00	00	37	00	28	00	15
40	41	42	43	00	38	07	29	00	16
44	00	00	00	00	00	00	00	00	00
45	46	47	48	49	50	51	52	53	54

Nodes Visited: 54

Heuristic Search

01	02	03	04	05	06	07	08	09	10
17	00	00	00	00	00	00	00	00	11
18	19	20	21	22	23	24	25	00	12
30	00	00	00	00	00	00	26	00	13
31	32	33	34	35	36	00	27	00	14
39	00	00	00	00	37	00	28	00	15
40	41	42	43	00	38	07	29	00	16
44	00	00	00	00	00	00	00	00	00
45	46	47	48	49	50	51	52	53	54

Nodes Visited: 54

BFS

01	03	05	08	11	15	19	24	29	34
02	00	00	00	00	00	00	00	00	38
04	07	10	14	18	23	28	33	00	42
06	00	00	00	00	00	00	37	00	45
09	13	17	22	27	32	00	41	00	48
12	00	00	00	00	36	00	44	00	50
16	21	26	31	00	40	07	47	00	52
20	00	00	00	00	00	00	00	00	00
25	30	35	39	43	46	49	51	53	54

Nodes Visited: 54

DFS

01	02	03	04	05	06	07	08	09	10
17	00	00	00	00	00	00	00	00	11
18	19	20	21	22	23	24	25	00	12
30	00	00	00	00	00	00	26	00	13
31	32	33	34	35	36	00	27	00	14
39	00	00	00	00	37	00	28	00	15
40	41	42	43	00	38	07	29	09	16
44	00	00	00	00	00	00	00	00	00
45	46	47	48	49	50	51	52	53	54

Nodes Visited: 54

Our Algorithm

01	02	04	06	08	09	11	13	15	17
03	00	00	00	00	00	00	00	00	19
05	10	26	28	35	40	00	00	00	21
07	00	00	00	00	00	00	22	00	00
12	32	00	00	00	00	00	00	00	23
14	00	00	00	00	00	00	00	00	25
16	24	38	00	00	00	00	00	00	27
18	00	00	00	00	00	00	00	00	00
20	29	30	31	33	34	26	37	39	41

Nodes Visited: 41 (α=3)

Input #2

1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	1
1	0	1	1	1	1	1	1	1	0	1
1	0	1	0	0	0	0	0	1	0	1
1	0	1	0	1	1	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	1	1
1	0	0	0	1	0	1	0	0	0	0
1	1	1	1	1	0	1	1	1	1	1

A*

01	02	03	04	05	06	07	08	09	10	11
34	0	0	0	0	0	0	0	0	0	12
35	0	29	28	27	26	25	24	23	0	13
36	0	30	0	0	0	0	0	22	0	14
37	0	31	0	49	50	51	0	21	0	15
38	0	32	0	48	0	52	0	20	0	16
39	0	33	0	47	0	53	0	19	18	17
40	0	0	0	46	0	54	0	0	0	0
41	42	43	44	45	0	55	56	57	58	59

Nodes Visited: 59

Heuristic Search

01	02	03	04	05	06	07	08	09	10	11
34	0	0	0	0	0	0	0	0	0	12
35	0	29	28	27	26	25	24	23	0	13
36	0	30	0	0	0	0	0	22	0	14
37	0	31	0	49	50	51	0	21	0	15
38	0	32	0	48	0	52	0	20	0	16
39	0	33	0	47	0	53	0	19	18	17
40	0	0	0	46	0	54	0	0	0	0
41	42	43	44	45	0	55	56	57	58	59

Nodes Visited: 59

BFS

01	03	05	07	09	11	13	15	17	19	21
02	0	0	0	0	0	0	0	0	0	23
04	0	00	00	00	51	49	47	45	0	25
06	0	00	0	0	0	0	0	43	0	27
08	0	00	0	32	34	36	0	41	0	29
10	0	00	0	30	00	38	0	39	0	31
12	0	00	0	28	0	40	0	37	35	33
14	0	0	0	26	0	42	0	0	0	0
16	18	20	22	24	0	44	46	48	50	52

Nodes Visited: 52

DFS

01	02	03	04	05	06	07	08	09	10	11
34	0	0	0	0	0	0	0	0	0	12
35	0	29	28	27	26	25	24	23	0	13
36	0	30	0	0	0	0	0	22	0	14
37	0	31	0	49	50	51	0	21	0	15
38	0	32	0	48	0	52	0	20	0	16
39	0	33	0	47	0	53	0	19	18	17
40	0	0	0	46	0	54	0	0	0	0
41	42	43	44	45	0	55	56	57	58	59

Nodes Visited: 59

Our Algorithm

01	02	05	05	06	08	09	10	12	14	15
03	0	0	0	0	0	0	0	0	0	17
07	0	00	00	00	49	47	33	32	0	19
11	0	00	0	0	0	0	0	31	0	20
13	0	00	0	40	41	42	0	29	0	21
16	0	00	0	39	0	43	0	28	0	23
18	0	00	0	38	0	44	0	27	26	25
22	0	0	0	37	0	45	0	0	0	0
24	30	34	35	36	0	46	48	50	51	52

Nodes Visited: 52 (α=3)

Table 2: α v/s The number of nodes visited

Input No.	α	The number of nodes visited
I	1	54
	2	54
	3	41
	4	44
	5	54
	Infinity	54
II	1	59
	2	52
	3	56
	4	59
	5	56
	6	56
	7	56
	8	54
	9	56
	10	56
	11	55
	12	55
	13	54
	14	52
	Infinity	52

5.1 Relation between α and the number of nodes visited

It can easily be predicted that for very small value of α the algorithm behaves like heuristic search and for very large value, the behavior is similar to a breadth first search. If we study the various values of α against the number of nodes for the previous inputs, we see the results as given in table 2

6. Problems where the algorithm can be applied

Some of the problems that may serve better using this algorithm are:

- Consider the 8 queens problems. If we lay down a constraint that a set of final positions can never be the final answer, then this algorithm would prove better.
- Consider the problem of Hannibal and the Cannibals, if we lay a constraint that a specific set of configurations is not possible, then this algorithm may be better.
- Consider a game where the player is expected to move from one specific position to the other such that it takes the minimum points during its path, where the points are scattered all over the board. If the player moves over a point, the points associated with that point are awarded to the player. If the points can take values in any range, this algorithm would prove better.
- Consider the 8 puzzle problem where we slide pieces on a grid, so as to reach a final configuration. If we say that the puzzle automatically shuffles at some state of the board, this algorithm might be useful.

7. Conclusion

In this paper we saw the new search algorithm and we studied the various cases in which it may beat its counterparts ie A* algorithm, Breadth First Search, Depth First Search. We saw the algorithm working very well in conditions where a heuristic function is available, works well, but still lags behind in certain conditions. In searching we must be prepared well in advance of the case of failure of the heuristics. This is exactly what this algorithm tried to do.

We studied the effect of this algorithm and saw it beating all it counterparts on the given data. This result proves the might of this algorithm and ensures a better result in certain conditions.

So far we have fixed value of α. The relation between α and the input size needs to be studied. If we are able to predict the correct α for the input, our search efficiency would improve a lot.

8. References

[1] Yang Zhang and Eric A. Hansen, "Parallel Breadth-First Heuristic Search on a Shared-Memory Architecture", Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications Boston, MA July 17, 2006

[2] Meyer Harald and Weske Mathias, "Automated Service Composition using Heuristic Search"

[3] Sitompul Opim Salim, Noah Shahrul Azman Mohd, "Multidimentional Model Visualization Using Depth-First Search Algorithm", Proceedings of the 2nd IMT-GT Regional Conference on Mathermatics, Statics and Applications Universiti Sains Malaysia, Penang, June 13-15, 2006

[4] Yeoh William, Koenig Sven, Felner Ariel, "IDB-ADOPT : A Depth-First Search DCOP Algorithm"

[5] Ezzahir Redouane, Bessiere Christian, "Asynchronous Breadth-First Search DCOP Algorithm", Applied Mathematical Sciences, Vol. 2, 2008, no. 37, 1837 - 1854

[6] Bentley Jon L., Sedgewick Robert, "Fast Algorithms for Sorting and Searching Strings"

[7] Areibi Shawki, Moussa Medhat, Abdullah Hussein, "A Comparison Of Genetic/Memetic Algorithms And Other Heuristic Search Techniques"

[8] Liotta Giuseppe, Tollis Ioannis G., "Advances in Graph Algorithms Special Issue on Selected Papers from the Seventh International Workshop on Algorithms and Data Structures, WADS 2001 Guest Editors' Foreword"

[9] Jornsson Yngvi B and Marsland Tony, "Selective Depth-First Search Methods"

[10] Yoo Andy, Chow Edmond, Henderson Keith, McLendon William, Hendrickson Bruce, ÄUmit C, atalyÄurek, "A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L"

[11] Rao V. Nageshwara and Kumar Vipin,"Parallel Depth First Search"

[12] Korf Richard E. and Schultze Peter, "Large-Scale Parallel Breadth-First Search"

[13] Jensen Rune M, Bryant Randal E, and Veloso Manuela M., "An Efficient BDD-Based Heuristic Search Algorithm"

[14] Pearl J., "Heuristics: Intelligent Search Strategies for Computer Problem Solving", Addison-Wesley, Reading, MA

[15] Luger, G. F. and Stubblefield, W. A., "Artificial Intelligence: Structures and Strategies for Complex Problem Solving", The Benjamin/ Cummings Publishing Co., Menlo Park, CA

About the Authors

Dr. Anupam Shukla

Dr. Anupam Shukla is an Associate Professor in the ICT Department of the Indian Institute of Information Technology and Management Gwalior. He has 19 years of teaching experience. His research interest includes Speech processing, Artificial Intelligence, Soft Computing and Bioinformatics. He has published around 62 papers in various national and international journals/conferences. He is referee for 4 international journals and in the Editorial board of International journal of AI and Soft Computing. He received Young Scientist Award from Madhya Pradesh Government and Gold Medal from Jadavpur University.

Rahul Kala

Rahul Kala is a student of 3rd Year Integrated Post Graduate Course (BTech + MTech in Information Communication Technology) in Indian Institute of Information Technology and Management Gwalior. His fields of research are robotics, design and analysis of algorithms, artificial intelligence and soft computing. He secured 7th position in the ACM International Collegiate Programming Contest, Kanpur Regionals. He is a student member of ACM. He also secured Al India 8th position in Graduates Aptitude Test in Engineeging-2008 with a percentile of 99.84.