Comparative Analysis of Search Algorithms

Maharshi J. Pathak
Student, B.Tech (IT)
CSPIT, CHARUSAT
Anand, Gujarat, India

Ronit L. Patel
Student, B.Tech (IT)
CSPIT, CHARUSAT
Anand, Gujarat, India

Sonal P. Rami
Assistant Professor (IT)
CSPIT, CHARUSAT
Anand, Gujarat, India

ABSTRACT
Nowadays many artificial intelligence search algorithms are available to figure out the problem of shortest path finding. The paper presents the detailed study of informed search and uninformed search techniques. The paper focuses more towards uninformed search strategies such as BFS, DFS, and UCS and informed search strategies like A*, and Best First Search. The paper includes working of search techniques, their merits, and demerits, where these algorithms are applicable, also open and closed list for each algorithm are shown. At last comparison of search techniques based on complexity, optimality and completeness are presented in tabular form.

Keywords
Artificial intelligence**,** informed search, search algorithms, shortest path algorithms, uninformed search.

1. INTRODUCTION
Artificial Intelligence is a study of "How to make computer to think like a human". Artificial Intelligence is a broad area of research. As shown in Figure 1 Artificial Intelligence has sub areas like Machine learning and Deep learning. The fields such as Artificial intelligence, Machine Learning, Deep Learning, and Data Science are inter-related with each other.

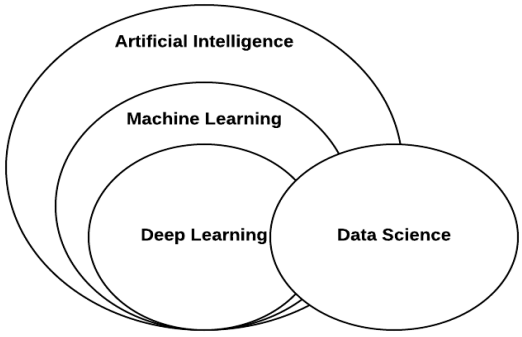

Fig 1: Co-relation of artificial intelligence with other fields

Artificial intelligence includes Problem Solving (Search techniques), Knowledge and Reasoning (Resolution, Knowledge Representation, Fuzzy Logic), Adversarial Search: Game Playing (Minimax Algorithm, Alpha-Beta Pruning), Uncertain Knowledge and Reasoning (Uncertainty, Probabilities, Bayesian Networks), and Expert system. The following paper presents more about problem-solving techniques in artificial intelligence. There are two problem-solving techniques in artificial intelligence, uninformed search, and informed search. The paper focuses more on BFS, DFS, UCS, A*, and Best First Search.

2. SEARCH ALGORITHM APPROACHES
2.1 Uninformed search
Uninformed Search is a blind search or brute force search which includes search techniques such as Breadth First Search, Depth First Search, Uniform Cost Search, Iterative Deepening, and Bidirectional Search. Uninformed search does not contain any information about the number of steps to reach from current state to goal state. The uniformed search will consider the path which is most promising at that moment, it will not consider the optimum path to reach the goal.

2.2 Informed search
Informed search is Heuristic search which uses heuristic function (To estimate how close a given state is from goal state) to solve a problem. Informed search uses generate and test approach. It includes hill climbing, steepest hill climbing, A*, AO*, and Best First Search algorithms. Informed search is more efficient than uninformed search. In informed search heuristic function is used as a model that will lead us to the goal state. The informed search will not traverse search tree blindly. It will consider the next node to traverse based on some evaluation function (Heuristic Function) to reach the goal state from current state.

3. WORKING OF SEARCH TECHNIQUES
3.1 BFS (Breadth first search)
In Breadth First Search all nodes are expanded level by level. It first expands all the nodes at first level in the search tree, then expands all the nodes of the second level and this way it reaches the goal. In Breadth First Search the frontier is actualized as a queue which works as First In First Out (FIFO). It is a poor strategy when all solution has a long way length or then again there is some heuristic information accessible [1]. It is not utilized when memory requirement is high. Time complexity is $O(b^d)$ and space complexity is $O(b^d)$, where b is branching factor and d is solution depth. For example, consider a graph (Figure 2). Table 1 shows open list and closed list for BFS. The open list is set of nodes yet to explore and closed list is set of nodes already been explored. At level 0 node 1 is expanded first. Children of node 1-2, 3, and 4 are added to the queue. Then according to First In First Out approach node 2 is expanded and child of node 2-6 is added to the queue. This way search reaches the goal state.

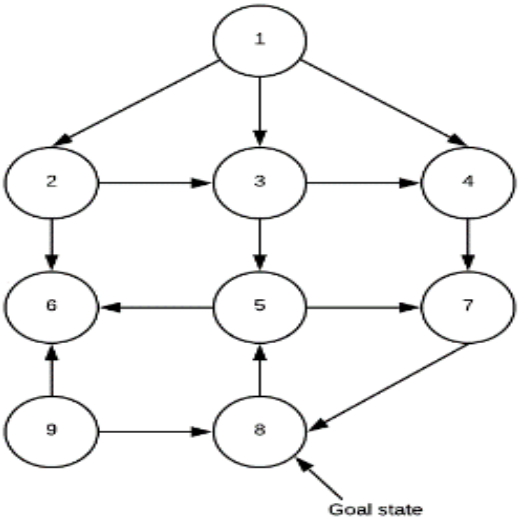

Fig 2: Graph for BFS and DFS

Table 1. Open and closed list for BFS

OPEN LIST	CLOSED LIST
1	1
2, 3, 4	2
3, 4, 6	3
4, 6, 5	4
6, 5, 7	6
5, 7	5
7	7
8–Goal state	-

3.1.1 Breadth First Search is applicable when:

- You need to discover the solution of problem containing the least curves.
- Few solutions may exist, and at least one has a short path length [1].
- Memory is not an issue.

3.1.2 Applications:

- For unweighted graph minimum spanning tree and shortest path.
- Shared network.
- Social Networking Websites.
- Global Positioning System Navigation systems.
- For testing whether the graph is bipartite or not.

3.2 DFS (Depth First Search)

In Depth First Search expansion starts from the source node and goes up to the deepest unexpanded node in a search of goal node. In Depth First Search the frontier is actualized as a stack which works as Last In First Out (LIFO). It is also known as recursive algorithm because it is implemented using the stack. Time complexity is $O(b^m)$ and space complexity is O(bm), where b is branching factor and m is maximum depth. For example, consider a graph (Figure 2). Table 2 shows open list and closed list for DFS. At level 0 node 1 is expanded first. Children of node 1-4, 3, 2 are added to the stack. Then according to Last In First Out approach node 2 is expanded and child of node 2-6 is added to the stack. Then node 6 is expanded. As it does not have any child it will backtrack to node 3 and node 3 will be expanded. These way nodes in the depth are expanded and search reaches the goal state.

Table 2. Open and closed list for DFS

OPEN LIST	CLOSED LIST
1	1
4, 3, 2	2
4, 3, 6	6
4, 3	3
4, 5	5
4, 7	7
4, 8-Goal state	-

3.2.1 Depth First Search is applicable when:

- Memory is limited.
- The order of the neighbours of a node are added to the stack can be tuned so that solutions are found on the first attempt [1].
- It is a poor technique when it is conceivable to get captured in endless ways, this happens when the graph is endless or at the point when there are cycles in the graph.

3.2.2 Applications:

- Identifying loops in a graph.
- Identifying components which are strongly connected in a graph.
- Finding the solution of the puzzle in which only one solution exists.

3.3 UCS (Uniform Cost Search)

Uniform Cost Search expand the node with low-cost path. It is implemented using the priority queue. To calculate cost of every node, consider this equation, c(m) = c(n) + c(n, m). where c(m) is the cost of the current node, c(n) is the cost of the previous node, and C (n, m) is the weight of the edge. The successor can be removed which are already in a queue with higher cost. Time complexity is $O(b^{\lfloor 1+C^*/e \rfloor})$ and space complexity is $O(b^{\lfloor 1+C^*/e \rfloor})$, where C is the optimal solution cost and each activity costs at least ε. For example, consider graph (Figure 3). Open list and closed list for UCS are shown in Table 3.

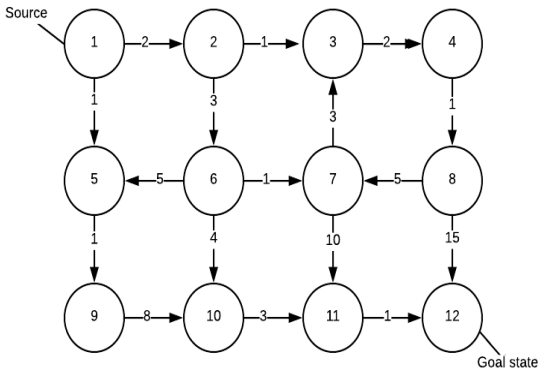

Fig 3: Graph for UCS

Table 3. Open and Closed list for UCS

OPEN LIST	CLOSED LIST
$1^{(0)}$	$1^{(0)}$
$2^{(2)}$, $5^{(1)}$	$5^{(1)}$
$2^{(2)}$, $9^{(2)}$	$2^{(2)}$
$9^{(2)}$, $6^{(5)}$, $3^{(3)}$	$9^{(2)}$
$6^{(5)}$, $3^{(3)}$, $10^{(10)}$	$3^{(3)}$
$6^{(5)}$, $10^{(10)}$, $4^{(5)}$	$6^{(5)}$
~~$10^{(10)}$~~, $4^{(5)}$, $7^{(6)}$, $10^{(9)}$	$4^{(5)}$
$7^{(6)}$, $10^{(9)}$, $8^{(6)}$	$7^{(6)}$
$10^{(9)}$, $8^{(6)}$, $11^{(16)}$	$8^{(6)}$
$10^{(9)}$, $11^{(16)}$, $12^{(21)}$	$10^{(9)}$
~~$11^{(16)}$~~, $12^{(21)}$, $11^{(12)}$	$11^{(12)}$
~~$12^{(21)}$~~, $12^{(13)}$-Goal state	-

3.3.1 Uniform Cost Search is applicable when:

- Space requirement is less.
- Intelligence is not required or when evaluation function (Heuristic function) is not required.

3.3.2 Applications:

- Solving Maze problem.
- Path finding.

3.4 A*

A* algorithm joins highlight of Uniform Cost Search and pure heuristic search to productively calculate optimal solution [1]. For computing cost of every node, consider this equation, f(m) = c(m) + h(m), where c(m) = c(n) + c(n, m), h(m) is heuristic function. f(m) compute the most reduced aggregate cost. Most reduced value of f is chosen at each node for expansion. Euclidean distance (used when allowed to move in any direction), Manhattan distance (used when allowed to move in only four directions-right, left, top, and bottom) or Diagonal distance (used when allowed to move in eight directions, same as the movement of king in chess) are used as Heuristic function. If the value of f of two nodes is same then the node having lowest h value is chosen for expansion. The calculation ends when an objective is decided for expansion. Admissible heuristic function ($h(n) \leq h^*(n)$) brings visited node back from the closed list to open list to get an optimal solution. Time complexity is $O(b^d)$ and space complexity is $O(b^d)$, where b is branching factor and d is solution depth [2]. For example, consider graph (Figure 4). Open and closed list associated with the example is shown in table 4.

Steps to reach goal state from source [3]:

1. Start from the source node, add it open list.
2. Explore all the nodes which are adjacent to the node which is in open list.
3. Calculate the cost for all the nodes discovered in step 2, and place them in open list in increasing order based on cost.
4. Move current working node, from the open list to closed list.
5. The first node in open list will become the current working node.
6. Repeat step 2 to 5, if the current working node is not goal state.
7. The closed list gives the shortest path and the value of last cost function obtained gives the optimal cost.

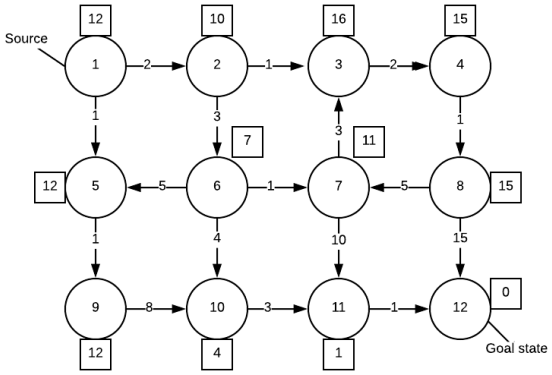

Fig 4: Graph for A*

Table 4. Open and Closed list for A*

OPEN LIST	CLOSED LIST
$1^{(12)}$	$1^{(12)}$
$2^{(12)}$, $5^{(13)}$	$2^{(12)}$
$5^{(13)}$, $3^{(19)}$, $6^{(12)}$	$6^{(12)}$
$5^{(13)}$, $3^{(19)}$	$5^{(13)}$
$3^{(19)}$, $7^{(17)}$, $10^{(13)}$, $9^{(14)}$	$10^{(13)}$
$3^{(19)}$, $7^{(17)}$, $9^{(14)}$, $11^{(13)}$	$11^{(13)}$
$3^{(19)}$, $7^{(17)}$, $9^{(14)}$, $12^{(13)}$-Goal state	-

3.4.1 Applications:

- Traffic navigation system [4].
- Games.
- Finding shortest path.
- Real-time path re-planning of an unmanned surface vehicle avoiding underwater obstacles [5].

3.5 Best First Search (Greedy search)

Best First Search is a merger of Breadth First Search and Depth First Search. Best First Search is implemented using the priority queue. The advantage of Depth First Search is that it gives a solution without calculating all node. while Breadth First Search arrives at a solution without search guaranteed that the procedure does not get caught. Best First Search, being a mixer of these two, licenses exchanging between paths. At each stage the nodes among the created ones, the best appropriate node is chosen for facilitating expansion, might be this node have a place to a similar level or different, hence can flip between Depth First and Breadth First Search [3]. It is also known as greedy search. Time complexity is $O(b^d)$ and space complexity is $O(b^d)$, where b is branching factor and d is solution depth [2].

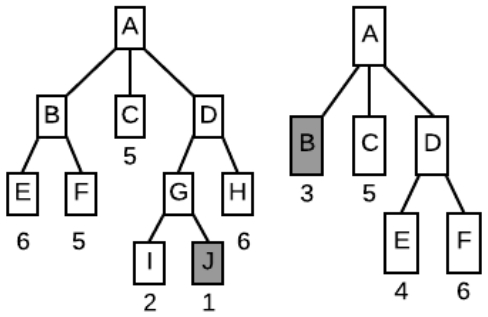

Fig 5: Example of Best First Search

3.5.1 Applications:

- Games.
- Web crawlers.

4. COMPARISON OF VARIOUS SEARCH ALGORITHMS

The performance of the algorithm is evaluated based on four parameters as follows:

1. Time complexity: Time taken by the algorithm to find a solution.
2. Space complexity: Memory required by the algorithm to perform the search.
3. Optimality: Solution provided by the algorithm will always be optimal or not.
4. Completeness: Is that calculation ensured to discover an answer when there would one say one is? This section is a Boolean marker regardless of whether the search algorithm is thorough.

Table 5. Comparison of search algorithm

Algorithm	**BFS**	**DFS**	**UCS**	**A***	**BFS (greedy search)**
Time Complexity	$O(b^d)$	$O(b^m)$	$O(b^{\lfloor 1+C^*/e \rfloor})$	$O(b^d)$	$O(b^d)$
Space Complexity	$O(b^d)$	O(bm)	$O(b^{\lfloor 1+C^*/e \rfloor})$	$O(b^d)$	$O(b^d)$
Optimality	Yes	No	Yes	Yes	No
Completeness	Yes	No	Yes	Yes	Yes

5. CONCLUSION

We conclude that heuristic search is more efficient and acceptable than blind search. From the example explained above (open and closed list for UCS and A*), it is clear that in UCS 11 nodes are required to expand to reach the goal state. While in A* only 6 nodes are required to expand to reach the goal state. Also, UCS is more time consuming then A*. So A* is more efficient and optimal than UCS. Also when all means costs are similar, instead of UCS, BFS is optimal because it always expands the shallowest unexpanded node. Hence heuristic search finds an optimal solution then blind search.

6. REFERENCES

[1] Deepika Garg, COMPARATIVE STUDY OF VARIOUS SEARCHING ALGORITHMS, National Conference on Innovative Trends in Computer Science Engineering (ITCSE-2015) held at BRCMCET, Bahal on 4th April 2015.

[2] Chandel, and Manu Sood, Searching and Optimization Techniques in Artificial Intelligence: A Comparative Study & Complexity Analysis, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 3 Issue 3, March 2014.

[3] Mr. Girish P Potdar, and Dr.R C Thool, COMPARISON OF VARIOUS HEURISTIC SEARCH TECHNIQUES FOR FINDING SHORTEST PATH, International Journal of Artificial Intelligence & Applications (IJAIA), Vol. 5, No. 4, July 2014.

[4] LIU Jingang, and LIU Yujun, Application of A* algorithm in Traffic Navigational System, Information Engineering and Electronic Commerce (IEEC), 2010 2nd International Symposium on. IEEE, 2010.

[5] Phanthong, Thanapong, et al. Application of A* algorithm for real-time path re-planning of an unmanned surface vehicle avoiding underwater obstacles. Journal of Marine Science and Application 13.1 (2014): 105-116.

[6] R.E. Korf, Scientific Paper on Artificial Intelligence Search Algorithms, University of California Los Angeles, June 1999.

[7] Eric A Hansen, and Rong Zhou, Anytime Heuristic Search, Journal of Artificial Intelligence Research 28, pp 267-287, 2007.

[8] Anne L. Gardner, Search: An Overview, AI magazine, Vol. 2, Number 1. Sept. 1980.

[9] A.Martelli, On the search complexity of admissible search algorithms, Al, Vol. 8, pp 1-13, 1977.

[10] R.K.Ahuja, K. Mehlhorn, J.B.Orlin and R.E.Tarjan, Faster algorithms for shortest path algorithms, Journal of the Association for Computing Machinery,Vol. 37, No. 2, pp 213-223, April 1990.