Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 7, July 2015, pg.513 – 522

RESEARCH ARTICLE

Shortest Path Searching for Road Network using A* Algorithm

Shrawan Kr. Sharma[1], **B.L.Pal**[2]
[1]M.Tech Scholar, Computer Science & Engineering, Mewar University, Gangrar, Chittorgargh-312901, India
shrawansharma7070@gmail.com
[2]Computer Science & Engineering, Mewar University, Gangrar, Chittorgargh-312901, India
Contect2blpal@rediffmail.com

ABSTRACT : In city area traffic the shortest path finding is very difficult in a road network. Shortest path searching is very important in some special case as medical emergency, spying, theft catching, fire brigade etc. There are various path searching algorithm like A* algorithm, Dijkstra etc in the market. The A* algorithm is fastest and provide the best shortest path. It uses heuristic to find the path. In present era only one directional search algorithm are used, we will use the Bi-directional search method for searching the path instead of Dijkstra algorithm due to demand of time and situations because of its robustness as well as its variants. This method reduces the searching time of the system and gets the fastest and best path. We will also compare the Dijkstra to A* algorithm s to show its performance in different-different ceases on the basis of different – different parameters.

Keywords – A* algorithm, Dijkstra algorithm, Bi-directional, Path Searching, heuristic

I. INTRODUCTION

Computing best possible routes in road networks from a given source to a given target location is an everyday problem. Many people frequently deal with this question when planning trips with their cars. There are also many applications like logistic planning or traffic simulation that need to solve a huge number of such route queries. Current commercial solutions usually are slow or inaccurate. The gathering of map data is already well advanced and the available road networks get very big, covering many millions of road junctions. Thus, on the one hand, using simple-minded approaches yields very slow query times. This can be either inconvenient for the client if he has to wait for the response or expensive for the service provider if he has to make a lot of computing power available [1]. On the other hand, using aggressive heuristics yields inaccurate results. For the client, this can mean a waste of time and money. For the service provider, the developing process becomes a difficult balancing act between speed and sub optimality of the computed routes. Due to these reasons, there is a considerable interest in the development of more efficient and accurate route planning techniques [2]. A* algorithm is a graph search algorithm that finds a path from a given initial node to a given goal node. It employs a "heuristic estimate" h(x) that gives an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. It follows the approach of best first search [3].

II. RELATED WORK

In A* Search a technique from the field of Artificial Intelligence, is a goal-directed approach, i.e., it adds a sense of direction to the search process. For each vertex, a lower bound on the distance to the target is required. In each step of the search process, the node v is selected that minimizes the tentative distance from the source s plus the lower bound on the distance to the target t. This approach can be combined with bidirectional search. The performance of the A* search depends on a good choice of the lower bounds. If the geographic coordinates of the nodes are given and we are interested in the shortest (and not in the fastest) path, the Euclidean distance from v to t can be used as lower bound. This leads to a simple, fast, and space-efficient method, which, however, gives only small speedups [3][4]. It gets even worse if we want to compute fastest paths. Then, we have to use the Euclidean distance divided by the fastest speed possible on any road of the network as lower bound. Obviously, this is a very conservative estimation. Goldberg et al. Even report a slow-down of more than a factor of two in this case since the search space is not significantly reduced but a considerable overhead is added [5].

A) Dijkstra Algorithm

Dijkstra algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.[4][5] The algorithm exists in many variants; Dijkstra original variant found the shortest path between two nodes,[6] but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest path tree.

B) A Algorithm*

A* algorithm is a graph search algorithm that finds a path from a given initial node to a given goal node. It employs a "heuristic estimate" h(x) that gives an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. It follows the approach of best first search. The secret to its success is that it combines the pieces of information that Dijkstra algorithm uses (favoring vertices that are close to the starting point) and information that Best- First-Search uses (favoring vertices that are close to the goal). In the standard terminology used when talking about A*, g(n)represents the exact cost of the path from the starting point to any vertex n, and h(n) represents the heuristic estimated cost from vertex n to the goal[7]. The A* algorithm depends on evaluating the best next step in searching for a path. This is done by evaluating each of the possible next steps against a heuristic to give a value that can be used to sort the list and hence determine the most likely next step. As you can imagine this makes choosing a good heuristic for your map and game really important to getting good path finding performance [8].

$F (n) =g(n)=h(n)$

III. A* Algorithm: working method

A. ***Unidirectional:*** It works by maintaining two lists; the open list and the closed list. The open list initially contains the start node; it contains all nodes that are yet to be considered. If the open list becomes empty then there is no possible path. The closed list starts out empty and contains all nodes that have already been considered /visited. The core loop of the algorithm selects a node from the open list with the lowest estimated cost to reach the goal. If the selected node is not the goal it puts all valid neighbouring nodes into the open list and repeats the process. Part of the magic in the algorithm is that all nodes that are created keep a reference to their parent. This means that from any node we can track backwards to get a path from the node and the start node [9].

i. Pseudo code

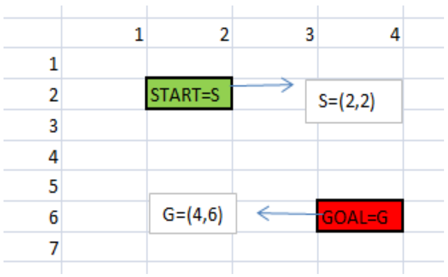
```
1) At initialization add the starting location to the open list and empty the closed list

2) While there are still more possible next steps in the open list and we haven't found the target:
   A) Select the most likely next step (based on both the heuristic and path costs)
   B) Remove it from the open list and add it to the closed
   C) Consider each neighbor of the step. For each neighbor:
          i) calculate the path cost of reaching the neighbor
          ii) If the cost is less than the cost known for this location then remove it from the open or
              closed lists (since we've now found a better route)
          iii) If the location isn't in either the open or closed list then record the costs for the location
               and add it to the open list (this means it'll be considered in the next search). Record how
               we got to this location
```

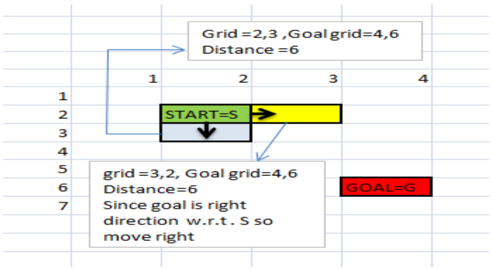
The loop ends when we either find a route to the destination or we run out of steps. If a route is found we back track up the records of how we reached each location to determine the path.

Example

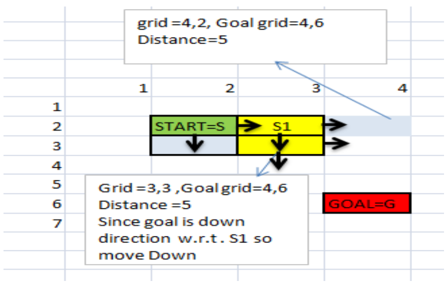
Initial step



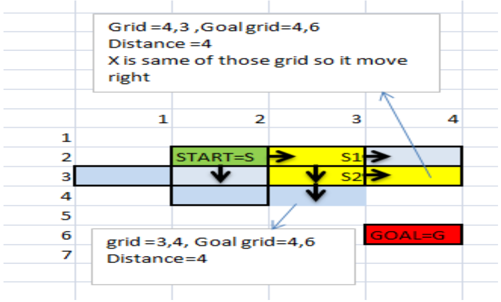
Step 1



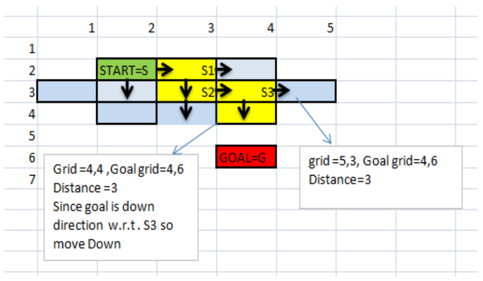
Step 2



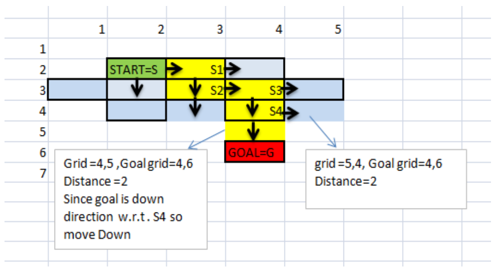
Step 3



Step 4



Step 5



Step 6

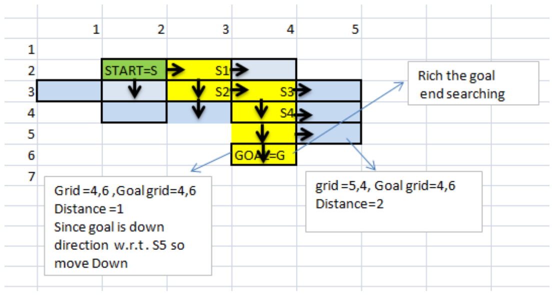


Figure 1: One direction method in A* algorithm

B) Bi-directional search

As the name suggests, bidirectional search suggests running 2 simultaneous searches, one from the initial state and the other from the goal state, those 2 searches stop when they meet each other at some point in the middle of the graph..

Pseudo code

```
1) At initialization add the starting and goal location to the open list and empty the closed list
2) Begin with a starting node with options for further travel: Also have a list of coordinates with
   nodes demonstrating movement options towards goal intersection
3) Consider each neighbour of the step. For each neighbor:
      i) calculate the path cost of reaching the neighbor
      ii) If the cost is less than the cost known for this location then remove it from
            the open or closed lists (since we've now found a better route)
      iii) If the start and goal path cost is same then intersect
```

The loop ends when we either find a route to the destination or we run out of steps.

Example:

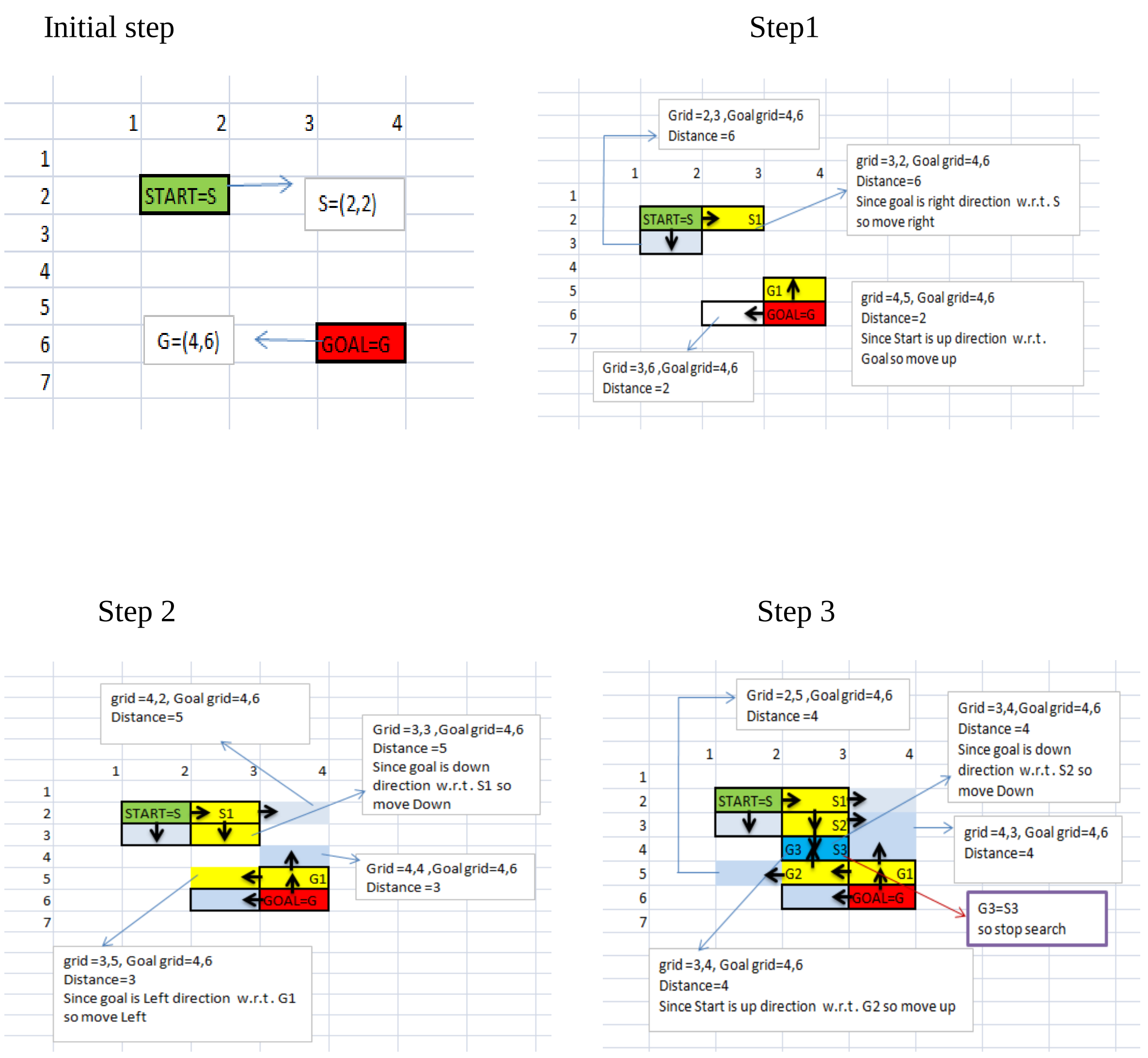
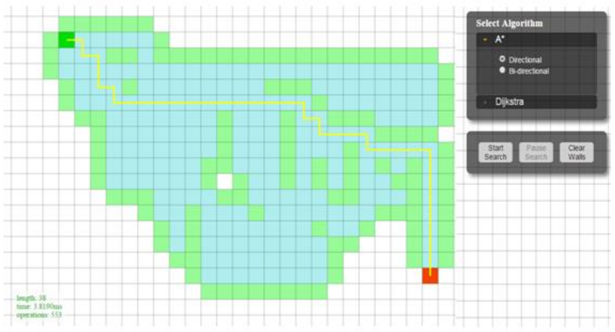


Figure 2: Bi-directional search in A* algorithm

IV. Experiment Result and Analysis

In this section, we conduct a experiment to find the best shortest path between the nodes. In that path was done by A* algorithm. I have also compare the A* algorithm to Dijkstra algorithm to searching the low cost path found. The compression is below. The compression is based on some parameters. I would like use the Bi-directional method in the A* star or Dijkstra algorithm.

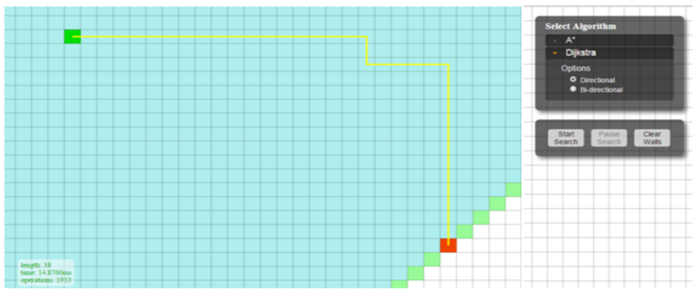
A) A* algorithm use one directional



```
length: 38
time: 3.8190ms
operations: 553
```

Figure 3: A* algorithm use one directional

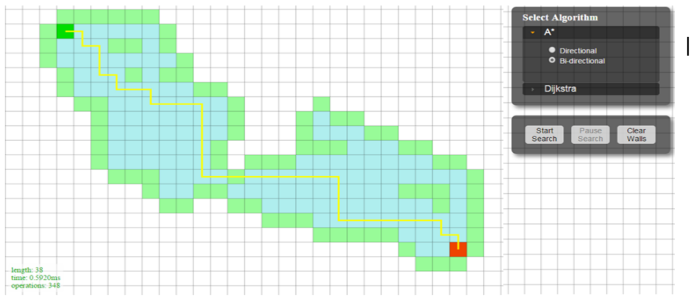
B) Dijkstra algorithm one directional



```
length: 38
time: 14.8760ms
operations: 1933
```

Figure 4: Dijkstra algorithm use one directional

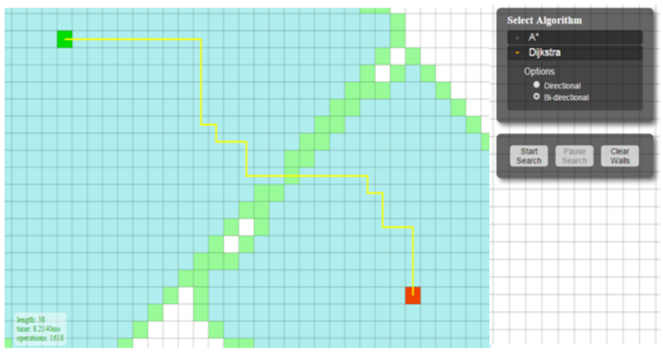
C) A* algorithm using Bidirectional



```
length: 38
time: 0.5920ms
operations: 348
```

Figure 5: A* algorithm using Bidirectional

D) Dijkstra algorithm use Bidirectional

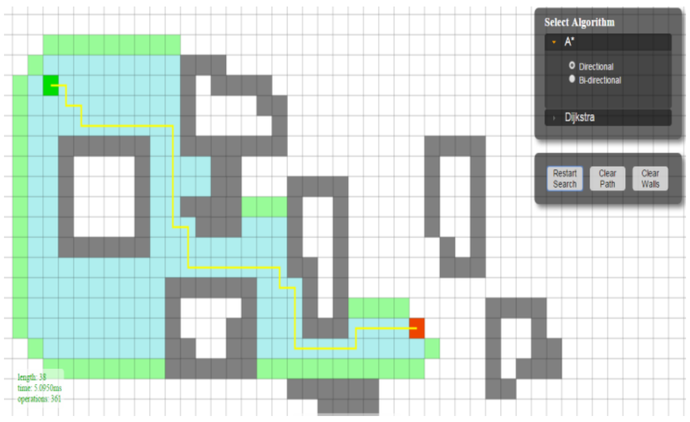


```
length: 38
time: 8.2140ms
operations: 1618
```

Figure 6: Dijkstra algorithm use Bidirectional

The obstacle problem: In many time we can search the path finding the lot of traffic is occur in that case the path searching is difficult .So we can get the solution which algorithm(A* or Dijkstra) is perform best in the obstacle as use traffic. The obstacle is find in the grid then the searching way is change and find the new path

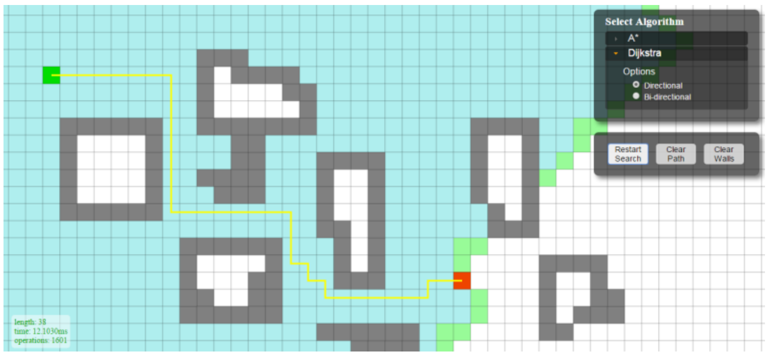
A) A* algorithm use one directional with obstacle



length: 38
time: 5.0950ms
operations: 361

Figure 7: A* algorithm use one directional with obstacle

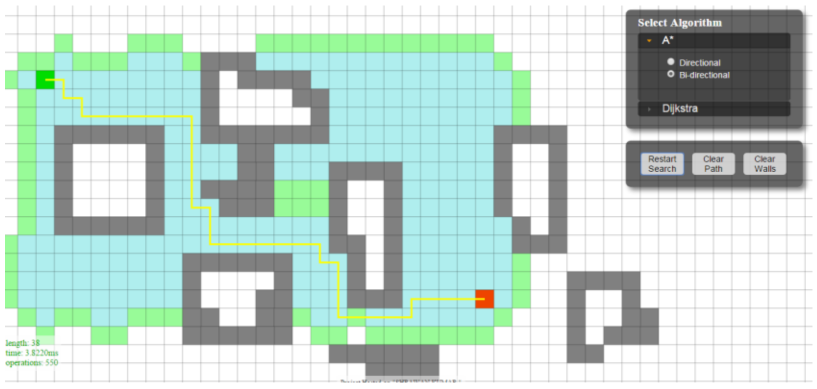
B) Dijkstra algorithm on directional with obstacle



length: 38
time: 12.1030ms
operations: 1601

Figure 8 Dijkstra algorithms one directional with obstacle

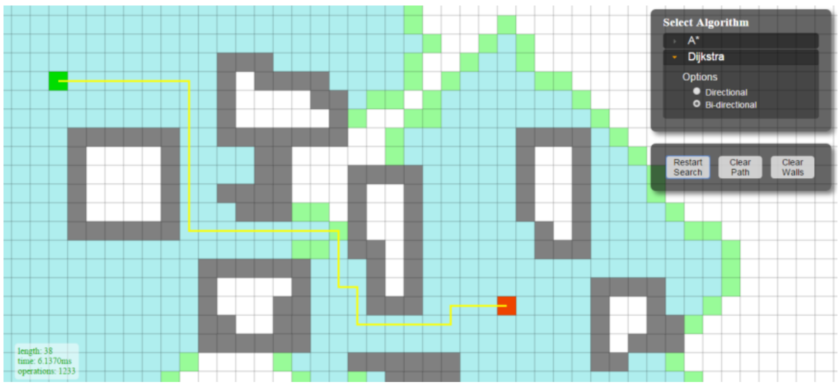
C) A * using Bi-directional with obstacle



length: 38
time: 3.8220ms
operations: 550

Figure 9 A * using Bi-directional with obstacle

D) Dijkstra algorithm Bidirectional use with obstacle


Select Algorithm
A*
Dijkstra
Options
Directional [✓ Bi-directional]

length: 38
time: 6.1370ms
operations: 1233

Figure 10 Dijkstra algorithm Bidirectional use with obstacle

V. RESULT ANALYSIS

Without obstacle: The result table show main difference of the time and number of operation. Time is very important for any searching manner so that this result show the A* algorithm is best performing in One or Bi-directional as per Dijkstra algorithm.

Algorithm	Condition	Length	Time	Operation
A star	One-Direction	38	3.8190	553
Dijkstra	One-Direction	38	14.8760	1933
A star	Bi-direction	38	0.5920	348
Dijkstra	Bi-direction l	38	8.2140	1618

Table 1: Compression of A* or Dijkstra without obstacle

With obstacle: The result table show main difference of the time and number of operation using in obstacle. We analysis that when the obstacle is occur in that case the A*algorithm performing much better than Dijkstra. So that A* algorithm performing better in all case as comparing Dijkstra algorithm

Algorithm	Condition	Length	Time	Operation
A star	One direction	38	5.095	361
Dijkstra	One direction	38	12.1030	1601
A star	Bi-direction	38	3.8220	550
Dijkstra	Bi-directional	38	6.1370	1233

.

Table 2: Compression of A* or Dijkstra with obstacle

CONCLUSION

According this study we can say that the A* algorithm is perform better then the Dijkstra algorithm in all the case (obstacle and without obstacle) except in uninformed search. .the A* algorithm has generated the best shortest path for other algorithm so that it is the best shortest path search algorithm.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor B.L. Pal for the continuous support of my study, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this paper. I could not have imagined having a better advisor and mentor for this paper.

Besides my advisor, I would like to Special thank Mr. Shiv Kumar and my Friends and well wisher: Bhupesh Sharma and Abhishek Sharma, for their encouragement, insightful comments, and hard questions. Last but not the least; I would like to thank my brother Nand Lal Sharma and my Wife Neelam Sharma supporting me spiritually throughout my life.

REFERENCES

[1] David Poole and Alan Mackworth. ,"ARTIFICIAL INTELLIGENCE FOUNDATIONS OF COMPUTATIONAL AGENTS"2010

[2]K.Rohila, P.Gouthami, Priya M,"Dijkstra's Shortest Path Algorithm for Road Network",IJIRCCE,Vol. 2, Issue 10, October 2014,ISSN(Online): 2320-9801

[3]Abhishek Goyal,Prateek Mogha,Rishabh Luthra,Ms. Neeti Sangwan,"PATH FINDING: A* OR DIJKSTRA'S?"IJITE Vol.02 Issue-01, (January, 2014) ISSN: 2321–1776

[4]Dominik Schultes,"Route Planning in Road Networks",page 22-29

[5][Kai Gutenschwager,Axel Radtke,"THE SHORTEST PATH: COMPARISON OF DIFFERENT APPROACHES AND IMPLEMENTATIONS FOR THE AUTOMATIC ROUTING OF VEHICLES",Proceedings of the 2012 Winter Simulation Conference,

[6] Fredman, Michael Lawrence; Tarjan, Robert E. (1984). Fibonacci heaps and their uses in improved network optimization algorithms. 25th Annual Symposium on Foundations of Computer Science. IEEE. pp. 338–346. doi:10.1109/SFCS.1984.715934.

[7] Sreekanth Reddy Kallem,"Artificial Intelligence Algorithms:,IOSRJCE) ISSN: 2278-0661, ISBN: 2278-8727 Volume 6, Issue 3 (Sep-Oct. 2012), PP 01-08

[8] Patrick Lester,"A* Pathfinding for Beginners",article apge 1-3

[9] Ben Coppin,"Artificial Intelligence Illuminated", A* algorithm, page 108-110,JONES AND BARTLETT PUBLISHERS,2004

[10] Pohl, Ira (1971), "Bi-directional Search", in Meltzer, Bernard; Michie, Donald, Machine Intelligence 6, Edinburgh University Press, pp. 127–140.

Author Profile

Shrawan Kumar Sharma is currently pursuing masters degree program {M.tech} in Computer science and engineering in Mewar University, India. His research interest includes soft computing, Mobile computing, DBMS, Network Security, Graphic programming, Fuzzy and Data Mining and etc.

B. L. Pal: working as an Assistant Professor in computer science and information technology department, Mewar University, Chittorgarh, India. He has done B. Tech. in Information Technology from AAI_DU Allahabad U.P. and his M.tech.(SIT) from DAVV Indore. He is also published some paper in international journals and has attended international conferences. He has more than three years teaching experience. His area of interest for research work includes Wireless Communication, Cryptography, and information technology.