ISSN: 0975-766X
CODEN: IJPTFI
Available Online through
Research Article
www.ijptonline.com

SHORTEST PATH FINDER USING GRID NORMALIZAITON

R. Mehta[1], K. Sanghvi[2], M. Thakore[3], M. Yamuna*[4]
[1,2,3]SCOPE, VIT University, Vellore-632014.
School of Advanced Science, VIT University, Vellore-632014.
Email: myamuna@vit.ac.in

Received on 19-05-2016 *Accepted on 25-06-2016*

Abstract

Pathfinding is a critical element of AI in many modern applications. The problem of Pathfinding in commercial applications has to be solved in real time, often under constraints of limited memory and CPU resources. The computational effort required to find a path, using a search algorithm such as A*, increases with size of the search space. Pathfinding on large maps can result in serious performance bottlenecks. In this paper, we will propose a method to find a path in grid converted map. Our method is automatic and does not depend on a specific topology. Both random and real-world maps are successfully handled using very little domain specific knowledge. The technique also has the advantage of simplicity and is easy to implement.

Keywords: Pathfinding, A* algorithm, Heuristic, Grid, Map, Maze, Shortest Route, Domain.

Introduction

Pathfinding or Pathing is the plotting, by a computer application, of the shortest route between two points [1]. It is a more practical variant of solving mazes. Today, Pathfinding algorithms are widely used in various fields of research and entertainment [6]. Some of the major fields are video games, exploration and industrial robots. Pathfinding generally refers to find the shortest route between two end points [1]. Examples of such problems include transit planning, telephone traffic routing, maze navigation and robot path planning. As the importance of game industry increases, Pathfinding has become a popular and frustrating problem in game industry. Games like role-playing games and real-time strategy games often have characters sent on missions from their current location to a predetermined or player determined destination [9].

The most common issue of Pathfinding in a video game is how to avoid obstacles cleverly and seek out the most efficient path over different terrain. Early solutions to the problem of Pathfinding in computer games, such as depth

first search, iterative deepening, breadth first search, Dijkstra's algorithm, best first search, A* algorithm, and iterative deepening A*, were soon overwhelmed by the sheer exponential growth in the complexity of the game. More efficient solutions are required so as to be able to solve Pathfinding problems on a more complex environment with limited time and resources.

Related works

Path-finding is commonly approached by a grid search technique. The world is abstracted as a graph model and searched, typically using some variant of A* algorithm. Common techniques often include those based on uniformly shaped grids, such as square or hexagonal tiles and quad trees or variable shaped tiles for adaption to more arbitrary terrain boundaries [1]. Heuristic roadmap information is widely used to further improve search efficiency. Hierarchical path-finding incorporates multiple graph or search-space decompositions of different granularity as a way of reducing search cost, perhaps with some loss of optimality.

Hierarchical information has been used to improve A* heuristics and proposed in terms of using more abstract, meta-information already available in a map, such as doors, rooms, floors, departments. Less domain-specific are graph reduction techniques based on recursively combining nodes into clusters to form a hierarchical structure [7]. Path-finding can also be based on the physics of dynamic character interaction.

In strategies based on potential fields or more complex steering behaviours a character's path is determined by its reaction to its environment [5]. This reactive approach can be combined with search-based models to improve heuristic choices during searching. There are many possible heuristics to exploit; in our implementations we use a "diagonal distance" metric to approximate the cost of unknown movement [7].

Proposed method

In the proposed method we aim to determine the shortest path between two locations. We approach this by using three steps. We now explain the three steps with suitable illustrations whenever needed. Next step is to find the path from start node to end node on the given grid. We select the most cost-optimal node as per the proposed algorithm. We recursively follow this step to reach the end/destination node. In the final step, we attempt to decrease the path length by applying superfluous path conditions.

1. Map to Grid Conversion

We first convert any given map into a grid system with four types of nodes:

Explored Nodes Explored nodes are already a part of the resultant path.

Open Nodes Open Nodes are the ones whose heuristic has been calculated and the most optimal open node, at the given state, becomes an explored node.

Blocked Nodes Blocked nodes are the ones through which we cannot move.

Closed Nodes Closed nodes are the ones whose cost has to be yet calculated.

We choose any random map, then this map is first fitted into a rectangular grid. Finally the rectangles including even the smallest part of any rectangular area is chosen to the grids for traversing to determine the shortest path. An illustration is seen in Fig 1.

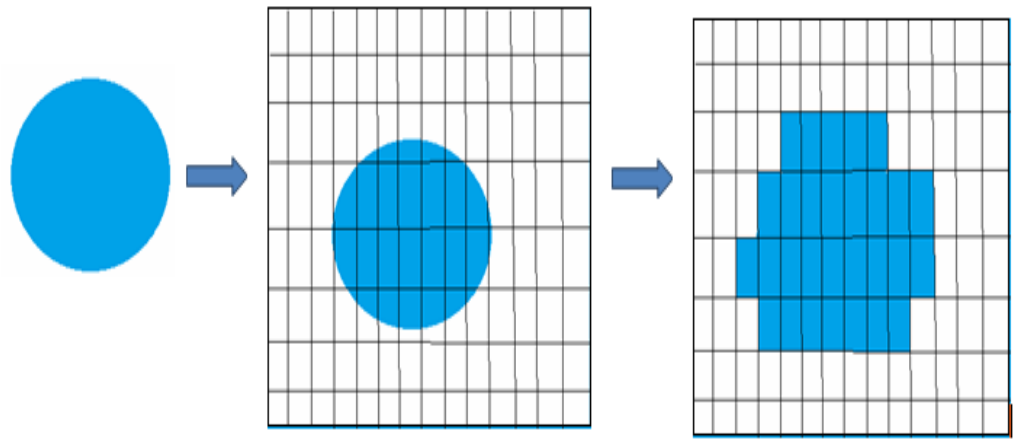


Fig 1: Map to Grid Conversion

Optimal Node Selection

- Choose the node with the least cost.
- **Low cost optimality collision rule 1** If there are at least 2 nodes with minimum cost among the group of open nodes, then choose the node farthest from the start node.
- **Low cost optimality collision rule 2** If the above two conditions fail, to get an optimal node then
- Form a set G of nodes which have filtered as optimal from the above two conditions.
- Open a node, given it's a neighbor to at least one node from the above set G
- Now among all these open neighboring nodes choose the optimal node as already done.
- Recursively follow these above steps till only 1 node emerges as the most optimal node.

Algorithm

1. Add the start node to the list of explored nodes.
2. Open all the 8 neighboring nodes of current node.

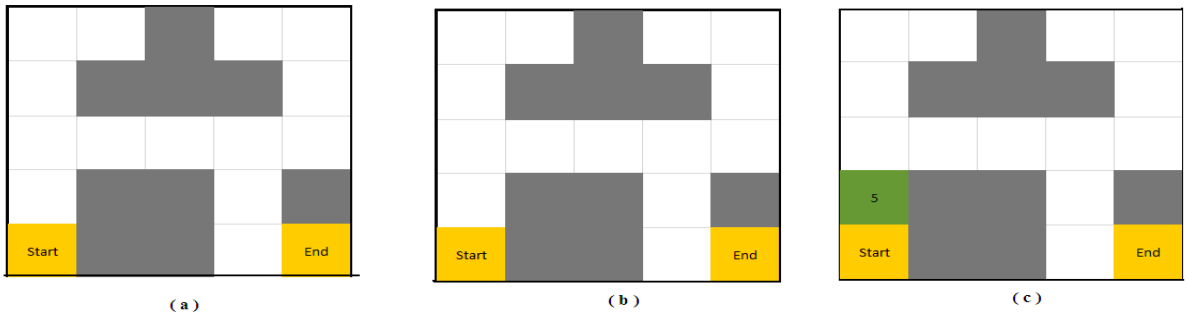
3. Calculate the cost for all neighboring nodes, the cost is the distance from the end node.

   Cost for a node (x, y) is: $|x - x_e| + |y - y_e|$, where (x_e, y_e) are the co-ordinates of end node. Choose the most optimal node and add it to resultant path and make it the current node.

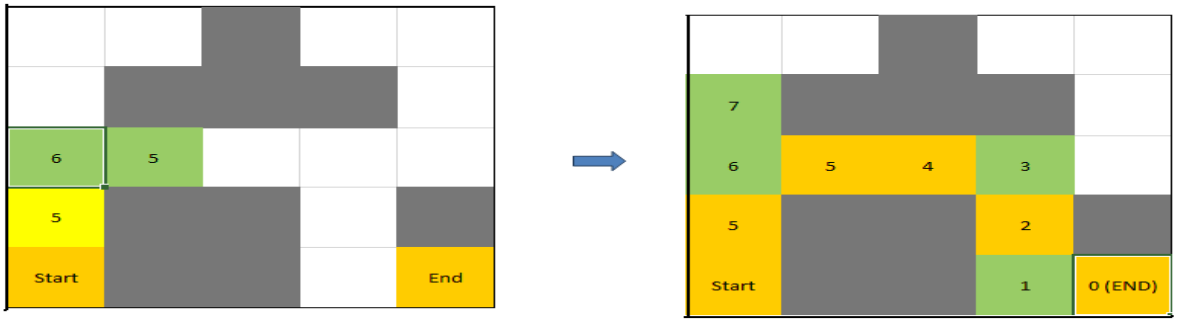
4. Repeat steps 2-4 iteratively until you reach the end node.

Illustration

We choose a 5x5 grid as sample with start and end node as seen in Fig 2(a). Now our current node is our start node. After exploring the grid we see that to reach the end node we traverse the shortest path as seen in Fig 2(b) which yields a value 5 seen in Fig 2(c)



On performing this step iteratively, we reach the final node. The different stages are seen in Fig 3.

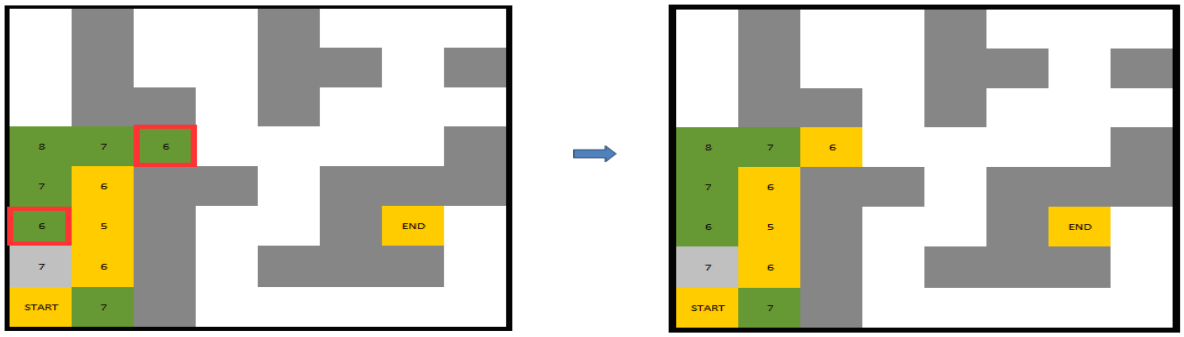


(a) (b)

Fig 3: Intermediate State and Final State.

Special Case Illustration

In the case in Fig 4(a) we see a case where more than one node with least cost is encountered (highlighted in red). To overcome this hurdle we apply the "low cost optimality collision rule 1". The result is as seen in Fig 4(b).



(a) (b)

Fig 4: Low cost optimality rule 1 case.

Thus on following the algorithm we get the path seen in Fig 5.

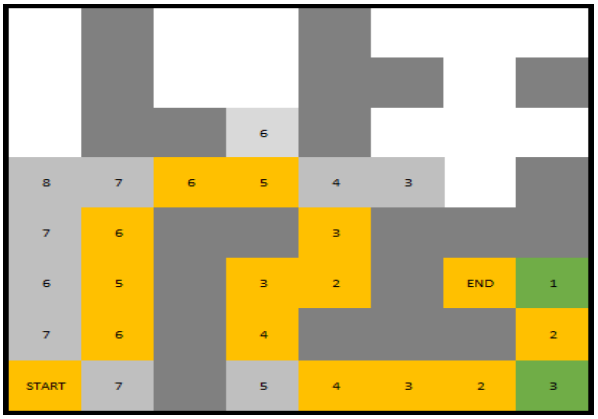


Fig 5: Result Path.

Removal of Superfluous nodes

Let us consider the path consists of n number of nodes (x_1, y_1); (x_2, y_2); ….(x_n, y_n)

From the right left iteratively select 3 adjacent nodes and check for this superfluous condition.

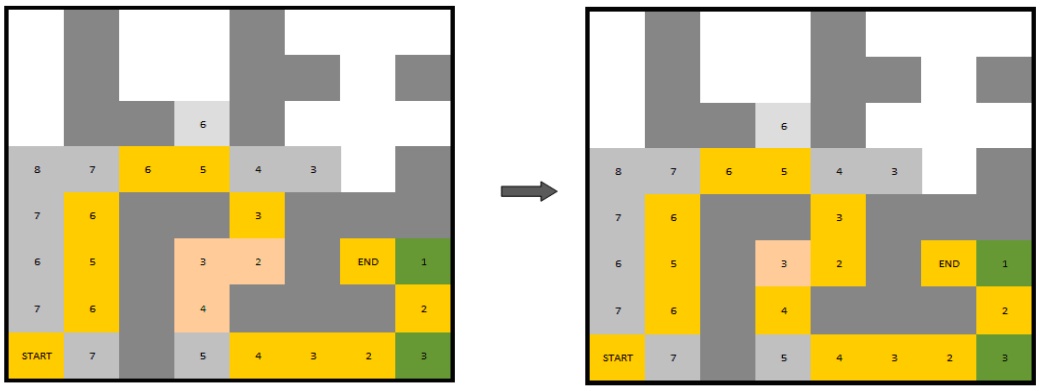
Assume we want to inspect whether nodes (x_{i-1}, y_{i-1}); (x_i, y_i);(x_{i+1}, y_{i+1})

a) $x_{i-1} = x_i != x_{i+1} \&\& y_{i-1} != y_i = y_{i+1}$

b) $x_{i-1} != x_i = x_{i+1} \&\& y_{i-1} = y_i != y_{i+1}$

If any of the above conditions is met then the center node is superfluous and can be eliminated from the resultant path.

There are peaks along the path, which has to be removed to optimize the path. The optimal path is seen in Fig 6(a) and Fig 6(b).



(a) (b)

Fig 6: Removal of superfluous node from path.

Conclusion

There are diverse algorithms for solving shortest path problem. The algorithm introduced in this paper has an advantage that it includes all the possible direction of motion that is up, down, left, right and diagonal. Our proposed

algorithm serves one of the best, as it considers the movement with respect to the priorities assigned to the directions, based on the finish point. It also has an advantage to calculate an alternate shortest path if available. So we can conclude that the suggested algorithm is efficient and time saving for calculating the shortest path between two locations.

References

1. Yachna Gupta, M Yamuna and Chiranjit Saha, Finding Shortest Path between two Locations Using Matrix Representaion (2014).
2. Jin Ho Kwak and Sungpyo Hong, Linear Algebra, Second edition, Springer (2004).
3. L.Fua,*, D. Sunb, L.R. Rilettc, Heuristic shortest path algorithms for transportation applications: State of the art, Computers & Operations Research 33 (2006) 3324-334.
4. UrI: http://www.vit.ac.in Arial view of the Vellore Institute of Technology, vellore campus map.
5. Sungchul Han and Sukchan Kang, Optimizing All-pairs Shortest-path Algorithm Using Vector Instructions.
6. Mohammad Reza SoltanAghaei, Zuriati Ahmad Zukarnain, Ali Mamat, Hishamuddin Zainuddin, A hybrid algorithm for finding shortest path in Network routing, Journal of Theoretical and Applied Information Technology-2005-2009.
7. G. Venkataraman, S. Sahni, and S. Mukhopadhyana," A blocked all-pairs shortest-paths algorithm", in Proc. Scandinavian Workshop algorithms and Theory,2000.
8. A. C. Mckellar and E. G. Coffman, Jr., "Organizing matrices and matrix operations for paged memory systems", Commun, ACM, vol. 12, issue 3, pp. 153-165, 1969.
9. Larsen E. Mcallister D.: Fast matrix multiplies using graphics hardware. Supercomputing, ACM/IEEE 2001 conference (10-16 nov. 2001), 43-43.
10. Han S. C., Kang S.-C.: Optimizing all-pairs shortest-path algorithm using vector instructions.http://www.ece.cmu.edu/peuschel/teaching/18-799B-CMU-spring05/material/sungchul-sukchan. pdf, 2005.

Corresponding Author:

M. Yamuna*,

Email: *myamuna@vit.ac.in*