Constraint Path Finding for Dijkstra Algorithm

Adeshpaul Singh Gill
Department of Computer Science
California State University
Fresno, California, United States
adeshpaulsinghgill@gmail.com

***Abstract*— Dijkstra's Algorithm is one of the popular algorithms in computer science it uses a greedy approach to find the shortest path between nodes. Over the years a lot of research work has been done on the algorithm to refine it. This paper presents one such refinement on the Dijkstra algorithm by constraining the search while finding the path. The proposed algorithm finds the shortest path between the start and the endpoint by evaluating the relative position of the endpoint from the starting point and then applying constraints based on it while searching for the path. The proposed algorithm's methodology and technique are described in detail along with the simulation results. The significance of the proposed method is also addressed and the conclusion for all of it is at the end of paper.**

***Keywords*—pathfinding, Dijkstra Algorithm, heuristic approach, constraint search, pathfinding**

I. INTRODUCTION

Pathfinding is basically plotting a feasible path between two points it can be based on criteria like a fastest, shortest or the longest path. Examples of such problems include traffic routing, maze navigation, robot path planning. Pathfinding has been one of the most studied problems in computer science and over the years have produced some incredible algorithms. Mainly pathfinding algorithms use two kinds of approaches in them heuristical or non-heuristical. The heuristical approach uses some kind of decision-making function when looking for the path which might not generate the optimal result but will guarantee a good result in reasonable time whereas the non-heuristical approach is blindly searching for the path by looking at all the options it guarantees an optimal result but is highly inefficient and expensive. Examples of heuristical search approach are A*, Best First Search while Dijkstra, Breadth First Search, Depth First Search are examples of the non-heuristical approach. The paper focuses on the Dijkstra Algorithm which was coined in 1956 by Edsger Wybe Dijkstra which uses a greedy approach for calculating the shortest path from a node to all the other nodes. Although it gives an optimal result bit is highly inefficient since it does a blind search as can be seen in Fig[1]. Considerable efforts have been made to optimize the algorithm and it has gone through tons of revisions. Some of the examples include optimizing the data structure, adding heuristics to find the optimal result like A* algorithm and many more. This paper presents one such way of improving the efficiency of the Dijkstra Algorithm.

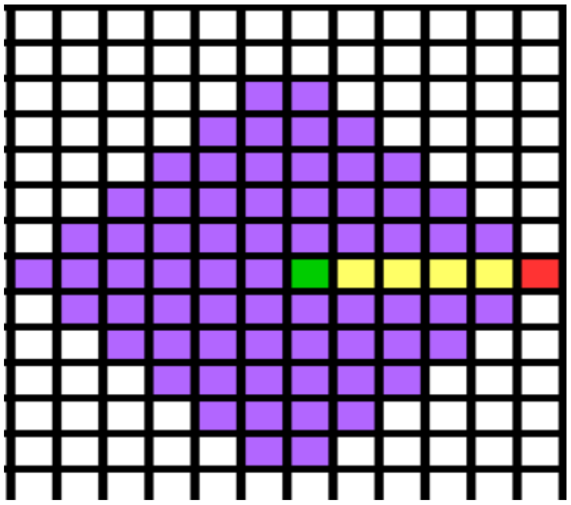
II. DIJKSTRA ALGORITHM

Dijkstra Algorithm specifically solves the single source shortest path where it starts from a node, and then it finds the shortest path from that node to every other node. So, if we have a goal node, we can keep track of the parents of each node and once we reach the goal node, we basically backtrack to trace the shortest feasible path between the start and the goal node. As you can see from Fig[1] which is a 2D representation of grid with each grid representing a node and each node having an edge of weight 1 going to top, bottom, left and right nodes. How Dijkstra works is it keeps track of every nodes distance from start node(green). Initially it sets all distance to infinity. Then it checks each node's neighbors(purple) and set a prospective new distance to equal the parent node plus the cost to get to the neighbor node. If the distance is less than the current neighbor's distance, we set its new distance and parent to the current node. Then it searches again from the nodes with smallest distance until it finds the goal state from where it backtracks to get the shortest path(yellow). The path generated by Dijkstra algorithm will always be an optimal path as can be seen in Fig[1] there is no alternate shortest path from green to red grid.

The traditional Dijkstra Algorithm has shortages as follow:

- It does a blind search thereby consuming a lot of time and resources.

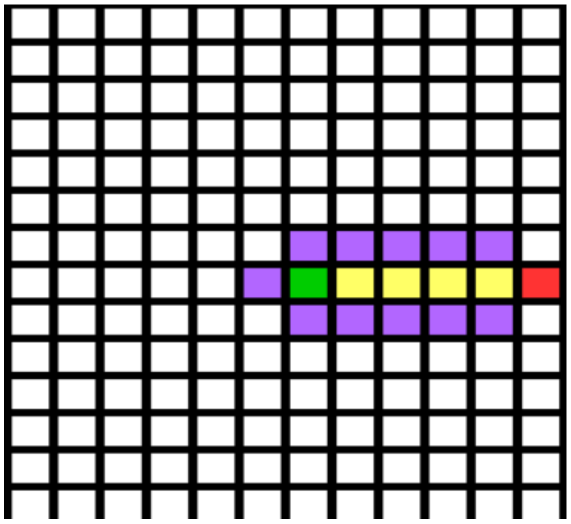
- It cannot handle negative edges since that lead to an acyclic graph which might not yield the optimal results.


Fig[1] 2D simulation of Dijkstra Algorithm

Pseudo Code of Dijkstra Algorithm

```
dist.[s] ← 0            (distance to source vertex is zero)
for all v ∈ V – {s}    (set all other distance to infinity)
    do dist[v] ← ∞
S ←Ø                    (S, set of visited vertices is initially
empty)
Q ← V                   (Q, contains all the vertices)
while Q ≠ Ø             (while Q is not empty)
do u ← minDistance(Q, dist)     (Select min distance in Q)
     S ← S U {u}                 (add u to visited vertices )
     for all v ∈ neighbors[u]
        do if dist[v] > dist[u] + w(u, v)  (if new short path
found)
             then d[v] ← d[u] +w(u, v)  (set value for it)
return dist           (return the shortest path)
```

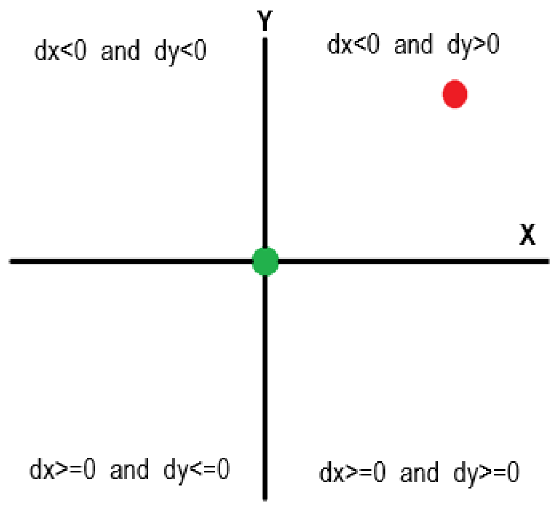
To improve the time efficiency of the Dijkstra algorithm heuristical approach was used which resulted in the making of A* algorithm Fig[2]. It uses a function $f = g + h$ where g is the distance between the current node and the start node and h is the heuristic-estimated distance from the current node to the goal node.


Fig[2] 2D simulation of A* algorithm

So, the f value helps the A* algorithm to find the shortest path more efficiently in a constrained way. This paper is aimed at improving the time and space efficiency of the Dijkstra algorithm by taking the inspiration from A* algorithm.

III. PROPOSED METHODOLOGY

The original algorithm has no constraint on the search, so from the start node, it starts searching in all directions(blind search) which although eventually guarantees a result but is highly resource extensive. So, what the proposed changes in the algorithm do is put a constraint on the neighbors that a current node can go to based on the location of the goal node, or the angle between current and the goal node. So, it uses a heuristical approach for decision making as to which neighbor to visit from the current node. It either calculates the angle that the goal node makes with the current node or simply calculate values dx and dy, this approach is basically used in a 2D representation where start node has values (x, y) on the grid and goal node has values (x', y'). $dx = x' - x$ and $dy = y' - y$ based on the values of dx and dy the search can be constraint to a quadrant as specified in Fig[3].



Fig[3] Four quadrants formed by intersection of X and Y

In the Fig[3] the current node(green) is assumed to be at the origin and end node(red) is the goal node so based on the values of dx and dy the search will be constrained to that quadrant only, this will improve the time and space required by the algorithm by a considerable amount since it will not have to blindly search in all the directions.

Pseudo code for the Modified Algorithm

- Initialization of all nodes with distance "infinite"; initialization of the starting node with 0
- Marking of the distance of the starting node as permanent, all other distances as temporarily.
- Setting of the starting node as active.
- Calculation of the temporary distances of all neighbor nodes of the active node by summing up its distance with the weights of the edges. *Apply the constraint methodology here***.**
- If such a calculated distance of a node is smaller as the current one, update the distance and set the current node as antecessor. This step is also called update and is Dijkstra's central idea.
- Setting of the node with the minimal temporary distance as active. Mark its distance as permanent.
- Repeating of steps 4 to 7 until there aren't any nodes left with a permanent distance, which neighbors still have temporary distances.

By applying the constraint method in step 4 the algorithm will work similar to Dijkstra but will search in a constraint way.

IV. IMPLEMENTATION OF THE MODIFIED ALGORITHM

Pathfinding algorithm usually works on graphs but to better understand the working you can represent the grids as graphs. For the paper grid representation is used to test the results of the algorithm.

Grid

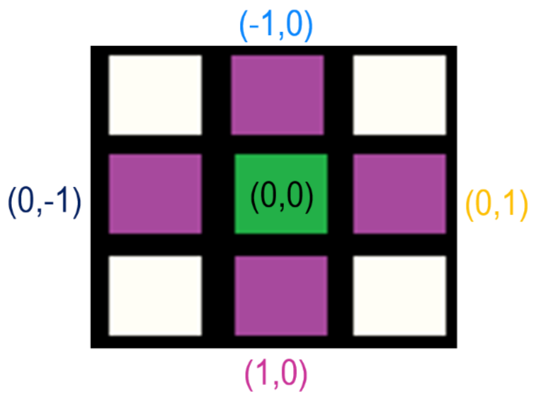
To test the outcome of the modification a simulation 2D model was developed in Python. The model consists of grids where each grid acts as a node and from each grid, you can visit the colored nodes as represented in the Fig[4]. From any node the movement possible are North, South, East and West. The diagonal movements are not considered as they yield a different result.

dx<0 and dy>0: neighbors set as (0,1) and (-1,0)

dx<0 and dy<0: neighbors set as (0,-1) and (-1,0)

dx>=0 and dy>=0: neighbors set as (0,1) and (1,0)

dx>=0 and dy<=0: neighbors set as (0, -1) and (1,0)



Fig[4] 2D representation of movement possible from a node

The grid pattern developed supports the use of obstacles and while searching for the path you have to go around the obstacle you can't go through the obstacle to reach the goal node. The path might not be feasible in cases where it is surrounded from all sides by obstacles. For the test results shown in the findings obstacles are not used.

Working

The proposed algorithm uses a direction based approach for pathfinding in a 2D grid, this approach can work for all the grid based environments like in maps or games.

The heuristic information needed is the relative position of the node with the end node which will help us designate the quadrant of search. Fig[4] gives the idea of how this representation will work as you can see the values to the right of the green node have increased value of y relative to its own y and a decreased value of y in case you go to the left, same is with the x values it increases when we go down from the current node and decreases when we go up.

So based on the relative values of x and y, dx and dy values are calculated for each node. Let's assume the current node has values (x, y) and the goal node has values (x', y'). So, the value of dx would be x'-x and for dy it would be y'-y. This will assign either a positive or a negative values to dx and dy and based on the Fig[3].

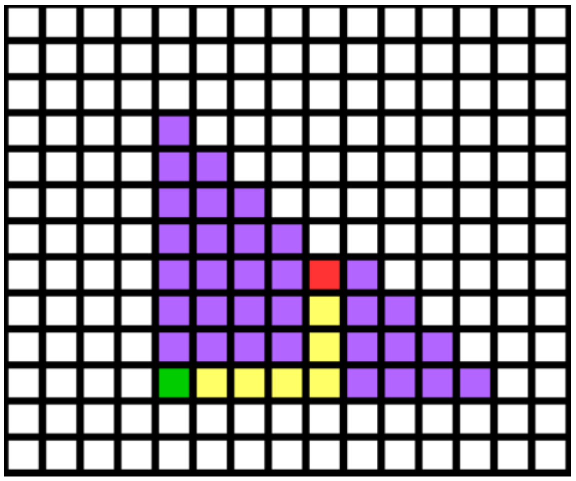
So based on the values of dx and dy if both are positive the search will be constrained in right and down direction if both the values are negative it will be constrained in up and left direction, for a positive value of dx and a negative value of dy it will be in the direction of left and down for negative value of dx and positive value of dy it will be constrained in up and right direction.

This approach will considerable improve the space complexity of the algorithm since we are constaring movement in only two directions and thus improving the overall performance as well.

The algorithm will still converge and hold the robustness properties just like the original one. The other approach can be the angle-based approach which can help improve the complexity for complex graphs.

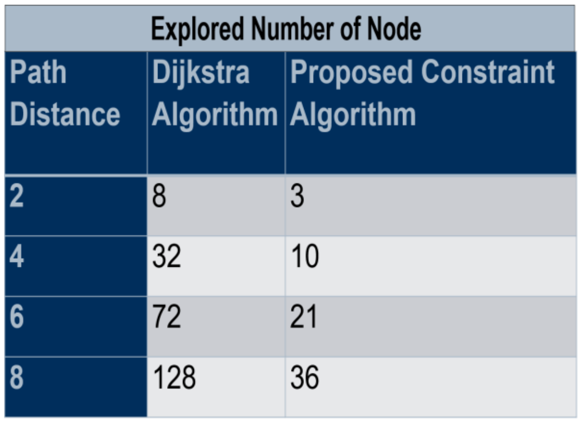
V. RESULT ANALYSIS

Results are based on the python implementation of the modified algorithm where start and end points are selected and then the algorithm is allowed to find the path between them. So, once you select the start and end node on the grid as in Fig[5]. The proposed algorithm will only explore the nodes visible colored rather than exploring blindly all the possible nodes.


Fig[5] visual representation of the algorithms working

The simulation model is further tested against the traditional Dijkstra Algorithm by selecting the same start and end nodes on the grid and results of the findings can be seen in Table[1].

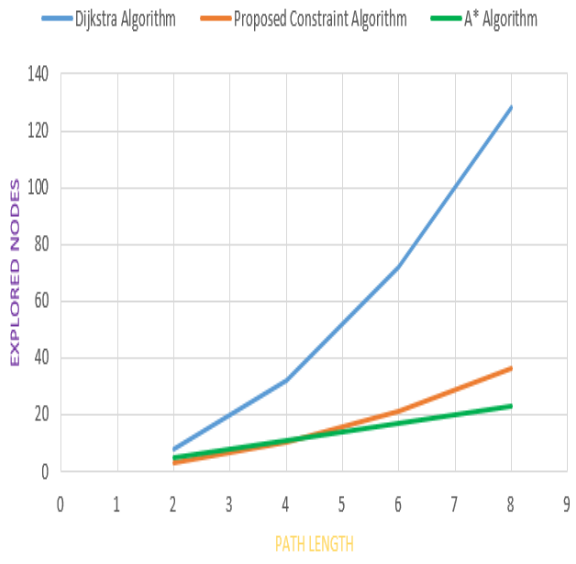
Table[1] Represents the number of Explored nodes based on the path distance



Explored Number of Node		
Path Distance	Dijkstra Algorithm	Proposed Constraint Algorithm
2	8	3
4	32	10
6	72	21
8	128	36

The modified algorithm results are also compared with A* simulation with the same settings of start and end node and results can be seen in Fig[6].

Table[1] and Fig[6] shows the comparison results where modified algorithm explores a comparatively fewer number of nodes than the Dijkstra algorithm, it works better than A* algorithm if the path length is less but A* gets better if the path length increases. The performance of the algorithm lies between that of Dijkstra and A* algorithm.



Fig[6] simulation results for Dijkstra modified and A* algorithm

Further improvements in time complexity are possible by using adequate data structures this implementation is solely determining the space complexity improvement of the algorithm.

VI. CONCLUSION

From the simulation model of the algorithm, we can conclude that the space complexity of the traditional Dijkstra Algorithm is significantly improved where the number of explored nodes in the modified algorithm is significantly less than the tradition algorithm with same environment settings. Also, the modified algorithm still yields the optimal result and holds the properties of convergence and robustness just like the traditional algorithm. Heuristic can be improved for the algorithm by choosing it smaller than the absolute value. This approach can be applied to 2-way search approach as well and that would be the next step for research.

ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to Dr. Matin Pirouz Nia for the guidance, suggestions, and words of encouragement.

REFERENCES

[1] P. W. Eklund, S. Kirkby and S. Pollitt, "A dynamic multi-source Dijkstra's algorithm for vehicle routing," *1996 Australian New Zealand Conference on Intelligent Information Systems. Proceedings. ANZIIS 96*, Adelaide, SA, Australia, 1996, pp. 329-333.

[2] Zeng, Mingbin & Yang, Xu & Wang, Mengxing & Xu, Bangjiang. (2018). Application of Angle Related Cost Function Optimization for Dynamic Path Planning Algorithm. Algorithms. 11. 127. 10.3390/a11080127.

[3] http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf

[4] https://medium.com/@nicholas.w.swift/easy-dijkstras-pathfinding-324a51eeb0f

[5] https://www.researchgate.net/profile/Xiao_Cui7/publication/267809499_A-based_Pathfinding_in_Modern_Computer_Games/links/54fd73740cf270426d125adc.pdf

[6] Geethu Elizebeth Mathew "Direction Based Heuristic for Pathfinding In Video Games", in press.