C-Theta*: Cluster based Path-Planning on Grids

Pramod Mendonca
School of Computer Science
University of Windsor
Windsor Ontario, Canada
mendoncp@uwindsor.ca

Scott Goodwin
School of Computer Science
University of Windsor
Windsor Ontario, Canada
sgoodwin@uwindsor.ca

Abstract— **Path planning is used to solve the problem of moving an agent towards a destination. Theta* is a well know any angle path planning algorithm which works by utilizing line of sight checks during the search. To find shorter paths that are not constraint to grid edges, there is a compromise in the time taken to reach the destination which makes Theta* undesirable as the grid map size increases.**
To solve this problem and enhance the search performance we propose a method which divides a map into high and low density regions using an unsupervised clustering algorithm based on the number of blocked nodes on a grid map.
After comparing the proposed model with theta* the results show the time taken to find the shortest path to be reduced significantly in comparison with Theta* while the path length will remain as short as Theta*.

Keywords—path finding; games; clustering; path planning;

I. INTRODUCTION

Path planning is one spoke in the wheel of artificial intelligence with the aim of finding the shortest path between two points on a given grid map**.** It is applied in a lot of domains such as robotics, logistics and computer games. Path planning introduces and tries to solve many challenges faced while trying to plot the most optimal and desirable path from a source to a destination, for example in a computer game the path for an agent to traverse would need to be generated in milliseconds if not nanoseconds. Consider Age of Empires a strategy game for which paths have to be plotted for all agents in the game, AOE (Age of Empires) has a player limit of 8 and each player has an army of 200 agents, paths have to be plotted and planned for each of these agents which totals to 1600 (worst case) which is an extremely intensive CPU task.
Dynamic changes to the terrain need to be compensated while plotting an optimal path, for example if a wall is an obstacle in the search space it is represented as a blocked region, removing this wall leads to changing the label to unblocked. Passing information among agents to avoid collision among themselves while plotting the path from source to destination is another problem. Another area is path smoothing and elimination of heading changes in free space to provide a realistic feel of an agent traversing from source to destination.
To solve many of these problems the most robust and commercially used path finding algorithm is A* and variant of A* like D*Lite which is used to find paths dynamic environment[4] or JPS(Jump Point Search) which is based on pruning techniques to find the path[3].
To solve the path finding problems the real world environment is usually represented as a grid. They are the most popular data structures used to test path finding algorithms because they are not complex and can be generated quickly unlike nav- mesh and waypoint which are complex and need hand tuning. Path finding algorithms makes use of the information provided by the underlying grid which is a special case of a graph to traverse through the environment, Traditional path finding algorithms like A* restricts the path movement of an agent along the grid edges which translates the movement of an agent from one tile to the next on a grid to 4-way or 8-way.Though these paths are optimal they are not the true shortest paths as shown in figure 1.1[1].
This restriction led to the introduction of any angle path planning on grids.

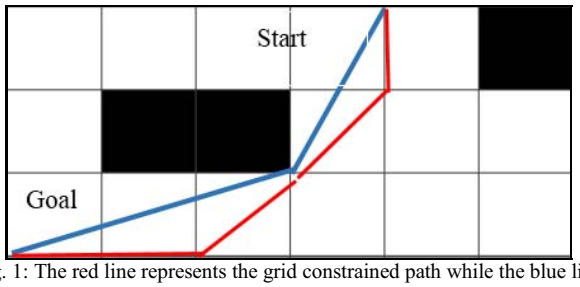

Fig. 1: The red line represents the grid constrained path while the blue line represents the true shortest path.

One of the earliest algorithms to address this issue was Field D*[2] and then Theta*[1]. We focus our research on Theta* which uses line of sight checks to improve path quality and remove the restriction of traversal along grid edges it always provides a path shorter than A* but takes more time than A* and this increases as the grid size increases.
In this paper we introduce C-Theta*, this is a variant of Theta* which tries to maintain the properties of its parent

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.92

and simultaneously improve the computational time, to achieve this C-Theta* uses additional information provided by clustering regions into high and low density areas based of the number of blocked nodes on the given grid map while performing the search from source to destination.

The rest of this paper is organized as follows. In the next section related work in the field of any angle path finding is briefly reviewed. In section 3 we introduce our proposed algorithm in detail. In section 4 the proposed will be compared with other algorithms, and our conclusions will be discussed in last section.

II. Related Work

Several variants of A* have been developed to address the issue of any angle path finding using line of sight checks to determine the path ahead and unlike A* which considers the center of a respective tile of a grid, in any angle path finding the grid corners are considered for node expansion from source to destination.

A* evaluates a nodes desirability based on an evaluating function which takes into account the actual cost and estimated cost to determine the best step to take towards a goal node. Fig: 2 displays the pseudo code for A*. [5][1]

```
Main()
S (start) := 0; (Start Node)
parent := S(start);
open := 0;     /*(Open list : = Set of node under consideration)*/
f(S(start)) := g(S(start))+ h(S(start));  /*(g(s) := Actual cost, h(s) := heuristic cost,
                                  f(s) = Total estimated cost to
                                  reach goal) */
open.Insert(S(start),f(S (start)));
closed := 0; /*( Closed List := Set of nodes already evaluated) */
while open != 0 do
  S := open.pop();
   If S == S(goal) then
     return "Path  Found" ;
closed := closed U {S}
 for each S' ∈ neighours (s) do
    if S' ∉ closed then
       if S' ∉ open  then
          g(S')  := ∞;
          parent(S') := NULL;
    NodeValue(S, S');
return "path not found";
end;
NodeValue(s, s')
       If g(s) + c(s,s') < g(s') then  /*(where c(s,s') cost to move
                                 from s to successor node s')*/
             g (s') := g(s) + c(s,s');
             parent (s') :=  s;
                if s' ∈ open  then
                  open.pop(s');
             open.push(s',f(s'));
end;
```

Fig. 2: Pseudo - Code A*.

Theta* is another path finding algorithm which is based on A*. To eliminate the restriction of traversal along grid edges it uses line of sight checks. It works by connecting nodes on a grid until it reaches a convex corner. (A corner is turning point along a blocked node.). The only difference in the algorithm is in the NodeValue method of the A* algorithm pseudo code defined above which considers two decisions while making a move from one node to the next to reach the goal node.

For the first decision, assume a node‘s’ under evaluation during the search process. Theta* considers the path from the start node to the parent of node s and from node s’ to the parent of node s in a straight line i.e. c (parent(s), s’) = cost of travel from parent(s) to s’. Where the actual cost travel is g (parent(s) + c (parent(s), s’)). This decision makes it possible for any angle traversal along a grid map. [1]

In the second Decision, the path is similar to the path considered by A* algorithm i.e. from a node‘s’ to s’ in a straight line. Resulting in the actual cost of traversal g(s) + c(s, s’). [1]

This allows a node s’ whose parent is not anchored to its predecessor. Figure 3 explains the pseudo-code of Theta*.

```
NodeValue(s,s')
If LOS (parent(s),s') then
/* Decision – 1 */
   If(g(parent(s))+c(parent(s),s')) < g(s') then
      g(s') := g(parent(s)) + c(parent(s),s');
      parent (s') = parent(s);
        if s' ∈ open then
          open.pop(s');
     open.push(s', f(s'))
else /*Decision -2 */
If g(s) + c(s,s') < g(s') then  /*(where c(s,s') cost to move
                                 from s to successor node s')*/
       g (s') := g(s) + c(s,s');
       parent (s') :=  s;
         if s' ∈ open  then
          open.pop(s');
       open.push(s',f(s'));
end;
```

Fig. 3: Pseudo-code for Theta*

Theta* always finds paths that are marginally longer that the actual shortest path and are shorter than A* but the time taken is more than A*.

Lazy Theta* to reduce the number of line of sight checks a lazy initialization technique is introduced which performs one line of sight check per expanded node unlike Theta* which performs line of sight checks for every unexpanded visible node. [6]

Anya Any angle path finding algorithms are constrained to traversal along grid edges but are not optimal, to introduce an online optimal any angle path finding algorithm ANYA was introduced. ANYA considers different states or intervals to reach the goal where a point is considered to represent the

f- value of a set of points. For each interval or state a representative f- value is calculated until the goal is reached. Which makes ANYA an optima any angle algorithm. [7]

To improve the performance of any angle algorithms a variant sub goal graphs were introduced which considers points on the corners of the grid and rather than the center and a point is a sub goal if and only if it lies on the convex corner a blocked node. [8] Also to improve the performance pre computed paths are stored in a database thus improving the computation time of the algorithm which was successfully implemented in Block A*[9].

III. PROPOSED APPROACH

As discussed earlier, Theta* finds paths that are shorter that A* but the time taken by Theta* to find the path from source to destination is almost twice the time taken by A*.This makes Theta* undesirable as the map size increases especially in modern day computer game maps which are growing in size as the years increase.

To address this problem we come up with a novel approach described in Fig. 4 where we have abstracted the map into regions and used the blocked areas in the map to help the algorithm decide which regions are desirable for line of sight checks and which are not. This is done by assigning a label of high and low density to region based on number of blocked tiles in them. If a region has a low density label the algorithm use A* path finding and otherwise it uses theta*.

For making region, we divide the grid into fixed size regions based on number of tiles. For example if a grid has 100 × 100 dimension and region size is 5 the algorithm will create 20 regions of 5 × 5 throughout the grid.

To decide which region is high or low density, a clustering algorithm must be integrated. Since prior data about the map and its features are not provided the supervised learning approach cannot be used. Thus an unsupervised learning approach must be implemented to perform this task. After comparing the various unsupervised clustering techniques, K-Means is selected because it is efficient, fast and perfectly suits the problem of labeling regions in the map.

We supply K-Means the regions of the map with the number of obstacles in them and it provides us with the result of labeling regions into high and low density. In fact, the input for K-Mean will be an array of the number of obstacles in each region. The output of this process will provide us two clusters one representing high and the other representing low density regions which are mapped to the regions abstracted from the map.

This information is now given to the search which decides when to perform a line of sight check thus creating an on demand line of check criteria.

In the online stage we consider the following decisions made by our algorithm

Decision 1: If the node belongs to a region which is considered a high density region we forego the line of sight check and expand the nodes just like A*.

Decision 2: If the node belongs to a region which is considered a low density region we perform a line of sight check and expand nodes just like Theta*.

For example in figure 5.a if an agent wants to find a path between blue and red points. The algorithm first divides the map into 9 regions, R1-9. Then based on the K-means, it calculates the density of these regions. As an instance R1 is high density denoted by R1-H. Since the source node lies in the low density region (R4), as shown fig 5.b, theta* is used as well as in R5 and R6 which are low density regions. As the goal lies in R9 which is the high density region and the path plotted will be A* path.

```
Main()
ConvertToRegions
do K-Means Clustering & Label Regions
/*start*/
S (start) := 0; (Start Node)
parent := S(start);
open := 0;      /*(Open list : =  Set of node under consideration)*/
f(S(start)) := g(S(start))+ h(S(start),); /*(g(s) := Actual cost, h(s) := heuristic cost,
                                            f(s) = Total estimated cost to
                                            reach goal) */
open.Insert(S(start),f(S (start)));
closed := 0; /*( Closed List := Set of nodes already evaluated) */
while open != 0 do
  S := open.pop();
    If S == S(goal) then
      return "Path  Found" ;
closed := closed U {S}
 for each S' ∈  neighours (s) do
     if S' ∉ closed then
         if S' ∉ open  then
            g(S')  :=  ∞;
            parent(S') := NULL;
     NodeValue(S, S');
return "path not found";
end;
NodeValue(s, s')
         If DetermineNodeRegion(s') then/*Low density Region*/
         If LOS (parent(s),s' ) then
           If(g(parent(s))+c(parent(s),s')) < g(s') then
                   g(s') := g(parent(s)) + c(parent(s),s');
                     parent (s') = parent(s);
                           if s' ∈ open then
                                   open.pop(s');
                    open.push(s', f(s'))

else

         If g(s) + c(s,s') < g(s') then  /*(where c(s,s') cost to move
                                  from s to successor node s')*/
                   g (s') := g(s) + c(s,s');
                   parent (s') :=  s;
                    if s' ∈ open  then
                           open.pop(s');
         open.push(s',f(s'));
end
```

Fig. 4: Pseudo-code C-Theta*

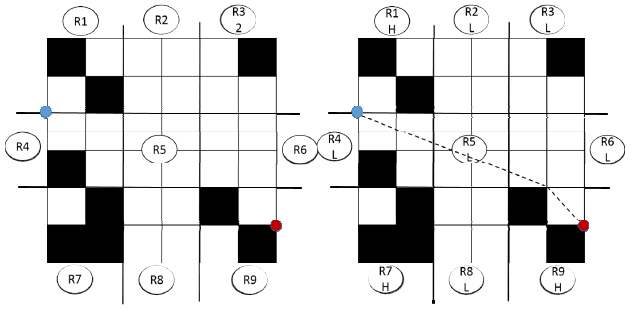

Fig. 5a: Regions created on a grid map. 5(b) represents the labelled regions after clustering and the dotted line represents the path from source to destination based on C-Theta*

IV. RESULTS

The experiments are conducted using 100 × 100 and 50 × 50 grids with random obstacles. The obstacle density of the grids maps is 20% and 50 % respectively. The maps are generated using a JAVA path finding framework which has been extended to support any angle path finding algorithms.
The experiments are conducted on a Lenovo ThinkPad X201 tablet with windows 7 64- bit system with 8 GB RAM and an Intel i7 processor. The heuristic used in all the algorithms is Euclidean distance. In this experiment the region size was fixed to 10
Table 1 compares the results of C-Theta* with A* and Theta*.

TABLE 1: ALGORITHM RESULTS AND PERFORMANCE

Grid Map Data	**Algorithms**	**Path Length**	**Time(s) ms**
Obstacle Density 20%	A*	67	5.04
	Theta*	60.8	10.99
	C-Theta*	**61.7**	7.78
Obstacle Density 50%	A*	56.8	7.16
	Theta*	54.2	15.08
	C-Theta*	**55.3**	10.09
100 × 100			
Obstacle Density 20%	A*	117.5	12.12
	Theta*	110.8	19.02
	C-Theta*	111.5	16.13
Obstacle Density 50%	A*	107.5	7.57
	Theta*	106.1	11.6
	C-Theta*	**106.1**	10.6
100 × 100			
Maze	A*	172.8	22.79
	Theta*	158	53.37
	C-Theta*	163.5	**31.95**

C-Theta* reports shorter paths A* and marginally longer paths than theta*.In some cases it also reported path length similar to Theta* in the result for grid size 100 × 100 of obstacle density 20% C-Theta* reported a path length equal to Theta*.
The path length of C-Theta* when compared with Theta* reports marginal degradation. Also there is an average 20 percent improvement in the time taken to find the path from the source to destination. Also there is only on average 1% path length degradation. In mazes the time taken by theta* when compared with A* is more than double, when Comparing theta* with C-theta* we see an almost 40% improvement in the time taken and also a shorter path than A*.

V. CONCLUSION

As game maps become more complex, detailed and the increase in game map size brings with it new challenges to optimize path planning algorithm. We have successfully shown that clustering can be used improve the information gain of a map and this can be used while performing the search in the case of C-Theta*. C-Theta* also exhibits its on demand line of sight checks can be successfully implemented to optimize theta*.

VI. FUTURE WORK

C-theta* has been applied to a static environment we would like to see if the same solutions can be applied to a dynamic environment and observe the results. Introduce a new technique of preforming LOS in time intervals to improve the performance theta*.

REFERENCES

[1] A. Nash, “Theta*: Any-Angle Path Planning for Smoother Trajectories in Continuous Environments,” AI Game Dev, 08-Sep-2010. [Online]. Available: http://aigamedev.com/open/tutorials/theta-star-any-angle-paths.
[2] Ferguson, D., and Stentz, A. 2006. Using interpolation to improve path planning:The field D* algorithm. Journal of Field Robotics 23(2):79–101.
[3] S. Koenig and M. Likhachev, “D*lite,” in Eighteenth national conference on Artificial intelligence, Menlo Park, CA, USA, 2002, pp. 476–483.
[4] “Jump Point Search,” Shortest Path. [Online]. Available: http://harablog.wordpress.com/2011/09/07/jump-point-search/. [Accessed: 10-Apr-2013].
[5] Hart, P. E., N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths in Graphs," IEEE Trans. on Systems Science and Cybernetics, Vol. SSC-4, No. 2, pp 100-107, (July 1968); also in Context-Directed Pattern Recognition and Machine Intelligence Techniques for Information Processing, Y-H Pao and G. W. Ernst (eds.) IEEE Computer Society Press (Silver Spring, MD., 1982).
[6] A. Nash, S. Koenig and C. Tovey, “Lazy Theta*: Any-Angle Path Planning and Path Length Analysis in 3D,” In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2010
[7] Daniel Harabor and Alban Grastien “An Optimal Any-Angle Pathfinding Algorithm” In Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, 2013.
[8] T. Uras and S. Koenig, “ Speeding-up Any-Angle Path-Planning on Grids” , In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 234-238, 2015
[9] Peter Yap and Neil Burch and Rob Holte and Jonathan Schaeffer, “Block A*: Database-Driven Search with Applications in Any-Angle Path-Planning” Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (2011).