1-13-2016

C – THETA* : PATH PLANNING ON GRIDS

Pramod Mendonca
University of Windsor

C – THETA* : PATH PLANNING ON GRIDS

By

PRAMOD MENDONCA

A Thesis

Submitted to the Faculty of Graduate Studies
Through the School of Computer Science

In Partial Fulfillment of the Requirements for
the Degree of Master of Science at the

University of Windsor

Windsor, Ontario, Canada

2016

C – Theta* : Path Planning on Grids

By

Pramod Mendonca

APPROVED BY:

_____

Dr. James Gauld, External Reader
Department of Chemistry & Biochemistry

_____

Dr. Dan Wu, Internal Reader
School of Computer Science

_____

Dr. Scott D Goodwin, Advisor
School of Computer Science

Jan 6, 2016

DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION

I. Co-Authorship Declaration

I hereby declare that this thesis incorporates material that is result of joint research, as follows: I am the sole author of Chapter 1, Chapter 2 and Chapter 4 up to Chapter 7. In Chapter 3 the proposed approach is co-authored with my supervisor Dr. Scott Goodwin.

In all cases, the key ideas, primary contributions, experimental designs, data analysis and interpretation, were performed by the author, and the contribution of co-authors was primarily through the provision of proof reading of the published manuscript.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Declaration of Previous Publication

This thesis includes 5 pages of original papers that have been previously published/submitted for publication in peer reviewed journals, as follows:

Thesis Chapter	Publication title/full citation	Publication status*
Chapter 3 – Proposed Approach	*P.Mendonca and Scott D.Goodwin "*C-Theta*: Cluster based Path-Planning on Grids*",* 2015 International Conference on Computational Science and Computational Intelligence	*Published*

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Finding the shortest path between two points on a given grid map is called path finding. Many algorithms have been devised, but the most widely used and efficient is A*. Theta* is an any-angle algorithm that finds shorter and more realistic paths when compared with A*.

Theta* is an any angle path planning algorithm which works by utilizing line of sight checks during the search to find shorter paths due to which the algorithms takes considerable amount of time to find the goal as the map size increases.

To solve this problem C – Theta* is proposed, It utilizes the concept of clustering to improve the search performance by implementing on-demand line of sight checks, this improves the time taken by C – Theta* to find the goal by at least 20% when compared with Theta* and the paths plotted are as short as Theta* and no longer than A* algorithms.

DEDICATION

To my loving family:

Mother: Late. Shalini Mendonca

Father: Marshal Mendonca

Siblings: Satish, Juliet and Rahul Mendonca.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Scott D Goodwin, for accepting my application as a Masters student at the University of Windsor and providing me with the opportunity of work in the field of Artificial Intelligence. His guidance, support and research conversations helped me a lot in understanding path finding and its concepts.I would like to thank my thesis committee comprising of Dr. Dan Wu and Dr James Gauld for their time and suggestions which has helped me in completing my thesis and being there for my proposal and defense.

I would like to thank Mrs. Karen Bourdeau, for her support and assistance without which I wouldn't be able to plan and execute my thesis in a timely manner.

Finally, I would like to thank my family for providing me the encouragement and support for the last two years being a student at the University of Windsor and my friends for being there with me on this incredible journey.

TABLE OF CONTENTS

DECLARATION OF ORGINALITY/ PREVIOUS PUBLICATION……iii

ABSTRACT……..…….……..iv

DEDICATION……...……...v

ACKNOWLEDGEMENTS……...……....….vi

LIST OF TABLES……...…….……..xii

LIST OF FIGURES……..……..…xiii

LIST OF ALGORITHMS……..…….……..….xv

LIST OF ACRONYMS & DEFINITIONS……..……..……..xvi

CHAPTER-1 INTRODUCTION…….…….……..1

1.1 Problem Domain……..…….……1

1.2 Components of Path finding.....…….……...…….2

1.2.1. Spatial Representation……..……2

1.2.2. Search Algorithms ……...……..…….……..5

1.2.3.Heuristics ……..……6

1.3 Contribution of this Thesis…….…..……..…8

1.4 Organisation of this Thesis…….…….…....10

CHAPTER - 2 BACKGROUND……..……...……...…...….11

2.1 A* ……..…….….....12

2.1.1 Evaluation of *f(n)*……...….……...14

2.1.2 Optimality and Completeness…….…14

2.2 Field D*……..……..……15

2.3 Theta* ……..……..……..17

2.4 Lazy Theta* ……..……..….....20

2.5 ANYA ……..…….....22

2.6 Block Guided Theta* …….……...……25

CHAPTER - 3 PROPOSED APPROACH…….…….…….....29

3.1 Motivation ……...……..…….……..…….....30

3.2 Clustering ……..……..……..……31

3.2.1 K - Means …….…….…….....32

3.3 C - Theta* …….…….…….…..35

3.4 C - Theta* Vs Theta*…….……..…….….....41

CHAPTER-4 EXPERIMENTAL SETUP…….…….……..44

4.1 Search Space representation and Heuristic …….…..…….44

4.2 Assumptions ……...…….45

4.3 Algorithms ……..…….....46

4.4 Process of Testing …….…….……46

4.5 Software and Hardware Environment …….…….…….47

CHAPTER - 5 EXPERIMENTAL ANALYSIS..……..……...48

5.1 Runtime Analysis …….…….…….…...48

5.1.1 User Designed Maps ……..……....……48

5.1.2 Random Maps ……...…..51

5.2 LOS Analysis ……..…….…53

5.3 Region Based Analysis …….…….……..56

5.4 Path Length Analysis ……..…..…….…..57

5.5 Standard Deviation ……...…….60

CHAPTER - 6 CONCLUSION…….……..……....……...…63

CHAPTER - 7 FUTURE WORK……..……...…..……65

REFERENCES……..……..…66

APPENDIX…….....68

VITA AUCTORIS……..……..…....72

LIST OF TABLES

1 Runtime results for the algorithms with varying obstacle density in map size 100 × 100……..……..……...49

2 Runtime results for the algorithms with varying obstacle density in map size 200 × 200……..51

3 Runtime results for the algorithms with varying obstacle density in map size 100 × 100…….……51

4 Runtime results for the algorithms with varying obstacle density in map size 200 × 200……...……..53

5 Avg Line of Sights for grid maps of size 100 × 100. …….……54

6 Avg Line of Sights for grid maps of size 200 × 200……54

7 Standard Deviations for user designed maps of size 100 × 100……60

8 Standard Deviations for random maps of size 100 × 100……..……60

9 Standard Deviations for user designed maps of size 200 × 200…….....61

10 Standard Deviations for random maps of size 200 × 200. ……61

LIST OF FIGURES

1.1 (a): The real world represented by N- sided polygon navmesh ……3

(b) The real world represented as a N- sided polygon navmesh with centroids placed in a convex polygon to plot the path from the node 's'(start) node 'g' (goal).…….……..……...…….....3

1.2 a): A real world representation of the environment with waypoints……4

b) Waypoint Graph…….…….....4

1.3 Spatial representation of a map using grid structure……..……4

1.4 Grid representation and the type of movement possible on a grid……5

1.5 Different methods to plot a path on grid map……...5

1.6 A grid map with start and goal node and the path plotted

using Manhattan distance heuristic ……..…..……..7

1.7 A grid map with start and goal node and the path plotted

using Euclidean distance as heuristic……...….....8

2.1 Before and after path plotted using string pulling……..…….....11

2.2(a) Spatial representation of the search space…….….….13

2.2(b) Pictorial representation of the A* algorithm search process…….…..13

2.3(a): Field D* corner vertices expansion[Stentz et al.] ..…….…….. 16

2.3(b) The traversal cost from node 's' utilizing the path cost of s_1 and s_2

the traversal cost 'c' and traversal cost 'b' of the bottom cell[Stentz et al.] …….16

2.4(a) Spatial representation of the search space …….……19

2.4(b)Pictorial representation of the Theta* algorithm search process ……..……19

2.5 The above figure represents the state (I,r) consisting of four states……23

2.6 Path finding using ANYA between n1 and n2 where the point

y1, y2, y3, y4 represent the point considered in different intervals

during its search for n2……...23

2.7 Shortest path selection as per Block guided Theta*…….25

2.8: v' depend – on 'p' and the current target vertex under 'v'……..……..27

3.1 A grid map with distributed obstacles, regions in the map where a

lot of unnecessary line of sight checks are performed ……..…….31

3.2 A graphical representation of clustering data scattered on a 2D plane. ……..32

3.3 Flowchart of K-Means ……..……...……..34

3.4(a): Grid map cut into regions based on region size……..35

3.4(b) Path Plotted by C – Theta*……..……..……..35

3.5 Flow chart explaining the pre-processing stage……..…….…..……..39

3.6 Flow chart explaining the structure of C-Theta* search……...……40

3.7 Theta* Vs C – Theta*…….41

4.1 The left image represents a user designed map and

the right a random map……..44

5.1 Avg. Runtime of each of the algorithms to plot the

paths on a user designed map of dimensions 100 × 10……..……..…48

5.2 Avg. Runtime of each of the algorithms to plot the paths on a user

designed map of dimensions 200 × 200…….……..….…….50

5.3 Avg. Runtime of each of the algorithms to plot the paths on a random

map of size 100 × 100……52

5.4 Avg. Runtime of each of the algorithms to plot the

paths on a random map of size 200 × 200……..…..52

5.5 Avg. Runtime of each of the algorithms to plot the

paths on a random maps with varying region sizes……...…….…...56

5.6 The average path length compared with the time taken

to find the goal from source on user designed maps of size 100 ×100…….…….58

5.7 The average path length compared with the time taken

to find the goal from the source on random maps of size 100 ×100……..…..58

5.8 The average path length compared with the time taken

to find the goal from the source on user designed maps of size 200 × 200……..…59

5.9 The average path length compared with the time taken

to find the goal from the source on random maps of size 200 × 200……...…..59

LIST OF ALGORITHMS

1 A* 12

2 Theta*. 18

3 Lazy Theta* ……...20

4 Block Guided Theta*……26

5 K-Means……33

6 C-Theta*……37

LIST OF ACRONYMS & DEFINITIONS

1 G – A graph consisting of nodes and edges.

2 S , s – Represents the start nodes in a graph.

3 Grid - A special variant of a graph consisting of repeating squares.

4 Tile - Represents a single square in a grid.

5 f (n) – The total cost of traversal from the current node to its successor node.

6 g(n) - The actual cost of traversal from current node to its successor node.

7 h(n) – The approximate cost of traversal from the successor node to the the goal

8 c(s, a, s') – Cost of traversal from node s to node s' based on some action 'a'

9 parent(s) – The predecessor node of s.

10 *f-value* – the total value of a node based on a heuristic.

11 Online - Searching the map interactively

12 Obstacle Density – The number of obstacles divide by the total number of obstacles.

CHAPTER - 1 INTRODUCTION

This chapter describes the domain of path finding and what are the components of path finding. It also looks at the problems in path finding since CPU resources are limited and game maps used are becoming more complex and realistic as well as the size increases in size of these maps with an increase in the number of agents on these maps [15] makes path finding an interesting research area.

1.1 Problem Domain

Path finding is the process of finding a low cost traversal path between two nodes in a graph these nodes are referred to as the start and goal node respectively. Let us consider a graph G on which two nodes ‘s’ and ‘t’ are selected. The process of path finding determines a path that is traversal from ‘s’ namely the start node and ‘t’ the goal or target node. Though there can be a number of paths that can be plotted from one node to another the aim of path finding is to find the least cost path if one exists. In general, if there exists a problem of finding a least cost path and this problem can be applied to a graph ‘G’ then the process of path finding can be applied to this problem.

Path finding is used in many real life applications such as GPS systems, networking, robotics and video games [14]. For example in a GPS system the maps can be represented as a graph G where the roads can be represented as edges and cities as nodes connected by these edges, when a user selects a source and destination the system using path finding can plot a least cost path for the user. In video games path finding can be applied in a similar way where a path needs to be plotted for an agent to traverse to various locations of the game map based on the users input.

Based on our observations of path finding the process of path finding can be reduced to searching. It can be applied to games such as chess and the 8- piece puzzle game, where

the goal is to arrange all tiles in a sorted order in such a game the next possible positions of a tile can be represented as the nodes. The problem of path finding can be applied to NP- hard problems such as the famous travelling salesman problem and the N-Puzzle problem.

1.2 Components of Path finding

There are three components of path finding are,

1. Spatial Representation.
2. A searching algorithm
3. A Heuristic

1.2.1. Spatial Representation

Spatial representation is also known as the process of representing the environment for a path finding problem. To perform any sort of path finding the environment needs to be discretized into a graph.

To improve the process of path finding the environment can be represented into special types of graphs such as

a) Navmesh.
b) Waypoints
c) Grids.

a) **NavMesh:**

Navmesh is a special type of graph that is used to abstract the real world into a graph. In short a navmesh can be defined as a set of convex polygons that describes the walkable surface of an environment [13]. The polygons in a nav mesh follow the following property: Consider a convex polygon P, then there exists a straight traversal path between any two points with the convex polygon P. Thus a convex polygon

guarantees a free walk for an agent as long as it is in the same polygon. The standard edges of polygons in a navmesh range from 3 to 6 anything above 6 edges becomes complex and expensive memory wise [11].

The Fig 1.1(a) and Fig 1.1(b) describe how the real world is represented as a navmesh and the method in which these paths are plotted on a navmesh.

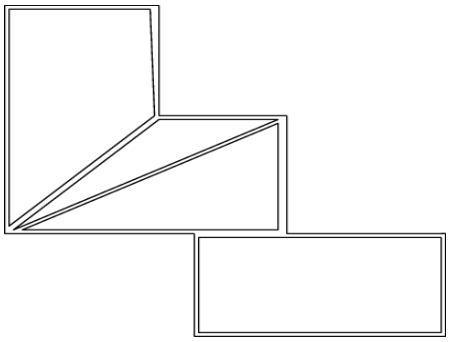

Fig 1.1 (a): The real world represented by N- sided polygon navmesh.

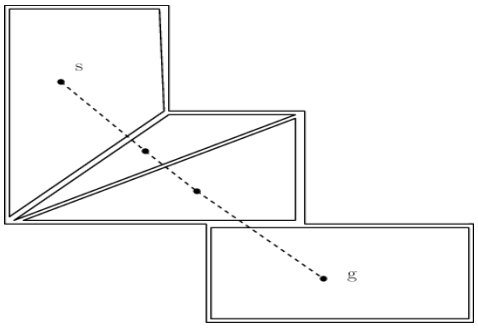

Fig 1.1(b) : The real world represented as a N- sided polygon navmesh with centroids placed in a convex polygon to plot the path from the node ‘s’(start) node ‘g’ (goal).

b) Waypoints:

A waypoint graph is another representation of the real world used in path finding. A waypoint graph consists of nodes placed at various locations of the map. These nodes are connected by straight line edges over which traversal is possible.

A waypoint graph is usually designed by a user, which requires a lot of tuning for a search to be efficient. It works well on maps that representing closed spaces for example a map of connecting rooms. As the number of waypoints increase on a map so does the complexity of the waypoint graph.

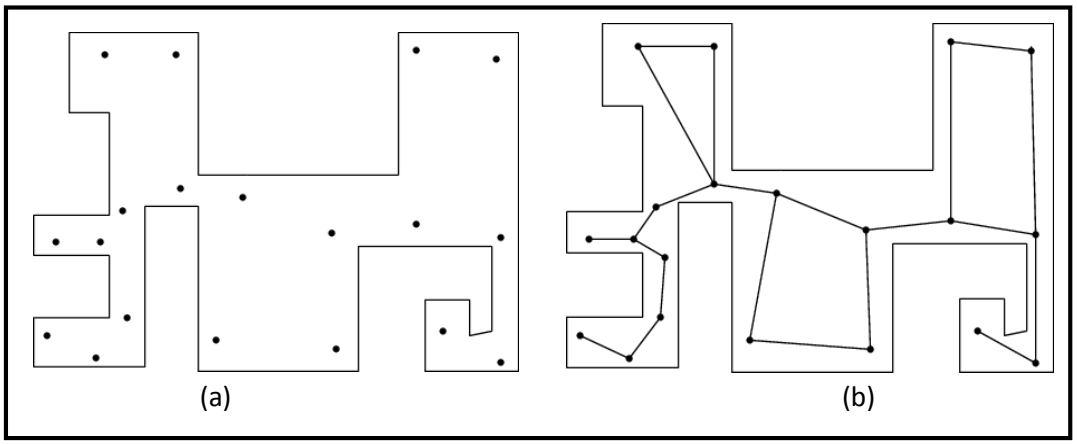

Fig 1.2 a): A real world representation of the environment with waypoints b) Waypoint Graph

The above Fig 1.2(a), Fig(b) provides a representation of strategically placed waypoints and how they are connected with straight lines (edges) to form a waypoint graph.

c) Grids

To represent the real world the most commonly used graph structure is a grid. A grid is a graph made up of repeating squares called tiles or a set of squares to represent the map terrain [12]. The popularity of grids is attributed to the fact that a grid can be generated quickly and efficiently to represent the environment and are utilized heavily in path finding research to test new algorithms. The below figure shows how the real world environment can be represented as a grid.

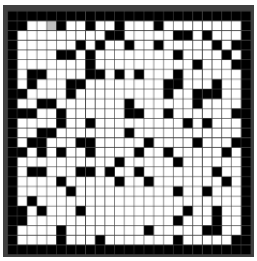

Fig 1.3 Spatial representation of a map using grid structure.

The tiles in the grid shown in the above Fig 1.3 can be used to store information about a region or a single tile in a grid map. For example a map of a strategy game called Age of Empires can store the amount of resources in the tiles of the grid map (a gold mine or quarry).

If the real world is represented as a grid the movement of an agent on a grid is restricted to 4-way movement (cardinal) or 8- way movement (octile) as shown in Fig 1.4 below.

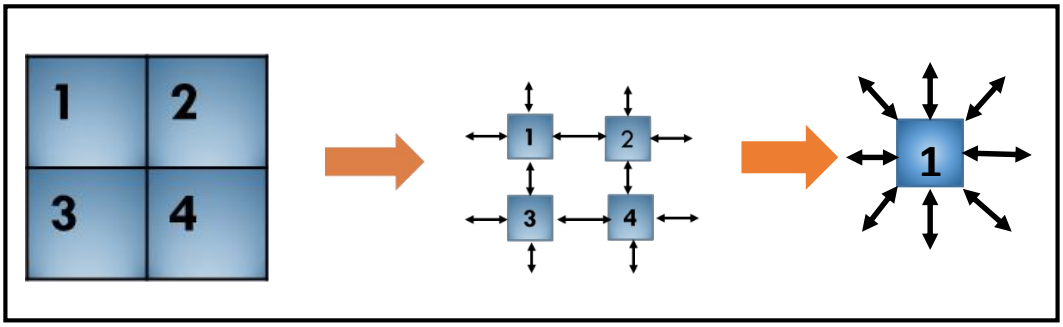

Fig 1.4: Grid representation and the type of movement possible on a grid.

Most path finding problems when using grids for spatial representation can move based on the following movement described in Fig 1.5 below.

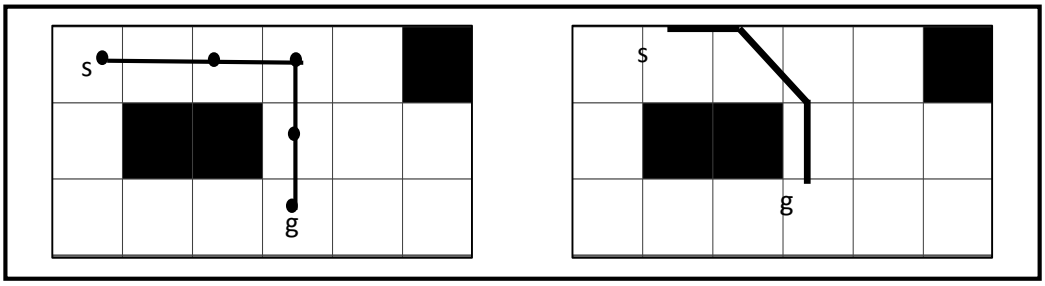

Fig 1.5: Different methods to plot a path on grid map.

The map on the left represents a path using the centroids of the tile and the right represents a path along the grid edges.

In this thesis the maps are represented as grids because they are the most popular type of spatial representation used by the research community.

1.2.2. Search Algorithms

There are a number of search algorithms that work on a graph to find a goal from the start node. Some of the early search algorithms in this area are breath first search, depth

first search, iterative deepening search, these algorithms will find the destination if a path exists but they aren't intuitive and they approach the problem utilizing the strategy of brute force , i.e., the algorithm will search every node on its way to the goal on the graph.

To provide a more intelligent and intuitive method of searching the environment (graph) a group of algorithms were implemented known as informed search algorithms. Dijkstra's search algorithm is an informed search algorithm that is used to solve the single source shortest path problem, it uses the actual cost to traverse from one node to another until the goal node is discovered and will select the path with the least cost from the source to destination[16][17]. Similarly best first search estimates the cost to traverse from node to node instead of using actual traversal costs until the goal is reached.

A* algorithm was developed by merging the properties of Dijkstra's and best first search. It is described in detail in Chapter 2.

1.2.3. Heuristics

Heuristics are the cornerstones of modern path finding methods. They provide the search algorithm with ability to estimate the cost to reach the goal. A heuristic is a method that aids in decision making and problem solving in human beings and machines [18].

In path finding consider a map on which a path finding algorithm needs to move from the start node to the goal node, the algorithm needs to decide which node to expand based on cost of traversal from the current node to its successor node on the path to the goal. A heuristic will aid the algorithm in deciding the most desirable node to expand, i.e., the least cost node.

In path finding some of the most popular heuristics that are applied to solve a particular problem are,

a) Manhattan Distance.

b) Euclidean Distance.

a) Manhattan Distance

Manhattan distance is calculated by measuring the length between two points on a grid along right angle axes. The heuristic is inspired based on the road network of the city of Manhattan which represents a grid layout. The Fig 1.6 represents the plotted path based on Manhattan distance.

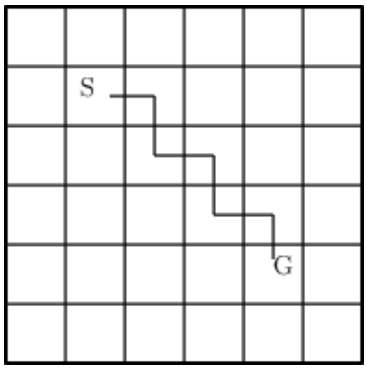

Fig 1.6: A grid map with start and goal node and the path plotted using Manhattan distance heuristic.

Consider two point x (x_i, y_i) and y (x_j, y_j) at right angle to calculate the distance between them using Manhattan distance the following formula is defined below.

$$h\,(n) = \sum\left(\left|x_i - x_j\right|\right) + \left(\left|y_i - y_j\right|\right)$$

b) Euclidean Distance

The heuristic Euclidean distance is defined as the straight line distance between two points of a tile on a grid. In a 2D space the heuristic is defined as the Pythagorean Theorem.

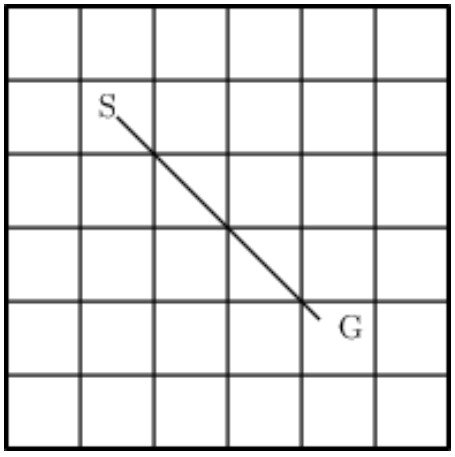

Fig 1.7: A grid map with start and goal node and the path plotted using Euclidean distance as heuristic.

Consider two points x (x_i, y_i) and y (x_j, y_j) is a 2D space the Euclidean distance is defined as follows.

$$h\,(n) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

In Euclidean space ω^3 the distance between two point x, y and z is defined as,

$$h\,(n) = \sqrt[2]{(x_j - x_i)^2 + (y_j - y_j)^2 + (z_j - z_i)^2}$$

This thesis uses Euclidean distance for evaluation of the performance of the search algorithms under review.

1.3 Contribution of this Thesis

In this thesis we introduce a new any-angle algorithm. It is a variant of the Theta* and has been christened C-Theta*.

C-Theta* introduces the concept of clustering in the field of any-angle search algorithms which are a part of single agent search. The details of C – Theta* are explained in chapter 3 Proposed Approach.

Any angle algorithms are a group of algorithms that successfully remove the constraint of path traversal on a grid map along the edges of the grid map. Consider the agent needs to move from the current node to its neighbor or successor node in a grid map. The movement is restricted to 45 degrees in the map thus an agent can move only on the grid edges as defined above. The movement is unrealistic especially in free space and cannot be considered as the true shortest path.

Though these algorithms successfully remove the constraint of movement and find shorter paths as explained above. The time taken to find the path from a given start node to a goal node on a grid map is considerable. They become especially slow as the map size increases which makes them undesirable.

This thesis looks at improving the performance of Theta* an any-angle path finding algorithm. The aim of C-Theta* which is a variant of Theta* is to improve the performance and at the same time maintain the structural properties of Theta* whose properties make it an algorithm that is easily adopted by users and used extensively on grid maps.

Thesis Claim:

C-Theta* the proposed algorithm in the thesis performs better that its predecessor Theta* in terms of run time as well as reducing the number line of sight checks introduced by its predecessor and plotting paths that are shorter than A*.

The experiments performed to compare the performance of C – Theta* with other path finding algorithms (A* and Theta*) are showcased in chapter 4 to validate our claim.

1.4 Organization of this Thesis

The thesis is organized as follows: As seen above chapter 1 provides a brief introduction to the world of path finding. In Chapter 2 we discuss about any-angle path finding and the milestones in this area of research. The chapter provides a brief overview on Theta* and the latest work done on this any- angle algorithm to improve its performance or tweak it to perform under certain environments. Chapter 3 introduces C - Theta* which is the pillar of this thesis or the main contribution. The features and properties of the algorithms are explained in detail. Especially the concept of clustering, the process of region creation and determining the node region during search are explained in detail. Chapter 4 describes the experimental setup implemented to test the path finding algorithm without bias. Chapter 5 showcases our results comparing C – Theta* with the following existing path finding algorithms A* and Theta*. Chapter 6 documents the observations made and the concluding remarks for C-Theta* and Chapter 7 provides a brief insight into the future work possible on C-Theta*.

CHAPTER - 2 BACKGROUND

This chapter looks at what work has been done in path planning under any-angle path planning. It documents the efforts taken to increase the performance and applications of these genre of path planning algorithms.

Any-angle path planning algorithms are a relatively new area in the field of path planning or path finding algorithms. In the early part of this century extensive research was conducted on improving and optimizing traditional path finding algorithms such as A* and Dijkstra's. These algorithms are efficient, robust and widely used in many practical applications such as video games, GPS systems, network design layout, robotics etc.

Traditional algorithms such as A* and Dijkstra's find the shortest path. Though this is not the true shortest path because the movement of an agent is constraint along the edges of a grid tile in a grid map. This also causes the movement of an agent to look unrealistic because of unnecessary turnings in free space on a grid map. To smooth a path plotted by A* or Dijkstra's a post-processing step is introduced known as string-pulling which is an overhead cost and the difference in the path can be seen in Fig2.1.

To solve this problem in path finding any-angle path finding was introduced the main objective of these algorithms is to void the constraint of an agents path along grid edges and propagate search information along grid edges.

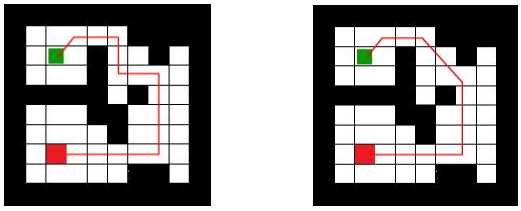

Fig 2.1: Before and after path plotted using string pulling.

2.1. A*

A* is an informed search technique. The algorithm merges the properties of two well-known search algorithms namely Dijkstra's and Best first search. A* was developed by Hart, is the most widely used and known algorithm in the domain of path-finding [1].

Algorithm -1 : A* Pseudo-Code [1]

```
1: Main()
2: S (start) := 0; (Start Node)
3: parent := S(start);
4: open := 0;
5: f (S (start) ) := g(S(start))+ h(S(start));
6: open. Insert(S(start),f(S (start)));
7: closed := 0;
8: while open != 0 do
9:      S: = open. Pop();
10:        If S == S(goal) then
11:              return "Path  Found" ;
12:    closed := closed U {S}
13:  for each S' ∈  neighbors (s) do
14:        if S' ∉  closed then
15:             if S' ∉  open  then
16:               g(S')  :=  ∞;
17:               parent(S') := NULL;
18:   nodeValue(S, S');
19: return "path not found";
20: end;
21: nodeValue(s, s')
22:      If g(s) + c(s, s') < g(s') then
23:               g (s') := g(s) + c(s, s');
24:               parent (s') :=  s;
25:                  if s' ∈ open  then
26:                       open. Pop(s');
27:               open. Push(s', f(s'));
28: end;
```

Algorithm -1: A*[1] pseudo-code describes how A* [1] finds the goal from the start node. The algorithm maintains two lists the open list (line 4) and the closed list (line 7). At the start of A* [1] search the open list contains the start node while the closed list is empty. The neighboring nodes around the start node are expanded by evaluating their *f-scores* and are placed on the open list (lines 21-28).The node with the lowest *f-score* (low - cost) is selected for expansion(line 9). This process is repeated until the algorithm finds the goal node or there are no nodes left for expansion, i.e., the open list is exhausted (line 8).

If the algorithm finds the goal node it traces the path back from the goal node to the start node. This is achieved by maintaining a pointer reference in the child node, i.e., the node that gave birth to it.

In Fig 2.2 (a), Fig2.2(b), A* [1] utilizes the open list to process nodes by either adding nodes to the list or updating nodes in the open list as it searches for the goal node. The nodes visited and already processed are placed on the closed list. The nodes represented in teal are the nodes in the closed list and the nodes surrounding the nodes in the closed list are on the open list represented by light blue.

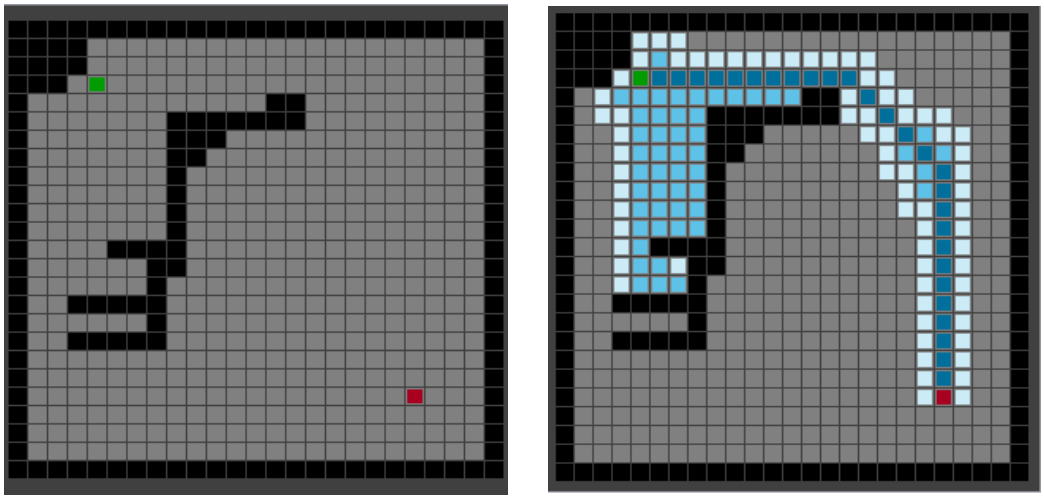

Fig 2.2(a): Spatial representation of the search space. 2.2(b) Pictorial representation of the A*[1] algorithm search process.

2.1.1. Evaluation of *f (n)*

To evaluate which nodes are to be expanded to reach the goal, A* uses the following evaluating function to calculate the cost of each node 'n'.

$$f(n) = g(n) + h(n)$$

Where,

g (n) is the actual cost of traversal, i.e., the cost from the start node to the node 'n' by adding up cost in between to reach 'n'. At the start g (n) = 0.

h (n) is the estimated cost to reach the goal from the node under consideration 'n'. It is also known as the heuristic cost or function. It is an approximate estimate as to the cost of traversal from the node under consideration 'n' to the goal. For example, in a GPS device calculating the straight line distance from the user's current location to the destination using Euclidean distance.

2.1.2. Optimality and Completeness

A* [1] is an optimal search algorithm by implementing an admissible heuristic like Euclidean distance, Manhattan distance etc. because these heuristics will never overestimate the cost to reach the goal. A* [1] guarantees a shortest path using tree search if the heuristic used is admissible.

A heuristic is said to admissible if for every node s in a graph the estimated cost *h(s)* to reach the goal node is less than h*(s) ($h(s) \leq h^*(s)$), where h*(s) is the true cost to reach the goal from the start node. This called the optimistic property of an admissible heuristic.

For example, consider GPS systems using Euclidean distance as a heuristic to measure the straight line distance from the start to the goal node.

A* guarantees to find a shortest path using graph search if the heuristic used in A* is consistent. A heuristic is said to be consistent if for every node s, and every successor node s' of parent s generated by an action a, $h(s) \leq c(s, a, s') + h(s')$ where $c(s, a, s')$ is the cost of travel from s to s' based on some action 'a'. This states that the evaluation function f (n) should be monotonically increasing.

A* is complete, this means that A* is will always finds a path to the goal node if one exists.

2.2 Field D*

It was the first algorithm to consider the nodes as the corner vertices of a grid tile on a grid map. The motivation for developing field D*[4] was to develop an efficient path planning algorithm in the field of robotics. Field D*[4] is an extension of D*[19] and D*Lite[20] proposed by Sven Koenig they are novel path planning and re-planning algorithms that continuously repairs the path based on the environment and are based on A*. Field D* [4]uses a linear interpolation technique to reduce the cost of travel from the start node to a goal node, produce smoother paths from the start to the goal node as well as reduce unnecessary turning on a grid map.

This algorithm was developed with the motivation to be used in robotics. Most path finding algorithms restrict the movement of an agent as discussed above on a small set of headings like 0, $\frac{\pi}{2}, \frac{\pi}{4}$ etc. [4]. For example consider traditional path finding algorithms that traverse along a set of discrete possible solutions. In such scenarios a robot is able to traverse along headings restricted to $\frac{\pi}{4}$ which results in paths that are not optimal and when these methods are used with string pulling the trajectories of the agent's path can be expensive.

Field D*[4] uses the below described linear interpolation method to compute the cost to drive its search which results in paths that are low in cost and also have low cost trajectories. In Fig 2.3 (a) below the nodes are represented as a set of edges {s_1s_2, s_2s_3, s_3s_4, s_4s_5, s_5s_6, s_6s_7, s_7s_8, s_8s_1} on which the optimal path from node‘s’ must exist.

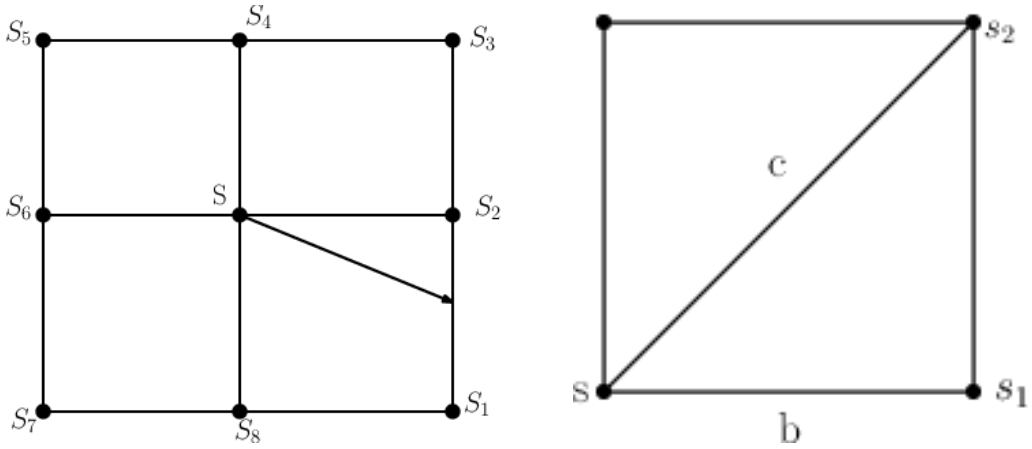

Fig 2.3(a): Field D* corner vertices expansion. (b):The traversal cost from node ‘s’ utilizing the path cost of s_1 and s_2.the traversal cost ‘c’ and traversal cost ‘b’ of the bottom cell.[4]

From the above Fig 2.3 consider the shortest path from the node‘s’ under consideration to pass through an grid edge connected to the neighbors of node‘s’. To compute the path cost using the set of edges describes in Fig 2.2 field D* also computes of the nodes connected by the edge s_1s_2 and the traversal costs from the node‘s’.

Field D* assumes a point s_y resides on the edge under consideration for example s_1s_2 is a linear combination of g (s_1) and g (s_2).

$$g(s_y) = yg(s_2) + (1-y)g(s_1) \ldots\ldots\ldots [4]$$

Where ‘y’ is the distance from s_1 and s_2.Though this is not an exact assumption in theory it works well with linear approximation. Thus the path cost of node ‘s’, s1, s2 and cost ‘c’ is computed by field D* using the following equation.

$$\min_{x,y} \; [bx + c\sqrt{(1-x)^2 + y^2} \; + g(s2)y + g(s1)(1-y] \ldots [4]$$

In the above equation x belongs to the interval [0, 1] which is the distance along the bottom edge 'b' from the node under consideration 's' before cutting across to reach the line edge s_1s_1 at a distance y from the node s_1 and y belongs to the interval[0,1].

2.3. Theta*

Theta*[2] is a variant of A*[1]. Though A*[1] is known for completeness and optimality in finding the path from the start node to the goal node, it does not find the true shortest path because the path is constrained along grid edges.

The motivation for Theta* was to remove this constraint and to find short and realistic looking paths [2]. Theta* performs line of sight checks on the fly to remove the constraint of traversal along grid edges this method is known as "string pulling". Thus eliminating the constraint of traversal along grid edges by broadcasting information along the grid edges of the map. On the next page the pseudo-code describes how Theta* traverses from one node to another.

Algorithm -2: Theta* Pseudo-Code[2]

```
1: NodeValue(s, s')
2: If LOS (parent(s),s' & ) then
/* Decision – 1 */
3:      If(g(parent(s))+c(parent(s),s')) < g(s') then
4:           g(s') := g(parent(s)) + c(parent(s),s');
5:           parent (s') = parent(s);
6:                if s' ∈ open then
7:                     open.pop(s');
8:                     open. Push(s', f(s'))
else /*Decision -2 */
9:     If g(s) + c(s, s') < g(s') then  /*(where c(s, s') cost to move from s to
successor node s')*/
10:          g (s') := g(s) + c(s, s');
11:          parent (s') :=  s;
12:               if s' ∈ open  then
13:                  open.pop(s');
14:                  open. Push(s', f(s')
15: end;
```

The above algorithm pseudo-code describes how Theta* explores the grid map to find a path from start to goal node. The process is similar to A* except for the function described above (lines1-15) this function is called in Algorithm - 1 (line 18) and the line of sight algorithm. Theta* considers two decisions while deciding which node to expand and the constraint on the predecessor node being the parent of the successor node is removed, i.e., in the case Theta* the node under consideration can be any of the evaluated predecessor nodes. The decisions are described in detail below.

Decision – 1

Consider a node s' under consideration and whose parent node is the predecessor node s. If a line of sight exists from s' to the parent of s, i.e., *parent(s).* Theta* then considers decision one (lines 2-8).In this case the sum of the actual cost of parent(s) and the cost of traversal from the *parent(s)* to s' (*g (parent(s)) + c (parent(s), s')*) is compared with the actual cost of the node s' under consideration, i.e., *g(s').*If the cost is less than the actual cost of s' g(s') (line 3) then the algorithm will replace the cost g(s') with the cost [*g(parent(s)) + c(parent(s),s')*] and anchor the parent of s with the node s'.[2]

Decision – 2

This decision is similar to A*, i.e., the sum of the actual cost of the node under consideration s' *g(s')* and the cost of traversal from s to s' *c(s, s')*. [2]

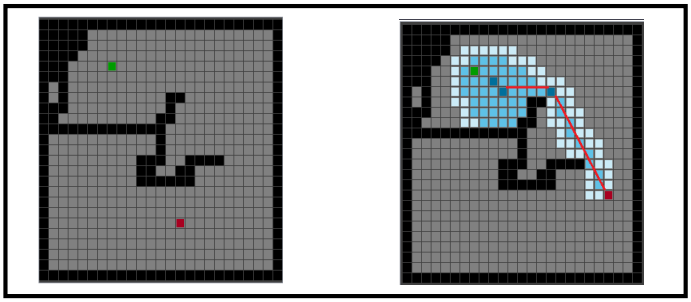

Fig 2.4(a): Spatial representation of the search space. (b) Pictorial representation of the Theta* algorithm search process.

In the above Fig 2.4 (a) and (b) Theta* utilizes the open list to process nodes by either adding nodes to the list or updating nodes in the open list as it searches for the goal node based on the output of the line of sight checks. The nodes visited and already processed are placed on the closed list. The nodes represented in teal are the nodes in the closed list and the nodes surrounding the nodes in the closed list are on the open list represented by light blue. The red line represent the path plotted by Theta*.

2.4. Lazy Theta*

Sven Koenig et al. proposed Lazy Theta*[8]. In a 3D environment the average shortest paths found between a start and goal node are 13% longer than paths found in a continuous environment, as per the authors, there is a need for a smart path planning algorithm for a 3D environment [8]. Theta* an any angle path planning algorithm can be tweaked to be adapted in a 3D environment, but since Theta* will perform a line of sight check for every unexpanded neighbor of a node under consideration and in a 3D grid map environment each node will have 26 neighbors under evaluation for a line of sight check before expanding the lowest cost neighbor which would make Theta* undesirable in a 3D environment.

The authors introduced Lazy Theta *[8] which is an extension of Theta* designed for a 3D environment grid map. Lazy Theta*[8] is similar to Theta but is based on lazy evaluation, i.e., it performs only one line of sight check for every expanded node [8].

Algorithm - 3: Lazy Theta* Pseudo-Code [8]

```
1. SetVertex(s)
2.      If ! (Lineofsight (parent(s), s')) then
3.         parent (s) := argmins' ϵ neighbor vis(s) ∩ closed (g(s') + c(s' , s));
4.         g(s) := mins' ϵ neighbor vis(s) ∩ closed (g(s') + c(s' , s));
5. end
6. ComputeCost (s, s')
7.          /*Decision – 1 */
8.      If g (parent(s)) + c(parent(s),s') < g(s')  then
9.              parent (s') := parent(s);
10.             g (s') = g(parent(s)) + c(parent(s),s');
11. end
```

According to the pseudo-code of Lazy Theta* it is similar to Theta* except for the addition of the function setVertex (s) and ComputeCost (s, s').

In Theta* the algorithm will update the g-value and the parent of every unexpanded visible node of the node under consideration's' based on the output of the line of sight check the algorithm will update the values of the neighboring nodes of 's' based on decision-1 and decision-2 as shown in the Theta* algorithm described above.

But in the case of Theta* the algorithm will assume that a line of sight exists between the node under consideration and its successor node s'. Thus it will anchor the parent of the node s' with the *parent(s)* and update the g-value of the node's', in this way lazy Theta* reduces the number of line-of-sight checks, Of course this assumption may be wrong and needs to be confirmed before expanding the node s'. This is implemented in the function setVertex(s) (lines 1-5).In the function setVertex the algorithm checks for a line of sight from the parent(s) to s' (line 2).If there exists a line of sight from a node s' to *parent (s)* then the algorithm confirms the assumption that a line of sight exists from the node s' to the parent of s. On the other hand if the line of sight check fails, .i.e., a line of sight does not exist between the nodes s and parent of s'. In this case the algorithm will update the g-values and the parent of s' based on decision -1. In this case the algorithm will consider the path from start node to the node s'' and from the node s'' to the node s' based on Euclidean distance (straight line distance) and will choose the shortest path. The reason the algorithm considers s'' is because s' will have at least one unexpanded neighbor in since the node s' was added to the open list when the algorithm expanded the neighbor nodes of the node s.

2.5 ANYA

Daniel Harabor and Alban Grastein introduced ANYA [5] an optimal any angle algorithm. Path finding implemented in robotics has been a known problem in the field of robotics. On a grid map with the selected nodes, i.e., start and goal node almost all any angle algorithms use a Euclidean solution to plot a path. As per the authors many online solutions exist but they require supra linear space and pre-processing time [5]. Most any angle algorithm forsake optimality for smoother, shorter paths and removal of unnecessary turning in free space. ANYA [5], any-angle path finding algorithm addresses this issue.

ANYA [5] considers a set of states when expanding nodes to reach a goal state from the start node. These states are represented by intervals in the algorithm. The algorithm will select a desirable point enclosed within a state and calculate its *f-value* which will represent the *f-value* for the whole state. The advantage of ANYA [5] is that it does all this online.

In Fig 2.5(a) below provides a graphical representation of how ANYA [5] considers different states while searching for the goal node. The four identified states in fig. 2.5 are (I'_1,r) , (I'_2,r) which are visible states and (I'_3,r') and (I'_4, r') which are not visible.[5]

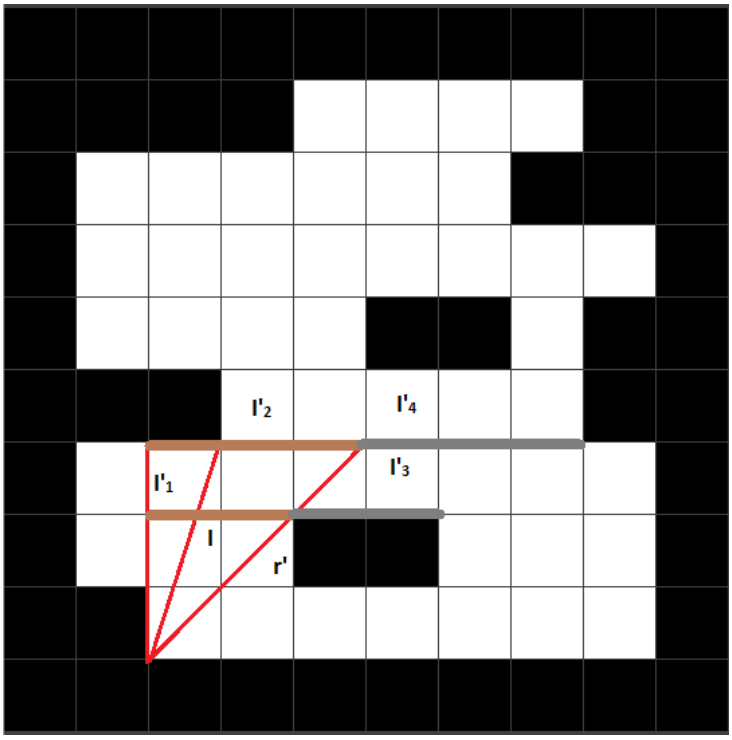

Fig 2.5: The above figure represents the state (I,r) consisting of four states.[5]

Example Trace of ANYA

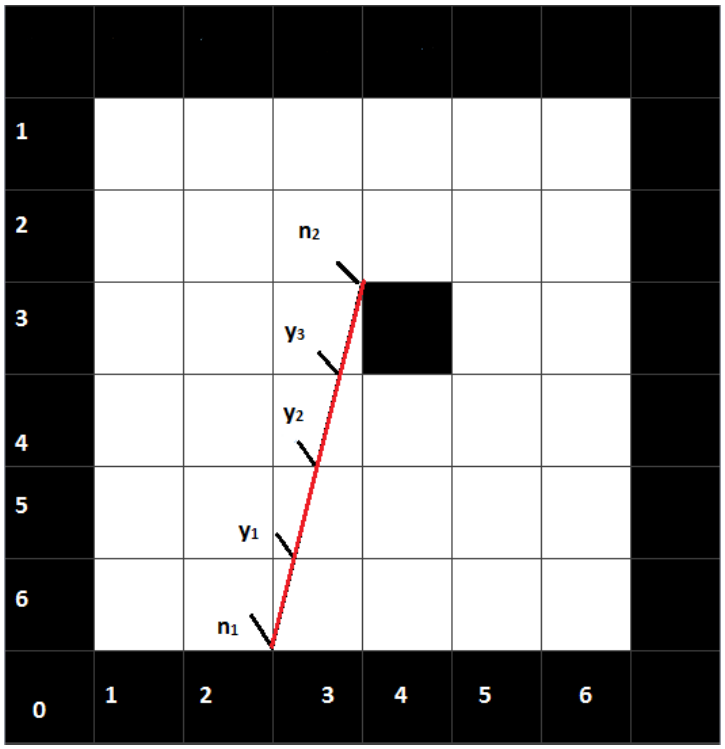

Fig 2.6: Path finding using ANYA between n_1 and n_2 where the point y_1, y_2, y_3, y_4 represent the point considered in different intervals during its search for n_2. [5]

Consider the following grid map of size 6 × 6 in fig 2.6. With start node n_1 and goal node n_2 .The node n1 is a point at (2, 0) the algorithm will plot a path from n1 to the goal node which is at point (3, 4). If we consider existing any angle algorithms like Theta* it will consider only the discrete points, i.e., the corner vertices of a grid tile while calculating the *f- values* of the points. Therefore as per traditional path finding algorithms the actual cost of traversal from the start to goal node differs, i.e., in traditional algorithm the actual cost is a grid – constrained path while in the case of Theta* it is the straight line distance between (n_1, n_2).

Due to this the condition for optimality as discussed before the evaluating function needs to be monotonically increasing which is broken in the case of Theta*.For example in fig. 7 the goal node n_2 can be reached from its parent node(s) (3,3). Applying the evaluating function for the node (3, 3) $f(n) = g(n) + h(n) \implies$ g (distance (n_1, s)) + h (s, n_2) $\implies$ 4.16. For the algorithm to be optimal it should be monotonically increasing thus $f(n2) \geq f(s)$. As per Theta* the evaluating function will compute the value for n_2 as follows $f(n)$ = g (distance (n_1, n_2)) + h (n_1, n_2) $\implies$ 4.12. Thus breaking the condition for optimality.

To address this issue ANYA [5] considers the corner vertices as well as the intermediate point between each of the grid edges. In this case the interval formed between to vertices of a grid tile is [0, 1], the problem encountered in this approach is the number of point can be very large. This problem can be reduced to a Farey sequence by considering the points between two corner vertices as a set of point's *yi* lying in the interval [0, 1] along which a point can be on the path of an optimal path from the start node to the goal node, this lies between $0 \leq (w/h) \leq 1$, where w(resp , h) is a set of integers between { 0,…..,W}(resp,{0,…….H}), in this case for any given n = max(W,H) [5] the cardinality for the corresponding set of elements quadratic in 'n'

[5].Thus in the case of ANYA instead applying the evaluation function to each point $y_{i..}$ It will evaluate together all the nodes from the corresponding interval in which the point y_i appears.

This makes ANYA a complete and optimal any angle algorithm.

2.6 Block guided Theta*

The authors Zi Yang and Wenshen Yu introduce Block guided theta*[3] which is a variant of Theta*[2]. Block guided Theta*[3] approaches the problem of any-angle path finding with a twist unlike Theta*[2], Lazy Theta*[8] which successively expand nodes until a goal state is reached. Block guided Theta* utilizes the blocked nodes in the grid map to guide the search to the goal state. The aim behind the algorithm is to reduce the number of vertex expansions, Unlike the previous algorithms which focus only on the nodes that are desirable (unblocked nodes) block guided Theta* doesn't consider these nodes but rather the unblocked nodes thus creating a finite set of nodes to examine during its search which greatly reduces the number vertex expansions.

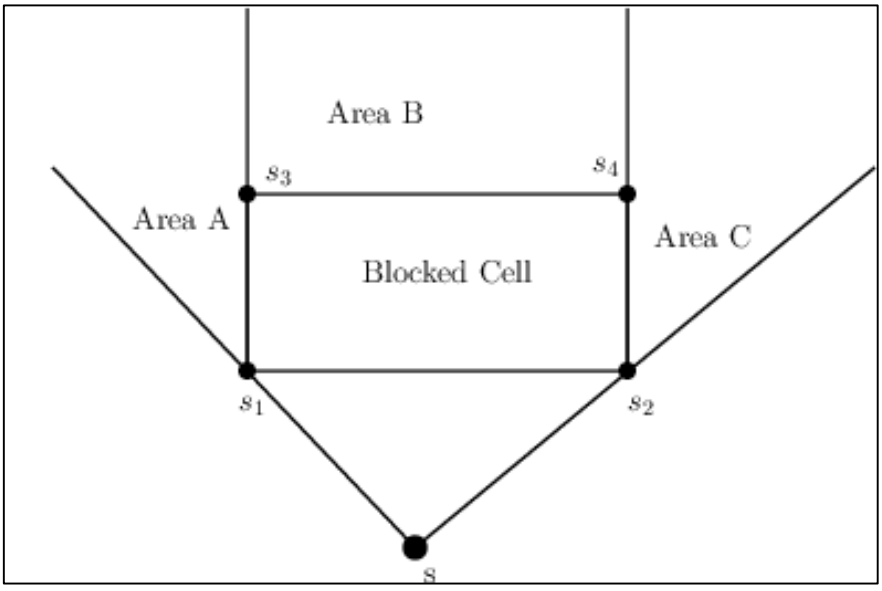

Fig 2.7: Shortest path selection as per Block guided Theta*[6].

In Fig 2.7 Consider the shortest paths based on one blocked cell (s, s'). As per the line axiom if s' is in area C then the shortest path P = {s, s_2, s'} because s_2 will be the intersection vertex of the path P. [3]

Algorithm - 4 : Block Guided Theta* [3]

2. Choose vertex v with the minimum *f (v)* from open,

If LineOfSight (v, t) is true then

Mark vertex t closed and go to Step5,

Else then

Put all vertices in BlockedVertices(Line(v, t)) into the target queueSet T(v) of vertex v.

3. For each non-closed vertex v ∈ *T (v)*, do the following check:

(a) If LineOfSight(v, v) is true,

Mark v open and calculate *f (v).* Then if *f (v)* has been changed in set open and there is a vertex q depend − on v, *f (q)* should be updated as *f(q) = g(v)+1+ h(q)* except that the former *f(q)* is smaller than the later.

(b) if LineOfSight(v, v) is false,

but there is a opened vertex p and v depend − on p, add v to the open set with

f (v) = g(p)+1+ h(v).

(c) If LineOfSight(v, v) is false

And v depends on no opened vertices, then add all vertices in BlockedV ertices (Line (v, v))to the back of T(v).

4. Go back to step2.

5. Clear set open to empty and do a re-find for the path generated from s to t.

6. The algorithm ends and return the final shortest path.

Block guided Theta* is divided into two parts, in the first part the algorithm will generate a path similar to Theta* in which case the nodes with the lowest cost are placed onto the OPEN list. Consider a vertex s under consideration by block guided Theta* it will try to find a path from the vertex s to the goal vertex g using the Euclidean heuristic - straight line distance.

If the algorithm encounters a blocked node on its path it will select one of the four vertices of the blocked node, If the path is relevant, i.e., it can plot a path to the goal vertex 't', it will place the select point on to the OPEN list. But if the algorithm cannot find a relevant path from the blocked node it will follow back on the depend-on situation. If the algorithm cannot find a suitable node to reach the goal it will record the blocked nodes. The algorithm will perform a search until it finds a suitable point from which the line of sight returns true.

The author describes the depend-on situation as follows: Consider a point under consideration's' and consider two points 'p' and v' which may or may not have line of sight to predecessor point's'. Then the algorithm will place the point v' on the open list. Fig 2.8. Describes the depend-on situation executed by block guided Theta*.

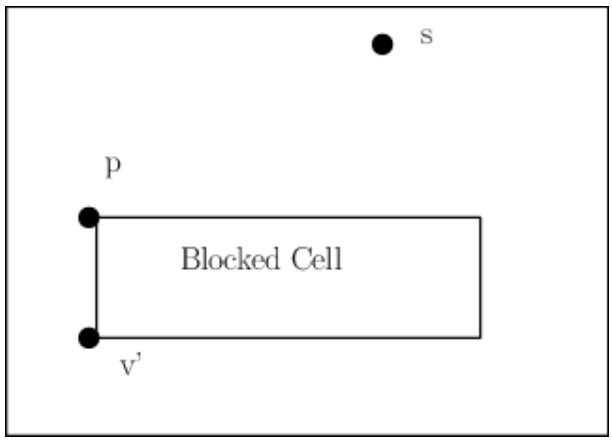

Fig 2.8: v' depend – on 'p' and the current target vertex under 'v'.[3]

Summary

This chapter discussed the work done in any-angle path finding by describing various algorithms. Thus providing a window to look at the state of any-angle path finding on a grid from the time it was implemented in Theta* by Sven Koenig et al. for a grid based spatial representation of the problem.

As seen most of the work focused on optimizing the algorithms runtime to provide better solution times as well as concentrating on reducing the path lengths of agents. All any-angle path finding algorithms are sub-optimal by nature research has also focused on creating an efficient optimal solution as described in ANYA. These solutions are important in terms of video games to provide a realistic feel to the movement of an agent as well providing an efficient run time solution. In terms of robotics it reduces the risk of dangerous heading changes and reduces the cost of trajectory movement by introducing line of sight checks at runtime instead of a post processing phase.

Theta * is an efficient, simple and finds shorter paths than traditional algorithms such as A* but has a runtime solution that can still be improved especially in the way it performs line of sight checks.

CHAPTER – 3 PROPOSED APPROACH

In this chapter the proposed approach is discussed. It explains the motivation for developing C-Theta*.

Clustering is an important concept and a cornerstone of C-Theta*. In this chapter the concept of clustering is explained and the clustering algorithm that is implemented is described in detail and how this process makes the map more intelligent to improve the search is discussed.

The proposed approach looks at the structure and properties of C-Theta* and how the algorithm tries to maintain the properties of Theta* and A* which are mainly ease of use in implementation for fast adoption, and finding shortest paths on the map based on where the start and goal node are located as well as eliminating unnecessary heading changes.

3.1 Motivation

Current research in the field of path planning focuses on any-angle path planning. Theta* an any-angle path planning algorithm is a fast and simple to understand path planning algorithm, On observing the behavior of the algorithm on grid maps it was noticed an agent performs a sizable amount of redundant line of sight checks to fulfill the property of any-angle path planning on grids, i.e., to remove the constraint of traversal along grid edges. For example consider a map on which the obstacles are distributed, there would be large spaces in the map where performing line of sight check are pointless shown in Fig 3.1.

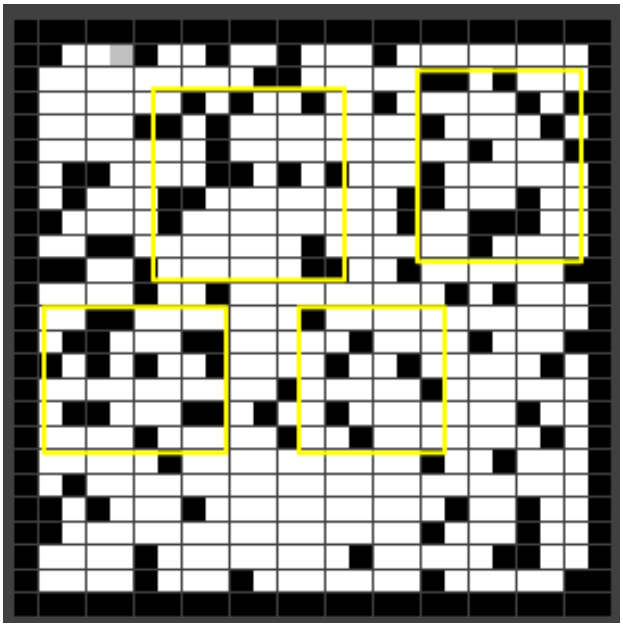

Fig 3.1: A grid map with distributed obstacles, regions in the map where a lot of unnecessary line of sight checks are performed.

The idea for cluster Theta* began with a question “How do we perform line of sight checks in an intelligent way?” If we consider a map “M” containing obstacles “O” on a map how can we best utilize this information to improve Theta* and make it viable as the map size increases.

One of the solutions is to use clustering on the map to provide or improve the information supplied to the agent about the terrain thus improving the path as well as navigation on grid maps.

3.2 Clustering

To understand C - Theta* the concept of clustering needs to be understood. Clustering comes under the domain of machine learning and is a subset of unsupervised learning. In unsupervised learning consider an algorithm(machine) that takes an input of data in a sequential pattern, let this sequence of data be represented as a_1, a_2 , a_3, a_4, a_5,a_6….a_n. Unlike in supervised learning where the algorithm is given a set of desired outputs and trains itself to provide the correct output, unsupervised learning is not provided with any of these options. The algorithm does not get any sort of feedback from the environment on which it is executed and still provides an approximate correct output which makes it difficult to understand since there is no prior knowledge to base your results hypothesis on. Unsupervised learning is based on the idea that given an input the algorithm can provide a formal representation of the input, i.e., data on which it can find patterns to assist in decision making by providing reports on the data and predicting future events based of patterns in the input. In short, this domain of machine learning mainly focuses its study on unstructured data by finding patterns. Unsupervised learning can be summarized in one line **“Finding order in chaos”**. In C-Theta* clustering is an important concept that is implemented to gain better understanding of the environment.

Clustering be defined as a process that works on a set of unstructured data and groups this data into groups based on some similarity measure. Thus clusters can be regarded

as groups of individuals (data) that are similar to each other in some characteristics and dissimilar with other groups of individuals (data).

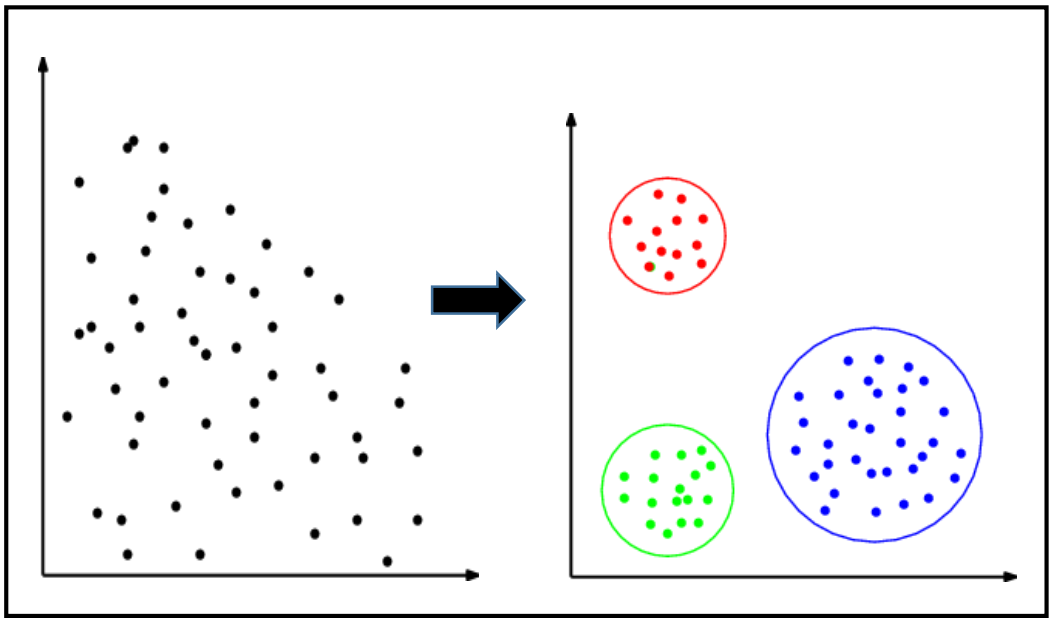

Fig 3.2: A graphical representation of clustering data scattered on a 2D plane. [6]

The most common feature utilized in clustering are distance measures like Euclidean distance and non-Euclidean distances like Jaccard and cosine distance. In Fig 3.2 above the data points are scattered in 2D space and by implementing clustering the points converge to form the clusters shown on the right.

In C-Theta* we implement the K-Means algorithm which is a clustering algorithm for the improving the information of the map and uses Euclidean distance as a distance measure.

3.2.1. K-Means

It is the most robust and widely used clustering algorithm and is also known as Lloyd's algorithm. To improve its output K-Means[21] utilizes an iterative technique which can be observed in the pseudo-code of the algorithm.

Algorithm 5: K-Means Algorithm[7]

1. Choose the number of k - clusters. k=1,2,3…….n
2. Select the points to represent a cluster center based on the number of clusters, a_1, a2, and a3 ….a_k.

   These cluster centers are selected based on one of the following ways

   a) Random or structured selection.

   b) Apply priori information (training data).

   c) Selection based on preliminary calculations.

   d) Applying a theoretical principle independent of the actual data.
3. Distribute the samples among K means

   The data samples must be assigned to the cluster based on its proximity to the cluster centers.

   $x \in S_i(n)$ if $|x - a_i(n)| \leq |x - a_j|$,

   for all j = 1, 2, 3 , ….,k; where $i \neq j$

   $S_i(n)$ is the set of samples whose cluster center is $z_i(n)$, where n indicates that this is the n^{th} iteration of this procedure.
4. Compute new clusters for each set $S_i(n)$

   Find a new value for each a_i.: In this case the a new cluster centre(centroid) $a_i(n+1)$ is chosen which is the mean of the points belonging to $S_i(n)$ such that:

   $$a_i(n+1) = \sum_{x \in Si(n)} x$$
5. Compare $a_i(n)$ and $a_i(n+1)$ for all i

   Compute the distance measures between each pair of points for consecutive iterations.

   a) If there is no substantial change, terminate the procedure. Some possible criteria for termination are,

      1. If $|a_i(n+1) - a_i(n)| < T, \forall i$
      2. If $\sum_{j=1}^{k} |a_i(n+1) - a_i(n)| < T, \forall i$

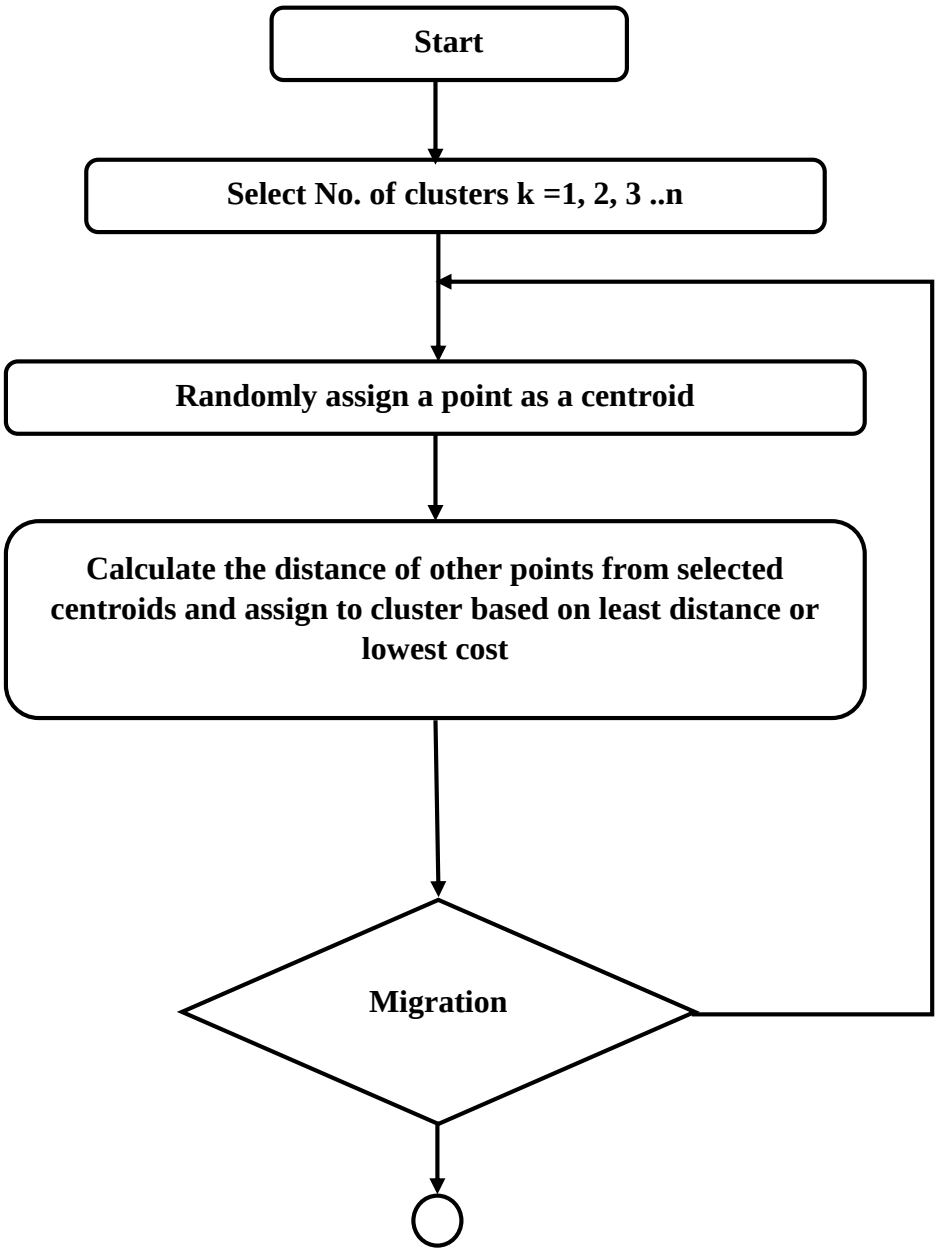

Fig 3.3. Flowchart of K-Means

3.3 C-Theta*

The below example trace provides an insight into how C-Theta* plots a path between two points.

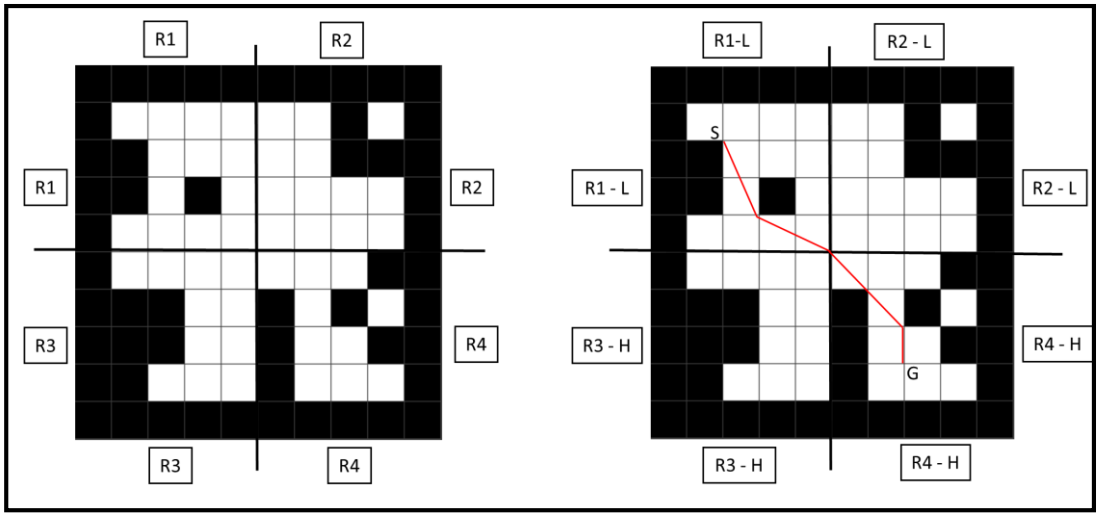

Fig. 3.4a: Grid map cut into regions based on region size. 3.4(b) Path Plotted by C – Theta*

In Fig 3.4(a) The grid map is processed to create regions based on region size. After region creation a clustering algorithm, in our case K-Means is used to classify a region into high and low density based on the number of obstacles in a region Fig 3.4(b) The red line plotted from start to goal node is the path found by C-Theta*.

In Chapter 2 Theta* is described in detail, as discussed in the above chapter it finds shorter paths than A*. Though the paths are shorter than A* the time taken is longer and as the map size increases this trend continues to increase.

To address this problem C-Theta* has been developed which attempts to maintain the structural integrity of Theta* which makes it a popular path planning algorithm as well as it tries to improve the time taken by Theta* to plot the path.

To understand how C-Theta* works, Consider Fig 3.4(a) and Fig 3.4(b), the map is created with obstacles (Fig3.4 (a)). This map is then divided into regions. Once the

regions are created the blocked nodes in a region under consideration provides a picture on whether the region is a high density region or a low density region. Based on this C-Theta* decides on whether to perform a line of sight or not. For example, to divide a grid map of size 200 × 200 into regions of size 10 the number of regions created is 20 of size 10 × 10 on the entire grid map.

Once the regions are created, how the algorithm does decides which density regions are high and low? It uses a clustering algorithm to decide whether a region is of high or low density.

In the case of C-Theta*, K-Means is used to decide whether a region is of high and low density by supplying it the number of blocked nodes in a region this data is stored in a single array and it will classify a region based on the number of obstacles into a low and high density region.

This new information is supplied to C-Theta* which considers two type of paths explained below

Smooth Path:

If a node of a region belongs to a label of low density. C-Theta* will perform a line of sight check to plot the path.

Raw Path:

If a node of a region belongs to a label of high density .C-Theta* will not perform a line of sight check.

For example in Fig 3.4(b) if the start node selected is represented by ‘S’ and the goal node is represent by ‘G’ the algorithm will first create the regions as discussed above where they are represented as R1-R4 in Fig 3.4(a).The K-Means algorithm is used to

label the regions as high and low density, as shown in Fig 3.4(b) R1-L is a region of low density. The start node is in a low density region (R1-L) and the goal lies in a region R4-H which is a high density region (Fig3.4 (b)).

When the algorithm begins, it will expand nodes as per the rule defined in "Smooth Path", Once the algorithm enters R4-H which is a high density region the "Raw Path" rule is followed where no line of sight checks are performed until it gets out of region R4-H or if the goal is found the search stops and the path is plotted.

The proposed algorithm is explained in great detail as shown by the pseudo-code derived below.

Algorithm - 6 Proposed Algorithm: C-Theta*

```
1: Main()
2: ConvertToRegions
3: do K-Means Clustering & Label Regions
/*start*/
4: S (start) := 0; (Start Node)
5: parent := S(start);
6: open := 0;      /*(Open list : =  Set of node under consideration)*/
7: f (S(start)) := g(S(start))+ h(S(start););
8: open.Insert(S(start),f(S (start)));
9: closed := 0; /*( Closed List := Set of nodes already evaluated) */
10: while open != 0 do
11:    S := open.pop();
12:       If S == S(goal) then
13:           return "Path  Found" ;
14: closed := closed U {S}
15: for each S' ∈  neighours (s) do
16:         if S' ∈  closed then
17:       if S' ∉  open  then
18:           g(S')  :=  ∞;
```

```
19:          parent(S') := NULL;
20       NodeValue(S, S');
21: return "path not found";
22: end;

23: NodeValue(s, s')
24: If DetermineNodeRegion(s') then/*Low density Region*/
25:      If LOS (parent(s),s' ) then
26:              If(g(parent(s))+c(parent(s),s')) < g(s') then
27:                      g(s') := g(parent(s)) + c(parent(s),s');
28:                parent (s') = parent(s);
29:                      if s' ∈ open then
30:                              open.pop(s');
31:               open.push(s', f(s'))
32: else if g(s) + c(s,s') < g(s') then
33:               g (s') := g(s) + c(s,s');
34:               parent (s') :=  s;
35:                if s' ∈ open  then
36:                      open.pop(s');
37:                          open.push(s',f(s'));
38: end
```

The Flowcharts below provide a graphical representation of how C-Theta* performs its search between a start and goal node on a given grid map.

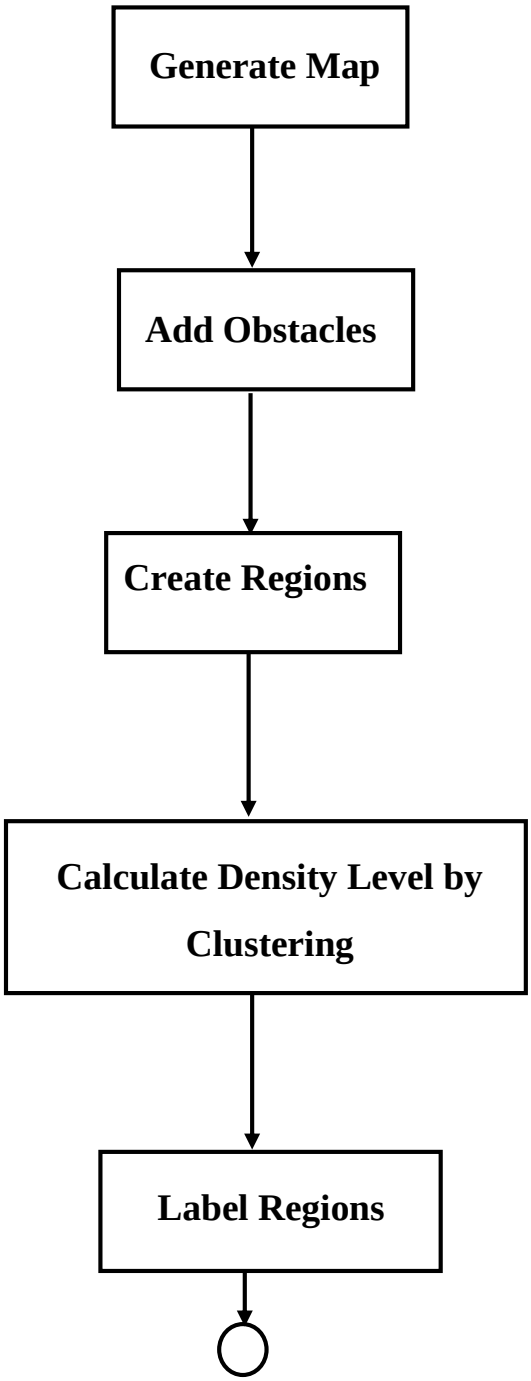

Fig 3.5: Flow chart explaining the pre-processing stage

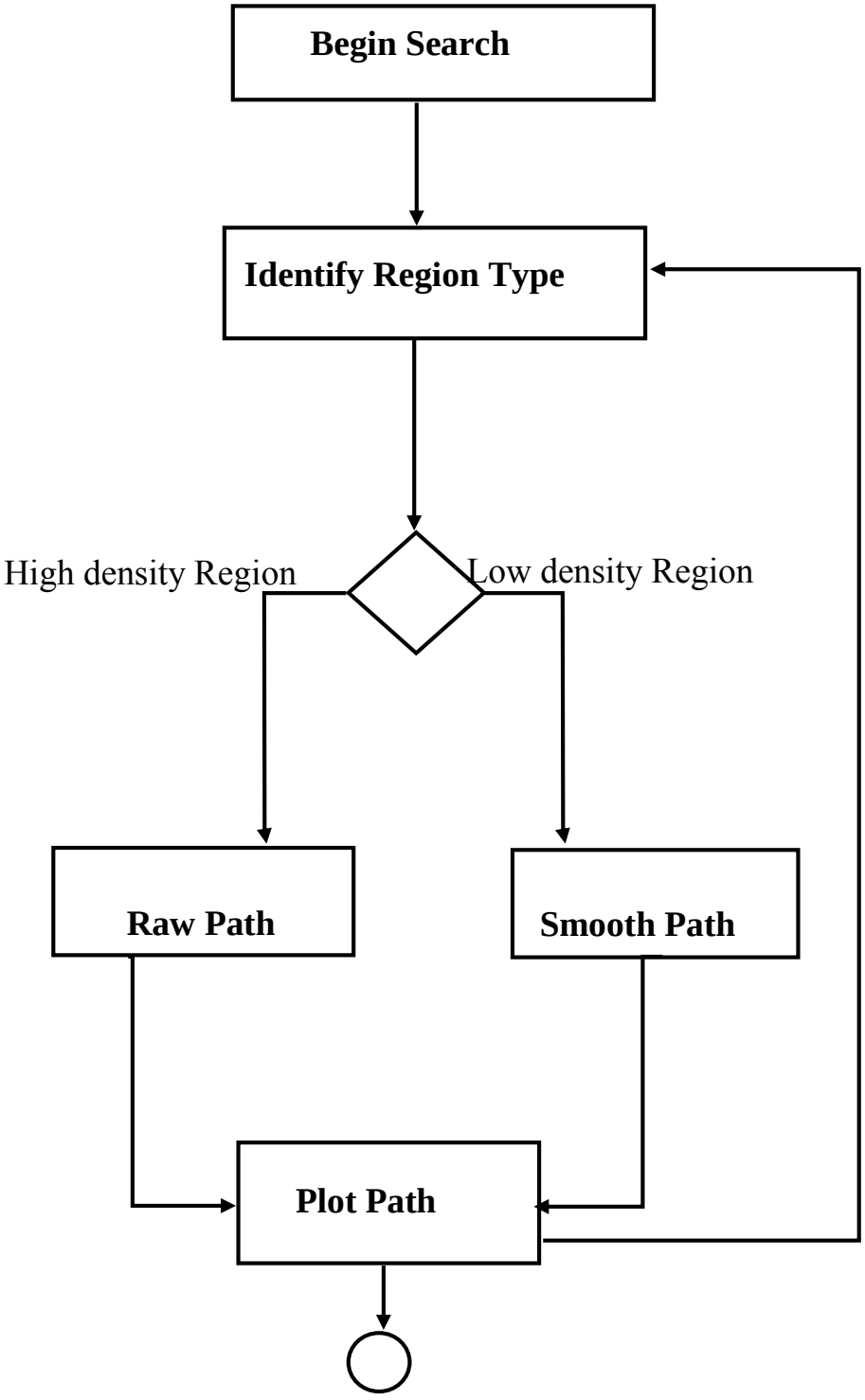

Fig 3.6: Flow chart explaining the structure of C-Theta* search

3.4 C-Theta* Vs Theta*

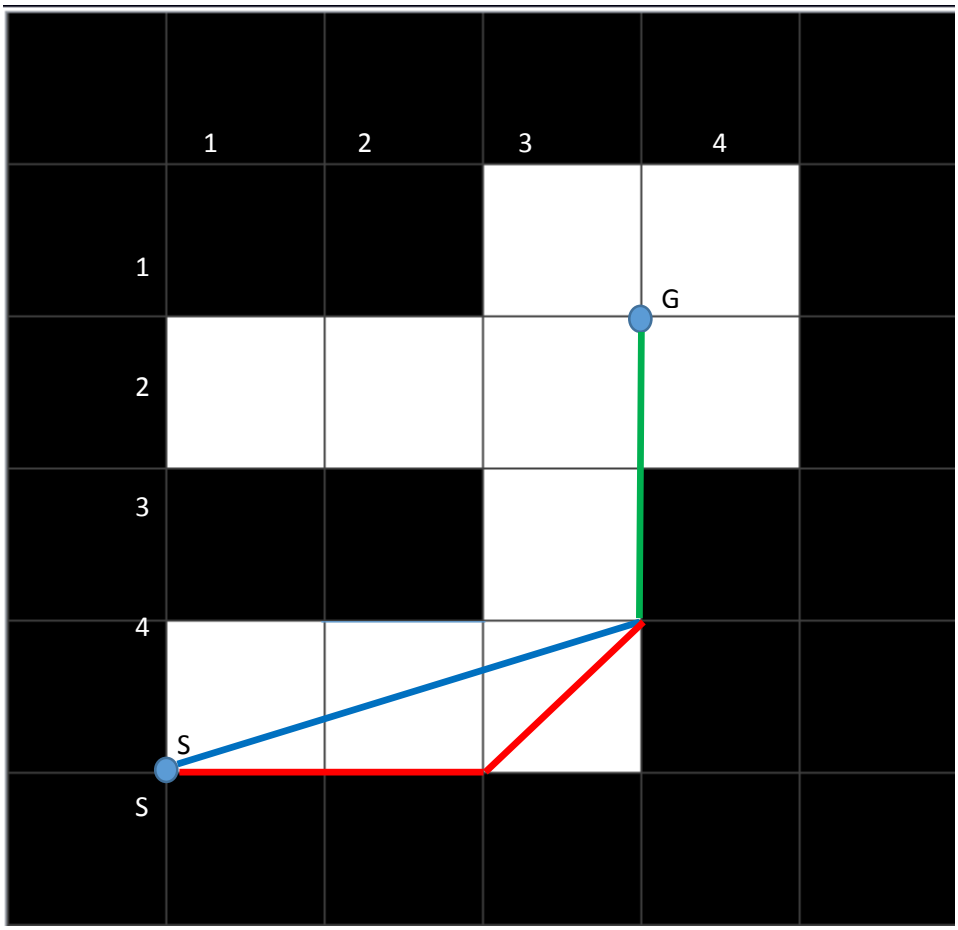

Fig 3.7: Theta* Vs C – Theta*

Fig 3.7 represents a high density grid region of a grid map where the yellow and red lines represent a path plotted by Theta* and C-Theta*.The green line represents the overlap of both algorithm on the path plotted to reach the goal.

Consider the above Fig3.7 which portrays the paths plotted by C-Theta* and Theta* in a high density region. These regions are where C-Theta* and Theta* differ.

For example in Fig 3.7 the start node ‘S’ at location [4, 4] and the goal node ‘G’ [3, 2] are in a high density region, if Theta* plots the path in this region it will perform a line of sight check of each unexpanded neighbor as moves from as it moves towards the goal

in this case to reach the node [3,3] it has already performed 6 line of sight checks at each of the corner vertices of the nodes [[4,4],[3,2],[3,3]], While as per C-Theta* no line of sight check will be performed since the nodes under consideration are in a high density region. To move from the node [3, 3] towards the goal Theta* will again perform at least 3 more line of sight checks to reach the goal node at [3, 2], while C-Theta* will not perform any line of sight checks and plot the same path as Theta* to reach the goal from [3,3].

Though these paths are different and the path length of Theta* is slightly shorter than and C-Theta* because of triangle inequality the time taken would exceed that of C-Theta* because of the redundant line of sight checks because one of the main objective of performing line of sight checks in the case of Theta* is to avoid unnecessary heading changes in a grid map by performing line of sight checks, but in a high density region the chances of performing unnecessary heading changes are minimal. Also the paths explored by Theta* and C-Theta* will overlap a fair bit as shown in Fig 3.7. in high density regions.

Summary

Chapter 3 discussed the workings of C-Theta* and the motivation behind this novel algorithm.

Since the concept of clustering is an important section of C-Theta* a brief overview of how clustering is implemented is explained. The clustering algorithm adopted for C-Theta* is also explained in detail.

The structure of C-Theta* is explained via pseudo-code as well as by using flow charts for graphical representation of how C-Theta* performs a search to find the goal from a give start node.

The examples provided help in understanding how C-Theta* expands and plots a path between nodes as well as providing an insight into how this algorithm differs from Theta*.

CHAPTER - 4 EXPERIMENTAL SETUP

4.1 Search Space representation and Heuristic

The environments used to test the performance of C-Theta* is based on maps made up of grids. The performance of most path finding algorithms is tested on this type of graphical representation and very popular in video games [9]. The grids maps designed have a uniform traversal cost set at 1 to move from one grid cell to another.

The maps have been designed manually using a base framework developed in JAVA by Arttu Viljakainen, Teemu Turunen [10] and modified and enhanced to support any-angle path finding for experimental purposes of C-Theta*, Random maps of with randomly placed obstacle maps are used to test the performance of algorithms which assists in testing the algorithm in environments not controller by the user where the maps can be tailor-made for a particular problem.

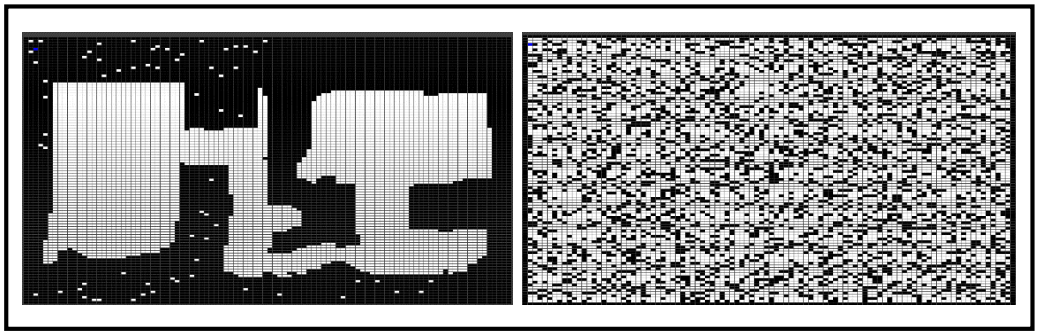

Fig 4.1: The left image represents a user designed map and the right a random map.

The heuristic used to drive the search of all the path finding algorithms under consideration is Euclidean distance in a 2D space. This heuristic is explained in great detail in the introduction.

4.2 Assumptions

a) **Static:** The graphs generated are static for the experimental results. The reason for using static grid maps is because of the preprocessing step ahead of C-Theta* where the map is broken into regions. If the maps were dynamic the results would be invalidated because of continuous change in information about the grid map.

b) **Information**: The information provided at the start, i.e., creation of the grid map is constant, i.e., there will be no change in the information supplied to the algorithms during run time. This is an important factor because C-Theta* is dependent on a pre-processed step.

c) **Type of Search:** The search will be performed between two single nodes anywhere on the grid map these points will be referred to as that start and goal node. In single agent search this is a common method used in path finding algorithms research.

d) **Connectivity:** The grid map generated will always be connected, i.e., the edges of nodes are connected such that a path always exists between any node in the grid map as well as a path from one node to any other node on the grid map.

e) **Tie Breaking:** The strategy used to take care of any ties between the *f-values* of any two nodes in the experimental set up is taken care of by the priority queue implemented in JAVA version "1.8.0_45".Therefore the node with the smallest *f-value* is selected and no explicit tie breaking rule has been defined.

d) **Region Size:** For the experiments, on user as well as random maps the region size is set at 10×10.

4.3 Algorithms

The algorithms under experimentation are:

a) A*

b) Theta*

c) C-Theta*.

Among these algorithms A* is used as the base algorithm against which both algorithms Theta* as well as C-Theta* are compared because A* is the most widely used path finding algorithm. This algorithm is used to measure the performance of C-Theta* in terms of time and the difference in path length when we compare all of the three algorithms together.

As discussed above Theta* is a variant of A* and is known to find shorter paths than A* but the time taken is considerable when compared with A*.We use Theta* as a benchmark to compare C-Theta* which is a variant of Theta*.

The last algorithm under consideration is C-Theta* which used clustering improve the search time in finding a traversable path between a start and goal node.

The heuristic used in all three algorithms is Euclidean distance which has been explained in detail in chapter 1.

4.4 Process of Testing

The maps used to test the algorithms are of dimensions 100 × 100 and 200 × 200 user designed maps and random maps of the same dimensions. The start and goal node are selected at random on the maps. Our assumptions hold that the selected paths will

always be connected any point that are unreachable due obstacle blockages are rejected. Also all paths that have their start node equal to the goal node on the grid maps will be excluded from analysis.

The algorithms have been tested on maps with varying obstacle density which has been limited to 40% on user designed maps and 30% on random maps. The search is executed in cycles of 10 iterations on each instance of a maps with the locations of each start and goal nodes spread across the map and no two location under consideration are equal, i.e., each start and goal node is unique.

The algorithms have also been tested on maps of varying region sizes to test the performance of C-Theta*.

4.5 Hardware and Software Environment

The hardware and software environment used to test the algorithms performance is as follows

a) Hardware

Machine: Lenovo X201 Thinkpad Tablet

Processor: Intel i5 second generation

RAM: 8GB

b) Software

The framework used is developed in java and the JVM version used for experimentation is "1.8.0_45".

CHAPTER - 5 EXPERIMENTAL ANALYSIS

5.1 Runtime Analysis

5.1.1 User Designed Maps

The algorithms were tested based on the obstacle density of each map under consideration and the time taken to plot the path from a start node to a goal node taken at random on the map. The below graph (Fig 5.1) displays the time taken by each of the algorithms to discover the path on the map size of 100 × 100.

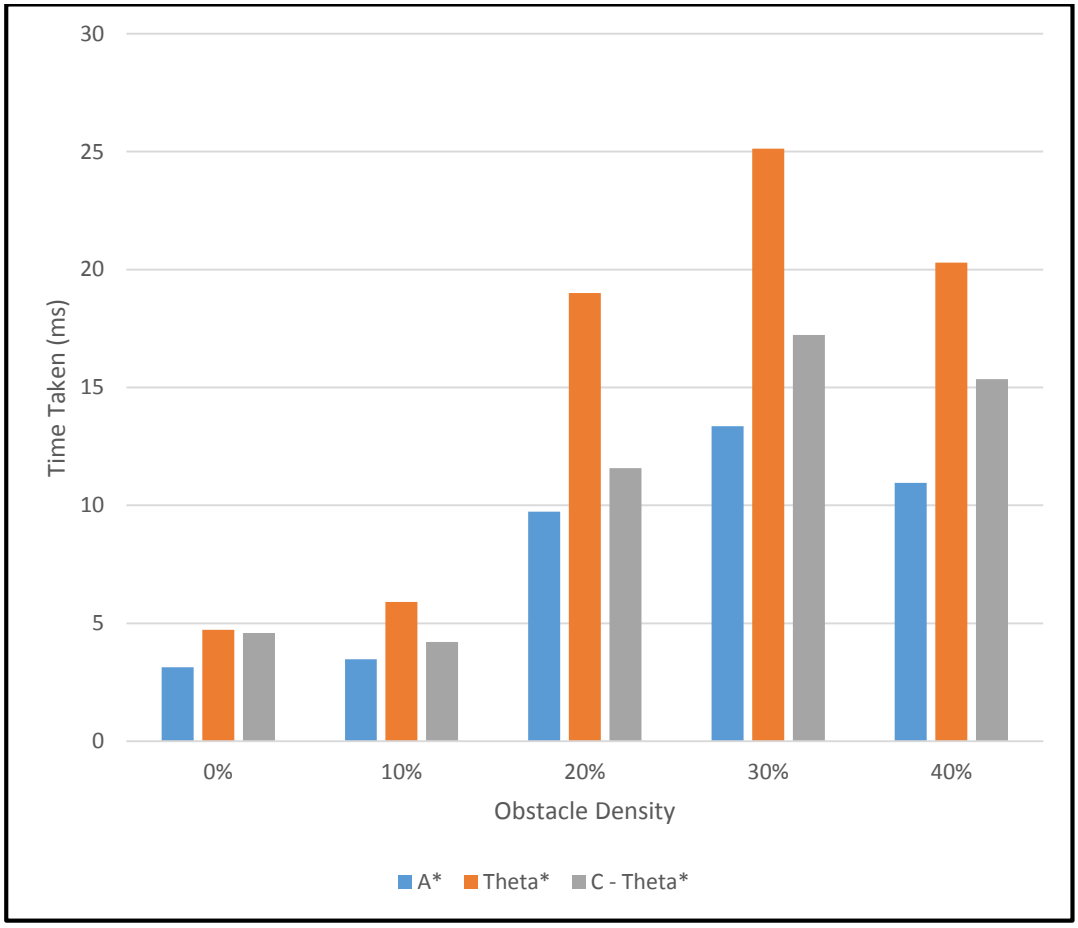

Fig 5.1: Avg. Runtime of each of the algorithms to plot the paths on a user designed map of dimensions 100 × 100.

Observing the runtimes of each algorithm to find the path from a given start node to goal node based on varying obstacle densities on can observe that A* takes the least amount of time in finding the path to goal and as expected Theta* the any-angle

algorithm on which C-Theta* is based takes the longest. On a high level C-Theta* oscillates between A* and Theta*.

On average the C-Theta* performs better than Theta* is most cases a peculiar thing to note is that as the obstacle density increases the performance of C-Theta* improves too. For example in the case of 10% obstacle density the performance is negligible as it oscillates between 5% and 10% improvement on average and while the obstacle density is 0% ,i.e., no obstacles on the map, the performance of C - Theta* is negligible and also may take more time than Theta* in some cases. This gradually improves as the obstacle density increases from 10 percent - 40 percent. The maximum improvement observed in time is 30% when the obstacle density reaches 40%.

The performance improvement in C-Theta* can be attributed to the clustering algorithm and the structure of C-Theta*. By structure, i.e., in the case of C-Theta* the algorithm tries to minimize the number of unnecessary line of sight checks using the additional information provided by K-Means in the pre-processing step. This makes C – Theta* intelligent in the terms of how it executes the line of sight checks on the given grid maps C – Theta* is executed on.

Below in Table 1 the runtime for results in user designed maps of dimensions 100 × 100 are documented.

Obstacle Density	**A***	**Theta***	**C-Theta***
0%	3.1395	4.722615	4.587843
10%	3.472443	5.898572	4.209572
20%	9.733683	19.0066	11.58277
30%	13.35421	25.12258	17.22577
40%	10.95463	20.28975	15.35478

Table 1: Runtime results for the algorithms with varying obstacle density in map size 100 × 100.

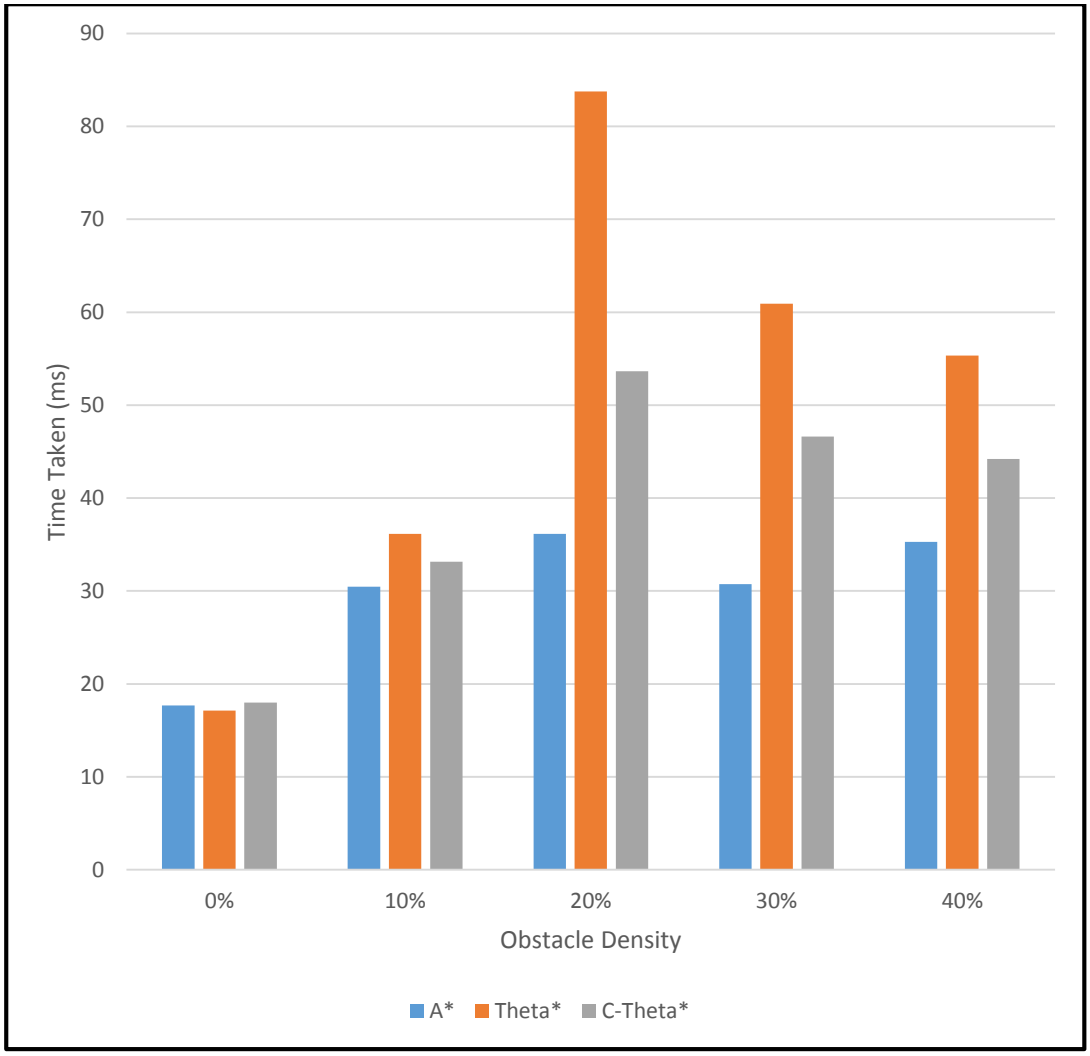

Fig 5.2: Avg. Runtime of each of the algorithms to plot the paths on a user designed map of dimensions 200 × 200.

The pattern repeats itself in map sizes of 200 × 200 shown in Fig 5.2, where C – Theta* again outperforms Theta* when plotting a path from a start node to a goal node but unlike in the before results where there was an average improvement of 30 % in the maps where the obstacle density was between 30 – 40 % we see a decrease in performance by 10% in our results on the 200 × 200 map size.

Though it may look like the performance degrades as the map size increases but that is not the case as an average improvement of 20 % was observed throughout all variations in the obstacle densities of the map. This proves that the performance C – Theta* is

dependent on the way the clustering algorithm will label the different regions of the map.

It is observed that when the obstacles are tightly coupled there is a decrease in performance as observed in the results in the above graphical representation of Fig 5.2.

Obstacle Density	A*	Theta*	C-Theta*
0%	17.6736195	17.12443	17.97635
10%	30.4726895	36.15652	33.15816
20%	36.15390695	**83.76186**	**53.64893**
30%	30.7462387	60.93086	46.60938
40%	35.2859	55.32058	44.22221

Table 2: Runtime results for the algorithms with varying obstacle density in map size 200 × 200.

5.1.2 Random Maps

The algorithms were compared on random maps with varying obstacle densities and the region size was fixed at 10 thus the dimensions each region is 10 × 10.In terms of performance C-Theta* performs better than Theta* on the maps.

The results for random maps can be observed in the graphical representation of the results based on the runtime of each algorithm and Table 3 displays the results of the algorithms.

Obstacle Density	A*	Theta*	C-Theta*
10%	4.257459	5.523796	4.528788
20%	5.40997	7.600321	5.707198
30%	6.037147	9.128994	6.128788

Table 3: Runtime results for the algorithms with varying obstacle density in map size 100 × 100.

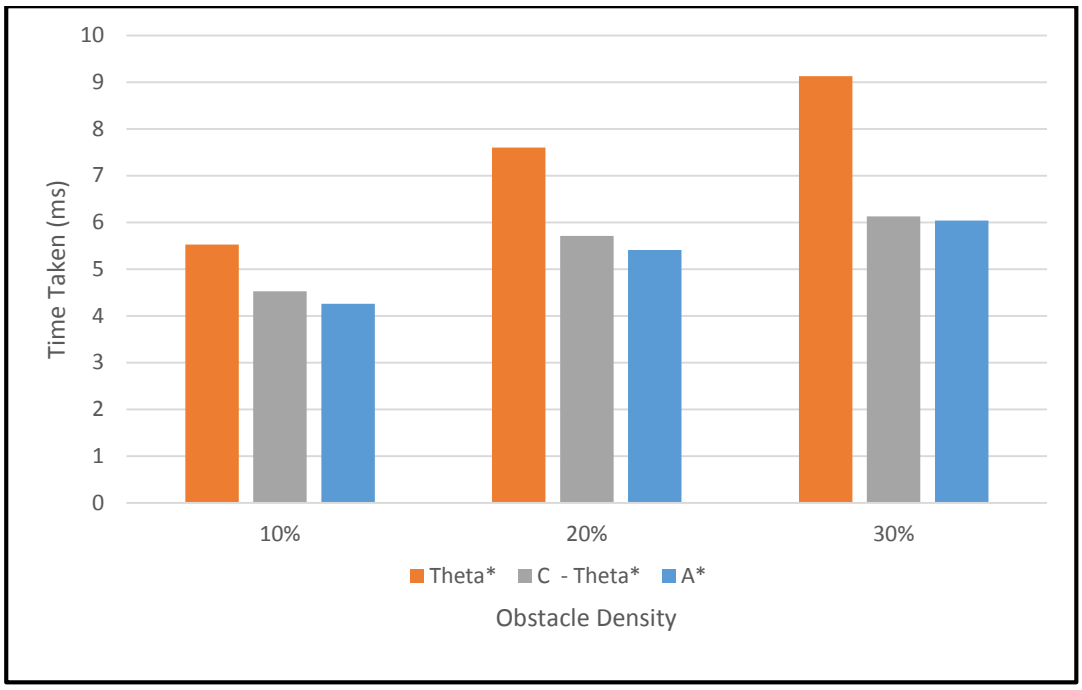

Fig 5.3: Avg. Runtime of each of the algorithms to plot the paths on a random map of size 100 × 100.

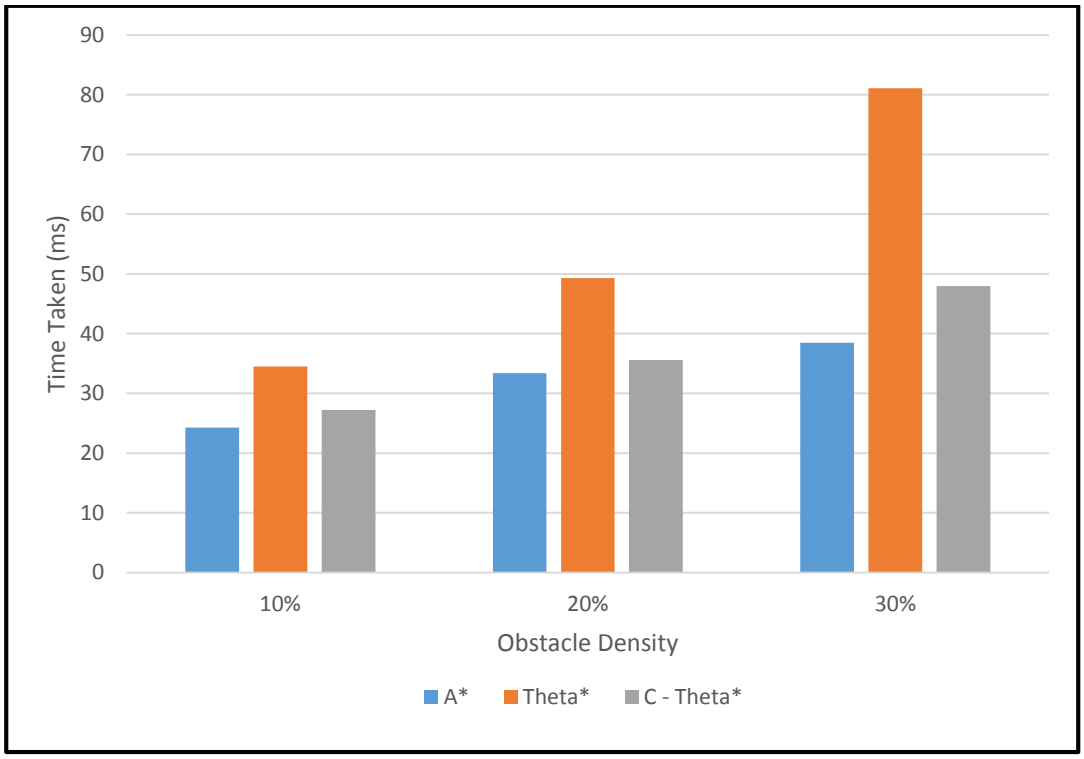

Fig 5.4: Avg. Runtime of each of the algorithms to plot the paths on a random map of size 200 × 200.

The above results reflect an almost similar pattern seen in the results of the user designed maps where on average the time taken by C-Theta* is 20 % less than that of Theta* though the time taken is still more when compared with A*.

Though there are spikes in improvement in time on random maps for example for the random map of 30% obstacle density with map size 200 × 200 we observe an improvement of 40% which is very good when we compare the result with Theta* the algorithm we aim to improve.

Also there seems to exist a pattern in the results though the improvement is of 20% on average. When the obstacle density increases the time taken by C – Theta* is closer to the result of A* and when the obstacle density decreases the time taken by C-Theta is closer to the time taken by Theta*.

Obstacle Density	**A***	**Theta***	**C-Theta***
10%	24.25541	34.51403	27.21528
20%	33.3874	49.29604	**35.58314**
30%	38.45055	81.07007	**47.97218**

Table 4: Runtime results for the algorithms with varying obstacle density in map size 200 × 200.

5.2. LOS Analysis

The Line of sight is used by any-angle path finding algorithms to eliminate unnecessary turns in free space and plot a path that looks realistic. In this thesis the focus has been to optimize any-angle algorithms by tweaking the line of sight checks to reduce the time taken by these algorithms in finding the path. The core of C-Theta* has been to reduce the number of unnecessary line of sight checks which makes Theta* (any angle algorithm) an undesirable option as the map size increases.

We observe the effects of reduction in line of sights based upon the time taken by C – Theta* and Theta* to find the path for a given start and goal node. Table 5 and Table 6 provide a tabular representation of the reduction in line of sight in C – Theta* when compared with Theta*.

User Designed Maps		
Obstacle Density	**Theta***	**C- Theta***
10%	7869	4974.33
20%	10350	6455.25
30%	5766	2315.5
40%	6724.5	2166.75
Random Maps		
10%	4996	2256
20%	4684.8	1798
30%	3856	1255

Table 5: Avg Line of Sights for grid maps of size 100 × 100.

User Designed Map		
Obstacle Density	**Theta***	**C-Theta***
10%	6643	5802
20%	17807	9758
30%	31678	25877
40%	44540	37712
Random Map		
Obstacle Density	**Theta***	**C-Theta***
10%	15730	7545.5
20%	16287	8581.333
30%	18275	6091.767

Table 6: Avg Line of Sights for grid maps of size 200 × 200.

The main reason C- Theta* performs better than Theta* is because of the way it reduces the line of sight checks in Theta*.

Based on the way the clustering algorithm labels regions we are able to reduce the line of sight for a particular map based on where the source and goal node are located. On observing the results one deduction can be made, i.e., as the line of sight checks decrease the performance of C-Theta* will improve. This phenomenon can be observed when we compare the results of random map and user designed map.

In the case of user defined maps, on average the number of line of sight performed is between 35 – 50% less than the line of sight checks performed by Theta*. While this number shoots up to on random maps where the line of sight checks are reduced anywhere between 55 - 65%. This is in itself proves that as the line of sight checks are reduced the time taken to plot the path from the start to the goal node will reduce.

We can prove this by taking single instances from data tables provided above. If we look at Table 2 where the results to find the time taken to plot are path is observed and we look at the second row displaying the results for a map with a 10% obstacle spread the performance improvement is just 8% while for the same table and a 20% obstacle spread the performance improvement is 36% this improvement can be linked to the reduction in line of sight checks shown in table 6 where the reduction in line of sight checks for a 10% spread for the same results in table 2 is 12% while for the 20% obstacle spread the reduction is line of sight checks is 45%.

Even though we see an overall improvement in the results from the reduction in line of sight checks, the performance is also affected the way in which the obstacles are spread across the map. The results reflect this when we look at the results of the random map and user designed maps. Where the average performance improvement is 20% as stated above but the performance increase to 30 % is observed frequently in random maps as opposed to user designed maps where there are few instances of C – Theta* gaining a huge performance improvement.

5.3 Region Based Analysis

To test whether the region size affects the improvement of results in C- Theta* we conducted experiments on 3 random maps with obstacle densities varying form 10 – 30%. The maps were divided into regions of 10, 15, 20 and the start and goal location were chosen at random on the maps. The tests were for 10 iteration each.

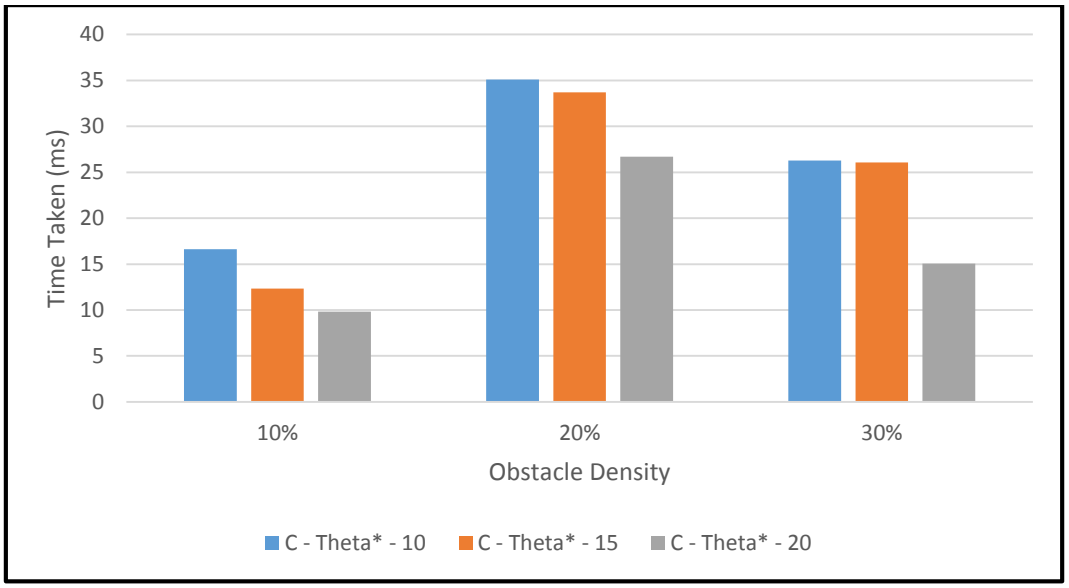

Fig 5.5: Avg. Runtime of each of the algorithms to plot the paths on a random maps with varying region sizes.

In our experiments we look at two parameters that largely affect the performance of C - Theta* namely the region size and the obstacle density. To correlate whether there is a difference in performance based on region size the above experiment was conducted. We also wanted to see if a link exists between the region size and obstacle density.

We shall first look at region size from the above graphical representation we can conclude that region size does affect the performance of C – Theta*

If we observe the result on an abstract level we can conclude that the performance of C – Theta* is affected by region size if we look at the difference between a region size of 10 and a region size of 20 there is a considerable performance improvement. For

example if we look at the graph and compare the results of a map with 10% obstacle density we see an improvement of almost 40% when the region size is increased from 10 to 20 this also seen when the obstacle density is 30%.

Thus as the region size increases the performance of C-Theta* also improves.

If we look closer there is also a relation between the obstacle density and region size for example in the above graph when the region size is larger the obstacle density is high the performance of C-Theta* improves considerably and this trend continues as observed in Fig 5.5.

Thus we can also conclude that if a grid map with low obstacle exists then tune the region size appropriately for best results.

5.4 Path Length Analysis

As explained in the previous chapters to improve the path length or shorten the path length most path finding algorithms would need to compromise on the time taken to find this shortest path. This can be clearly seen in the case of Theta* where is plots paths that are shorter than A* but the time taken to find these paths is significant especially as the maps size increases.

Below is a graphical representation of the path length in user designed maps as well as random maps.

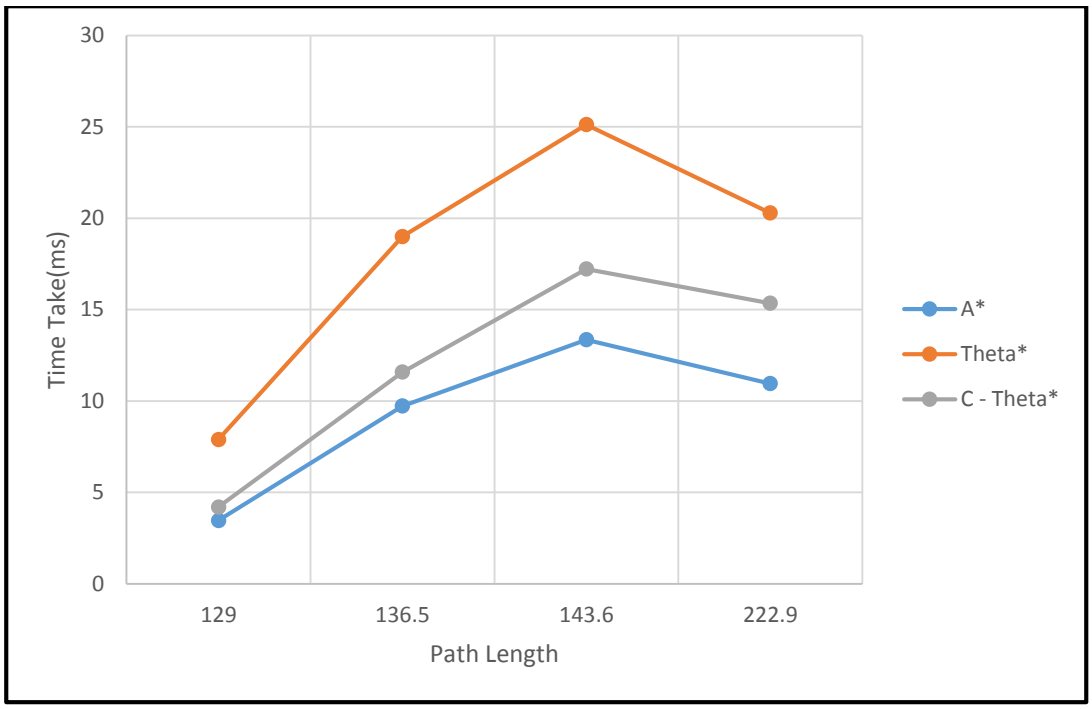

Fig 5.6: The average path length compared with the time taken to find the goal from source on user designed maps of size 100 ×100.

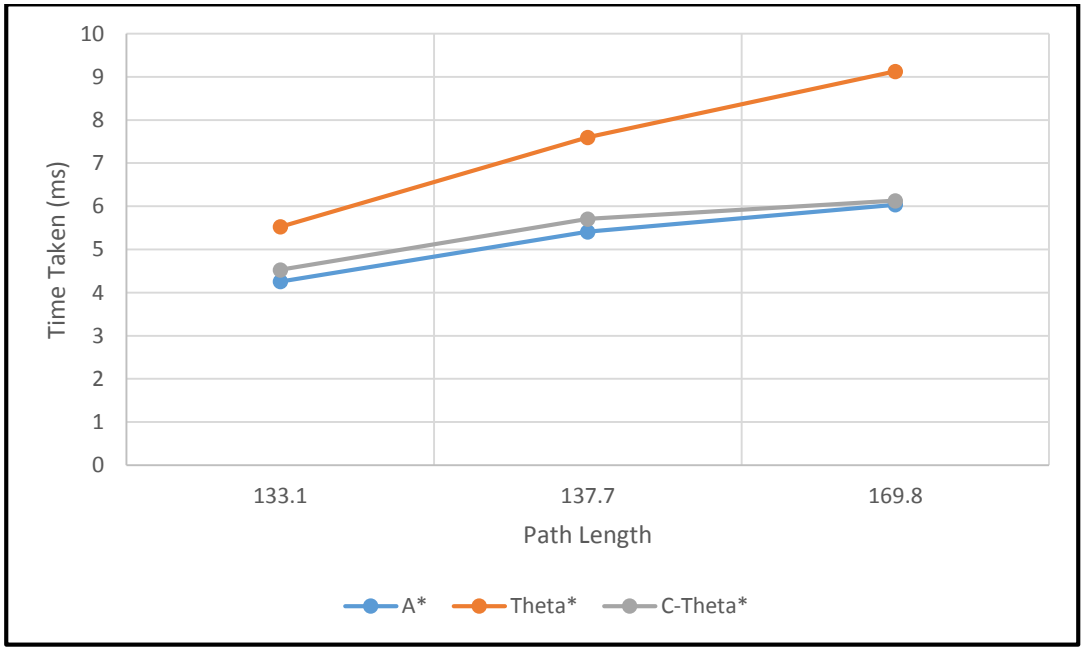

Fig 5.7: The average path length compared with the time taken to find the goal from the source on random maps of size 100 ×100.

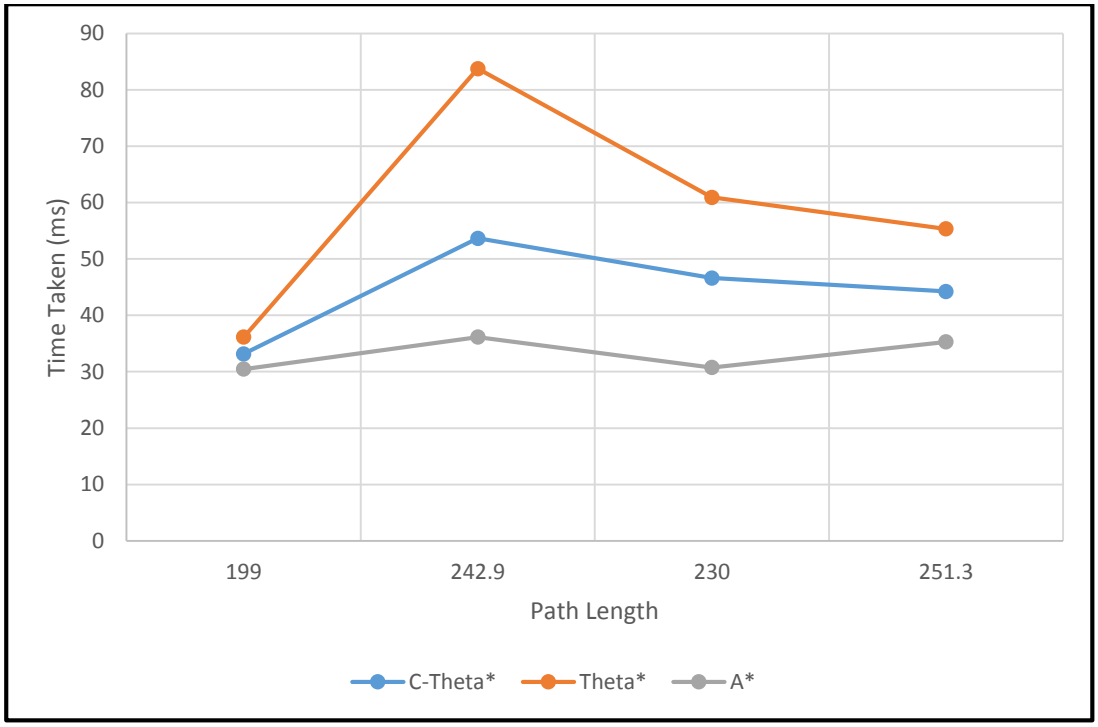

Fig 5.8: The average path length compared with the time taken to find the goal from the source on user designed maps of size 200 × 200.

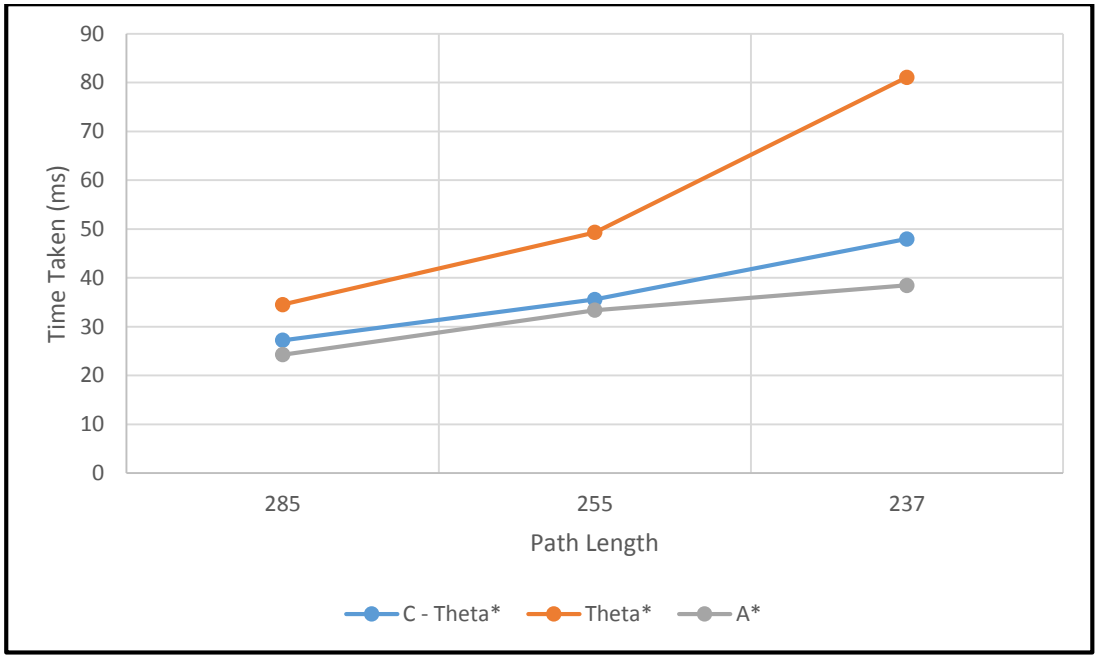

Fig 5.9: The average path length compared with the time taken to find the goal from the source on random maps of size 200 × 200.

From the figures of path length vs time taken above in figs 5.6, 5.7, 5.8, 5.9 C- Theta* still finds shorter paths than A* and the paths are as long as Theta*. Also it follows a

similar pattern noticed in our earlier analysis of the LOS and region based analysis, i.e., as the obstacle density increases the path plotted by C – Theta* is extremely close to the path plotted by A* though still shorter than A*.

It was observed the paths plotted were similar to Theta* when the obstacle densities are low and as the obstacles increase the path plotted by C – Theta* were similar to A*, but an important observation to be made is that the path quality does not degrade as we try and optimize the time to find the shortest path from goal to destination.

5.5 Standard Deviation

To validate the results of our experiments we have calculated the standard deviation for each of the dataset for one particular problems on grid maps. The standard deviation is calculated for a random as well as user designed map for the dimensions used in the experiments above in the same pattern in which all the results were documented, i.e., by ignoring the first 5 iterations and collecting the data of the next 10 iterations.

Obstacle Density	A*	Theta*	C- Theta*
10%	0.34	1.312472	1.381999
20%	0.2277	0.978325	0.143853
30%	0.30925	0.270195	0.092681
40%	0.193893	0.093759	0.154389

Table 7: Standard Deviations for user designed maps of size 100 × 100.

Obstacle Density	A*	Theta*	C- Theta*
10%	0.387187	0.119655	0.5092
20%	0.135549	0.238611	0.294189
30%	0.279773	0.307399	0.152067

Table 8: Standard Deviations for random maps of size 100 × 100.

Obstacle Density	A*	Theta*	C - Theta*
10%	0.699237	0.928006	1.022301
20%	0.844314	1.129114	1.238258
30%	0.935076	1.345235	1.312908
40%	1.284351	1.01688	1.57082

Table 9: Standard Deviations for user designed maps of size 200 × 200.

Obstacle Density	A*	Theta*	C - Theta*
10%	1.468292	1.928006	1.23006
20%	0.539145	1.424891	1.614983
30%	1.168435	1.388527	1.109081

Table 10: Standard Deviations for random maps of size 200 × 200.

The tables 7, 8, 9, 10 displays the standard deviation for each map with different obstacle densities. On observing the value for the standard deviation for each of the datasets we can conclude the experimental set up for testing the performance of each of the algorithms is very good because the individual values of each run lie between an extremely tightly coupled interval of +1.3 to -1.3 approximately.

Summary

In our experimental analysis we looked at two types of maps and performed experiments related to obstacle density, region sizes, and time and path length. In our Experimental analysis we provided an insight into how C- Theta* performs better than Theta* and how clustering can be successfully used to make maps more informative without manually increasing knowledge of grid maps.

It was observed that on average C- Theta* performed 20-30% better than Theta* and still plotted paths shorter than A*.

Based on our results we can make the following conclusions.

1. The results of C- Theta depends on the pattern in which K-Means labels the regions.

2. As the obstacle density increases C- Theta* performs better and reports times closer to A* as well as plotting shorter paths than A*.

3. As the region size increases the performance of C- Theta* improves considerably the best results were reported using a region size of 20.This can be attributed to the fact as the region size increases the number of line of sight checks reduce thus the gain in time.

4. The path quality of C- Theta* does not degrade, i.e., it still plots paths shorter than A* our base algorithm while reducing the time to plot such paths.

5. The time taken by the C – Theta* also depends on the Obstacle spread of the map.

CHAPTER - 6 CONCLUSION

The thesis has explained several concepts of path finding. It looks at any-angle path finding and how it differs from traditional path finding methods. This book builds upon the work done in any-angle path finding and specifically looks at Theta* in detail. Theta* is an any-angle path finding algorithm that uses line of sight checks to eliminate the constraint of traversal on the edges of a grid map. Theta* differs from A* by performing line of sight checks along with the search to find the goal from a particular start node.

C - Theta* uses the same characteristics of Theta*, i.e., performing line of sight checks during the search and maintaining its structural properties of being easy to implement and use just like Theta* and plot paths longer than Theta* if not as short as A*. In C - Theta* we introduce three concepts that makes C – Theta* unique in the field of any-angle path finding.

Regions: Dividing the map into regions reduces the overall complexity this concept has been influenced by real life, i.e., When an individual needs to search over a large area our brain tends to divide the area into regions while performing a search.

Clustering: To improve the time of any-angle algorithms and enhance their searching capabilities we build upon our real life example explained above of searching an individual in a large area though our brain creates regions it also associates priority with these regions based on some attributes. Similarly C – Theta* implements clustering to assign labels of high and low density to the created regions to make the map considerably informative.

Selective LOS: Based on the results of clustering, C – Theta* can make intelligent guesses of when a line of sight check needs to be performed or not on a node under consideration based on its location in a region of a map.

Obstacle Spread: While comparing the results of C – Theta* an important attribute was discovered, the influence of the pattern of the obstacle spread on a map.

This thesis compares C – Theta* with Theta* the algorithm it hopes to improve and A* the algorithm against which Theta* performs considerably poorly thus A* .Thus A* and Theta* can be consider as the min and max time and the ideal time for C- Theta* would be in between these two maxima and minima.

Based on the results we can conclude that C – Theta* manages to perform considerably well against Theta* and manages to catch up to A* to some extent at times. Some key observations were made based on the results of C – Theta* from which we can conclude that C – Theta* works well in environments where the obstacle density is high. It also performs well where the obstacle are spread across the map creating evenly distributed clusters of high and low density regions. Also the performance of C – Theta* can considerably improve in maps where the obstacle density is low by increasing the region size of the grid map under consideration.

This algorithm can be used in game maps which are considerably huge. It can be used in GPS systems to make the maps more informative as well as improving the paths plotted for a user based on division of the regions into high and low density.

CHAPTER - 7 FUTURE WORK

In this thesis we have introduced C – Theta* an any-angle algorithm used on static grid maps. In the future we would like to extend C – Theta* on dynamic maps and document its performance with other path finding algorithms used in dynamic environments like D* and D* lite. Another interesting possibility is to explore the use of C – Theta* on different representations of graphs such as navmesh and way points.

In the future work could be done to improve the line of sight process such that it performs better if not as good as A* with respect to time taken to plot the path from start to goal node, for example introducing time based strategies to execute line of sights on grid maps. Currently in this thesis we have implemented only the Euclidean distance heuristic to test our results. In the future, experiments can be conducted to test the performance of C- Theta* with different heuristics. This would be an interesting test to observe the best suited heuristic to be used with C – Theta*.

Currently we have used clustering to improve the information of the terrain in a map, in the future work can be done to explore the possibilities of other methods to improve the information of a map. For example creating quad trees algorithm to divide the grid into regions based on obstacles and comparing the results with clustering would be an interesting topic to explore. Also testing whether the performance of C – Theta* can be improved by implementing other clustering algorithms and comparing the results would be a nice path to explore the advantages of clustering in path finding.

While performing experiments it was observed that an important factor to measure the performance of C – Theta* is the obstacle spread. In the future we would like to perform experiments to prove whether the obstacle spread of a map has an effect on the performance of path finding algorithms.

REFERENCES

[1] 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).

[2] 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.

[3] Yang, Zi, and Wensheng Yu. "Block-Guided Theta* Path Planning Algorithm." The 26th Chinese Control and Decision Conference (2014 CCDC) (2014): n. pag. Web.

[4] Carsten, Joseph, Dave Ferguson, and Anthony Stentz. "3D Field D: Improved Path Planning and Replanning in Three Dimensions." 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (2006).

[5] 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.

[6] "Clustering - Introduction." Clustering - Introduction. N.p., n.d. Web. 18 Nov. 2015.

[7] "Semi-Supervised Classification Using Pattern Clustering." Novel Strategies Semi-Supervised and Unsupervised Machine Learning (2013): 127-81. Http://ceeserver.cee.cornell.edu/. Web.

[8] 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.

[9] Algfoor, Zeyad Abd, Mohd Shahrizal Sunar, and Hoshang Kolivand. "A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games." International Journal of Computer Games Technology 2015 (2015): 1-11. Web.

10] Xiao Cui and Hao Shi “An Overview of Pathfinding in Navigation Mesh” IJCSNS International Journal of Computer Science and Network Security, VOL.12 No.12, December 2012.Web.

[11] Topfat - Tool for Path Finding Algorithm Testing - Google Project Hosting." Topfat - Tool for Path Finding Algorithm Testing - Google Project Hosting. N.p., n.d. Web. 11 Dec. 2015.

[12] Yap,Peter,”Grid-Based Pathfinding” Advances in Artificial Intelligence Lecture Notes in Computer Science(2002):44-55.Web

[13] Board, Ben., Ducker, Mike., “Area Navigation: Expanding the
Path-Finding Paradigm”, Game Programming Gems 3, Charles River Media, 2002

[14] S. Russel and P. Norvig, "Artificial Intelligence: A Modern Approach," in Artificial Intelligence: A Modern Approach, Prentice Hall, 2002, pp. 59-94.

[15] Bulitko, Vadim, Yngvi Björnsson, Nathan R. Sturtevant, and Ramon Lawrence. "Real-Time Heuristic Search for Pathfinding in Video Games." Artificial Intelligence for Computer Games (2011): 1-30. Web.

[16] Misa, Thomas J. "An Interview with Edsger W. Dijkstra." Communications of the ACM Commun. ACM 53.8 (2010): 41. Web.

[17] Dijkstra, E. W. "A Note on Two Problems in Connexion with Graphs." Numer. Math. Numerische Mathematik 1.1 (1959): 269-71. Web.

[18] Farreny, Henri, and Henri Prade. "Heuristics—intelligent Search Strategies for Computer Problem Solving, by Judea Pearl. (Reading, Ma: Addison-Wesley, 1984)." International Journal of Intelligent Systems Int. J. Intell. Syst. 1.1 (1986): 69-70. Web.

[19] Stentz, Anthony. "Optimal and Efficient Path Planning for Partially Known Environments." Intelligent Unmanned Ground Vehicles (1997): 203-20. Web.

[20] Koenig, S., and M. Likhachev. "Improved Fast Replanning for Robot Navigation in Unknown Terrain." Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292) (2002): n. pag. Web.

[21] Lloyd, S. "Least Squares Quantization in PCM." IEEE Trans. Inform. Theory IEEE Transactions on Information Theory 28.2 (1982): 129-37. Web.

APPENDIX

IEEE COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE. Please read the form carefully and keep a copy for your files.

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

Pramod Mendonca , Scott Goodwin

2015 International Conference on Computational Science and Computational Intelligence

COPYRIGHT TRANSFER

The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the "IEEE") all rights under copyright that may exist in and to: (a) the Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this form.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. The undersigned agrees that publication with IEEE is subject to the policies and procedures of the IEEE PSPB Operations Manual.
4. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing grant of rights shall become null and void and all materials embodying the Work submitted to the IEEE will be destroyed.
5. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.
6. The author hereby warrants that the Work and Presentation (collectively, the "Materials") are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the author has obtained any necessary permissions. Where necessary, the author has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE

You have indicated that you DO wish to have video/audio recordings made of your conference presentation under terms and conditions set forth in "Consent and Release."

CONSENT AND RELEASE

1. In the event the author makes a presentation based upon the Work at a conference hosted or sponsored in whole or in part by the IEEE, the author, in consideration for his/her participation in the conference, hereby grants the IEEE the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record, digitize, broadcast, reproduce and archive, in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the "Presentation"). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IEEE and live or recorded broadcast of the Presentation during or after the conference.
2. In connection with the permission granted in Section 1, the author hereby grants IEEE the unlimited, worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IEEE from any claim based on right of privacy or publicity.

BY TYPING IN YOUR FULL NAME BELOW AND CLICKING THE SUBMIT BUTTON, YOU CERTIFY THAT SUCH ACTION CONSTITUTES YOUR ELECTRONIC SIGNATURE TO THIS FORM IN ACCORDANCE WITH UNITED STATES LAW, WHICH AUTHORIZES ELECTRONIC SIGNATURE BY AUTHENTICATED REQUEST FROM A USER OVER THE INTERNET AS A VALID SUBSTITUTE FOR A WRITTEN SIGNATURE.

Pramod Mendonca	06-11-2015
Signature	**Date (dd-mm-yyyy)**

Information for Authors

AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at http://www.ieee.org/publications_standards/publications/rights/authorrightsresponsibilities.html Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

RETAINED RIGHTS/TERMS AND CONDITIONS

- Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
- Authors/employers may reproduce or authorize others to reproduce the Work, material extracted verbatim from the Work, or derivative works for the author's personal use or for company use, provided that the source and the IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
- Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use.The IEEE Intellectual Property Rights office must handle all such third-party requests.
- Authors whose work was performed under a grant from a government funding agency are free to fulfill any deposit mandates from that funding agency.

AUTHOR ONLINE USE

- **Personal Servers**. Authors and/or their employers shall have the right to post the accepted version of IEEE-copyrighted articles on their own personal servers or the servers of their institutions or employers without permission from IEEE, provided that the posted version includes a prominently displayed IEEE copyright notice and, when published, a full citation to the original IEEE publication, including a link to the article abstract in IEEE Xplore. Authors shall not post the final, published versions of their papers.
- **Classroom or Internal Training Use.** An author is expressly permitted to post any portion of the accepted version of his/her own IEEE-copyrighted articles on the author's personal web site or the servers of the author's institution or company in connection with the author's teaching, training, or work responsibilities, provided that the appropriate copyright, credit, and reuse notices appear prominently with the posted material. Examples of permitted uses are lecture materials, course packs, e-reserves, conference presentations, or in-house training courses.
- **Electronic Preprints.** Before submitting an article to an IEEE publication, authors frequently post their manuscripts to their own web site, their employer's site, or to another server that invites constructive comment from colleagues. Upon submission of an article to IEEE, an author is required to transfer copyright in the article to IEEE, and the author must update any previously posted version of the article with a prominently displayed IEEE copyright notice. Upon publication of an article by the IEEE, the author must replace any previously posted electronic versions of the article with either (1) the full citation to the

IEEE work with a Digital Object Identifier (DOI) or link to the article abstract in IEEE Xplore, or (2) the accepted version only (not the IEEE-published version), including the IEEE copyright notice and full citation, with a link to the final, published article in IEEE Xplore.

Questions about the submission of the form or manuscript must be sent to the publication's editor.
Please direct all questions about IEEE copyright policy to:
IEEE Intellectual Property Rights Office, copyrights@ieee.org, +1-732-562-3966

VITA AUCTORIS

NAME: PRAMOD MENDONCA

PLACE OF BIRTH: PUNE, INDIA

EDUCATION: MASTER OF COMPUTER SCIENCE,

UNIVERSITY OF WINDSOR