INTELLIGENT TRANSPORT SYSTEM AND PATH FINDING IN CITY ENVIRONMENT

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

Computer Science & Engineering

under the Supervision of

Mr. Ravindra Bhatt

By

Prabhjot Singh Kalra-111297

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Contents

List of Figures

Certificate

This is to certify that project report entitled "**Intelligent Transport System and Pathfinding in city environment**", submitted by **Prabhjot Singh Kalra** partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:

Mr. Ravindra Bhatt

Assistant Professor Dept. of (CSE&IT)

ACKNOWLEDGEMENT

I would like to articulate my profound gratitude and indebtedness to those persons who helped me in the project. First of all, I would like to express my obligation to my project guide Mr. Ravindra Bhatt for his motivation, help and supportiveness. I am sincerely thankful to him for his guidance and helping effort in improving my knowledge on the subject. He has been always helpful to us in all aspects and I thank him from the deepest of my heart. An assemblage of this nature could never have been attempted without reference to and inspiration from the works of others whose details are mentioned in reference section. I acknowledge my indebtedness to all of them. At the last, my sincere thanks to all my friends who have patiently extended all sorts of helps for accomplishing this project.

Date: Prabhjot Singh Kalra

Abstract

Recently the modernization of transport system is a new area of research in wireless communication system. Such transport system aims to create a more secure, reliable and efficient system to improve road safety, driver assistance and user comfort on road. This gives the new idea of intelligence transportation system (ITS) which is very popular in developed countries. Shortest Path problems are inevitable in road network applications such as city emergency handling and drive guiding system, in where the optimal routings have to be found. We propose a path finding in the city map to the VANET application utilizing the well-known A* algorithm. In our work the details of A* algorithm are addressed as a basis of delivering a number of optimization techniques to find the best possible path. This project aims to investigate the single source shortest path problems and intends to obtain some general conclusions by examining A* algorithm approach. Our work is carried out into three different scenarios and respective results are obtained and compared with each other.

1 – Introduction

Now a day, wireless networks have been developed into a very important medium in the field of data communication. For the period of the last two decades an incredible improvement has been occurred in the area due to the technological advancement in Vehicular computers and wireless data communication devices. Therefore, wireless communication has become very popular for accessing information and various services in Vehicular environment where the users are not restricted to a particular location.

With the expected growth in mobile devices and mobile traffic, Vehicle-to-vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communications are expected to become more in demand and will continue to grow. VANETs can be used to provide a wide range of services, including both safety and non-safety related applications. Examples include vehicular safety traffic management services, surveillance services, and mobile vehicular cloud services.

1.2 What is VANET?

Vehicular networks are very fast emerging for deploying and developing new and traditional applications. In VANET we provide communications among nearby vehicles and between vehicles and nearby fixed equipment. A set of moving vehicles in the road which are competent of communication with each other without any fixed infrastructure or centralized administration, are used in VANET. These networks permit us to construct an intelligent transportation system (ITS) whose major element is the inter–vehicular communication (IVS). It is characterized by rapidly changing topology, high mobility, and ephemeral, one-time interactions. VANETs are characterized from the movement and self-organization of the nodes. To improve the security, efficiency and enjoyment in road transport through the use of new technologies for information and communication is the goal of this system.

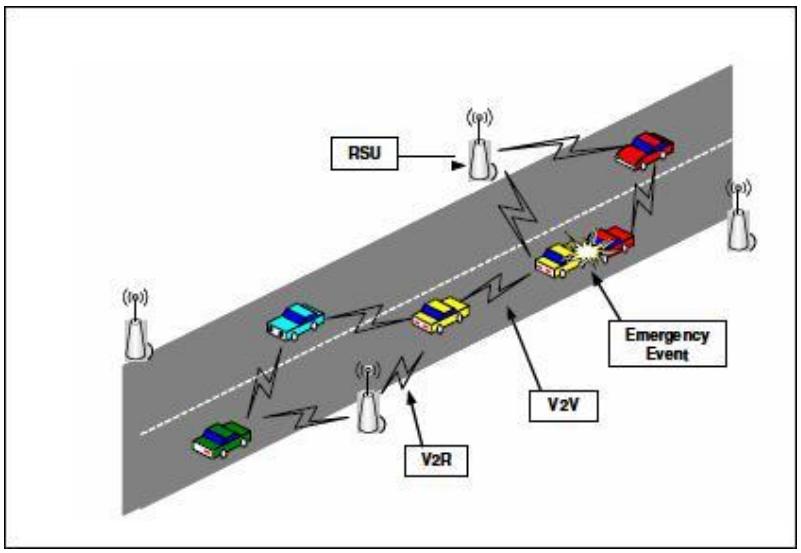


Figure-1.1: The Example of VANET [1]

1.2.1 Properties of Vehicle for VANET:

The moving vehicles are the main component in VANET. These vehicles are known as nodes in VANET. The following properties of vehicles are used for better operation in VANET.

a) **Sensing:** The different types of sensor are used to sense the Different vehicular and environmental conditions (state of the vehicle, state of road, weather condition, pollution and others).
b) **Processing:** The data or information coming from the different sensors are processed by the vehicles.
c) **Storage:** To store the different type of data and processing results for future uses require a large storage space.
d) Routing: The vehicles (nodes) should have the potential to communicate with each other in the VANET (with the help of IP or Cellular for example).

1.2.2 What is Smart Vehicle?

Smart vehicles are equipped with multi interface cards, sensors and on-board units.
Such vehicles are equipped with on-board wireless devices (*e.g.*, UMTS, IEEE 802.11*p*, Bluetooth, etc.) and sensors (*e.g.*, radar, etc.), for efficient transport and management applications.
The radar present on on-board could be used to sense traffic congestions and automatically slow the vehicle. In accident warning systems, sensors can be used to determine that a crash may be occurred if air bags were deployed; this kind of information is then relayed via V2I or V2V within the vehicular network.
Commonly, a smart vehicle is equipped with the following technologies and devices:
(i) A wireless transceiver for data transmissions among vehicles (V2V) and from vehicles to RSUs (V2I).

(ii) A Central Processing Unit (CPU) which implements the applications and communication protocols.

(iii) A Global Positioning Service (GPS) receiver for navigation and positioning services.

(iv) An input/output interface for the interaction of human with the system.

1.2.3 Application of VANET:

There are many applications of the VANET. These can broadly be divided into four major categories:

a) **Road Safety Application:** Road safety has become very important aspect in vehicular transport for most of the developed countries. Due to the increase of road accident in recent times, VANET has become very much popular. To improve road safety, the VANET provides necessary information regarding prevention of collisions with static or dynamic obstacles, condition of road and traffic congestion, weather related information etc.

b) **Driver Assistance Application:** Driver assistance and collaborative driving is another important application of VANET. It helps the driver in specific situation like vehicle over talking; prevention of straight or curved lane exists etc. It also improves the road safety measures.

c) **Electronic Toll Collections (ETCs):** Each vehicle can pay

the toll electronically when it passes through a Toll Collection Point (a special RSU) without stopping. The Toll Collection Point will scan the Electrical License Plate at the OBU of the vehicle, and issue a receipt message to the vehicle, including the amount of the toll, the time and the location of the Toll Collection Point.

d) **Comfort Application:** Comfort application is one of important application of VANET. By the help of VANET, Different communication services such as Vehicular access to internet, electronics messaging, inter-vehicle chat, internet accessing, network games etc. can also be done.

1.2.4 CHALLENGING ISSUES IN VANET

- **Routing protocols:** Routing plays an important role in VANET applications but the high-speed mobility of vehicles and their rapidly changing topology results in conventional VANET routing protocols being inadequate to efficiently and effectively deal with this unique vehicular environment as intermediate nodes cannot always be found between source and destination and end-to-end connectivity cannot always be established. This has prompted researchers to find scalable routing algorithms that are robust enough for the frequent path distributions caused by vehicle mobility.

- **Network Management:** Due to high mobility, the network topology and channel condition change rapidly. Due to this, we can't use structures like tree because these structures can't be set up and maintained as rapidly as the topology changed.

- **Congestion and collision Control**: The unbounded network size also creates a challenge. The traffic load is low in rural areas and night in even urban areas. Due to this, the network partitions frequently occurs while in rush hours the traffic load is very high and hence network is congested and collision occurs in the network.

- **Environmental Impact:** VANETs use the electromagnetic waves for communication. These waves are affected by the environment. Hence to deploy the VANET the environmental impact must be considered.

- **MAC Design:** VANET generally use the shared medium to communicate hence the MAC design is the key issue. Many approaches have been given like TDMA, SDMA, and CSMA etc. IEEE 802.11 adopted the CSMA based Mac for VANET.
- **Security:** As VANET provides the road safety applications which are life critical therefore security of these messages must be satisfied.

1.3 Motivation

The original impetus for the interest in ITS was provided by the need to inform fellow drivers of actual or imminent road conditions, delays, congestion, hazardous driving conditions and other similar concerns. In highly dynamic city environment the driver should be aware of the city conditions like, traffic, obstacles, accidents etc. These conditions are major concerns in the field of Intelligent Transport System. Thus there should be an application which is capable of assisting the driver about the road conditions and finding the best possible path.

1.4 Problem Statement

This work aims to design a framework for supporting path-planning analysis of city map, based on evaluation of transport cost and safety related objectives in order to maximize the overall productivity of vehicular movement. Further this application aims to enable the driver to make strategic decisions regarding the movement of vehicles on a city map, and layout planning that improves travel distance, safety and visibility to allow the examination of path scenarios and the selection of the best possible path. The path-planning IT platform can be applied to a number of instances of site layouts for the evaluation of a dynamic site layout. In our work we have used A* algorithm to find the shortest path in a city map.

1.5 Organization of the Work:

The report is organized as follows: **Chapter 2** includes the Literature survey and gives brief idea about the path finding history and various types of search methodologies. **Chapter 3** deals with the path finding applications and algorithms. **Chapter 4** presents the detailed description of the A* algorithm and its step by step explanation. **Chapter 5** includes the proposed work, flow chart and observations. **Chapter 6** concludes the report and discusses the future works.

Literature Survey

2.1- The History of Pathfinding

Pathfinding is like any other Artificial Intelligence problem that has to be solved. Predominately the problem is to "Find the shortest path from A to B", but other forms of pathfinding issues also exist, such as "Find any path from A to B" or "Find a path from A to B via C". All these pathfinding problems basically equate to the same problem; a Search problem. Graph search problems in particular date all the way back to 1736, with Leonhard Euler's Königsburg Bridges, where Euler proved it was impossible for someone to walk across Königsburg's 7 bridges without travelling over the same bridge more than once, and return to the same place that you started.

Another famous historical problem is the Travelling Salesman Problem (TSP), where a salesman wishes to visit a specified number of cities, one at a time. The salesman cannot visit the same city twice, and must visit all of the cities using the cheapest solution (i.e. the solution where the salesman travels the least distance overall). In 1954, George Dantzig et al calculated the solution to 49 cities, and over the years as computational power has increased, so have the number of cities that have been calculated. Pathfinding is used in many forms in a variety of different programs. In Route Generation programs such as AutoRoute 2005 (Microsoft, 2004), pathfinding is used to find the shortest path from one real world location to another on a 2 dimensional map.

2.2 - Terms Used in Pathfinding

Pathfinding is usually always thought of as being conducted in a Search Tree. In a search tree, there are several key terms which describe different parts of the search tree. A Branch refers to a section of the search tree, which is made up of several nodes. Any particular branch contains a 'parent' node (the stem of the branch), and children nodes (the other nodes attached directly/indirectly to the parent node. A Node is a point where two or more lines meet it (i.e. a sort of junction). In project terms, a node refers to a single square in the tile based environment. The Root refers to the highest Node in the tree.

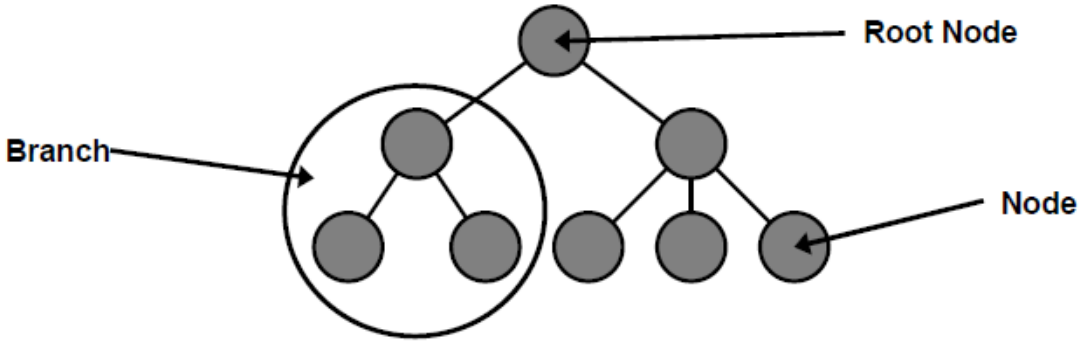


Figure 1 - Example Search Tree

There are several other phrases used when conducting searches. When we speak of the Cost in pathfinding terms, we refer to the time to get from one node to another. The cost can be anything from being derived from the distance between the two nodes, to a uniform cost. The term Heuristic has several different meanings, which have developed over the years that AI has been around. According to Norvig, R (1995. p.94), Heuristics has meant everything from "the study of methods for discovering and inventing problem solving techniques", to "the opposite of algorithmic". For this report, the definition of heuristic will mean "a function that provides an estimate of solution cost". This means that a heuristic algorithm must be able to work out an estimation of the solution in relation to the start, as a limiter to the search algorithm.

2.3 - Representing the Path finding in a Virtual Environment

In our work we have used a city map as a virtual environment for the purpose of implementation of the pathfinding algorithm.

There are several steps to achieve pathfinding in a virtual environment. The virtual environment must first be partitioned, where each section of the environment corresponds to a node which the pathfinding algorithm can use to navigate. Finally, one must choose the most effective search algorithm for the job at hand. Partitioning the virtual environment is vital for being able to achieve pathfinding, and can be done in several different ways. It can be partitioned as a Grid (or Tile); a uniform splitting process down to the lowest level where the smallest obstacle takes up one tile. The division of grid is important to analyze the degree of freedom of the nodes.

The degrees of freedom that the entities can enjoy must also be decided, either the 4-way adjacency, where the entity can only move forward, backwards, left or right, or alternatively an entity would be allowed to travel in any of the 8 directions (diagonally or straight movements), which would cut down on the distance the entity would have to travel. The figure below shows an example of a grid based world, containing an obstacle.

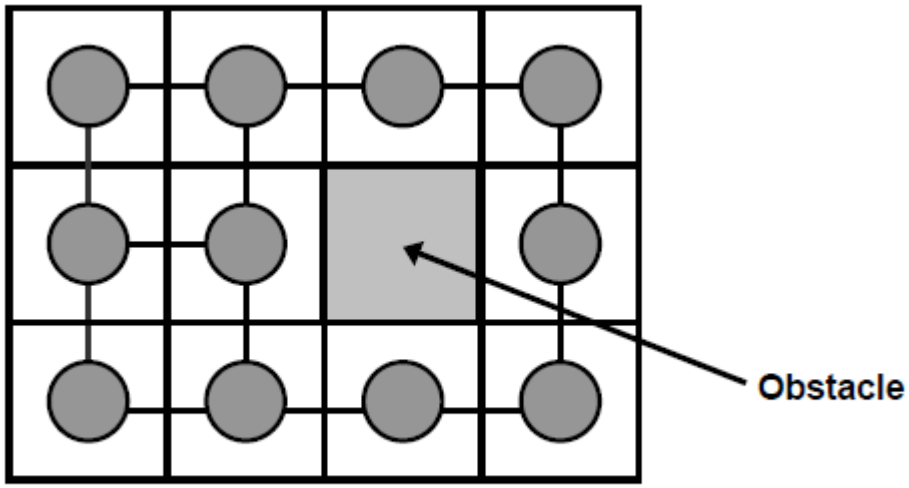


Figure 2 - Tile Based world with graph overlay

Another partitioning type includes Polygon partitioning, where each surface polygon is a node. This proves easier to create more realistic looking virtual obstacles, since there is not an ordered tile structure (obstacles can be of any geometrical shape rather than a square). In polygon partitioning finding out the cost of each link is a costly process if there are a large number of links (in grid partitioning each cost is uniform, refer figure 3). It can also leads to the problem that if there is a large polygon, applying only one node to the large polygon area means that detail in the graph is lost, which makes it harder to path-find to a specific location within the large polygon. The figure below shows an example of a grid based world, containing an obstacle. Note that the bottom left node, due to its large size, has lost some detail in the graph overlay, compared to the bottom right nodes, which are densely packed and so offer a good level of detail.

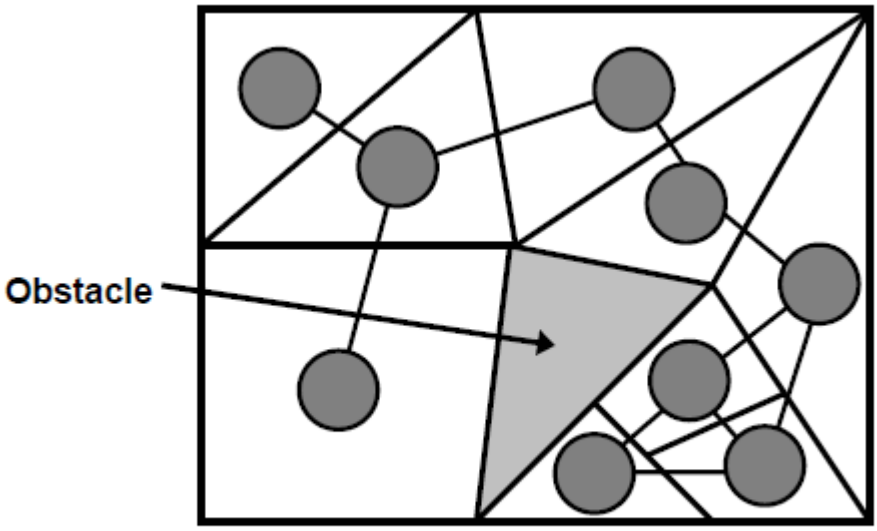


Figure 3 - Polygon Based world with graph overlay

2.4– Search Techniques

There are a number of search algorithms that can be adopted for use in pathfinding from source to destination. The major search techniques are discussed in the following few pages.

2.5- Breadth-First Search

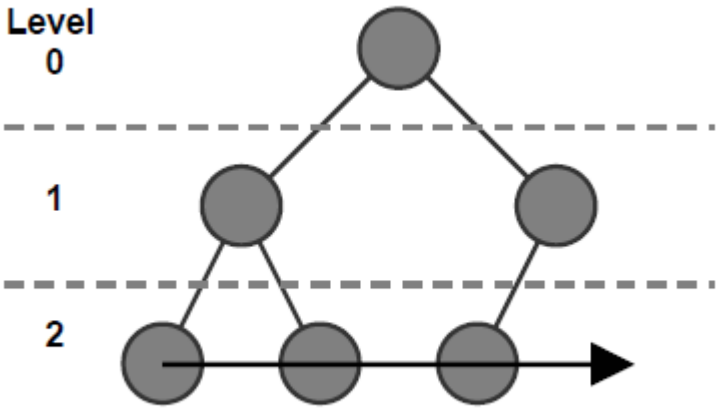


Figure 4 - Breadth-First Search

A Breadth-First search (Norvig, R. 1995. p.74) searches all the root nodes of the previous nodes before deepening the search another level. This way, relatively shallow solutions (i.e. close goals) can be found quickly. However, deeper solutions take longer to discover than the depth-first search. They also use a lot of memory, as each level the algorithm progresses, the nodes that are to be checked increase exponentially. It may also only find a local solution, rather than the optimal solution, unless an exhaustive search is conducted of the entire search tree.[2]

The figure above shows a breadth-first search in action, where it searches each node on the level before working on the next level. In the example it is currently working through level 3, and has one more node left to complete before moving onto level 4. A way to find the optimal solution is to modify the breadth-first search into a Uniform Cost search.

This expands the lowest cost node before expanding the other ones in the level, totaling up the cost of the branch. This way, if the algorithm finds a branch that leads to a goal, it knows a limit to place upon the other branches' costs, and if the other branches have higher costs and have not found the goal, that means that the branch with the goal is the optimal solution.[2]

2.6- Depth-First Search

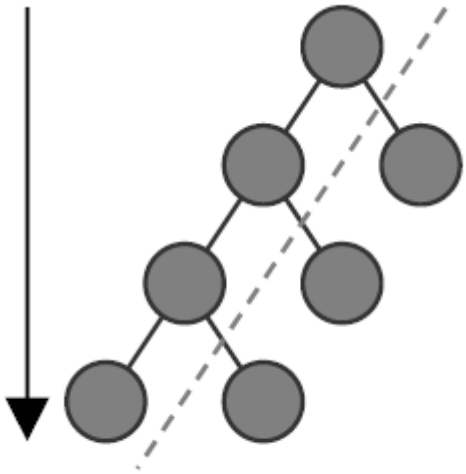


Figure 5 - Depth-First Search

A Depth-First search (Norvig, 1995, p.77) expands nodes towards the deepest levels of the search tree. This search does well when the solution is located quite far down the search tree. However, one of the problems with depth-first is that if it makes a mistake and takes the wrong path early on in the tree, it may take a lot longer to recover and go on the right track. Figure 2 shows an example depth-first search, where it searches down the left most branch nodes first, before working its way to the right of the search tree. There are a couple of modifications to counter this, as described by Norvig (1995) as being 'Depth-Limited' search, and 'Iterative deepening search'.[2]

Depth-Limited (p.78) does exactly as the title suggests; it limits how deep that the search algorithm is allowed to search before giving up on that branch and pursuing another one. This has the advantage of if it sets off on a wrong branch which could in theory be of infinite deepness, it will be able to search along a different branch of the tree if it reaches the limit of the search instead of continuing indefinitely. Iterative Deepening (p.79) searches for the solution in an ever-increasing depth limit, similarly to a Breadth-First search, but without the large memory usage.[2]

2.7- Best-First search

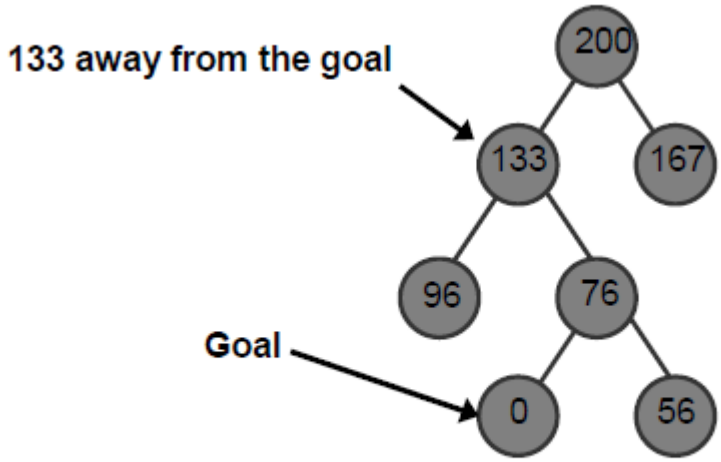


Figure 6 - Best-First search

A Best-First search (Norvig, 1995, p.92) expands the nodes which appear to be the best choice (i.e. the closest node to the solution, or the lowest cost solution). An example of a Best-First search is called a Greedy Search, (p.93) named so because it attempts to take the largest 'chunk' out of the remaining distance to the solution by choosing the node which gets the closest to the solution in the short term (not necessarily the best solution in the entire search space). [2]

This leads to problems where dead-ends are involved, as if a dead-end exists that is closer to the solution than another node which leads round to the solution, the greedy search will always choose the shortest path, not knowing it leads to a dead end. Even worse, if the search algorithm doesn't remember which nodes it has checked, it would endlessly cycle back and forth between the dead end and it's neighbour. The figure above shows an example of a Best-First greedy search, with the algorithm simply picking the closest node to the goal at each choice.[2]

The A* (A Star) pathfinding algorithm is the combination of a best-first search's efficiency and the uniform-cost search's completeness, as described by Norvig (1995, p.96). It was first presented by Hart et al in 1968, and it finds the shortest, most complete path in any graph.

The input of the A* Algorithm are the start and end points; as long as it knows where to start and where to finish, it can work out a path if there is one.

The basics of the A* algorithm are that for each node of the search tree, it calculates that node's Goal cost (the cost to get from the starting node to this particular node), the Heuristic (the estimated cost to get from this particular node to the goal) and the Fitness (the sum of the goal and the heuristic, basically assigning a cost value to this path). There are two other parts of the A* Algorithm; the Open and Closed lists. The Open list holds all the nodes which have not yet been expanded by the algorithm (i.e. the possible options left available to the pathfinder), and the Closed list holds all the nodes which have been expanded already (some nodes may be part of the path, while others are expanded paths which may have led to dead ends. The Closed list is used so the algorithm knows where it has searched so far, so it doesn't loop between two nodes.

In the next chapter we will study about the various path finding applications and see in what all field path finding is used.

Path Finding Applications

3.1 Path Finding Applications:

Path finding in the basic definition is moving an object from its initial position to the final location. Path Finding Algorithms (PFA) are used in many different application areas such as:

- Games and Virtual Tours
- Robot Motion and Navigation
- Driverless Vehicles
- Transportation Networks

3.1.1 PFAs in Games and Virtual Tours

Path finding is a very important part of game programming. Game characters move according to the path they (or computer) calculate. There are some algorithms used in games changing with the complexity and purpose of the path calculation. The most used algorithm in today's games is the A* Algorithm.

“What makes the A* algorithm so appealing is that it is guaranteed to find the best path between any starting point and any ending point, assuming, of course, that a path exists.”

3.1.2 Robot Motion and Navigation

Mobile robots moving in indoor or outdoor environments should have their route planning and navigation systems in order to be able to find their way. Generally the navigation and path finding units are placed on the robots, so that they can move by themselves. It is important to have this property, if we think about the usage of these robots. In military, detection of hazardous or explosive materials and discovery and investigation of unknown areas are very suitable for this type of robots.

3.1.3 Driverless Vehicles

Another application area of the path finding algorithms is the automated vehicles that will be able to find its way without making interaction with the obstacles. Once again its major usage is in military, but it can be implemented to different areas of our life. In the future, they can search for the lost outdoor adventurers, hikers, or they can be very useful in case of natural disasters, with the sensors they have.

3.1.4 Transportation Networks

In road networks it is getting more important to find the way to the destination point. If a person is new to a place, much time can be wasted until finding the destination. There exist some products developed to overcome this difficulty, providing a map of the region. After entering the starting point and the final location, it is possible to get the shortest path.

3.2. Path Finding Algorithms

There are many algorithms that are commonly known and used, ranging from simplex to complex, in order to be able to solve the path finding problem. The simplest approach is to walk directly towards the goal until met with any kind of obstacles. When met with any object, direction will be changed and it can be passed by tracing around the obstacle. This type of algorithm is an example of the “visually impaired Search” since it does not know or have any information about the path. Some other examples of the visually impaired Search are Breadth-First Search, Dijkstra's Algorithm and Depth-First Search.

There also exist some algorithms that plan the whole path before moving anywhere. Best-first algorithm expands nodes based on a heuristic estimate of the cost to the goal. Nodes, which are estimated to give the best cost, are expanded first. The most commonly used algorithm is A* algorithm, which is a combination of the Dijkstra algorithm and the best-first algorithm.

The different algorithms work in different ways. The breadth-first search begins at the start node, and then examines all nodes one step away (here we have 4 different nodes for Manhattan distance calculation and 8 different nodes for Euclidean type of distance calculation), then all nodes two steps away, then three steps, and so on, until the goal node is found. This algorithm is guaranteed to find a shortest path as long as all nodes have a uniform cost.

The bi-directional breadth-first search is where two breadth-first searches are started simultaneously, one at the start and one at the goal, and they keep searching until there is a node that both searches have examined. The final path is the combination of the path from the start to the intersection node, and the path from the goal to the intersection node.

3.2.1 Dijkstra's algorithm

Dijkstra's algorithm (named after its developer, E. Dijkstra [13]) looks at the unprocessed neighbors of the node closest to the start, and sets or updates their distances (in terms of cost, not number of nodes) from the start. The Dijkstra algorithm expands the node that is farthest from the start node, so it ends up "stumbling" into the goal node. Just like the breadth-first search; it is guaranteed to find the shortest path

Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph G = (V, E) for the case in which all edge weights are nonnegative. Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined. The algorithm repeatedly selects the vertex u V – S with the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u.

```
DIJKSTRA(G, s, t)
{
Define set S                                  // set of explored nodes
Define auxiliary d(u) for elements of S     // minimum distance from s to each node u
S = {s}                                       // start with source s as explored and
d(s) = 0                                     // no distance to get to it
while S ≠ V(G)
{
find a node v ∉S with at least one edge from S for which d '(v) = min(u,v)for u∈S d(u) +
w(u,v)
S = S ∪ {v}
d(v) = d'(v)
}
return d(t)
```

In the next chapter we will cover the detailed discussion of A* algorithm and various parameters used in the algorithm.

A* Detailed Approach

4.1 A* Algorithm

The algorithm was first described in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael [**15**]. It is a global space-search algorithm that can be used to find solutions to many problems, path finding being just one of them. It has been used in many real-time strategy games and is probably the most popular path finding algorithm.

The A* algorithm works much like the Dijkstra and best-first algorithms only it gives values to the nodes in a different way. Each node's value is the sum of the actual cost to that node from the start and the heuristic estimate of the remaining cost from the node to the goal. In this way it combines the tracking of previous length from Dijkstra's algorithm with the heuristic estimate of the remaining path from the bestfirst search. In fact, the Dijkstra search is an A* search, where the heuristic is always 0. This algorithm also makes the most efficient use of the heuristic function, meaning that no other algorithm using the same heuristic will expand fewer nodes and find an optimal path.

A* algorithm uses a starting point and a destination point to produce the desired path, if it exists (Figure 2.10). In figures, the cell marked with "O" is our starting point/node and the cell marked with "X" is the destination. White squares are walkable nodes and the black ones are walls, shelves or any other obstacles.

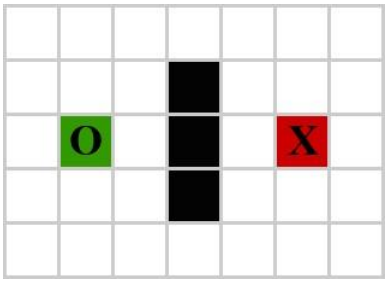


Figure 7: Starting and Destination Points in A* Search Algorithm

A* algorithm starts to execute by looking at the starting node first and then expanding to the surrounding nodes (Figure 2.11).

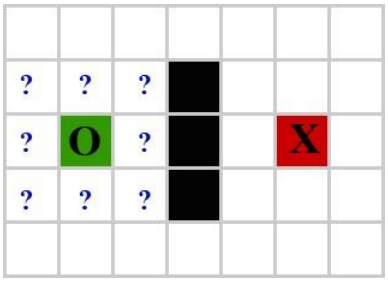


Figure 8: Starting the A* Search Algorithm

This operation continues until the destination node is found. In order to know the nodes, which will be used in search, A* algorithm needs a way to keep track of the nodes. So the nodes to be examined are held in a list, called Open List. At the beginning we place the starting node to the Open List, and after examining all of its surrounding nodes we will move it from the Open List and place in another list called the Closed List (Figure 2.12). Closed List holds the nodes that are visited and there is no need to re-visit its members. When building the Open List, algorithm checks if the node is walkable. If the node is not walkable, it is not added to the Open List.

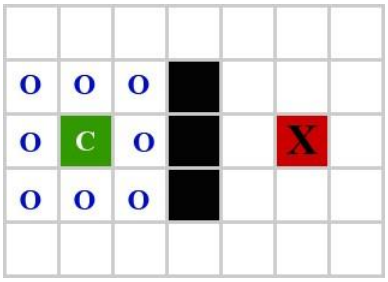


Figure 7: Putting the neighboring cells to the Open List

This process made for one cell is the main iteration through one A* loop; however, we need to track some additional information. We need to know how the nodes are linked together.

Although the Open List maintains a list of adjacent nodes, we need to know how the adjacent nodes link together as well. We can do this by tracking the parent node of each node in the Open List. A node's parent is the single node that the user steps from to get to its current location. On the first iteration through the loop, each node will point to the starting node as its parent (Figure 2.13).

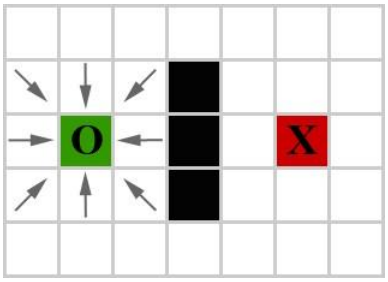


Figure 8: Parent relation of Starting Node

We will use the parent links to trace a path back to the starting node when we reach the destination. At this point we start over process. We now have to choose a new node to check from the Open List. At the first iteration we had only a single node in the Open List. We now have eight nodes in the Open List, and the node which will first be inspected is determined by assigning a score to each node. This score, f(n), is the combination of two scores:

We select the node with the lowest score. We use a Priority Queue to build the Open List, and the nodes that will be added to Open List is sorted by this score. So when we pop an element from this Queue, we always get the node with the lowest score.

There are some well-known heuristics used in scoring.

4.2 Heuristic Functions Used in A* Algorithm

We calculate each node's score by adding the cost of getting there from the starting location to the heuristic value, which is an estimate of the cost of getting from the given node to the final destination. We use this score when determining which tile to check next from the Open List. We will first check the tiles with the lowest cost. In this case, a lower cost will equate to a shorter path.

The g(n) value shown in each open node is the cost of getting there from the starting node. In this case, each value is 1 because each node is just one step from the starting node. The h(n) value is the heuristic. The heuristic is an estimate of the number of steps from the given node to the destination node. We do not take obstacles into consideration when determining the heuristic. We have not examined the nodes between the current node and the destination node, so we do not really know yet if they contain any obstacles. At this point we simply want to determine the cost, assuming that there are no obstacles. The final value is f(n), which is the sum of g(n) and h(n). This is the cost of the node. It represents the known cost of getting there from the starting point and an estimate of the remaining cost to get to the destination.

Heuristics used in A* algorithm have some variations, but for grid based maps there are three heuristic functions that work well.

A. Manhattan Distance:

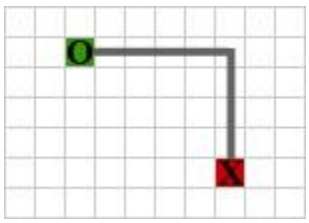


Figure 9: Manhattan distance calculation

The Manhattan heuristic is computed by adding the differences in the x and y components together. The advantage of using this heuristic is that, it is computationally inexpensive, and it can run faster than the others. The major disadvantage of the Manhattan heuristic is that it tends to overestimate the actual minimum cost to the goal (unless 4-adjacency is used) which means that the road being found may not be an optimal solution. If we are not interested in an

optimal solution, but just a good one, then using an overestimating heuristic can speed up the road finding.

B. Diagonal Distance:

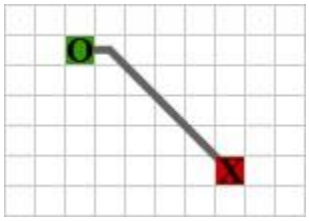


Figure 10: Diagonal distance calculation

Diagonal method balances the h(n) and g(n) in calculation of the total cost, but it is a bit slower than Manhattan method.

C. Euclidean Distance:

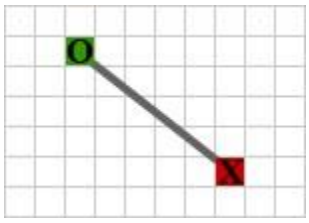


Figure 11: Euclidean distance calculation

The Euclidean heuristic (Figure 2.16) is admissible, but usually underestimates the actual cost by a significant amount. This means that we may visit too many nodes unnecessarily which, as a result, increases the time it takes to find the road. The Euclidean distance heuristic is also computationally more expensive to apply compared to the Manhattan heuristic, as it additionally involves two multiplication operations and calculating the square root. As we search the nodes toward the goal we add them to Open List and transfer to Closed List when

our job with them completed. These operations continue until we reach to the goal. When we reach to the destination, we do a back-track by using the parent links for all of the nodes and construct the path (Figure 2.17).

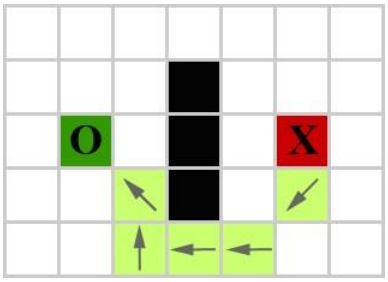


Figure 12: Constructing Final Path in A* Search

4.3 Path Scoring

The key to determining which squares to use when figuring out the path is the following equation:

F = G + H

Where

G = the movement cost to move from the starting point A to a given square on the grid, following the path generated to get there.

H = the estimated movement cost to move from that given square on the grid to the final destination, point B.

As described above, G is the movement cost to move from the starting point to the given square using the path generated to get there. In this example, we will assign a cost of 10 to each horizontal or vertical square moved, and a cost of 14 for a diagonal move. We use these numbers because the actual distance to move diagonally is the square root of 2 (don't be

scared), or roughly 1.414 times the cost of moving horizontally or vertically. We use 10 and 14 for simplicity's sake.

Since we are calculating the G cost along a specific path to a given square, the way to figure out the G cost of that square is to take the G cost of its parent, and then add 10 or 14 depending on whether it is diagonal or orthogonal (non-diagonal) from that parent square. The need for this method will become apparent a little further on in this example, as we get more than one square away from the starting square.

H can be estimated in a variety of ways. The method we use here is called the Manhattan method, where you calculate the total number of squares moved horizontally and vertically to reach the target square from the current square, ignoring diagonal movement, and ignoring any obstacles that may be in the way. We then multiply the total by 10, our cost for moving one square horizontally or vertically. This is (probably) called the Manhattan method because it is like calculating the number of city blocks from one place to another, where you can't cut across the block diagonally.

F is calculated by adding G and H. The results of the first step in our search can be seen in the illustration below. The F, G, and H scores are written in each square. As is indicated in the square to the immediate right of the starting square, F is printed in the top left, G is printed in the bottom left, and H is printed in the bottom right.

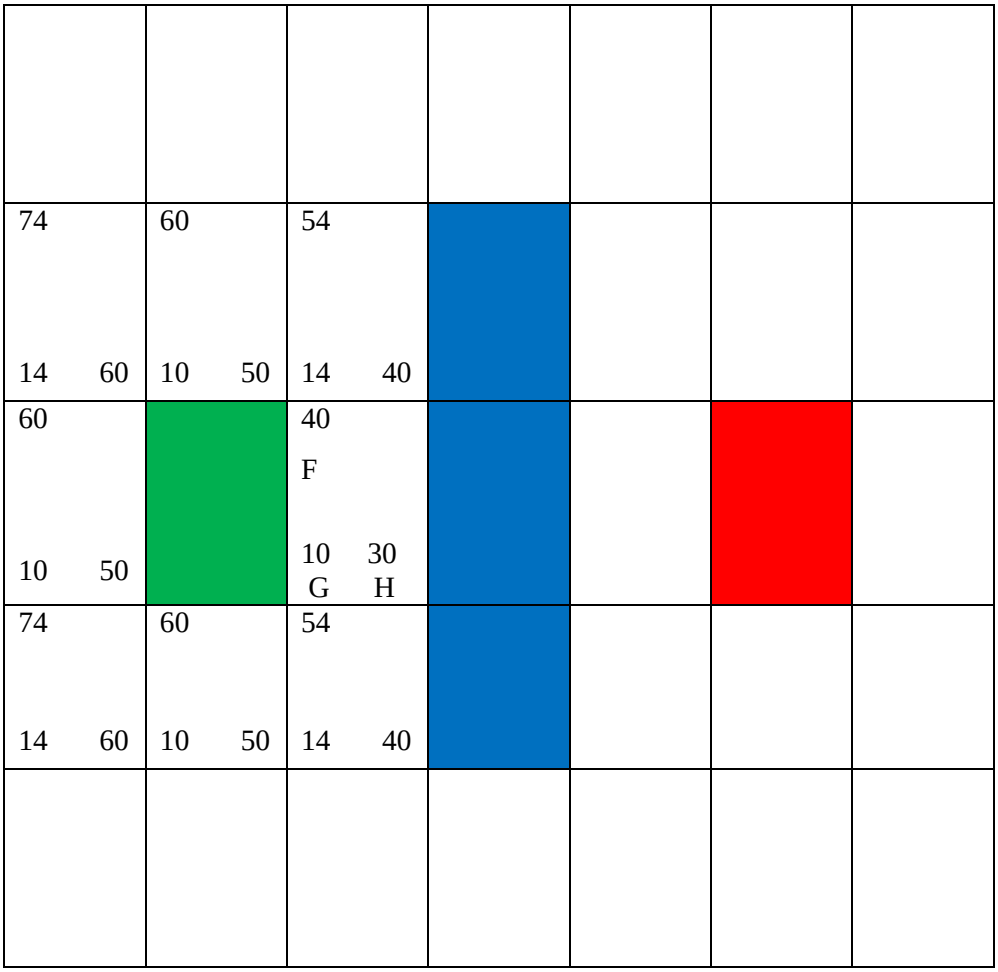


Figure 13: Cost calculation grid

So let's look at some of these squares. In the square with the letters in it, G = 10. This is because it is just one square from the starting square in a horizontal direction. The squares immediately above, below, and to the left of the starting square all have the same G score of 10. The diagonal squares have G scores of 14. The H scores are calculated by estimating the Manhattan distance to the red target square, moving only horizontally and vertically and ignoring the wall that is in the way. Using this method, the square to the immediate right of the start is 3 squares from the red square, for a H score of 30. The square just above this square is 4 squares away (remember, only move horizontally and vertically) for an H score of 40. You can probably see how the H scores are calculated for the other squares. The F score for each square, again, is simply calculated by adding G and H together.

In the next chapter we will discuss the proposed work, algorithm, flowchart and observations & results

5.1 Proposed Work

Step 1: In our work we have simulated a city map in which we have used the start node, the target node and the obstacles in the city road. In our work we can simulate a scenario of n number of nodes.

Step2: The source node includes into each packet a route vector composed of a list of anchors or fixed geographic points, through which packets are passed to the destination

Step3: We have generated obstacles in the city map through which no vehicle can pass and the vehicle has to find an alternate route.

Step 4: We have used A* algorithm to show the simulation and find the minimum possible path between the start node and target node.

5.2 The Algorithm of our work is as follows:

A1 Add the starting node to the open list.

A2. Repeat the following steps:

a. Look for the node which has the lowest f on the open list. Refer to this node as the current node.

b. Switch it to the closed list.

c. For each reachable node from the current node

i) If it is on the closed list, ignore it.

ii) If it isn't on the open list, add it to the open list. Make the current node the parent of this node. Record the f, g, and h value of this node.

iii) If it is on the open list already, check to see if this is a better path. If so, change its parent to the current node, and recalculate the f and g

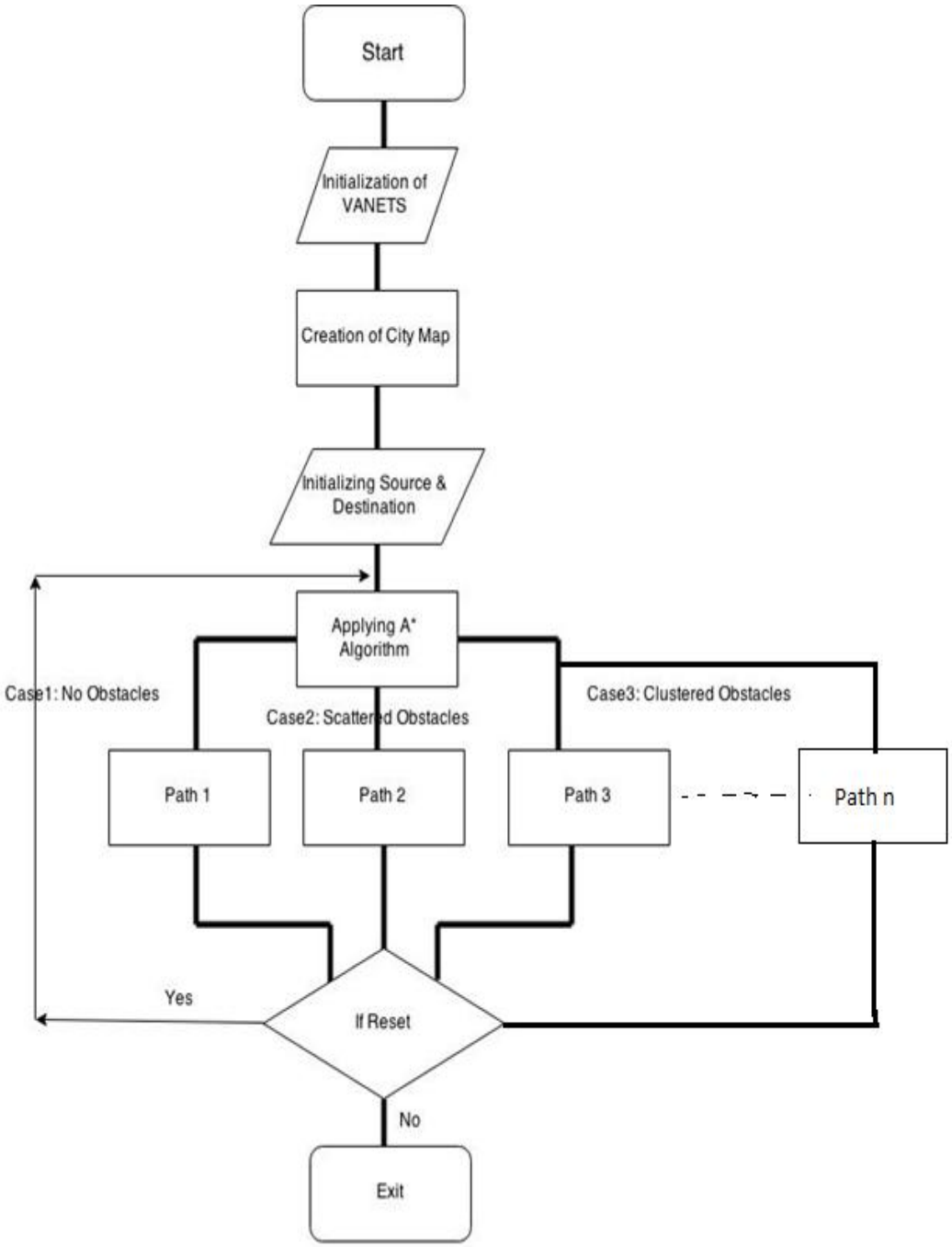
d. Stop when

i) Add the target node to the closed list.

ii) Fail to find the target node, and the open list is empty.

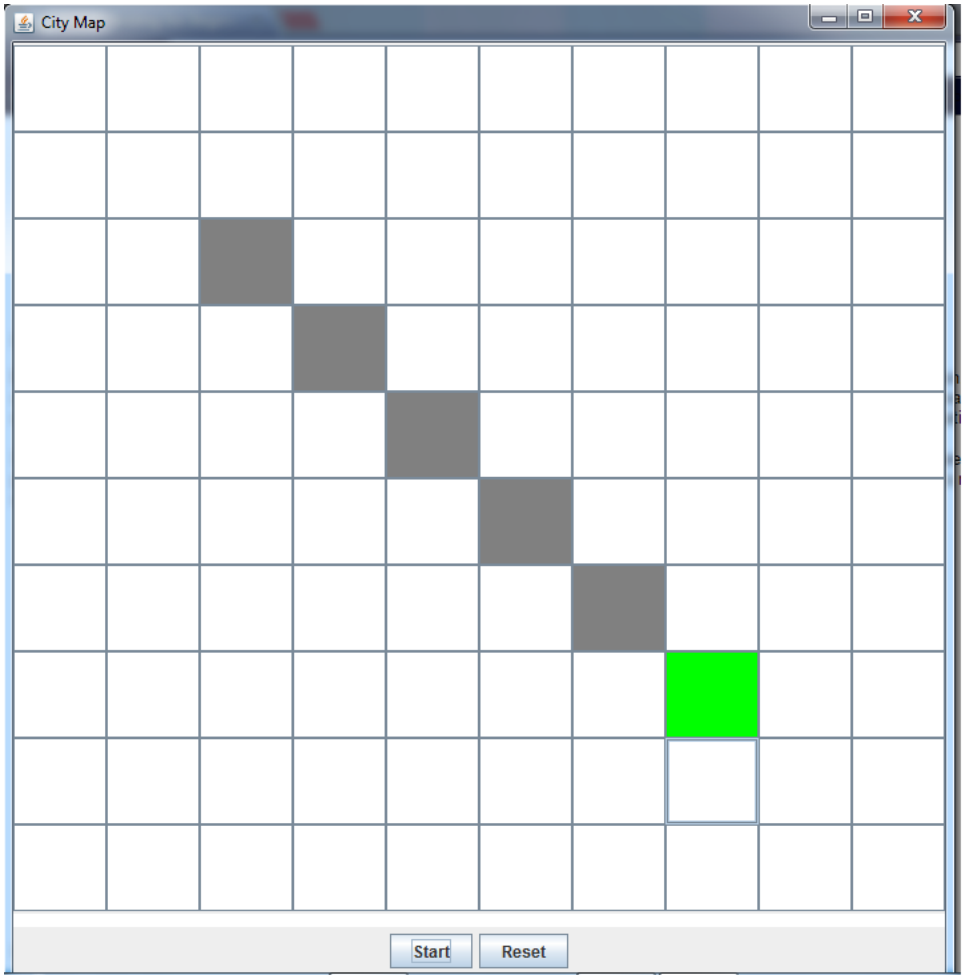
A3 Tracing backwards from the target node to the starting node. This is the path obtained.

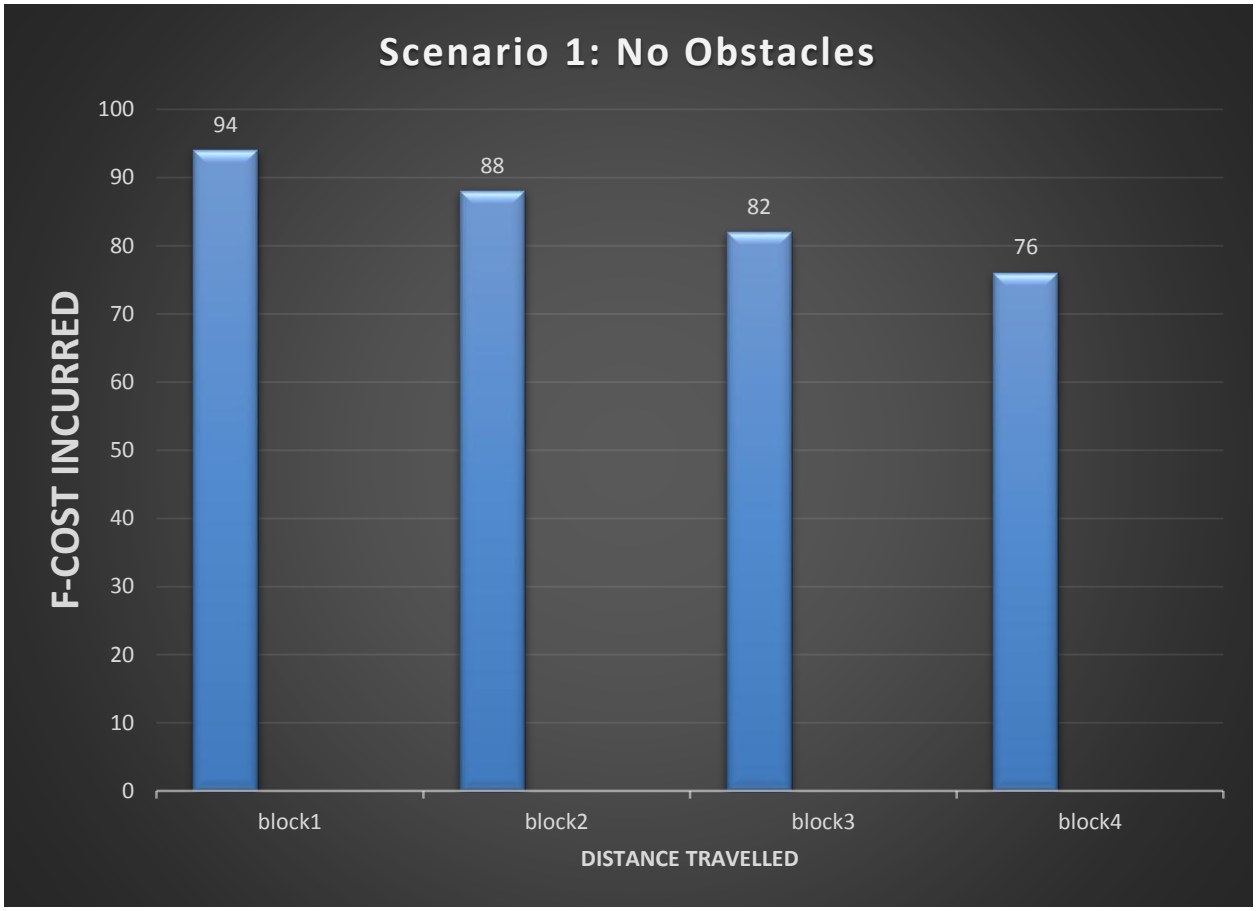
5.3 Flow chart:



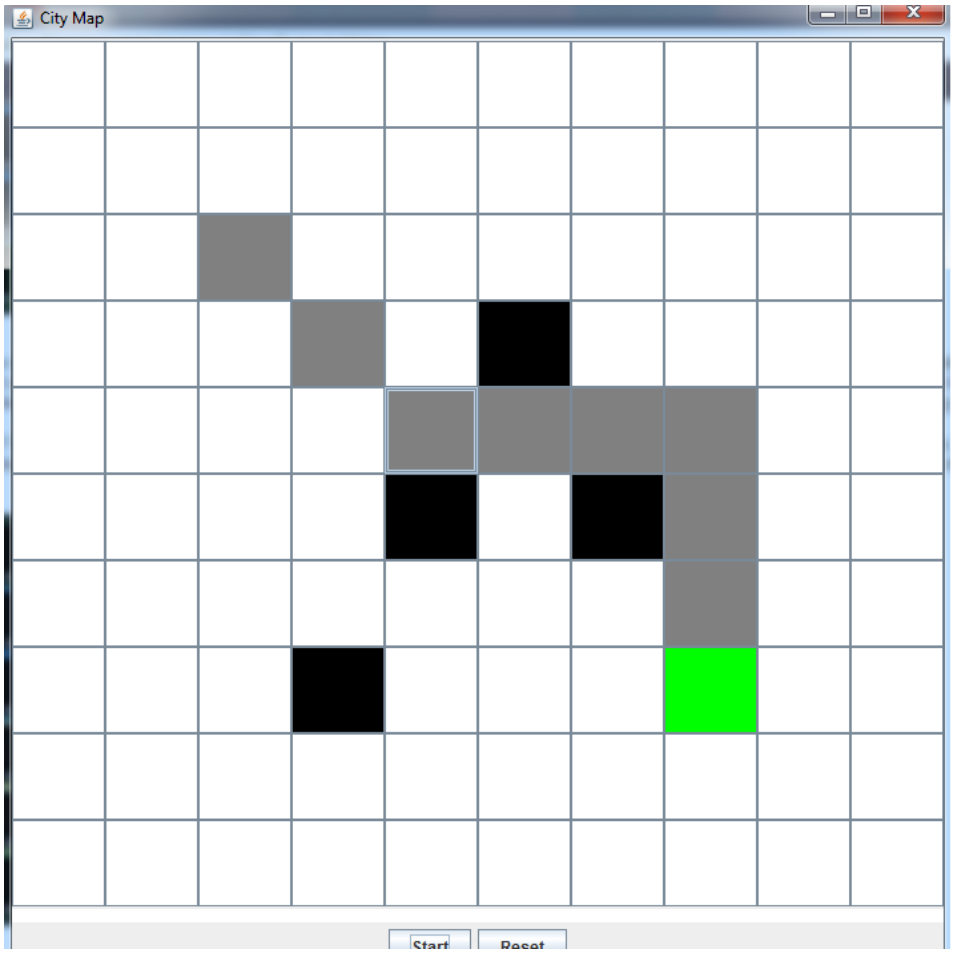
5.4 Observation and Result

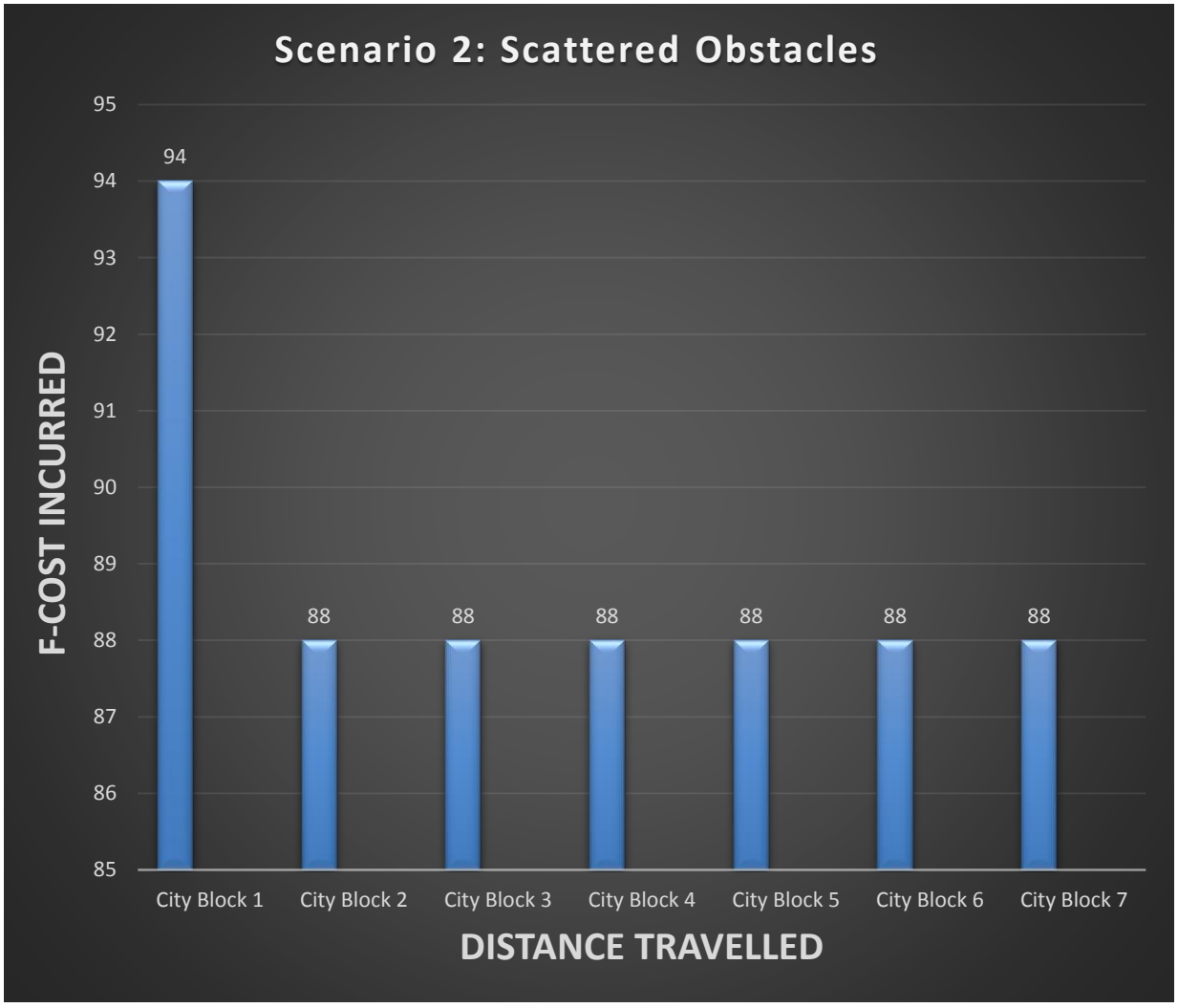
Scenario 1:



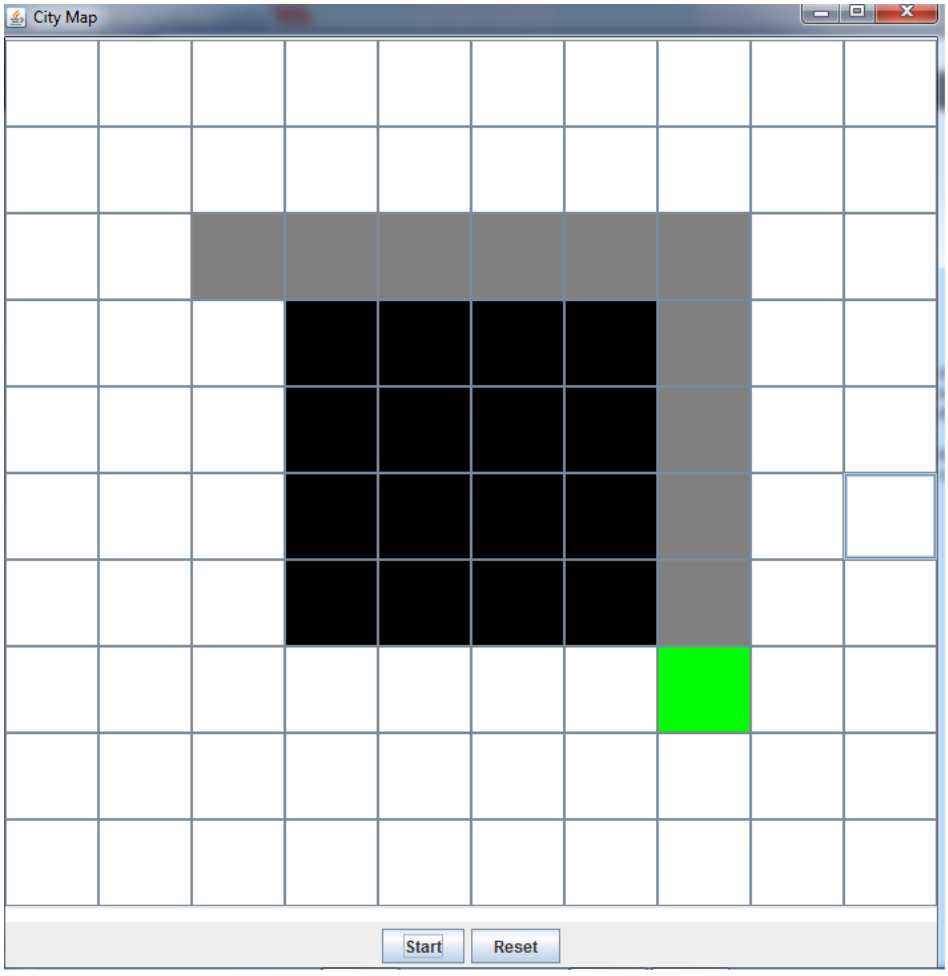

Scenario 1: No Obstacles
F-COST INCURRED
100
90
80
70
60
50
40
30
20
10
0
94
88
82
76
block1
block2
block3
block4
DISTANCE TRAVELLED

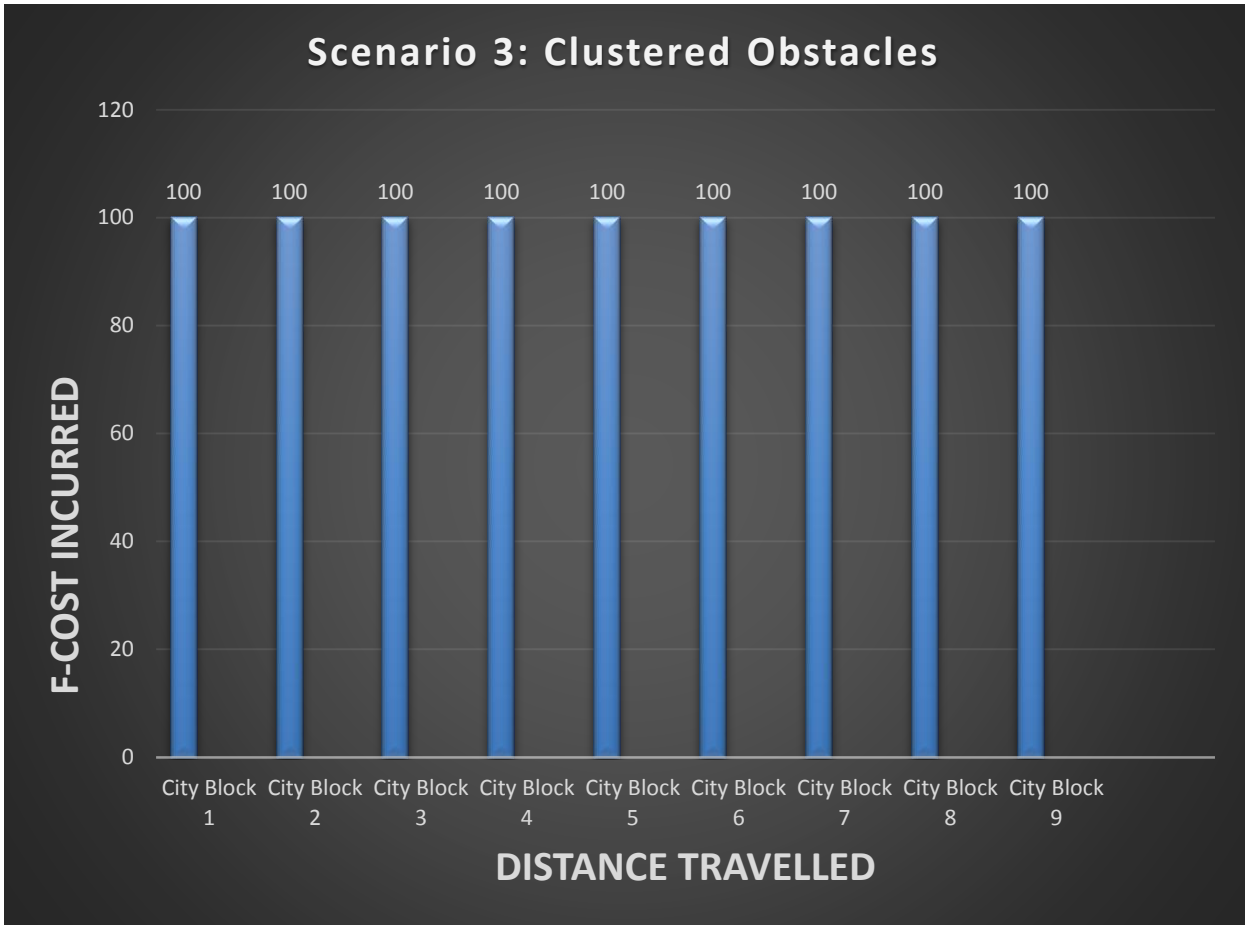
Scenario 2:

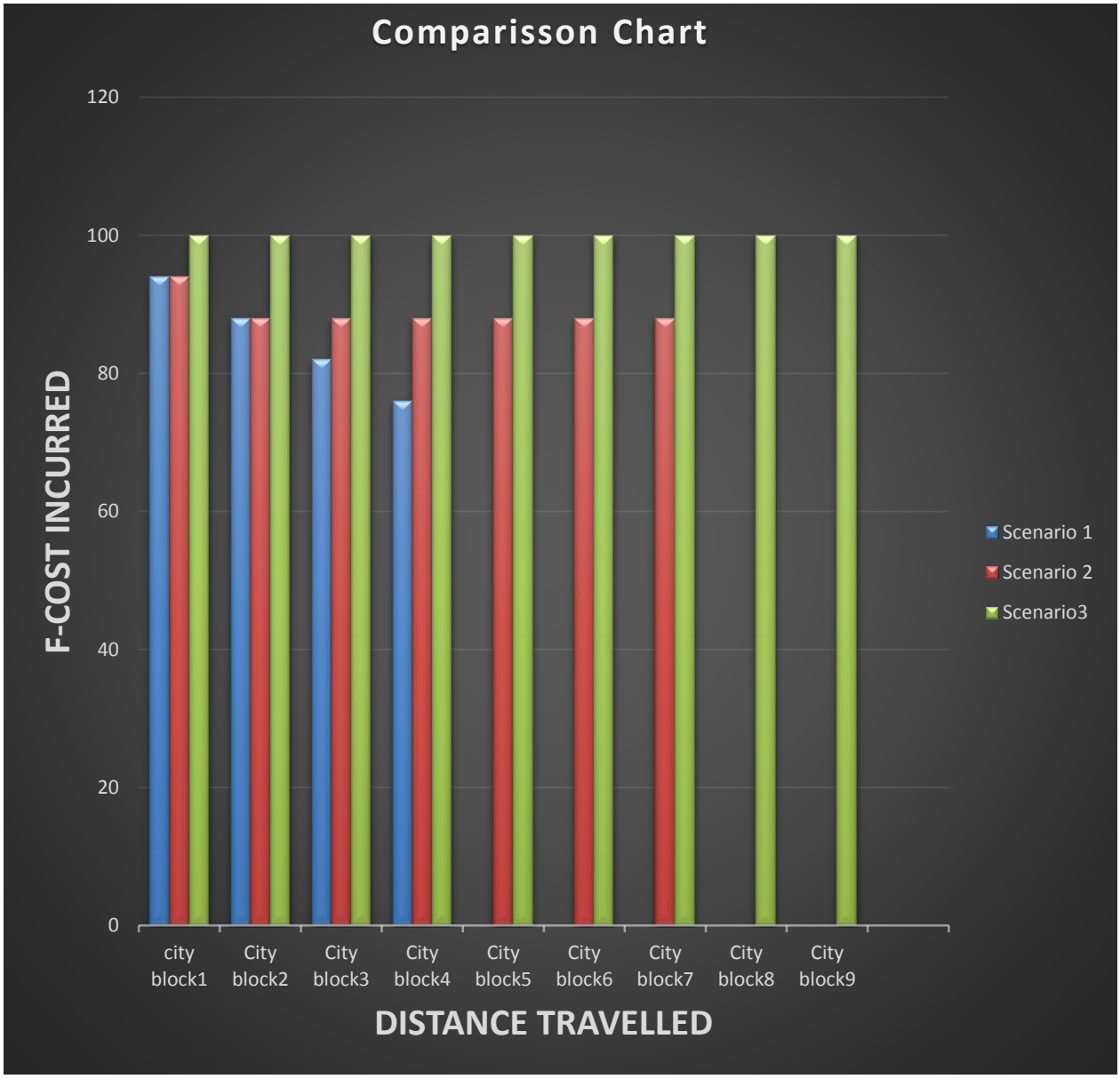



Scenario 2: Scattered Obstacles
F-COST INCURRED
95
94
93
92
91
90
89
88
87
86
85
94
88
88
88
88
88
88
City Block 1
City Block 2
City Block 3
City Block 4
City Block 5
City Block 6
City Block 7
DISTANCE TRAVELLED

Scenario 3:




Scenario 3: Clustered Obstacles
120
100
80
60
40
20
0
F-COST INCURRED
100
100
100
100
100
100
100
100
100
City Block 1
City Block 2
City Block 3
City Block 4
City Block 5
City Block 6
City Block 7
City Block 8
City Block 9
DISTANCE TRAVELLED


Comparisson Chart
120
100
80
60
40
20
0
F-COST INCURRED
city block1
City block2
City block3
City block4
City block5
City block6
City block7
City block8
City block9
DISTANCE TRAVELLED
Scenario 1
Scenario 2
Scenario3

Chapter 6

Conclusion

With the introduction of intelligent vehicles, we argue that road transportation and traffic management are also demanding for intelligent traffic signs. Based on the emergent vehicular communication technologies, we have described a system that enables to find the shortest path in a city map.

A* path finding algorithm is one of the best path finding algorithm in Vehicular Ad-Hoc Networks (VANETs) for city environment. The performance of A* path finding algorithm is better than other algorithms like, Dijkstra's algorithm as the search area is reduced in A* algorithm due to the cost function. In literally, A* is the one of the important path finding algorithm used in city environment.

The evaluation of path planning on the above city layout provided an in-depth analysis of transport operation that proved to be one of the major operations that can greatly influence the overall travelling cost in terms of safety management, time, productivity, and travelling distance.

BIBLIOGRAPHY

.

[1] A. K. Saha, D. Johnson," Modeling Mobility for Vehicular Ad Hoc Networks", Poster Session, 1st ACM Workshop on Vehicular Ad Hoc Networks (VANET2004), October 2004.

[2] Hall, Matt. "Pathfinding in an Entity Cluttered 3D Virtual Environment." PhD diss., University of Leeds, School of Computer Studies, 2005.

[3] Karp Brad and Hsiang-Tsung Kung. "GPSR: Greedy perimeter stateless routing for wireless networks." In Proceedings of the 6th annual international conference on Mobile computing and networking, pp. 243-254. ACM, 2000.

[4] Seet, Boon-Chong, et al. "A-STAR: A mobile ad hoc routing strategy for metropolis vehicular communications." NETWORKING 2004. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications. Springer Berlin Heidelberg, 2004. 989-999.

[5] Carlos De Morais Cordeio, Dharma Prakash Agarwal, "Ad Hoc & Sensor Network", Published by World Scientific Publication, 2006.

[6] Z. Hass, "A New Routing Protocol for the Reconfigurable Wireless Networks (VANETs)", IEEE 6th International Conference on Universal Personal Communications (ICUPC '97), pages 562-566, 1997.

[7] Davis, V.: Evaluating Mobility Models Within an Ad Hoc Network. Master's Thesis, Colorado School of Mines(2000).