Target Detection and Interception Using Modified POMDP Based Motion Planning of Mobile Robots in a Known Environment

Author

EHSAN ELAHI BASHIR

NUST201261232MCEME35512F

Supervisor

Col. Dr. KUNWAR FARAZ AHMED KHAN

DEPARTMENT OF MECHTRONICS ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

JULY, 2015

Target Detection and Interception Using Modified POMDP Based Motion Planning of Mobile Robots in a Known Environment

Author

EHSAN ELAHI BASHIR

NUST201261232MCEME35512F

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Mechatronics Engineering

Thesis Supervisor:

Col. Dr. KUNWAR FARAZ AHMED KHAN

Thesis Supervisor's Signature:____

DEPARTMENT OF MECHATRONICS ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

JULY, 2015

Declaration

I certify that this research work titled "*Target Detection and Interception Using Modified POMDP Based Motion Planning of Mobile Robots in a Known Environment*" is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Ehsan Elahi Bashir

NUST201261232MCEME35512F

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student

Ehsan Elahi Bashir

NUST201261232MCEME35512F

Signature of Supervisor

Acknowledgements

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which You setup in my mind to improve it. Indeed I could have done nothing without Your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life.

I would also like to express special thanks to my supervisor Dr. Kunwar Faraz Ahmed Khan for his help throughout my thesis and also for AMDT and Motion Planning for Mobile Robots courses which he has taught me. I can safely say that I haven't learned any other engineering subject in such depth than the ones which he has taught.

I would also like to thank Brig. Dr. Javaid Iqbal, Dr. Umar Shahbaz and Dr. Khurram Kamal for being on my thesis guidance and evaluation committee and express my special Thanks to Dr. Adnan Masood for his help. I am also thankful to Ahsan Gul, Saad Farooq, Mansoor Ghazi, Kamran Nazir, Sir Sami-ud-Din, Adnan and Sir Rashid for their support and cooperation.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Dedicated to my exceptional parents and adored siblings whose tremendous support and cooperation led me to this wonderful accomplishment

Abstract

This thesis presents an approach to locate an adversarial, mobile evader in an indoor environment using motion planning of mobile pursuers. The approach presented in this thesis uses motion planning of mobile robots to search a target in a graph and clear the workspace. The algorithm used is Modified Partially Observable Markov Decision Process (POMDP), a probabilistic search method to clear the indoor workspace in a pursuit evasion domain. In this thesis, the indoor environment is assumed to be known beforehand and the mobile evader to be adversarial with no motion model given. The workspace is first discretized and then converted to a graph, whose nodes represent the rooms and corridors and edges represent connection between them. The task of pursuer is to clear the whole graph with no contaminated node left in minimum possible steps. Such path planning problems are NP-hard and the problem scales exponentially with increased number of pursuers and complex graph.

Key Words: *Motion Planning, Mobile Robots, Reinforcement learning, POMDP, Search, Pursuit Evasion*

Table of Contents

Table of Contents

List of Figures

CHAPTER 1: INTRODUCTION

This chapter will briefly give an overview about the research work motivation, problem statement and thesis outline.

1.1 Motivation

Many times in our everyday life we have to search for a person in a building, while exploring an already explored location multiple times, lastly expecting that the individual may have moved to an effectively investigated area and re-defiled that area. Consider a scenario where a hostile person or object enters a building and is meant to cause some sort of harm, or a person who has been separated from his team. Consider a person trapped inside a building that is on fire or is collapsed, or an assistant robot who has to find the old man in the house to take care of him. A rescuer who has to rescue a friendly team member from a given environment without the knowledge of friend's location. In all these scenarios, there is one thing in common and that is "search". Search in pursuit-evasion plays the most important role, when one does not know the exact location of the evader. Pursuit Evasion games have played an important role in robotics, surveillance, search and rescue, network routing, network security and modeling animal behavior etc.

Search (otherwise called "one-sided search") and pursuit evasion (otherwise called "adversarial search") issues have been generally tended to by two distinctive methodologies. One point of view has been to plan methodologies that expand execution of searcher against an antagonistic,, worst-case target [1]. In such settings, the evader is viewed as having limitless speed, complete information of pursuit environment, and familiarity with follower position. This type of search problem is called adversarial search and is considered to be NP-Hard. A number of iterative and greedy search algorithms are proposed through the course of history to find such an evader. Such methods offers guarantees on the accomplishment of the search, i.e. by capture of the evader in limited time [1]. There are certain search methods that assume the evader and pursuers to be aware of each other's location. In these type of search algorithms the pursuers and evader have finite equal or different speed. The pursuers try to move towards the evader to capture it. On the other hand, there are probabilistic search perspectives where the motion model of evader is known to the pursuer and the purser has to find the probability of presence of evader at some place according to the evader's motion model. This type of search problem is easy to solve as compared to the adversarial search problem. In adversarial search problem, it takes a lot of resources, time and effort to find a solution. But, the solution is always complete and it finds an evader in an environment, if one exists. On the other hand, there are some search problems where cost is the main factor of search, and the pursuer has to minimize the overall cost. The cost can be number of steps taken, time spent to search the environment or number of pursuers used to find the solution. Such solutions sacrifice completeness over cost minimization. They may or may not find an evader, even if there exists one.

Another, one of the most important aspects of the pursuit-evasion or search problem is to find out the Graph search number or search number of Polygon. The search number is to find the minimum no of pursuers needed to find an evader in a graph or in a polygon. A lot of work has been done on finding search number for a graph or polygon, but there is no 100% correct, hard and fast method to find the upper and lower bound of search number *s(G)* other than hit and trial.

The above discussion shows that the available pursuit-evasion algorithms or search algorithms try to find an adversarial or non-adversarial evader either with guarantee or in minimum possible steps. The available methods either minimize cost or guarantee the capture, but not the both at same time.

1.2 Problem Statement

In search and pursuit-evasion problem, there are a number of parameters that require to be kept in mind while deriving an algorithm.

Some of these parameters are given in Figure.1.2

The problem in this thesis is to find and adversarial evader in a graph environment, given that the evader is:

1. Adversarial
2. Has an infinite speed

3. Knows the environment
4. Knows the pursuer’s position all the time
5. Moves along edges of the graph
6. Cannot teleport
7. Tries to avoid capture
8. Is mobile (moving evader)
9. Has an unbounded turning angle

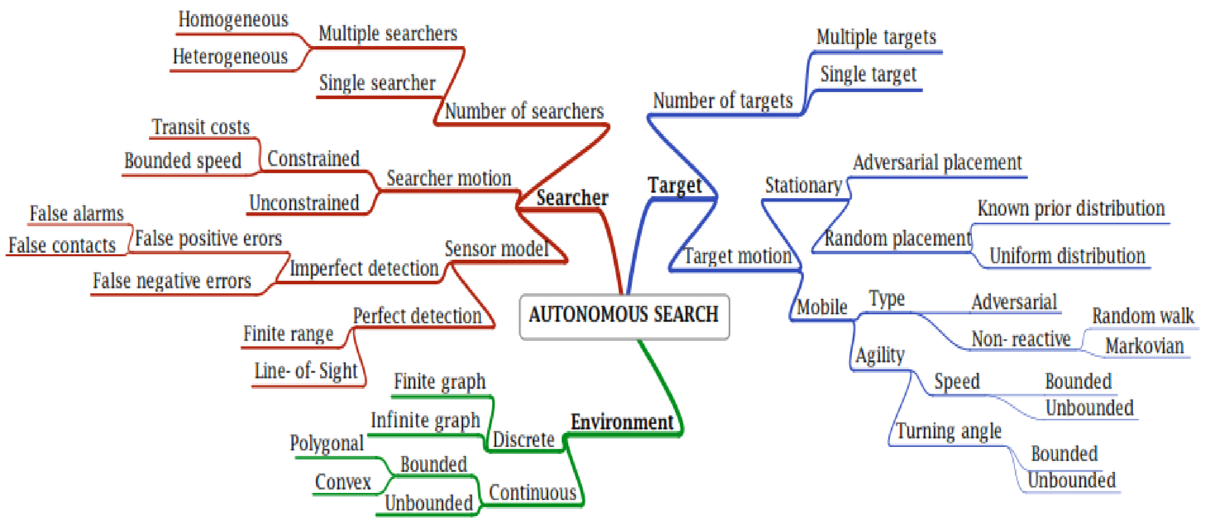


Figure 1.2 Different parameters of Search and Pursuit-Evasion [1]

The Environment is:

1. Discrete
2. Finite graph

The pursuer is:

1. Single searcher
2. Has a finite speed
3. Has a constrained motion (four point connectivity)

4. Has a perfect detection (line of sight)
5. Does not know the evader location
6. Knows the environment (map is given)
7. Has a 360 degree field of view.

The important tasks to be completed in this thesis are:

1) To convert the given environment into a graph.
2) To find the search number of graph *s(G).*
3) To find a balance between cost minimization and guaranteed capture.

The first task is to convert the given polygonal environment in a graph. This is done by discretizing the environment according to the critical visibility changes.

The second assignment is to discover the search number s(G) of converted graph, that is to locate the least number of searchers expected to clear the graph.

The third task is to derive an algorithm that uses positive properties of both adversarial search methods and probabilistic search methods to find a balance between them. This balance ensures the guaranteed capture while minimizing the cost at the same time.

This thesis presents a method to find an adversarial target in an indoor environment using probabilistic search techniques on a graph. The environment is first discretized and then converted into a graph. The nodes in graph correspond to the rooms and corridors, and edges correspond to the pathways between rooms and corridors. Since it is an adversarial search problem, so it is assumed that the evader has an infinite speed, it can move from one node of graph to another node instantaneously, but it cannot teleport. It has to follow the edges to move from one node to another. The evader has complete knowledge of search environment including awareness of pursuer position and intention. The goal of the evader is to hide from the pursuer and to avoid capture. The pursuer on the other hand has a finite speed, it can move from one node to only its adjacent nodes in a step. It has a four point connectivity movement. The pursuer has a prior knowledge of environment, which includes the positions of obstacles, the nodes adjacency, node visibility and directions of nodes with respect to the other nodes. The pursuer however does not know the evader position at any time instant. The pursuer is equipped with a 360 degree field of view camera and can see a target in straight line. The capture happens when an evader lies in the same node as pursuer, or when an evader comes in straight line of sight of pursuer.

1.3 Classification of Thesis

Chapter 2 provides information about research work previously implemented on search and pursuit-evasion.

Chapter 3 illustrates the detailed problem formulation.

Chapter 4 illustrates the detailed proposed algorithm.

In chapter 5 the results of the proposed algorithm on different environments are elaborated.

Chapter 6 concludes the thesis and proposes future work regarding search and pursuit-evasion for different scenarios.

CHAPTER 2: LITERATURE REVIEW

Chapter 2 provides a deep insight of research done by different researcher in the field of search and pursuit-evasion. The chapter provides information about different types of environments, search algorithms and techniques of pursuit-evasion.

2.1 Search Algorithms

This section provides information about different search algorithms. Some of these include, greedy search algorithms, Dijkstra's Algorithm, graph traversal and Kruskal's algorithm.

2.1.1 Dijkstra's Algorithm

The acclaimed Dijkstra's search algorithm is an algorithm that has been utilized as a part of search and pursuit evasion. Dijkstra's algorithm is an algorithm for discovering the briefest paths between vertices in a graph. It is a greedy algorithm. For a given starting node in the graph, the algorithm finds the briefest path between that node and all other neighboring nodes. The algorithm is generally utilized as a part of system network routing protocols and briefest path search issues. It is the fastest known shortest path algorithm for directed graphs. The A* algorithm is a hypothesis of Dijkstra's algorithm that wipes out the span of the subgraph that must be explored, if additional information is available that gives a lower bound on the "distance" to the target. The Dijkstra's algorithm has been used in [2] for finding zero-sum dynamic games with deterministic transitions and alternating moves of the players. The problem with Dijkstra's algorithm is that it is greedy. It only tries to minimize immediate cost but does not consider long term cost. Another problem with this algorithm is that it is a directed graph, and does not consider re-contamination. That makes this algorithm unsuitable for guaranteed search of evader. Dijkstra's algorithm lives up to expectations by going by vertices in the graph beginning with the object's starting point. It then more than once analyzes the nearest not-yet-inspected vertex, adding its vertices to the arrangement of vertices to be analyzed. It extends outwards from the starting point until it achieves the objective. Dijkstra's algorithm is ensured to locate a most brief way from the starting point to the objective, the length of none of the edges have a negative cost.

The point by point steps utilized as a part of Dijkstra’s algorithm to locate the shortest path from a solitary source vertex to all different vertices in the given graph.

(1) Create a set *sptSet* (most brief way tree set) that stays informed concerning vertices included in briefest way tree, i.e., whose base separation from source is ascertained and finished. At first, this set is vacant.

(2) Assign a separation quality to all vertices in the info diagram. Instate all separation values as INFINITE. Relegate separation esteem as 0 for the source vertex so it is picked first.

(3) While *sptSet* does exclude all vertices

(a) Pick a vertex *u* which is not there in *sptSetand* has least separation esteem.

(b) Include *u* to *sptSet.*

© Update separation estimation of all contiguous vertices of *u*. To upgrade the separation qualities, emphasize through all neighboring vertices. For each contiguous vertex *v*, if whole of separation estimation of *u* (from source) and weight of edge *u*-*v*, is not exactly the separation estimation of *v*, then repeat the separation estimation.

2.1.2 Graph Traversal

Graph traversal is the type of search algorithm that is used to visit all the nodes in a graph, in a specific manner. In graph traversal some nodes are needed to be visited multiple times, as it is unknown before transitioning to a node that the node has already been visited. As graph becomes denser and complex, the computation times increases as the redundancy becomes more prevalent. So the algorithm needs to know the visited nodes, to avoid visiting them again and again. The problem with graph traversal algorithm is the lack of predefined root node. The famous Breadth-First and Depth-First search algorithms are types of graph traversal algorithm.

2.1.3 Kruskal’s Algorithm

Kruskal’s algorithm is an algorithm that is a minimum spanning tree algorithm that finds a node in the graph with having a minimum cost that connects to the current node. It is a greedy algorithm that finds a initial traversing tree in a coordinated graph.

2.1.4 Greedy Best First Search

The Greedy Best First Search algorithm works in a comparative path as Dijkstra's Algorithm, with the exception of that it has some estimate (called a heuristic) of how a long way from the objective any node is. As opposed to selecting the vertex nearest to the starting node, it chooses the vertex nearest to the objective. Greedy Best First Search is not ensured to locate a most limited way. On the other hand, it runs much faster than Dijkstra's algorithm in light of the fact that it utilizes the heuristic function to guide its direction towards the objective rapidly. Case in point, if the objective is toward the south of the starting node, Greedy Best First Search will have a tendency to concentrate on ways that lead southwards. It demonstrates that Greedy Best First Search can discover path immediately contrasted with Dijkstra's algorithm. The inconvenience is that Greedy Best First Search is "greedy" and tries to move towards the objective regardless of the possibility that its not the right path. Since it just considers the cost to get to the objective and disregards the cost of the path as such, it continues going regardless of the possibility that the path it’s on has turn out to be truly long.

2.1.5 A* Algorithm

A* is the most generally utilized algorithm as a part of recreations. It is one of a group of graph search algorithms that take after the same structure. These algorithms represent the map as a graph and after that discover a path in that graph. Dijkstra's Algorithm functions admirably to locate the briefest path, yet it squanders time searching in directions that aren't promising. Greedy Best First Search investigates in encouraging directions yet it may not locate the most limited path. The A* algorithm utilizes both the actual distance from the start and the assessed distance to the goal. A* is similar to Dijkstra's algorithm in that it can be utilized to locate a most limited path. A* is similar to Greedy Best First Search in that it can utilize a heuristic to direct itself. In the straightforward case, it is as quick as Greedy Best First Search. The key to its prosperity is that it joins the bits of data that Dijkstra's algorithm uses (favoring vertices that are near to the beginning stage) and data that Greedy Best-First-Search uses (favoring vertices that are near to the objective). In the standard phrasing utilized when discussing A*, g(n) speaks to the precise cost of the path from the beginning stage to any vertex n, and h(n) speaks to the heuristic evaluated cost from vertex n to the objective. A* equalizations the two as it moves from the beginning stage to the objective. Every time through the principle circle, it looks at the vertex n that has the most reduced $f(n) = g(n) + h(n)$.

Admissibility. If there is any solution then the methodologies which ensure ideal solution, are called admissible. There are couple of things which should have been fulfilled for A* to be admissible. In the event that a tree-search is being used, the optimality of A* is clear to be broke down. For this situation, A* is ideal, if h(n) is an acceptable heuristic.

h*(n) = the true minimum cost to objective from n. A heuristic h is permissible if h(n) <= h*(n) for all states n.

2.2 Search Number of Graph or Polygon

A lot of work has been done on finding the search number of a graph or a polygon. The minimum number of pursuers s(G) that are required to clear a polygon (free space) relies on upon the geometry and topology of the free space [3]. $H(F) \geq 2$ when free space F is multiply connected (the evader can hole up behind a gap to abstain from being caught by the evader), where H(F) is the search number of polygonal free space [3]. It is shown in [3] and [4] for a simply connected free space F with n edges, even from a pessimistic standpoint H(F) = O(lg n), and with the expectation of complimentary spaces with h holes, best case scenario H(F) = O(h + lg n). From [3] and it can be demonstrated that for exploring a tree of passages as indicated in Figure 2.2 obliges k + 1 followers, where k is the stature of the tree. Thus the search number in this type of free space is $H(F) = \Omega(lg\ n)$, and $H(F) = \Omega(\sqrt{h} + lg\ n)$.

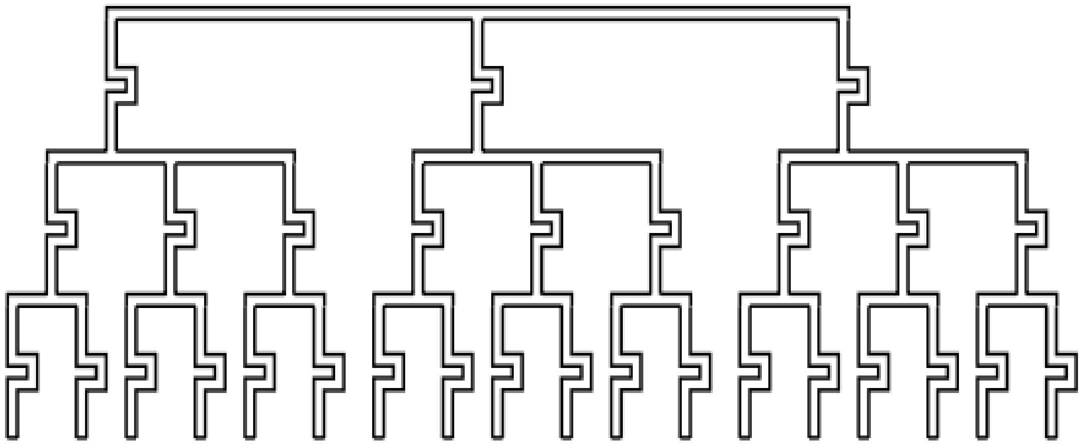


Figure 2.2 A tree of corridors

In [5] a special case of mobile searchers is considered, where pursuer has a torch and can see only the area of free space that is lit by the torch in the straight line. Some upper and lower bound on search number of polygonal free space are discussed for the proposed approach. If *ps(P)* is the minimum number of pursuers needed for a simple polygon *P, n, r, b and g* be the quantity of edges, the quantity of reflex vertices, the bushiness and the extent of least guard set for polygon P individually, then it is demonstrated in [5] that the the upper and (thinking pessimistically) lower limits of 1 + [log3 (2b + 1)] on ps(P). The upper limits on ps(P) as far as n, r and g are displayed as; ps(P) ≤ 1 + [log3 (n – 3)], ps(P) ≤ 1 + [log3 r] and ps(P) ≤ 2 + [log2 g]. For any common number ≥ 2, there exists a polygon P such that ps(P) = log3 (n + 1) = log3 (2r + 3) = 1 + log3 (2g–1) = s [5]. The probabilistic search methods on graphs are generally used with only one pursuer, as it becomes very complex and resource intensive for such algorithms to find an evader as the graph becomes denser or the nodes size increases.

Determining the minimum number of searchers (search number) of a graph is NP-hard [6], and NP-complete due to monotonicity of optimal edge search schedule [7][8].

2.3 Coverage Based Algorithms

Target tracking is also done using coverage based wireless sensor networks. Different approaches have been proposed to find the minimum number of wireless sensors to cover a whole environment for tracking a target. In [9] a grid based approach is proposed that provides the maximum coverage for a mobile target in a hybrid sensor network. The maximum coverage in [9] is attained by moving mobile sensor nodes in the network based on a distributed technique. The main problem with coverage based systems is the cost of sensors. It is also hard to determine the minimum number of sensors needed to cover a whole environment. The key issues in a cluster based wireless sensor networks are tracking accuracy and energy efficiency. An inner cluster optimization algorithm for tracking a mobile target is proposed in [10]. In [11] a Coverage-Hole Trap Model based on a system that contains one mobile pursuer, one mobile evader and a distributed relay robot network with sensing coverage holes is proposed. The coverage holes are destructive for target following in remote portable robot system. The complexity of a coverage based wireless sensor network increases with increased number of sensors. Most study in coverage based remote sensor systems concentrate on the scope of the entire detecting field. The famous art gallery problem [12] is a type of coverage based target tracking using static guards.

2.4 Graph Search and Pursuit Evasion

One of the essential amusements that happen on graph is the cops and robbers game. In this game cops (followers) attempt to catch a robber (evader) along the vertices of a graph [1][13][14].

The question that arises in this type of problem is:

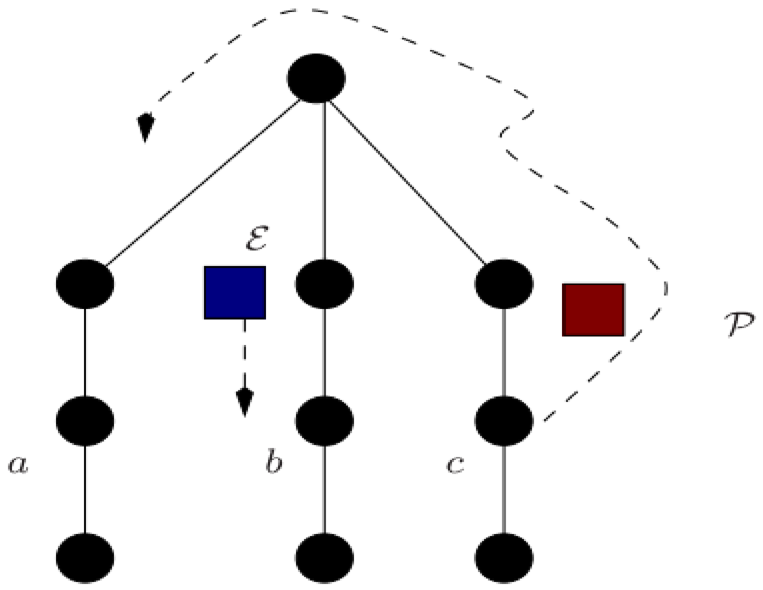
1) What is the search number of graph? Irrespective of the pursuer's initial position and

2) What is the class of graph?

Another problem with the above given approaches is that, during the play, both pursuer and evader know each other's position all the time. An approach with local visibility of evader was proposed in [15]. In this approach both the players (i.e. pursuer and evader) move simultaneously, both evader and pursuer are considered to be having a local visibility or *i*-visibility of the environment. They cannot see each other unless there is a distance '*i*' between them. If the evader has 1-visibility, two pursuers with 1-visibility can capture an evader in any graph with a higher probability. The expected time of capture with two pursuers is polynomial in the vertices of graph. It was also shown in [8] that when the evader has 2-visibility, the pursuers number becomes unbounded [1].

Pursuit-evasion games, presented in this section, seek to amplify the most pessimistic scenario execution on search or capture. Conversely, probabilistic pursuit strategies consider advancement of the normal estimation of a search target, for example, maximal likelihood of identification or insignificant time to location.

In [13] Nowakowski and Winkler proposed a cops and robbers game in graph where a Graph G is given and player I, from this time forward known as the cop, picks a vertex the "station" on which to begin. Player II, the criminal, then picks his starting vertex, and the players proceed onward the other hand starting there beginning with the cop. A move includes staying in one's spot or moving along an edge of G to a connecting vertex, and the cop wins if he "gets" the robber after a set number of moves. Since there is done information in this redirection, either the cop or the robber must have a triumphant procedure; in the past case G will be known as a cop-



win

Figure 2.4 Both the players have 1-visibility. On this graph, the follower P can't catch the evader E utilizing a deterministic procedure. On the other hand, a randomized capture method exists.[15]

graph, generally G is burglar win. The objective of proposed methodology in [13] was to characterize cop-win graphs and to connect them with the structure theory of graphs.

In [14] Aigner and Fromme studied the cops and robbers game in [13] and studied special cases of the game, where more than one pursuers or cops were needed to find a cop-win graph. They then proposed their method to find such cop-win graphs.

In [14] it was exhibited that the cop number of planar graphs is at for the most part three. Another methodology is to surmise the minimum number of cops. This charming piece of the issue has not got basic thought which emerges the question that:

Are there powerful algorithms for approximating the cop-number of a given graph? The issue with above methodologies is that they don't describe the part of the order of play in the cops and robbers game. Another question is that is it genuine that a framework for the turn-based model can be changed into a capable randomized system under the simultaneous move model? In the above cleared up essential cops and robbers philosophies it is expected that the players know each other's positions at all times.

In [16] a hunter and rabbit game was introduced, in this game it was assumed that both the players cannot observer each other unless they are present in the same vertex.

In [17] an offline, greedy and iterative algorithm is proposed for indoor pursuit-evasion that searches a graph for an adversarial evader, who is actively trying to avoid capture. The algorithm guarantees the capture of even an adversarial evader. The problem with this greedy and iterative algorithm is that it sometimes need more pursuers to search the graph for a guaranteed capture than other already present algorithms. The evader was assumed to be adversarial with no teleporting ability and an infinite speed.

2.5 Search and Pursuit-Evasion in Polygonal Free Space

Sometimes the temporal aspects of the game is lost, when the game is taken place on a graph that is an abstraction of a geometric environment. It was showed in [18] that a pursuer can catch a non-deterministic evader in any simply-connected polygonal environment using a randomized strategy.

A pursuer can even capture a mobile evader even if the pursuer is stationary. There exists a random time when an evader may lie in the same free space where pursuer is present and in return get caught.

In [19] it was shown that three pursuers can capture an evader in a polygon (even with holes and obstacles). In this problem, the holes were considered to be finite, the problem was not adversarial as both pursuers and evader can see each other all the time. It was demonstrated that a pursuit-evasion game in which one or more cops attempt to catch a robber by moving onto the criminal's present area. All players should be having equivalent most extreme speeds. They can watch one another at all times. It was demonstrated that three cops can catch the robber in any polygonal environment (which can contain any limited number of holes).

A visibility based pursuit evasion was introduced in [20], in which one or more searchers must travel through a given domain in order to ensure location of all evaders, which can move self-assertively quick. The objective was to create methods for arranging groups of robots to execute this undertaking in application areas, for example, clearing a building, for reasons of security or wellbeing.

Another visibility based pursuit evasion problem was discussed in [21], the issue of arranging the movement of one or more pursuer in a polygonal domain to in the long run "see" an evader that is capricious, having obscure beginning position, and fit for moving self-assertively quick was mulled over. A visibility locale was connected with every pursuer and the target was to ensure that the evader will at last lie in no less than one visibility region. The study of this issue was persuaded to some degree by mechanical autonomy applications, for example, reconnaissance with a portable robot outfitted with a camera that must locate a moving focus in a jumbled workspace.

A couple limits were presented, and a complete algorithm was introduced for processing an effective motion strategy. For a simply connected free space, a logarithmic bound was set up on the minimum number of pursuer required. Loose bounds for increase associated free spaces were likewise given. A set of problems that were feasible by a solitary follower and obliged a direct number of re-contamination was demonstrated. The complete algorithm searches a limited cell complex that was built on the premise of basic data changes. This idea had the capacity be connected on a fundamental level to different pursuer problems, and the instance of a single pursuer was executed. A few solution techniques were demonstrated, the vast majority of which were figured in almost no time on a standard workstation.

A Guaranteed Search with Spanning Trees (GSST), an anytime algorithm for multi-robot search was proposed in [22]. The issue was as per the following: clear the environment of any adversarial target utilizing the least number of searchers. This issue is NP-hard on subjective graphs however can be settled in linear time on trees. The proposed algorithm created traversing trees of a graphical representation of the environment to control the search. At any time, spanning tree generation can be stopped yielding the best strategy so far. The subsequent method can be adjusted online if extra data gets to be benefit capable. In spite of the fact that GSST does not have performance guarantees after its first emphasis, it was demonstrated that few varieties had the capacity to find an ideal solution given adequate runtime.

2.6 Probabilistic Search and Pursuit Evasion

Unlike traditional search strategies that try to find a guaranteed solution for a graph or a polygonal environment, the probabilistic search methods consider enhancement of the normal estimation of a search target, for example, maximal likelihood of recognition or insignificant time to catch. Numerous probabilistic search techniques have been proposed so far. A mixed observability based robotic tasks planning algorithm under uncertainty is proposed in [23]. These type of probabilistic algorithms try to maximize the capture probability of an evader. The probability is calculated in many ways, such as using the probabilistic motion model of the evader or calculating the evader probability on the basis of previous search history. It is NP-hard to find an evader in a polygon or in a graph using probabilistic search techniques. Unlike classical pursuit-evasion and graph search which rely on evader motion model or uncertainty in search strategy, an alternative formulation of multi-robot search problems uses a probabilistic approach to model the location of the evader or the movement of the searchers [1].

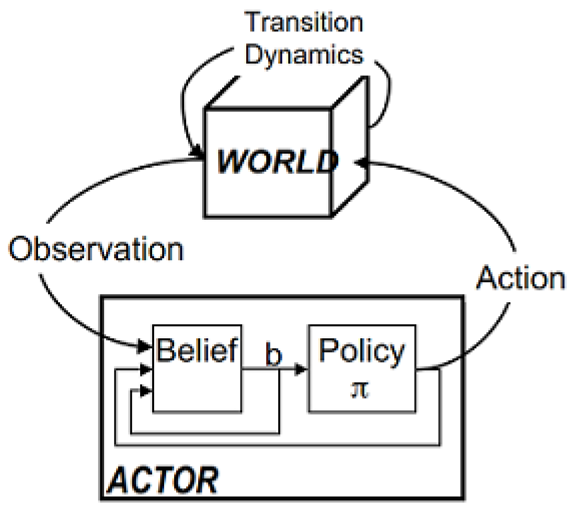


Figure 2.6: A POMDP Algorithm example

CHAPTER 3: PROBLEM FORMULATION

The objective is to locate an adversarial evader in a graph with minimal computational effort. In this study, PE is set to take place in an indoor environment. Moreover it is assumed that, both pursuer and evader cannot leave the environment. The evader is adversarial having infinite speed, but can only move along edges and can hide in nodes. The pursuer has a unit speed and can move only one step at a time. Both the pursuer and evader know the map, evader also knows the pursuer position but pursuer does not know the evader position. The capture happens when either the evader and pursuer reside in the same node or evader comes in line of sight of pursuer. The pursuer has a 360 degree field of view camera mounted on it and can see an evader in its line of sight [17].

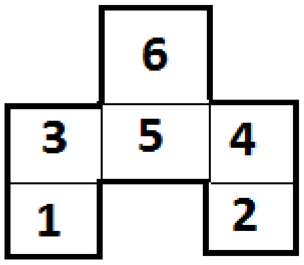
Consider the map of Figure 3.1.a. This map can be converted to a graph using discretization, as shown in Figure 3.1.b.

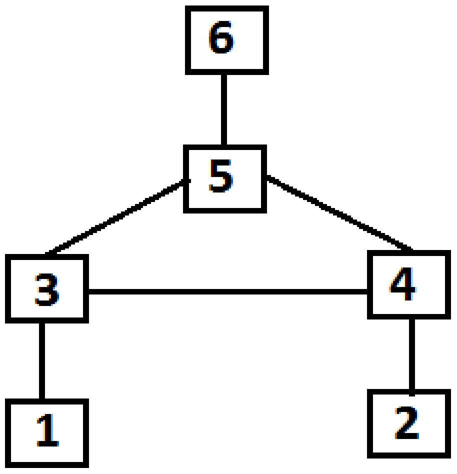
Some steps to convert the polygonal indoor environment to a graph are as following:

1. The indoor environment is discretized into cells, each of these cells relate to a node of an (undirected) navigation graph. The discretization happens on discriminating visibility occasions [17].
2. A (*directed*) *information space* is obtained from the *navigation graph*. The information graph is a *state transition diagram* of the search process [17].
3. Search happens in the information graph to discover a path from a beginning state to a terminal or clear state [17].

As we know that a graph G= (V, E) is described by a set of nodes or vertices V and edges E, such that $E \subseteq V \times V$. The nodes are labelled by integers i.e. for a graph of N vertices we take $V = \{1, 2, 3, \ldots, N\}$. We also have $|V| = N$ (using the *set cardinality notation*). The map to graph conversion is performed by:

1. Discretize the environment into cells.
2. Associate a node to each cell
3. Insert edges to nodes which correspond to adjacent nodes and also an edge to the node itself.





(a) **(b)**

Figure 3.1 (a) A map of an indoor environment (b) Graph of the map

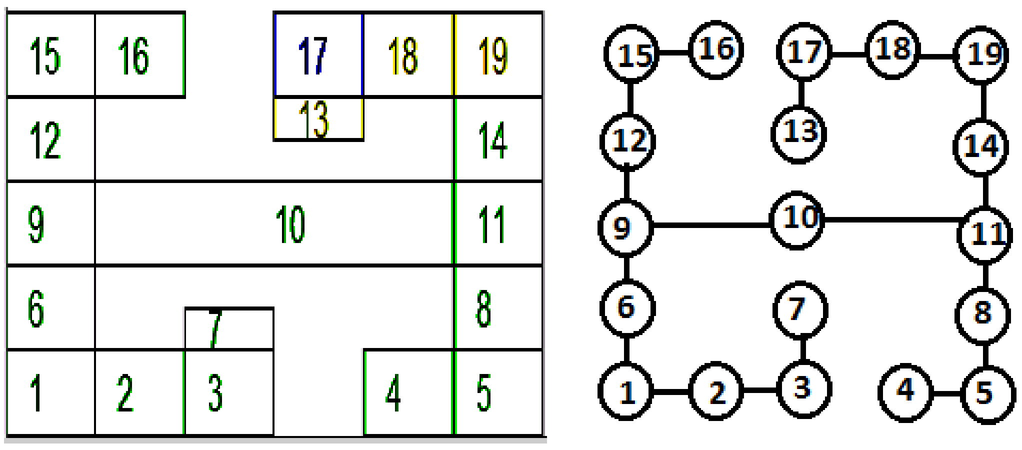
Given a graph G = (V; E) with N nodes, the adjacency of node $n \in V$ is denoted by N (n) and defined by

$$N(n) = \{m \in V: (n, m) \in E\}.$$

The $N \times N$ adjacency matrix **A** of the graph is defined by

$$A_{mn} = \begin{cases} 1 \; iff \;\; n \in N(m); \\ \quad\quad 0 \; else. \end{cases}$$

The adjacency matrix of graph in Figure 3.1.2.b is



(a) (b)

Figure 3.1.2. Conversion of a map to graph (a) a map of an indoor environment (b) a graph of 3.1.2.a

$$A=\begin{bmatrix}
1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&1&1&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&1&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1
\end{bmatrix} \quad (1)$$

For a given graph the *visibility region* of node $n \in V$ is denoted as V (n) and is defined as

$$V(n) = \{m \in V: m \text{ is visible from } n\}.$$

The condition “*m* is visible from *n*” is evaluated under “straight line visibility” and *iff* every point of *m* is visible from every point of *n*. The $N \times N$ visibility matrix **C** of the graph is defined by

$$\mathrm{C}mn = \begin{cases} 1\ iff\ \ n\ \in V\ (m); \\ \quad\quad 0\ else. \end{cases}$$

The visibility matrix of graph in Figure 3.1.2b is

$$\mathbf{C} = \begin{bmatrix}
1&1&1&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&0&1&0&0&1&0&0&1&0&0&0&0&1\\
1&0&0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0&0&1\\
1&0&0&0&0&1&0&0&1&1&1&1&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&1&1&1&1&0&0&1&0&0&0&0&1\\
1&0&0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0&0&1\\
1&0&0&0&0&1&0&0&1&0&0&1&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&1&1&1
\end{bmatrix}$$

(2a)

The direction matrix **F** of graph in Figure 3.1.2b is given as

$$\mathbf{F} = \begin{bmatrix} \mathbf{0} & \boldsymbol{N} & \boldsymbol{S} & \boldsymbol{E} & \boldsymbol{W} \\ \mathbf{1} & \mathbf{6} & \mathbf{0} & \mathbf{2} & \mathbf{0} \\ \mathbf{2} & \mathbf{0} & \mathbf{0} & \mathbf{3} & \mathbf{1} \\ \mathbf{3} & \mathbf{7} & \mathbf{0} & \mathbf{0} & \mathbf{2} \\ \mathbf{4} & \mathbf{0} & \mathbf{0} & \mathbf{5} & \mathbf{0} \\ \mathbf{5} & \mathbf{8} & \mathbf{0} & \mathbf{0} & \mathbf{4} \\ \mathbf{6} & \mathbf{9} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{7} & \mathbf{0} & \mathbf{3} & \mathbf{0} & \mathbf{0} \\ \mathbf{8} & \mathbf{11} & \mathbf{5} & \mathbf{0} & \mathbf{0} \\ \mathbf{9} & \mathbf{12} & \mathbf{6} & \mathbf{10} & \mathbf{0} \\ \mathbf{10} & \mathbf{0} & \mathbf{0} & \mathbf{11} & \mathbf{9} \\ \mathbf{11} & \mathbf{14} & \mathbf{8} & \mathbf{0} & \mathbf{10} \\ \mathbf{12} & \mathbf{15} & \mathbf{9} & \mathbf{0} & \mathbf{0} \\ \mathbf{13} & \mathbf{17} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{14} & \mathbf{19} & \mathbf{11} & \mathbf{0} & \mathbf{0} \\ \mathbf{15} & \mathbf{0} & \mathbf{12} & \mathbf{16} & \mathbf{0} \\ \mathbf{16} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{15} \\ \mathbf{17} & \mathbf{0} & \mathbf{13} & \mathbf{18} & \mathbf{0} \\ \mathbf{18} & \mathbf{0} & \mathbf{0} & \mathbf{19} & \mathbf{17} \\ \mathbf{19} & \mathbf{0} & \mathbf{14} & \mathbf{0} & \mathbf{18} \end{bmatrix} \qquad (2b)$$

Note that the graph can be cleared by only one pursuer, so it is assumed that the PE takes place in G= (*V*, *E*) with *V*= {1, 2, …. , N}. The position of pursuer at time *t* is *x* (*t*). If a node *might* contain evader, the node is called *dirty* otherwise *clear.* A node is clear if pursuer is present in it or it is a part of the pursuer's *visibility region* and remains *clear* until there is no free (pursuer's *visibility free*) path from it to an already contaminated node. The node is re-contaminated if there is a pursuer's *visibility free* path from it to an already contaminated node after it is cleared. A set of all the dirty nodes is denoted by *D*.

The indicator vector for *D* is **d**= {d_1, d_2, …., d_N }, where $d_N = 1$ *iff* $n \in D$, 0 else. The task of the pursuer is to capture the evader by clearing the nodes and in return converting the *dirty* set to *clear* set. At any time *t*, the *state vector* is the *position* of pursuer at time *t* and the *dirty* node set. The *state vector* is denoted as

$$\mathbf{z}\,(t) = [x\,(t), \mathbf{d}\,(t)] \qquad (3)$$

The task of the pursuer is to make the state vector at some time *t* as **z** = (*x*, 0), that is no more dirty nodes should be present in the graph.

Assume for a minute that the follower is expelled from the graph and the evader moves with unit speed. At that point, the conceivable areas of the evader at time t +1 are the ones adjoining its conceivable areas at time t. Scientifically this is expressed by [17].

$$\mathbf{d}\,(t + 1) = \mathbf{d}\,(t) * A; \qquad (4)$$

On the off chance that the evader has speed M (it crosses M edges in a solitary time step) then rather than (4) we have

$$\mathbf{d}\,(t + 1) = \mathbf{d}\,(t) * A^{[M]} \qquad (5)$$

CHAPTER 4: METHODOLOGY

Since it is assumed that the evader is adversarial so the pursuer does not know the evader position. At any time *t* the pursuer knows its own position with certainty but the evader position is unknown, only the probability of evader can be calculated from the dirty node set, that is why the search algorithm used is a probabilistic algorithm and is called Partially Observable Markov Decision Process (POMDP)[24][25]. The algorithm is slightly modified according to the problem, as the terminal state is unknown, the standard policy and value iteration solutions cannot be applied.

In the PE process, suppose that the pursuer starts at an initial node **z**in= (*x*in; **d**in). The pursuer must travel through nodes in a way such that the 1's in the d part continuously diminish (the dirty set shrinks) until in the long run the follower achieves one of the clear states, say zfin= (xfin; dfin), where d fin= 0.

As the given graph in Fig.3.1.2b contains 19 nodes, a dirty node vector of 19 elements is defined as

$$\mathbf{d} = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$$

The pursuer's task is to make all the elements of this dirty node vector zero. A (pursuer's *visibility region*) free path between a dirty node and a cleared node makes the cleared node re-contaminated. In the given problem, the set of actions is denoted by **E** and is defined as

$$\mathbf{E} = [\text{North, South, East, West}]$$

It is assumed that the sensors of pursuer are accurate, that is the pursuer knows its position with certainty, and the observations are accurate. The actuators however are not accurate enough, and the action probability of pursuer is 90%. However the state transition probabilities are considered to be 100% for simplicity and to minimize the resources used.

4.1 The Search Algorithm

```
The Search Algorithm
Input Graph G= (V, E); Adjacency matrix A; Visibility matrix C; Direction matrix F; Dirty vector d
 Find hard nodes; H = (nodes visible from minimum nodes, from visibility matrix C)
  Input Starting node i (must be a member of H)
Initialization
 Assign value to i = 0;
 Assign values to remaining H = -100;
  While (~assigned values to all the nodes)
        Assign value to nodes adjacent to previously assigned nodes;
If     (previous node is adjacent to initial node)
node value = abs (previous adjacent node value) + 5
      else
         node value = abs (previous adjacent node value) – 5
end if
end while
Main
While (d is not empty)
   Move to the adjacent node with maximum value;
    Assign current node value = 0;
      Previous node value = previous value + 3;
       Add previous node to visited set;
        H nodes value = –100;
          If (a clear node is re-contaminated)
            Assign value to contaminated node;
                              Contaminated node value = +90;
          end if
end While
Total cost = [{(no of steps taken before d becomes empty) × –10} + {(no of times contaminations occurred) × –30}]
```

Output the policy generated (from the movement history of robot)

The algorithm can be defined as.

1. In the first step of the algorithm, the Hard Nodes vector **H** is formed by finding the nodes that are visible from only one node from the visibility matrix **C**.
2. Then a node is selected from the **H** and is assigned a value equal to zero. All the other nodes of **H** are assigned values equal to -100. The hard nodes should be assigned a negative value because, if the pursuer enters a hard node it will lose visibility of the graph and in return the whole graph will be re-contaminated.
3. After assigning a value to the initial node and hard nodes, values are assigned to the adjacent nodes of **H**. The nodes adjacent to the initial node are assigned a value equal to the value of initial node + 5. The nodes adjacent to the remaining hard nodes are assigned a value equal to the absolute value of hard node – 5;
4. The process of assigning values to nodes continues until all the nodes are assigned with some value. The values of nodes adjacent to the initial node are assigned in increasing order, while the nodes adjacent to the remaining hard nodes get a value in decreasing order as compared to their parent node.
5. After all the nodes are assigned with a value, the pursuer starts to move from the initial node by first finding possible actions from direction matrix **F**, and then finding an action that maximizes the overall value. At the same time avoiding recontamination by entering the hard nodes.
6. In order to maximize the overall value, the negative cost at each step forces the pursuer to find the terminal state in minimum steps possible.
7. The node containing the pursuer is assigned a value equal to zero. The previous node is assigned a value equal to the node's previous value + 3. In this way the value of each visited node again starts increasing that helps the pursuer to return from a corridor like structure.
8. Every time a node is cleared, it is added to the visited node set. If at some time a node that was previously a member of visited node set again becomes a member of dirty node vector, it is considered re-contaminated and is removed from visited set, as well as assigned a value equal to +90. So that to force the pursuer to clear the node again.

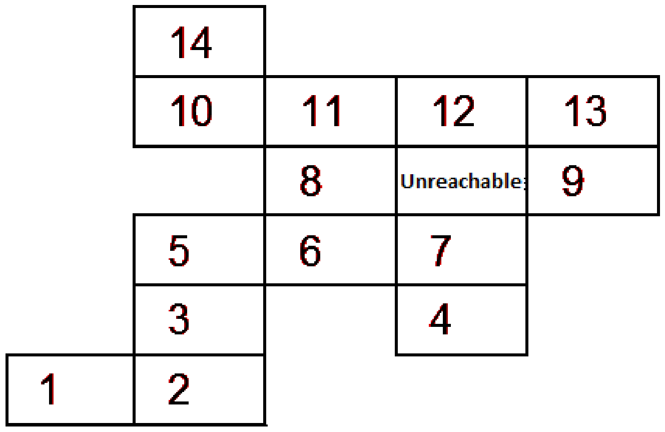
9. The process continues until the dirty node set becomes empty.
10. After the terminal state is obtained, a policy is generated that is a mapping of pursuer movements in graph.

The algorithm proposed in this thesis resembles POMDP in a way that it uses *value iteration* to assign values to all the nodes. After values are assigned, the algorithm uses *policy iteration* to find the actions that maximize the overall value. The algorithm applies value iteration after each step to prevent the pursuer from going into a loop. It also tries to minimize the cost by finding the terminal state in minimum steps possible, while at the same time guaranteeing capture. It should be kept in mind that the algorithm is a modified form of the POMDP algorithm, which has been modified according to the problem requirements. It is not essentially the pure POMDP algorithm.

CHAPTER 5: IMPLEMENTATION

Consider the Workspace and graph in Figure 5.1.a and Figure 5.1.b respectively.
The graph contains four Hard Nodes, as the node set {1, 4, 9, 14} are the nodes that are only visible from one node {2, 7, 13, 10} respectively. The first step according to the proposed search algorithm is to assign numbers to the nodes of the graph. After numbers are assigned, the nodes adjacency matrix is formulated. After node adjacency matrix, the node visibility matrix is determined. After both the node adjacency matrix and node visibility matrix are determined, the Hard Nodes are calculated. The Hard Nodes are those nodes that are visible only to minimum number of nodes in a graph. They are determined from the node visibility matrix.

5.1 Workspace No 1



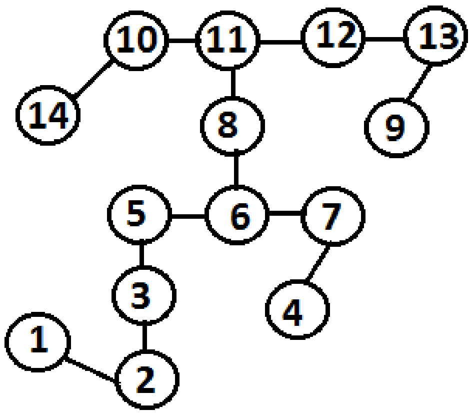


Figure 5.1 (a) The Workspace (b) Corresponding Graph

The node adjacency matrix **A** of figure 5.1 is given as

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.1.a)$$

And the node visibility matrix **C** is given as

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.1.b)$$

The node direction matrix **F** is given as

$$\mathbf{F} = \begin{bmatrix} 0 & N & S & E & W \\ 1 & 0 & 0 & 2 & 0 \\ 2 & 3 & 0 & 0 & 1 \\ 3 & 5 & 2 & 0 & 0 \\ 4 & 7 & 0 & 0 & 0 \\ 5 & 0 & 3 & 6 & 0 \\ 6 & 8 & 0 & 7 & 5 \\ 7 & 0 & 4 & 0 & 6 \\ 8 & 11 & 6 & 0 & 0 \\ 9 & 13 & 0 & 0 & 0 \\ 10 & 14 & 0 & 11 & 0 \\ 11 & 0 & 8 & 12 & 10 \\ 12 & 0 & 0 & 13 & 11 \\ 13 & 0 & 9 & 0 & 12 \\ 14 & 0 & 10 & 0 & 0 \end{bmatrix} \quad (5.1.c)$$

After calculating the node adjacency matrix, node visibility matrix and node direction matrix, the Hard Nodes are calculated.

The Hard nodes in Figure 5.1.b are

Hard Nodes = {1, 4, 9, 14}

And they are visible from {2, 7, 13, 10}

Initially the dirty node set is given as

D = [1 1 1 1 1 1 1 1 1 1 1 1 1 1]

After finding the hard nodes, a member of hard nodes is selected at random and assigned a value of -100. All other hard nodes are given values equal to +100. After assigning the values to the hard nodes the nodes adjacent to hard nodes are assigned a value and then this continues like a brush fire algorithm until all nodes have a value assigned to them.

After initial values are assigned, the pursuer starts its movement from the starting node and performs action from the actions set. The action is chosen that maximizes the overall value according to the node value and reward or penalty. This process continues until the evader is found or the graph is cleared. If no evader is found, the graph is declared as clear.

If node **‘1’** is selected as starting node then the initial values of nodes are

Initial Values = [0   9   19 -100   29   82   91   71 -100   92   83   82   91 -100]

And the node track after clearing the whole graph is

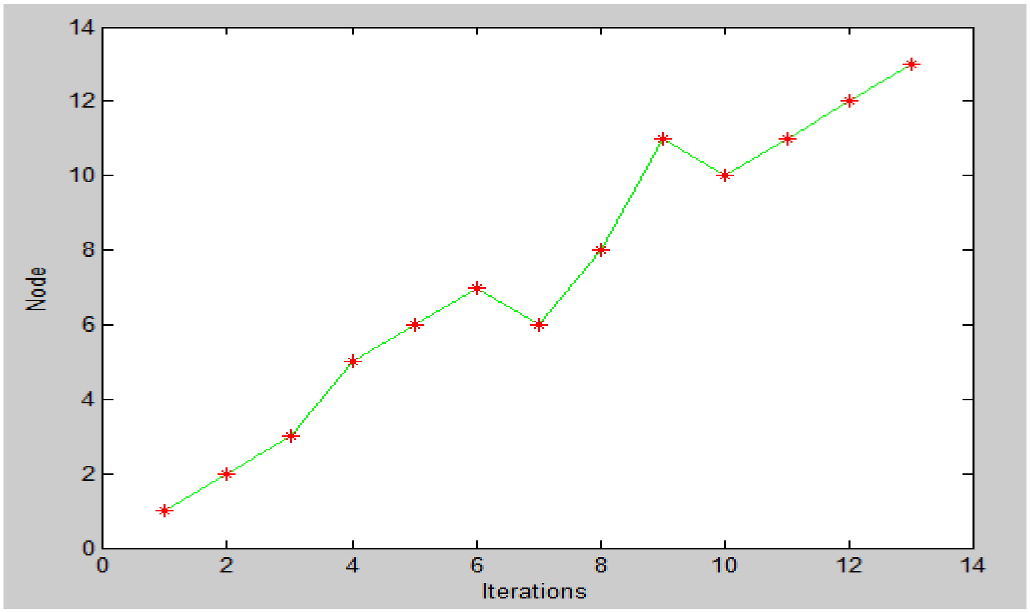


Figure 5.1.1: Node track after starting from node 1

The policy generated is

Final generated policy =   1   2   3   5   6   7   6   8 11 10 11 12 13

If the starting node is **‘4’** then the initial values of nodes are

Initial Values = [-100   91   81   0   71   18   9   29 -100   92   83   82   91 -100]

And the node track after clearing the whole graph is

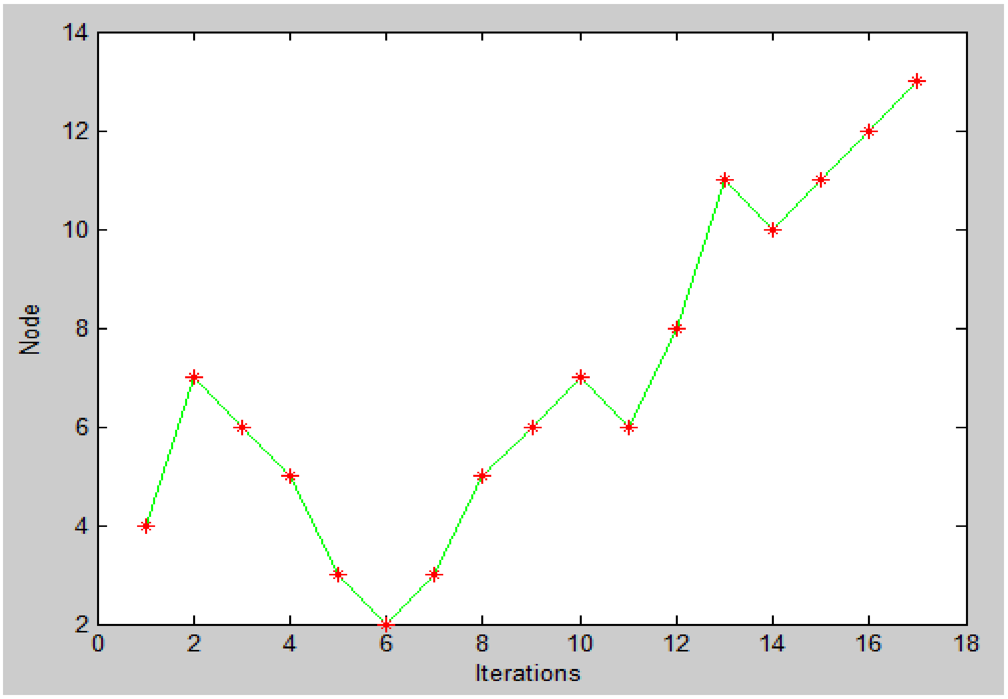


Figure 5.1.2: Node track after starting from node 4

The policy generated is

Final generated policy = 4 7 6 5 3 2 3 5 6 7 6 8 11 10 11 12 13

It can be noted that in case of starting from node '4', the graph gets contaminated, that is why it takes more iteration to clear the whole graph. It can also be seen from the node track that even after contamination, the algorithm successfully cleared the graph.

If the starting node is **'9'** then the initial values of nodes are

Initial Value = [-100 91 81 -100 71 82 91 71 0 92 83 18 9 -100]

And the node track after clearing the whole graph is

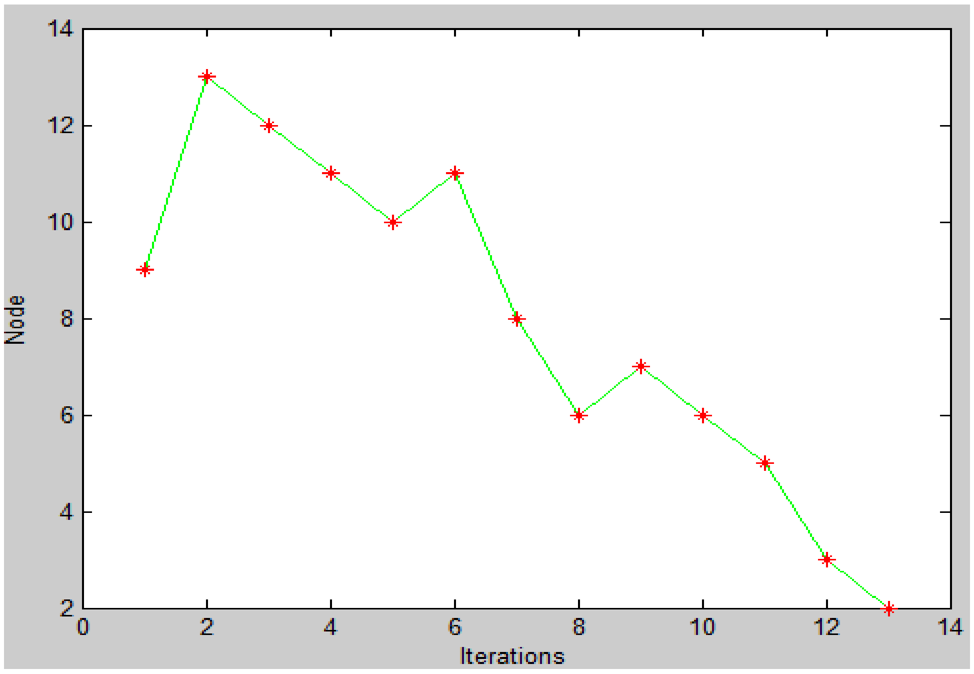


Figure 5.1.3: Node track after starting from node 9

The policy generated is

Final generated policy = 9 13 12 11 10 11 8 6 7 6 5 3 2

If the starting node is **'14'** then the initial values of nodes are

Initial Value = [-100 91 81 -100 71 82 91 71 -100 8 17 82 91 0]

The generated policy is

Final generated policy = 14 10 11 12 13 12 11 8 6 7 6 5 3 2

And the node track is given as

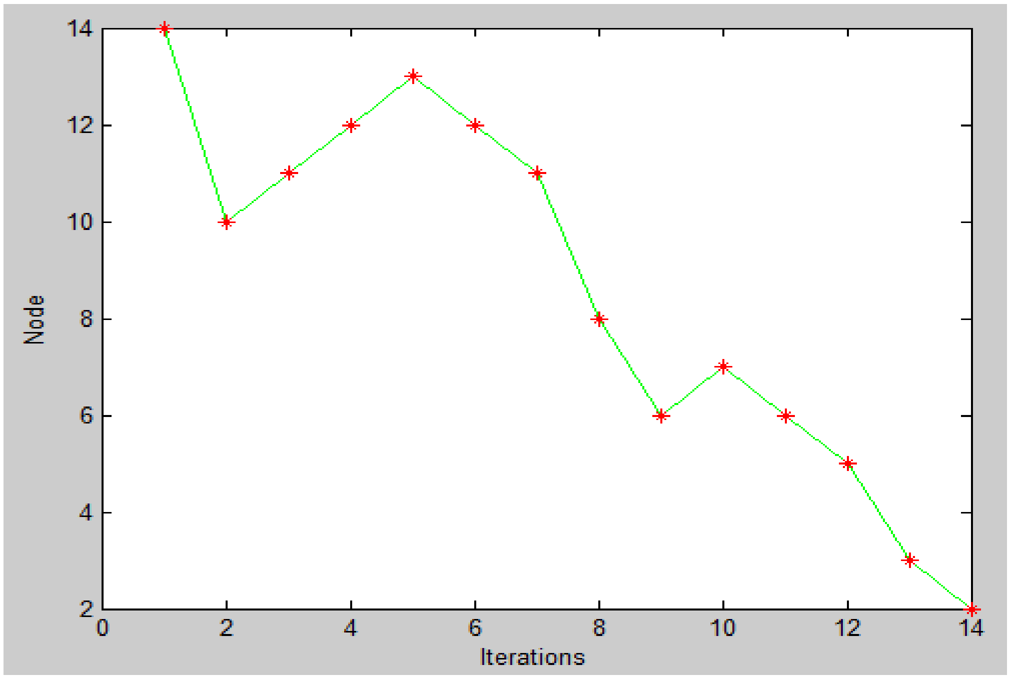
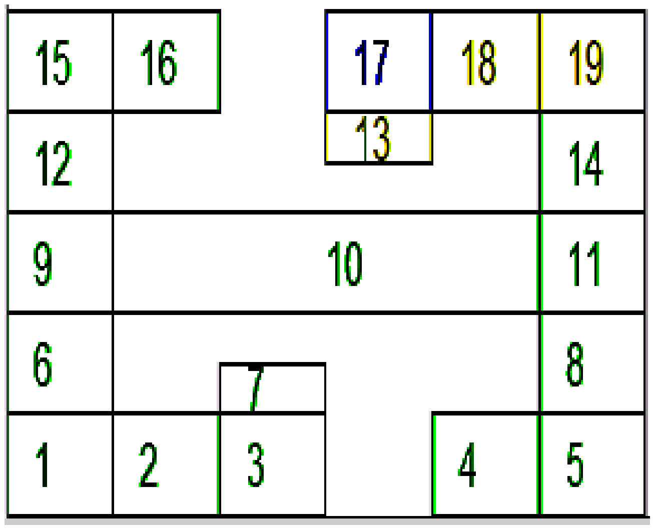


Figure 5.1.4: Node track after starting from node 14

5.2 Workspace No 2

Similarly consider the workspace and graph in Figure 5.2.a and Figure 5.2.b respectively.



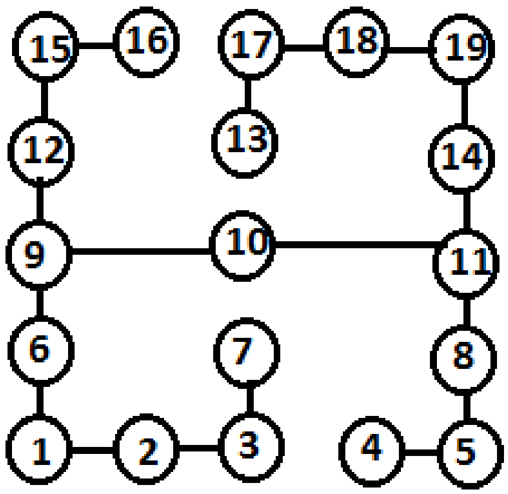


Figure 5.2: (a) The Workspace (b) The Graph

The node adjacency matrix **A** of the Figure 5.2.b is given as

$$
\mathrm{A}=\begin{bmatrix}
\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}
\end{bmatrix} \quad (5.2.a)
$$

And the node visibility matrix **C** is given as

$$
\mathbf{C}=\begin{bmatrix}
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}\\
\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}\\
\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}\\
\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}\\
\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}\\
\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}
\end{bmatrix} \quad (5.2.b)
$$

The node direction matrix **F** is given as

$$\mathbf{F} = \begin{bmatrix} \mathbf{0} & \boldsymbol{N} & \boldsymbol{S} & \boldsymbol{E} & \boldsymbol{W} \\ 1 & 6 & 0 & 2 & 0 \\ 2 & 0 & 0 & 3 & 1 \\ 3 & 7 & 0 & 0 & 2 \\ 4 & 0 & 0 & 5 & 0 \\ 5 & 8 & 0 & 0 & 4 \\ 6 & 9 & 1 & 0 & 0 \\ 7 & 0 & 3 & 0 & 0 \\ 8 & 11 & 5 & 0 & 0 \\ 9 & 12 & 6 & 10 & 0 \\ 10 & 0 & 0 & 11 & 9 \\ 11 & 14 & 8 & 0 & 10 \\ 12 & 15 & 9 & 0 & 0 \\ 13 & 17 & 0 & 0 & 0 \\ 14 & 19 & 11 & 0 & 0 \\ 15 & 0 & 12 & 16 & 0 \\ 16 & 0 & 0 & 0 & 15 \\ 17 & 0 & 13 & 18 & 0 \\ 18 & 0 & 0 & 19 & 17 \\ 19 & 0 & 14 & 0 & 18 \end{bmatrix} \quad \textbf{(5.2.c)}$$

The Hard Nodes in Figure 5.2.b are

Hard Nodes = {4, 7, 13, 16}

And they are visible only from {5, 3, 17, 15}

The initial dirty node set is

D = [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

If node **'4'** is selected as starting node then the initial values of nodes are

Initial Value = [76 84 92 0 9 66 -100 19 76 38 29 84 -100 49 92 -100 91 81 71]

The node track is

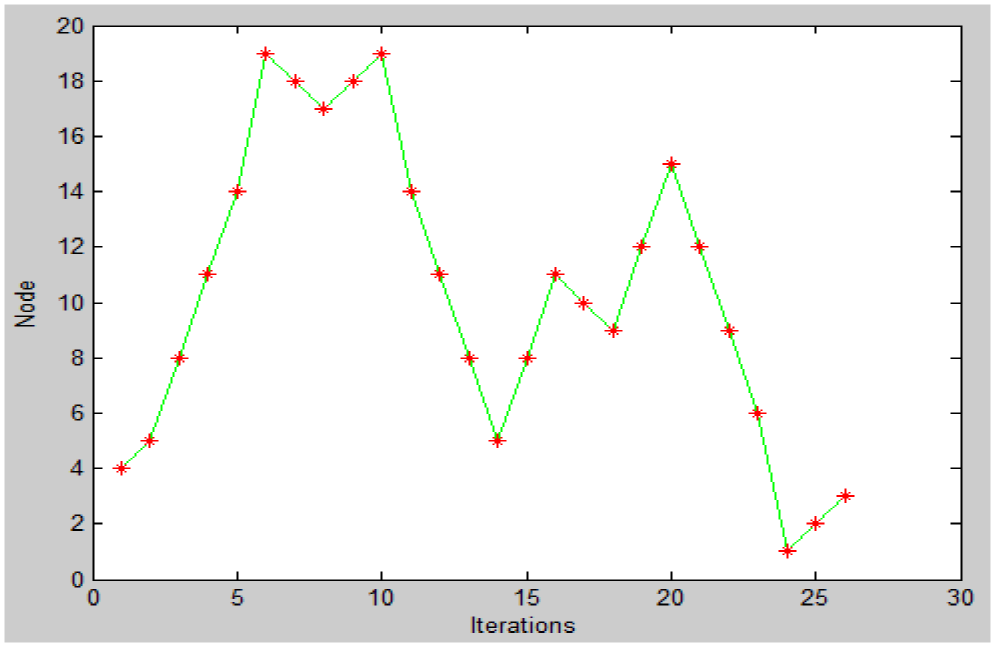


Figure 5.2.1: Node track after starting from node 4

And the generated policy is

Final generated policy =

4 5 8 11 14 19 18 17 18 19 14 11 8 5 8 11 10 9 12 15 12 9 6 1 2 3

If the starting node is **‘7’** then the initial nodes value are

Initial Value =

24 16 8 -100 91 34 0 81 76 62 71 84 -100 51 92 -100 91 81 71

The generated policy is

Final generated policy = 7 3 2 1 6 9 12 15 12 9 10 11 8 5 8 11 14 19 18 17

And the node track is given as

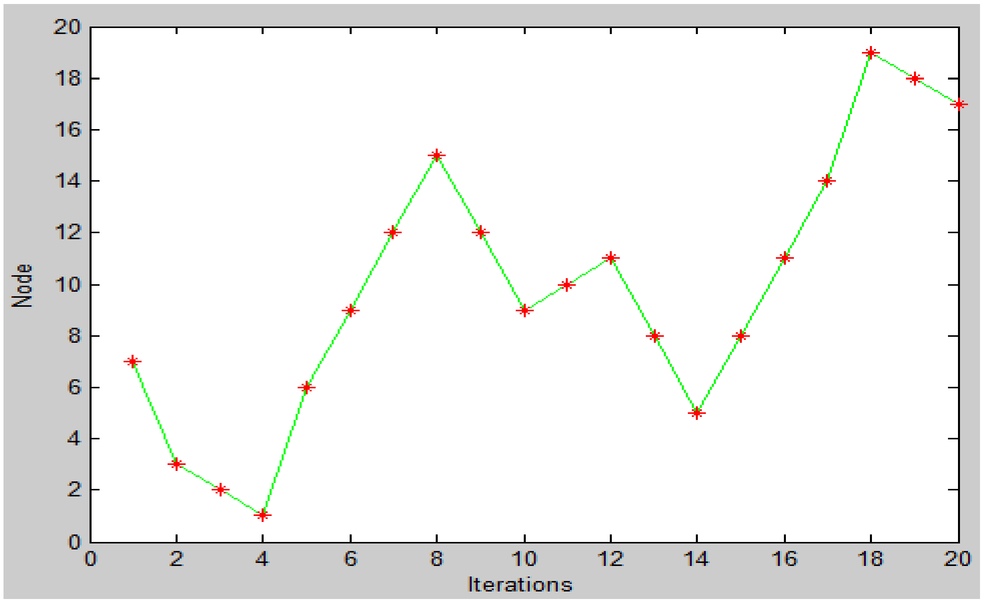


Figure 5.2.2: Node track after starting from node 7

If the starting node is **'13'** then the initial values of the nodes are

Initial Value =

76 84 92 -100 91 66 -100 81 76 62 71 84 0 51 92 -100 9 19 29

The node track is given as

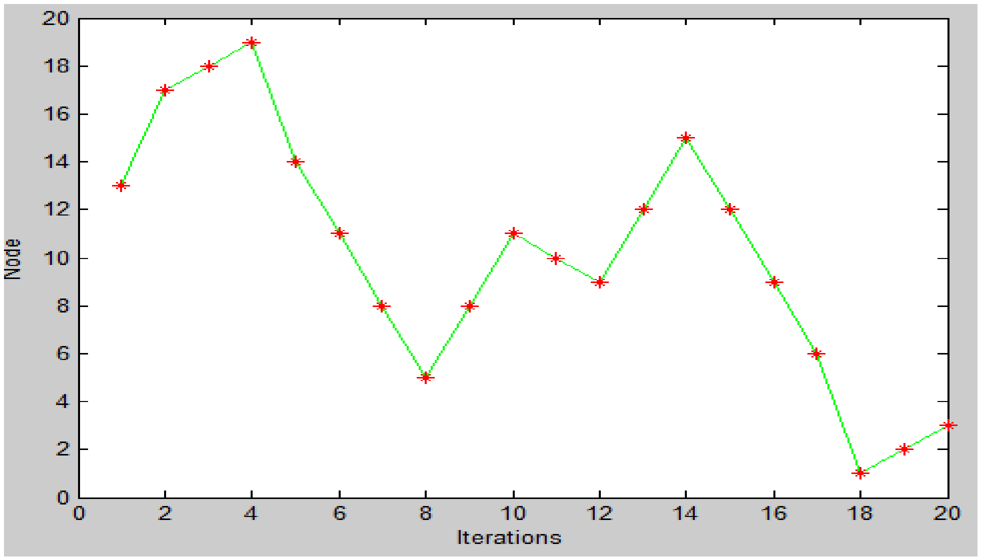


Figure 5.2.3: Node track after starting from node 13

The generated policy is given as

Final generated policy = 13 17 18 19 14 11 8 5 8 11 10 9 12 15 12 9 6 1 2 3

5.3 Workspace No 3

Now consider the Workspace in Figure 5.3.a and the Graph of Figure 5.3.b.

The graph has 30 nodes, with 6 hard nodes. The hard nodes are

Hard Nodes = {10, 11, 12, 13, 15, 29}

The graph has certain areas where pursuer cannot clear the graph without recontamination of the graph. This graph is difficult even for a single pursuer using exhaustive search. The proposed algorithm successfully cleared this graph in all scenarios.

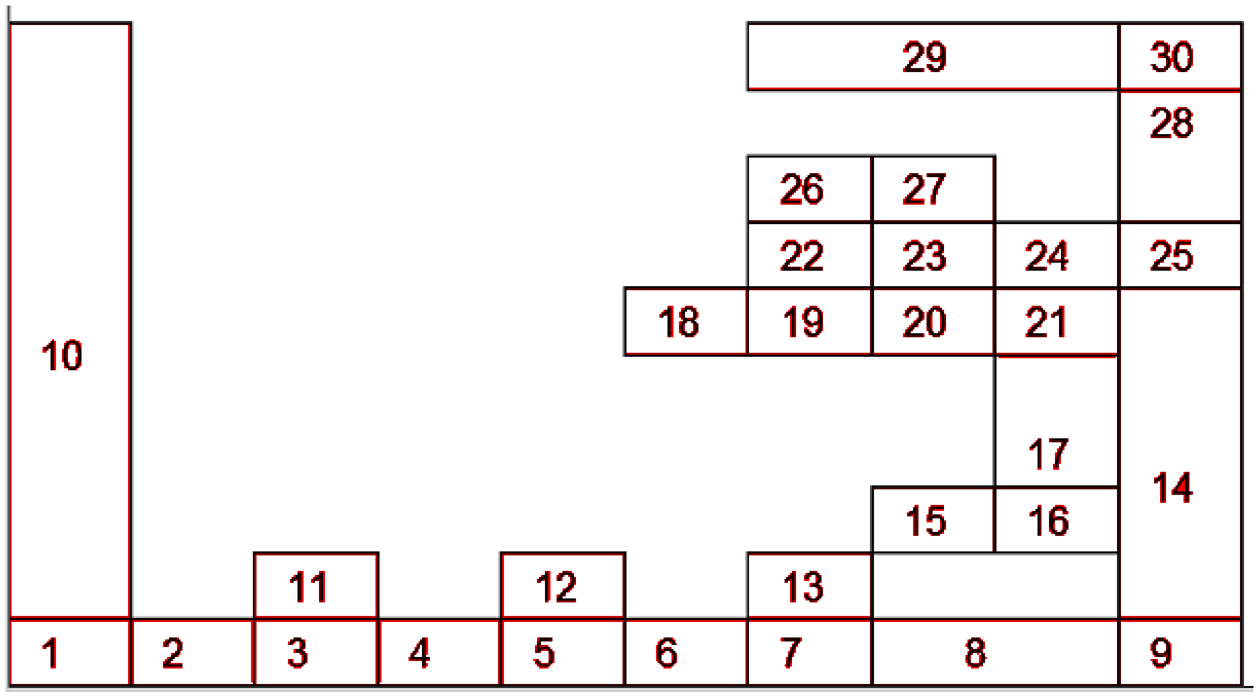


Figure 5.3 (a) The Workspace no 3

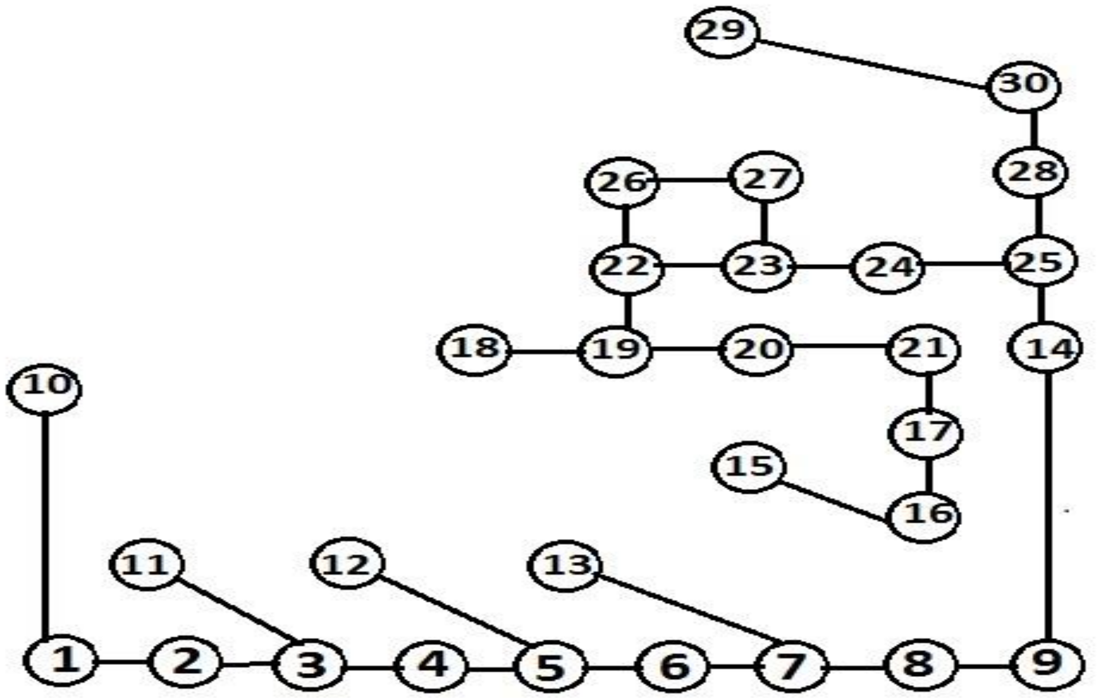


Figure 5.3 (b) The Graph no 3

The node adjacency matrix **A** of graph shown in Figure 5.3.b is given as

```
1 2 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 3 4 0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 4 5 6 0 0 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 5 6 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 6 7 8 0 0 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 7 8 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 8 9 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 3 0 0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 5 0 0 0 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 7 0 0 0 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 9 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 25 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 15 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 15 16 17 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 17 0 0 0 21 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18 19 0 0 0 0 0 0 0 0 0 0 0
```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18 19 20 0 22 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 19 20 21 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 17 0 0 20 21 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 19 0 0 22 23 0 0 26 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 22 23 24 0 0 27 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 23 24 25 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 14 0 0 0 0 0 0 0 0 0 24 25 0 0 28 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 22 0 0 0 26 27 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 23 0 0 26 27 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 25 0 0 28 0 30
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 29 30
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 28 29 30

The node visibility matrix **C** is given as

1 2 3 4 5 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 0 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 25 0 0 28 0 30
1 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 3 0 0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 5 0 0 0 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 7 0 0 0 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 9 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 25 0 0 28 0 30
0 0 0 0 0 0 0 0 0 0 0 0 0 0 15 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 15 16 17 0 0 0 21 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 17 0 0 0 21 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18 19 20 21 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18 19 20 21 22 0 0 0 26 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18 19 20 21 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 17 18 19 20 21 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 19 0 0 22 23 24 25 26 27 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 22 23 24 25 26 27 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 22 23 24 25 0 0 0 0 0
0 0 0 0 0 0 0 0 9 0 0 0 0 14 0 0 0 0 0 0 0 22 23 24 25 0 0 28 0 30

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 19 0 0 22 23 0 0 26 27 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 22 23 0 0 26 27 0 0 0
0 0 0 0 0 0 0 0 9 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 25 0 0 28 0 30
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 29 30
0 0 0 0 0 0 0 0 9 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 25 0 0 28 29 30
```

The node direction matrix **F** is given as

0	N	S	E	W
1	10	0	2	0
2	0	0	3	1
3	11	0	4	2
4	0	0	5	3
5	12	0	6	4
6	0	0	7	5
7	13	0	8	6
8	0	0	9	7
9	14	0	0	8
10	0	1	0	0
11	0	3	0	0
12	0	5	0	0
13	0	7	0	0
14	25	9	0	0
15	0	0	16	0
16	17	0	0	15
17	21	16	0	0
18	0	0	19	0
19	22	0	20	18
20	0	0	21	19
21	0	17	0	20
22	26	19	23	0
23	27	0	24	22
24	0	0	25	23
25	28	14	0	24
26	0	22	27	0
27	0	23	0	26
28	30	25	0	0
29	0	0	30	0
30	0	28	0	29

There are 6 hard nodes in this graph. These hard nodes are

Hard nodes = {10, 11, 12, 13, 15, 29}

The initial dirty node vector is given as

D = [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

If the starting node selected is **‘10’** then the initial values are

initial_value =

```
 8    17    92    82    92    82    92    82    72     0  -100  -100  -100    62  -100    91    81    46    54
62    71    35    58    66    75    24    47    83  -100    91
```

The node track is given as

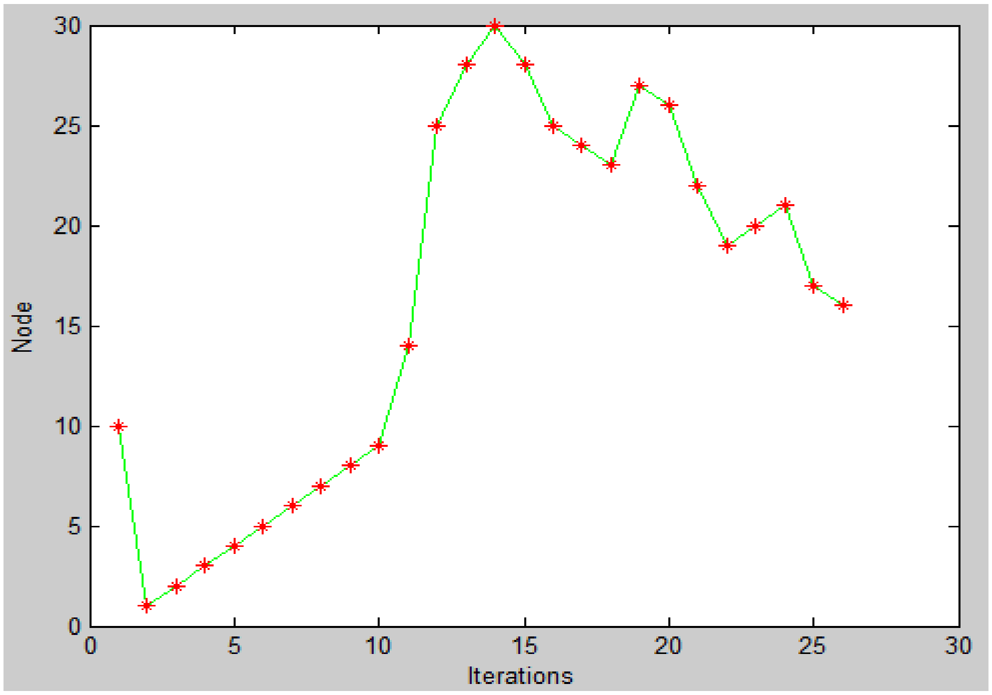


Figure 5.3.1: Node track after starting from node 10

And the generated policy is given as

Final generated policy =

10 1 2 3 4 5 6 7 8 9 14 25 28 30 28 25 24 23 27 26 22 19 20 21 17 16

If the starting node is **‘15’** then the initial values are

```
initial_value =
 92    83    92    82    92    82    92    82    72  -100  -100  -100  -100    62     0 9    19    54    46
38    29    65    58    66    75    76    47    83  -100    91
```

The node track is given as

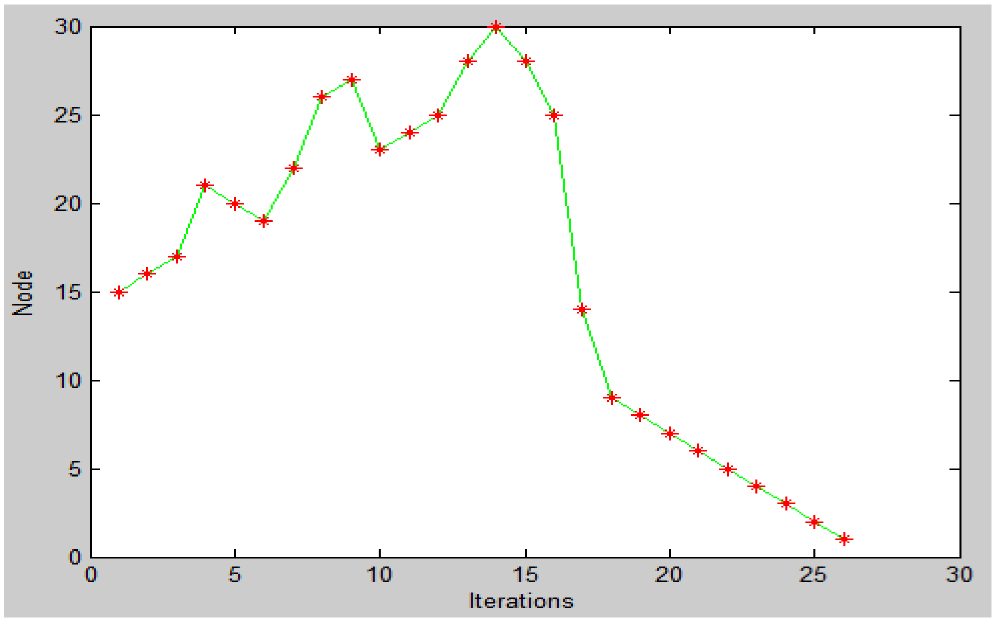


Figure 5.3.2: Node track after starting from node 15

The generated policy is given as

Final generated policy =

15 16 17 21 20 19 22 26 27 23 24 25 28 30 28 25 14 9 8 7 6 5 4 3 2 1

CHAPTER 5: CONCLUSIONS AND FUTURE SUGGESTIONS

5.2 Conclusion

The modified POMDP algorithm proposed in this thesis effectively generated policies for graphs that are clearable by single pursuer. The algorithm first discretized a workspace in the form of cells. These cells were then converted to a graph. The graph is then cleared for an adversarial evader by a single pursuer successfully.

5.2 Future Suggestions

Pursuit-evasion (PE) has been extensively explored in the fields of software engineering, mathematics and robotics. Many different approaches have been proposed to capture one or more evaders throughout the course of history after the pursuit-evasion games in a graph were proposed. The approaches include search in graph, search in a polygon, adversarial search, non-adversarial search, probabilistic search, and search with only one pursuer and search with multiple pursuers etc. Many proposed strategies focused on finding a target in minimum time or minimum steps, others proposed guaranteed search algorithms irrespective of the cost incurred, time elapsed or steps taken. One of the main problem in pursuit evasion is to find multiple evader using multiple pursuers. The use of multiple pursuers is difficult as the pursuers are required to communicate their pursuit policy with one another. The pursuers can generate their local policy or a pursuer can generate policies for all its teammates and the team executes that policy. The robots in real life are not accurate enough as they have both sensor uncertainties and actuators uncertainties. Due to these uncertainties the best approach the plan a motion for pursuers is by using Partially Observable Markov Decision Process (POMDP). The POMDP can be effectively used to generate policies for one pursuer, but it is very difficult to generate policies for multiple pursuers.

REFRENCES

[1] T. H. Chung, G. a. Hollinger, and V. Isler, “Search and pursuit-evasion in mobile robotics A survey,” *Auton. Robots*, vol. 31, no. 4, pp. 299–316, 2011.

[2] M. Bardi, J. Pablo, M. Lopez, M. Bardi, J. Pablo, M. Lopez, and A. Dijkstra-type, “A Dijkstra-type algorithm for dynamic games To cite this version :,” 2015.

[3] S. M. LaValle, D. Lin, L. J. Guibas, J. C. Latombe, and R. Motwani, “Finding an Unpredictable Target in a Workspace With Obstacles,” *IEEE Int. Conf. Robot. Autom.*, p. 6, 1997.

[4] L. J. GUIBAS, J.-C. LATOMBE, S. M. LAVALLE, D. LIN, and R. MOTWANI, “A VISIBILITY-BASED PURSUIT-EVASION PROBLEM,” *International Journal of Computational Geometry & Applications*, vol. 09, no. 04n05. pp. 471–493, 1999.

[5] M. (Department of E. E. U. Yamashita, E. D. D. C. A. C. Umemoto, Hideki (System Engineering Section, I. (Department of E. E. and C. S. of W. Suzuki, and T. (School of C. S. F. U. Kameda, “Searching for Mobile Intruders in a Polygonal Region by a Group of Mobile Searchers.”

[6] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, “The complexity of searching a graph,” *22nd Annu. Symp. Found. Comput. Sci. (sfcs 1981)*, vol. 35, no. 1, pp. 18–44, 1981.

[7] D. Bienstock and P. Seymour, “Monotonicity in graph searching,” *J. Algorithms*, vol. 12, no. 2, pp. 239–245, 1991.

[8] A. S. LaPaugh, “Recontamination does not help to search a graph,” *J. ACM*, vol. 40, no. 2, pp. 224–245, Apr. 1993.

[9] J. W. Lin and S. C. Tang, “Coverage improvement for target tracking in hybrid sensor networks,” *2010 2nd Int. Conf. Comput. Autom. Eng. ICCAE 2010*, vol. 4, pp. 126–130, 2010.

[10] J. Zhang, C. Wu, Z. Jia, H. Chu, and P. Si, “Inner-cluster optimization based on coverage for target tracking in wireless sensor networks,” *J. Inf. Comput. Sci.*, vol. 8, no. 14, pp. 2959–2966, 2011.

[11] H. Miao, C. C. Ooi, X. Wu, and C. Schindelhauer, “Coverage-hole trap model in target tracking using distributed relay-robot network,” *Proc. 2010 ACM Symp. Appl. Comput. - SAC ’10*, p. 1299, 2010.

[12] D. S. Scott and J. Vuillemin, *Art Gallery Theorems and Algorithms the International Series of.* .

[13] R. Nowakowski and P. Winkler, “Vertex-to-vertex pursuit in a graph,” *Discrete Math.*, vol. 43, no. 2–3, pp. 235–239, 1983.

[14] M. Aigner and M. Fromme, “A game of cops and robbers,” *Discret. Appl. Math.*, vol. 8, no. 1, pp. 1–12, 1984.

[15] V. Isler, S. Kannan, and S. Khanna, “Randomized Pursuit-Evasion with Local Visibility,” *SIAM J. Discret. Math.*, vol. 20, no. 1, pp. 26–41, 2006.

[16] M. ADLER, H. RCKE, N. SIVADASAN, C. SOHLER, and B. VCKING, “Randomized Pursuit-Evasion in Graphs,” *Combinatorics, Probability and Computing*, vol. 12, no. 3. pp. 225–244, 2003.

[17] A. Kehagias and S. Singh, “A Graph Search Algorithm for Indoor Pursuit / Evasion,” no. July, pp. 1305–1317, 2008.

[18] V. Isler, S. Kannan, and S. Khanna, “Randomized Pursuit evasion in a polygonal environment,” *IEEE Trans. Robot.*, no. Wafr, 2005.

[19] D. (department of C. S. and E. of M. Bhadauria and V. (Department of C. S. and E. of M. Isler, “Capturing an Evader in a Polygonal Environment with Obstacles,” no. June, pp. 16–26, 2011.

[20] B. P. Gerkey, “Visibility-based Pursuit-evasion with Limited Field of View,” *The International Journal of Robotics Research*, vol. 25, no. 4. pp. 299–315, 2006.

[21] L. J. Guibas, J. L. Steven, M. L. David, and L. Rajeev, “A Visibility-Based Pursuit-Evasion Problem,” pp. 1–29.

[22] G. Hollinger, A. Kehagias, and S. Singh, “GSST: Anytime guaranteed search,” *Auton. Robots*, vol. 29, no. 1, pp. 99–118, 2010.

[23] S. C. W. Ong, Shao Wei Png, D. Hsu, and Wee Sun Lee, “Planning under Uncertainty for Robotic Tasks with Mixed Observability,” *Int. J. Rob. Res.*, vol. 29, no. 8, pp. 1053–1068, 2010.

[24] K. P. Murphy, “A survey of POMDP solution techniques,” *Environment*, vol. 2, no. September, p. X3, 2000.

[25] A. Barto, “Reinforcement learning: An introduction,” 1998.

APPENDIX A

```
function first_code()

close all
clear
clc

% disp(' 1 for local visibilty graph ');
% disp(' 2 for straight line visibilty ');
% choice = input(' 3 for Big Gaph of one pursuer = ');
workspace_no = input('please enter workspace no = ');
workspace(workspace_no)
text(210,190,'Possible nodes with values')
  valid1 = text(80,247,' ');
        text(5,225,'Visible Nodes =   ')

h1text = text(5,235,'Current Node = ');
h2text = text(28,230,' ');
h5text = text(210,180,' ');
h6text = text(210,172,' ');

[A_Decimal,B,C] = graph_select(workspace_no);
A=A_Decimal~=0;
len = length(A);

disp(' A is a nodes adjacency matrix. ');
%A_Decimal;
%A;
disp(' B is a nodes visibilty matrix. ');
%B;
disp(' C is a Direction Matrix, Node North South East West. ');
%C;
%%
for i = 1:len % Hard nodes Finding
    L = nnz(B(:,i));
    T(i,1)=L;
end
```

```
T;
[row,col] = find(T == min(T(:)));
Hard_nodes=row;
 Hard_nodes = Hard_nodes'
text(5,245,'Hard Nodes = ');
h3text = text(25,245,sprintf(' %.0f',Hard_nodes));
%%

disp('1 = North, 2 = South, 3 = East, 4 = West ');
Actions = [1 2 3 4];

 [P1,P2,P3,P4] = trans_prob(Actions,C);
 P1
 P2
 P3
 P4

total_nodes = A(1,:);
t_nodes = length(total_nodes);
initial_dirty_nodes = length(total_nodes);
dirty_nodes_vector = ones(1,initial_dirty_nodes);

disp(' D is a dirty nodes vector. ');

D = dirty_nodes_vector
contamination = 0;
n = nnz(D) ;  % find number of non zero elements

possible_nodes = linspace(1,t_nodes,t_nodes);

abc = length(possible_nodes);   % calculates the initial node probabilty
def = 1/abc;
initial_probabilty = zeros(1,abc);
for jij = 1:abc
   initial_probabilty(1,jij) = def;
end
```

```
initial_probabilty; % initial node probabilty
node_probabilty = initial_probabilty;

node = 0; % initially no node is selected

node_track = zeros(1,60); % tracks the movement of pusuer
i = 0;
iterations = 0;

prompt={'Enter Choice:'};
 titled = ('Enter first node');
 answerd = inputdlg(prompt,titled);
 node = str2double(answerd{1});

 % START TIMER
 Tic

 current_node = node;
 first_node = current_node;

initial_value = i_value(first_node,Hard_nodes,A_Decimal)
% initial_value = i_value_mex(first_node,Hard_nodes,A_Decimal);

 %%
 % temporary initial value calculation for workspace three

 %%
 updated_value = initial_value;
 visited_nodes = zeros(1,len);

 TOTAL_VALUE = 0;

while (n > 0)% && contamination < 1)% iterations < 30)

   % A; % adjacency matrix
    possible_nodes; % next possible nodes movements from current node
    prev_node = current_node;
    n_check = 0;
    prevD = D; % previous node to calculate next nodes probabilty

    % current node to calculate next nodes probabilty
% next possible actions finder
     delete(h1text)
     clear handles
    h1text = text(5,235,sprintf('Current Node = %.0f',current_node));

    vis_node = nonzeros(B(node,:));
    visible_node = (vis_node)';

    delete(h2text)
    h2text = text(28,225,sprintf(' %.0f',visible_node));

    modified_A = newA(visible_node,A);
    power = length(D);
    modified_A = (modified_A)^power;

newDD = D*modified_A;
D = newD(newDD);
%D;
n = nnz(D);  % find number of non zero elements
new_D = D;

running_node_color(current_node,visible_node,new_D,workspace_no); % changes the nodes
color during running of code

contamination = cont_check(prevD,new_D);
possible_nodes = nonzeros(C(node,:));
possible_nodes = (possible_nodes)';
%node_probabilty = prob_check(prev_node,current_node,D,possible_nodes,visible_node);

i = i+1;

node_track(1,i) = node;
iterations = iterations+1;
```

```
 while(n_check <= 0 && n>0)
     % possible_nodes;
     %  node_probabilty;

       possible_actions = possible_actions_finder(current_node,C(current_node,:));
       leng = length(possible_nodes);
       nodes_with_values = zeros(1,leng);

       for act_val = 1: leng

           nodes_with_values(1,act_val) = updated_value(1,possible_nodes(1,act_val));
       end

      % nodes_with_values_vector = [possible_nodes; nodes_with_values];

       delete(h5text)
       delete(h6text)
       h5text = text(210,180,sprintf(' %.0f  ',possible_nodes));
       h6text = text(210,172,sprintf(' %.0f   ',nodes_with_values));

     %  prompt={'Enter Choice:'};
% titled = ('Enter action no');
 %answerd = inputdlg(prompt,titled);
% current_action = str2double(answerd{1});
                                  % node = input(' please enter the current node = ');
                                   % [value, position] = max(node_probabilty);
                                     % node = possible_nodes(:,position)



    current_CCC = C(current_node,:);
    current_CCC(:,1)=[];
    current_CCC = nonzeros(current_CCC)';

    possible_actions = nonzeros(possible_actions)';

    non = length(possible_actions);
    VALUES_OF_NODES = zeros(1,non);
```

```
        for act_v = 1:non

        VALUES_OF_NODES(1,act_v) = updated_value(1,current_CCC(1,act_v));
        end

        M = max(VALUES_OF_NODES);
        MAX_POSITION = (VALUES_OF_NODES == M);
        ACTION = possible_actions(1,MAX_POSITION);

        current_action = ACTION;

   % pause(0.50)

   n_check = node_check(current_action,possible_actions);
   if (n_check <= 0)

     valid1 = text(80,247,'Invalid Action Selected');
     pause(1);
         delete(valid1)
   else
   end
end

   row_of_C  = C(prev_node,:);
   node = row_of_C(1,current_action+1);
   current_node = node;

   prev_value = updated_value;
  % visited_nodes(1,prev_node) = 1;

   [visited_nodes,updated_value,TOTAL_VALUE] =
U_value(prev_node,current_node,prev_value,visited_nodes,Hard_nodes,TOTAL_VALUE);

   %updated_value(1,4)=-50;

   updated_value = contaminated_value(updated_value,new_D,visited_nodes,Hard_nodes);
%    visited_nodes
```

```
   % TOTAL_VALUE;
    % waits for Button press and then resumes the code

 w = waitforbuttonpress;
    end
node_track = nonzeros(node_track)';
%iterations;

Penalty = penalty_calc_mex(iterations);
TOTAL_VALUE = TOTAL_VALUE - (Penalty);
Generated_policy = node_track;

fprintf('No of iterations = %d \n',iterations);
fprintf('TOTAL VALUE = %d \n',TOTAL_VALUE);
fprintf('Final generated policy = \n %s \n',num2str(Generated_policy));

GG = length(node_track);
XX = linspace(1,GG,GG);

hold all

figure(2);

plot(XX,node_track,'g')
hold on
plot(XX,node_track,'r*')
xlabel('Iterations')
ylabel('Node')

% STOPPING TIMER
Toc
```

```
function initial_value = i_value(first_node,Hard_nodes,A_Decimal)


% Current_hard_check = ismember(first_node,Hard_nodes);
% if isequal(Current_hard_check,0)
%
%     Hard_nodes = [first_node Hard_nodes];
% else
% end

LO_length = length(A_Decimal);
H_N_L = length(Hard_nodes);

assigned_check = zeros(1,LO_length);
node_value = zeros(1,LO_length);
initial_vector = find(Hard_nodes == first_node);

b = nnz(assigned_check);

   for i = 1:H_N_L
      if (Hard_nodes(1,i) == first_node)

         node_value(1,Hard_nodes(1,i)) = 0;
         assigned_check(1,Hard_nodes(1,i)) = Hard_nodes(1,i);
      else
         node_value(1,Hard_nodes(1,i)) = -100;
         assigned_check(1,Hard_nodes(1,i)) = Hard_nodes(1,i);
      end
   end
   last_updated = Hard_nodes;
   b = nnz(assigned_check);

while(b<LO_length)
   for j = 1:H_N_L

      next_row = last_updated(1,j);
      next_cand_row = nonzeros(A_Decimal(next_row,:))';
      % remove_previously updated
      PPP = next_cand_row == last_updated(1,j);
      next_cand_row(1,PPP) = 0;
```

```
        run = length(next_cand_row);
        for k = 1:run

          T  = ismember(next_cand_row(1,k),assigned_check);

            if(T == 0) && (j == initial_vector) && (b<LO_length)

        %     if(T == 0 && isequal(j,initial_vector) && b<LO_length)
                    node_value(1,next_cand_row(1,k)) = node_value(1,last_updated(1,j))+(7+k);
                    assigned_check(1,next_cand_row(1,k))= next_cand_row(1,k);
                    last_updated(1,j) = next_cand_row(1,k);
                    b = nnz(assigned_check);

            else if(T == 0) && (j ~= initial_vector) && (b<LO_length)

           % else if(T == 0 && isequal(j,~initial_vector) && b<LO_length)
                    node_value(1,next_cand_row(1,k)) = abs(node_value(1,last_updated(1,j)))-(7+k);
                    assigned_check(1,next_cand_row(1,k))= next_cand_row(1,k);
                    last_updated(1,j) = next_cand_row(1,k);
                    b = nnz(assigned_check);
                else
                end
            end
        end
    end

    b = nnz(assigned_check);

end

initial_value = node_value;




function [visited_nodes,updated_value,TOTAL_VALUE] =
U_value(prev_node,current_node,prev_value,visited_nodes,Hard_nodes,TOTAL_VALUE)

visited_nodes(1,prev_node) = prev_node;
```

```
%visited_nodes(1,current_node) = 1;

lengthh = length(prev_value);

for i = 1:lengthh

    if (visited_nodes(1,i) ~= 0 && current_node ~= i && prev_node ~= i)
        updated_value(1,i) = prev_value(1,i)+4;

%     else if (visited_nodes(1,i) ~= 0 && new_D(1,i) ~= 0)
%
%             updated_value(1,i) = initial_value(1,i);

        else if (current_node == i) % && visited_nodes(1,i) == 1)

               updated_value(1,i) = 0;
               TOTAL_VALUE = TOTAL_VALUE + prev_value(1,i);

            else if (visited_nodes(1,i) ~= 0 && prev_node == i)

                    updated_value(1,i) = 4;

                else
                             updated_value(1,i) = prev_value(1,i);
                end
            end
       % end
    end
end

[bob,Capital_L] = size(Hard_nodes);

for L = 1:Capital_L

    updated_value(1,Hard_nodes(1,L)) = -100;

end

updated_value;
visited_nodes;
```

```
function node_color(workspace_no)

X1X = [0;20;20;60;20;40;60;40;80;20;40;60;80;20];
Y1Y = [0;0;20;20;40;40;40;60;60;80;80;80;80;100];
ZX1Z = [20;20;20;20;20;20;20;20;20;20;20;20;20;20];
ZY1Z = [20;20;20;20;20;20;20;20;20;20;20;20;20;20];

X2X = [0;20;40;80;100;0;40;100;0;20;100;0;60;100;0;20;60;80;100];
Y2Y = [0;0;0;0;0;20;20;20;40;40;40;60;70;60;80;80;80;80;80];
ZX2Z = [20;20;20;20;20;20;20;20;20;80;20;20;20;20;20;20;20;20;20];
ZY2Z = [20;20;20;20;20;20;10;20;20;20;20;20;10;20;20;20;20;20;20];

X3X =
[0;20;40;60;80;100;120;140;180;0;40;80;120;180;140;160;160;100;120;140;160;120;140;160;1
80;120;140;180;120;180];
Y3Y =
[0;0;0;0;0;0;0;0;0;20;20;20;20;20;40;40;60;100;100;100;100;120;120;120;120;140;140;140;180;
180];
ZX3Z =
[20;20;20;20;20;20;20;40;20;20;20;20;20;20;20;20;20;20;20;20;20;20;20;20;20;20;20;20;60;20]
;
ZY3Z =
[20;20;20;20;20;20;20;20;20;180;20;20;20;100;20;20;40;20;20;20;20;20;20;20;20;20;20;40;20;2
0];

if (workspace_no == 1)

   X1 = X1X;
   Y1 = Y1Y;
   ZX = ZX1Z;
   ZY = ZY1Z;


   rectangle('Position',[62,62,16,16],'FaceColor','r')
text(61,70,'Unreachable','FontSize',11);
```

```
else if (workspace_no == 2)

        X1 = X2X;
        Y1 = Y2Y;
        ZX = ZX2Z;
        ZY = ZY2Z;
    else if (workspace_no == 3)

        X1 = X3X;
        Y1 = Y3Y;
        ZX = ZX3Z;
        ZY = ZY3Z;
        else
        end
    end
end

K = length(X1);

for i = 1:K

    X = X1(i,1);
    Y = Y1(i,1);
    Z1 = ZX(i,1);
    Z2 = ZY(i,1);

    hold all
  rectangle('Position',[X,Y,Z1,Z2],'FaceColor','r','LineWidth',2)
    end
text(235,240,'Dirty','BackgroundColor',[0.99,0,0],'FontSize',13)
text(235,230,'Clear','BackgroundColor',[0,0.99,0],'FontSize',13)
text(235,220,'Current','BackgroundColor',[0,0,0.99],'FontSize',13)
text(235,210,'Visible','BackgroundColor',[0.9,0.9,0],'FontSize',13)
```

```
function [P1,P2,P3,P4] = trans_prob(Actions,C)

disp('1 = North, 2 = South, 3 = East, 4 = West ');
Actions;

[m,n] = size(C);
P1 = zeros(m,m);
P2 = zeros(m,m);
P3 = zeros(m,m);
P4 = zeros(m,m);

for Action = 1:4

    for i = 1:m

        node = C(i,1);
        action = Action;
        row = C(i,:);
        backup_row = row;

        prob = probab_check(node,action,row);

        for mm = 1:5

   if (prob(1,mm) == 0)
        row(1,mm) = 0  ;
   else
     row(1,mm) = row(1,mm);
   end
        end
   row1 = (row);

   for nn = 1:5

    if (row1(1,nn) == 0 && prob(1,nn) ~= 0)
     prob(1,1) = prob(1,1) + prob(1,nn);
     prob(1,nn) = 0;
     row1(1,1) = backup_row(1,1);
       else
           prob(1,nn) = prob(1,nn);
    end
  end
   prob = prob;


prob = nonzeros(prob);
prob = (prob)';
row1 = nonzeros(row1);
row1 = (row1)';

tut = length(prob);

for mun = 1:tut

    ji = row1(1,mun);

    if (Action == 1)
    P1(ji,i) = prob(1,mun);
    else if (Action == 2)
            P2(ji,i) = prob(1,mun);
        else if(Action == 3)
                P3(ji,i) = prob(1,mun);
                else if(Action == 4)
                        P4(ji,i) = prob(1,mun);
                    else
                    end
            end
        end
    end
end
   end
   end


end
```

Completion Certificate

It is to certify that the thesis titled "***Target detection and interception using modified POMDP based motion planning of mobile robots in a known environment***" submitted by Registration no. NUST201261232MCEME35512F, NS Ehsan Elahi Bashir of MS-74 Mechatronics Engineering is complete in all respects as per the requirements of Main Office, NUST (Exam branch).

Supervisor: _____________

Lt. Col. Dr. Kunwar Faraz Ahmed Khan

Date: ____ July, 2015