Society for Science and Education United Kingdom
TMLAI Transactions on Machine Learning and Artificial Intelligence
Volume 3 Issue 3
ISSN 2054-7390

Probabilistic Search and Pursuit Evasion on a Graph

[1]E. Ehsan and [2]F. Kunwar
National University of Sciences and Technology Islamabad Pakistan
[1]ehsan74@mts.ceme.edu.pk; [2]kfaraz@gmail.com

ABSTRACT

This paper presents an approach to locate an adversarial, mobile evader in an indoor environment using motion planning of mobile pursuers. The approach presented in this paper uses motion planning of mobile robots to search a target in a graph and clear the workspace. The algorithm used is Partially Observable Markov Decision Process (POMDP), a probabilistic search method to clear the indoor workspace in a pursuit evasion domain. In this paper, 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.

1 Introduction

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 Parsons [1] proposed the pursuit-evasion games in a graph. 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.

This paper presents an approach to find an adversarial target using probabilistic search algorithm. The task is to capture the evader by clearing a graph with guarantee and keeping the cost, and capture time as low as possible.

2 Related Work

One of the basic games that take place on graph is the cops and robber game. In this game cops (pursuers) try to capture a robber (evader) along the vertices of a graph[2][3][4]. 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 [5]. In this approach evader is considered to be having a local visibility or i-visibility of the environment. 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

DOI: 10.14738/tmlai.33.1313
Publication Date: 27th June, 2015
URL: http://dx.doi.org/10.14738/tmlai.33.1313

pursuers is polynomial in the vertices of graph. It was also shown in [5] that when the evader has 2-visibility, the pursuers number becomes unbounded. One approach of worst-case pursuit-evasion is to consider evader as adversarial target having an infinite speed, as compared to slow pursuers. The search number was defined in [1] by T.D. Parsons as the minimum number of pursuers necessary for capture. Determining the search number of a graph was found to be NP-Hard [6], and to be NP-Complete due to monotonicity of optimal edge search schedules [7][8][2]. In [9] 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.

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 [10] that a pursuer can catch a non-deterministic evader in any simply-connected polygonal environment using a randomized strategy. In [11] 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. A visibility based pursuit-evasion was proposed in [12][13][14]. A visibility based pursuit-evasion in a polygonal environment was proposed in [15]. The evader is considered to be adversarial, with unknown initial and current position. A guaranteed search with spanning trees is proposed in [16], an anytime algorithm for multi-robot search. The proposed GSST algorithm clears the environment of any adversarial evader using fewest number of pursuers. The NP-hard problem on arbitrary graph can be solved using spanning trees in linear time. In [17] a group of mobile searchers have to find mobile evaders in a polygonal region. Upper and lower case bounds on search number of polygon are also discussed.

Unlike traditional search strategies that try to find a guaranteed solution for a graph or a polygonal environment, the probabilistic search methods consider optimization of the expected value of a search objective, such as maximal probability of detection or minimal time to capture. Many probabilistic search methods have been proposed so far. A mixed observability based robotic tasks planning algorithm under uncertainty is proposed in [18]. 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[2].

Many search problems can be formulated as a Markov Decision Process (MDP) if the target's position is known [19] or a Partially Observable Markov Decision Process (POMDP) if it is unknown. In [20] it is shown that how belief compression can be used to make a POMDP search problem tractable for a single pursuer. This approach fails when the team size scales up or the state space is increased. The POMDP search problem can be searched with two pursuers using mixed observability (e.g. when the pursuer's position is fully known but the evader's position is unknown) and the Markovian target motion model is given, as shown in [18]. Most of the probabilistic search models assume the target to be non-adversarial and the search environment to be known beforehand. Most if not all the probabilistic search techniques suffer with scalability issues as the team size scales up or the environment becomes more complex.

From the above discussion it is evident that the graph search algorithms and polygon search algorithms are greedy in nature and guarantee the capture of an adversarial evader in finite time [2], but they need a lot of resources and incur a high cost. They are not feasible for higher dimensional environments. On the other hand the probabilistic algorithms either need a motion model of the evader or they incur a high cost in finding the evader, probabilistic algorithms also do not guarantee the capture. Another problem with these algorithms is that they are complicated and become hard to implement as the team size scales up or the environment or graph becomes more complex. Such solutions sacrifice completeness over cost minimization. They may or may not find an evader, even if there exists one.

This paper presents an approach to find an adversarial evader using a modified form of the Partially Observable Markov Decision Process (POMDP) to minimize the overall cost, while guaranteeing capture.

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 [9].

Consider the map of Fig.1.a. This map can be converted to a graph using discretization, as shown in Fig.1.b.

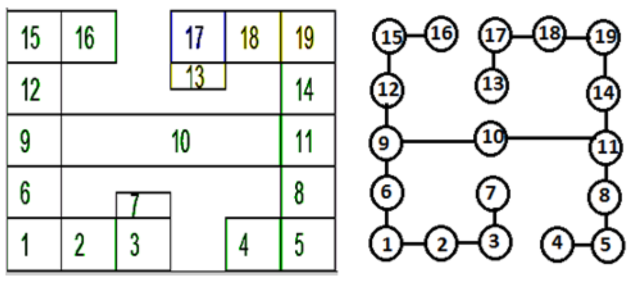
Some steps to convert the polygonal indoor environment to a graph are as following:

1. The indoor environment is discretized into cells, each of these cells correspond to a node of an (*undirected*) *navigation graph*. The discretization takes place on *critical visibility events* [9].
2. A (*directed*) *information space* is obtained from the *navigation graph*. The information graph is a *state transition diagram* of the search process [9].
3. Search takes place in the information graph to find a path from a starting state to a terminal or clear state [9].

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.

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



(a) (b)

Figure 1. Conversion of a map to graph (a) a map of an indoor environment (b) a graph of 1.a.[9]

N (n) = {*m* ∈ *V*: (*n, m*) ∈ *E*}.

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

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

The adjacency matrix of graph in Fig.1.b is

$$A=$$

$$\begin{bmatrix}
1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&1&1&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&1&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1
\end{bmatrix}$$

(1)

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

V (n) = {*m* ∈ *V*: *m* 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

$$C_{mn} = \begin{cases} 1 \; iff \;\; n \in V\,(m); \\ \qquad 0 \; else. \end{cases}$$

The visibility matrix of graph in Fig.1.b 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 Fig.1.b is given as

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

(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 falls inside the pursuer's *visibility region* and remains *clear* until there is no free (pursuer's *visibility free*) path from it to a dirty node. The node is re-contaminated if there is a pursuer's *visibility free* path from it to a dirty 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* ∈ *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.

Suppose for a moment that the pursuer is removed from the graph and the evader moves with unit speed. Then, the possible locations of the evader at time t +1 are the ones adjacent to its possible locations at time t. Mathematically this is expressed by [9].

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

If the evader has speed M (it crosses M edges in a single time step) then instead of (4) we have

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

4 The Search Algorithm

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)[21][22]. 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 zin= (xin; din). The pursuer must move through nodes in a manner such that the 1's in the d part progressively decrease (the dirty set shrinks) until eventually the pursuer reaches one of the clear states, say zfin= (xfin; dfin), where d fin= 0.

As the given graph in Fig.1.b contains 19 nodes, a dirty node vector of 14 elements is defined as **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

E = [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.

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 paper 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.

5 Implementation

Potential advantages of the proposed algorithm are highlighted through its implementation. Consider the graph illustrated in Fig.1.a. A single pursuer is used to clear the graph. The graph is composed of 19 nodes. First of all hard nodes are determined, after that values are assigned to these and remaining nodes of the graph. After initial values have been assigned the pursuer starts its search for the evader. At each step, the values of nodes are updated in order to facilitate pursuer in its search for the evader. If a node is re-contaminated it is assigned a higher value, so that pursuer has to clear it again.

From the graph of Fig.1.b it can be seen that the hard nodes are 4, 7, 13 and 16.

H = [4 7 13 16]

If initial node selected is 7, then the policy generated is given as:

Π = [7 3 2 1 6 9 12 15 12 9 10 11 8 5 8 11 14 19 18 17]

This policy is exactly the same as was generated in [9] for the same graph using same initial node.

If initial node is 4, the generated policy is given as:

Π = [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 initial node is 13, the generated policy is given as

Π = [13 17 18 19 14 11 8 5 8 11 10 9 12 15 12 9 6 1 2 3]

If initial node is 16, the generated policy is given as

Π = [16 15 12 9 6 1 2 3 2 1 6 9 12 15 12 9 10 11 8 5 8 11 14 19 18 17]

The above generated policies show the effectiveness of the proposed algorithm in finding a solution to the problem with certainty and in minimum steps possible. The algorithm successfully generated the capture policies for multiple initial nodes, as compared to [9] which was considered as rooted and was used only for one initial node. The algorithm returns a policy only when the graph is cleared and any evader in the graph is successfully captured, or returns a clear graph if there is no evader found in the graph. The proposed Algorithm assigns values to the nodes on the basis of probability of presence of evader in them. The policy generated by the algorithm is in just 20 iterations for node 7 as initial node, which is in fact the minimum no of steps for any algorithm to clear the graph, 26 iterations for node 4, 20 for node 13 and 26 for node 16. All of these generated policies are generated in minimum possible iterations while considering re-contamination for any algorithm.

6 CONCLUSION

A probabilistic algorithm is proposed that uses value iteration and policy iteration concept of POMDP in finding an adversarial evader in a graph in minimum steps possible while guaranteeing capture by maximizing the overall value.

ACKNOWLEDGMENT

We would like to thank Allah Almighty. Our parents for their prayers. Mr. Mansoor Ghazi for his guidance in writing this paper, Mr. Saad Farooq, Mr. Ahsan Gull and Mr. Adnan for their motivation. NUST College of E&ME Pakistan for providing a research oriented environment.

REFRENCES

[1] T. D. Parsons, “Pursuit-Evasion in a Graph,” vol. 642, 1978.

[2] 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.

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

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

[5] 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.

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

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

[11] 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.

[12] 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.

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

[14] 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.

[15] L. Guibas, J.-C. Latombe, S. Lavalle, D. Lin, and R. Motwani, “Visibility-based pursuit-evasion in a polygonal environment,” *Algorithms Data Struct.*, pp. 17–30, 1997.

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

[17] 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.”

[18] 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.

[19] J. H. Eaton and L. A. Zadeh, “Optimal Pursuit Strategies in Discrete-State Probabilistic Systems,” *J. Basic Eng.*, vol. 84, no. 1, p. 23, Mar. 1962.

[20] N. Roy, G. Gordon, and S. Thrun, “Finding approximate POMDP solutions through belief compression,” *J. Artif. Intell. Res.*, vol. 23, pp. 1–40, 2005.

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

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