Detecting a Stationary Target on an Undirected Network

Anusman Panda
109CS0084
Manjeet Singh Harsh
109CS0593

Department of Computer Science and Engineering
National Institute of Technology
Rourkela-769 008, Odisha, India
May 2013

Detecting a Stationary Target on an Undirected Network

Thesis submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Anusman Panda
[Roll: 109CS0084]

Manjeet Singh Harsh
[Roll: 109CS0593]

under the guidance of

Prof. S.K. Rath
NIT Rourkela

Department of Computer Science and Engineering
National Institute of Technology
Rourkela-769 008, Odisha, India
Academic Year: 2012-13

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

May 13, 2013

Certificate

This is to certify that the work in the thesis entitled ***Detecting a Stationary Target on an Undirected Network*** by ***Anusman Panda and Manjeet Singh Harsh*** is a record of an original research work carried out under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Prof. S. K. Rath
Professor
CSE Department of NIT Rourkela

Acknowledgment

We express our sincere and heartfelt gratitude towards our guide Prof. S.K. Rath for his expert guidance and motivation during the course of the project which served as a spur to keep the work on schedule.

We convey our regards to all the other faculty members of Department of Computer Science and Engineering, NIT Rourkela for their valuable guidance and advices at appropriate times. Finally, we would like to thank our friends for their help and assistance all through this project.

(Anusman Panda)

(Manjeet Singh Harsh)

Abstract

Search processes are widely used in the field of Computer Science. Many well defined algorithms also have been established to perform search in various areas of computer science. The motive behind all the search process is to perform search in efficient manner, either time efficient or space efficient. These well-defined algorithms do not always lead in a polynomial solution for an arbitrary problem. Sometimes there are NP-hard problems which are very tough to solve using standard algorithms. In this project, an attempt is made to search a uniformly distributed and single immobile entity on an undirected network. The objective here is to search for the entity in a minimum expected search time with no limit on the length of the search. An immobile entity can be defined as one which cannot move. Its position is stationary throughout the search process. The network is represented by a graph. A methodology is proposed to carry out the search process in minimum possible time resulting in optimal solution.

Keywords: Search Process; NP-hard; Immobile entity.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Search processes are widely used in the field of Computer Science. For every now and then there is a need to search for one thing or another, be it in real world or in the domain of computer networks. There are numerous algorithms defined to carry out the search process. The main objective of these algorithms is to carry out the search process in an efficient manner. Some deal with efficiency pertaining to space while some deal with minimizing the time complexity.

An immobile entity can be defined as an entity that cannot move. The position of such type of entity remains stationary throughout the time period of search [1]. A graph G = (V,E) is used for representing the network where E denotes the set of edges or nodes, V denotes the set of vertices or lines and l_i , i $\in$ E represent the length of an edge in the network. The network in reference is an undirected network i.e. the graph which is used to represent the network is undirected. An undirected graph is one in which the edges do not have any orientation. The edge (x,y) is same as that of (y,x).

Some of the assumptions made to carry out the search process are [1]:

- To find the entity, both the searcher and the entity should be in the same place.
- The entity does not have any information about the search strategy which the searcher is going to follow to search it.
- The searcher does not know the whereabouts of the entity prior to the search process.
- There is no connection between the entity and the searcher before or during the search process.

The required search time is the collective sum of the time which the searcher takes to move from its current position to a specific point of entry on the network plus the time it then takes to find the entity in the network once it reaches the network. As the searcher has no idea or information about the location of the entity in the network, it might be required to cover the whole network at least once covering each and every edge of the network.

1.2 Motivation

The research in search problem has been one of the most popular research directions in search theory. In case of emergency situations like natural calamities, disasters etc. there is an urgency of dispatching emergency vehicles [2] to the particular location as soon as possible so as to avoid the loss of life and property. Some real life situations could be an injured person stuck under the remains of a damaged or a building, collapsed due to earthquake or any other cause. During military operations the mines placed by the enemy to create destruction could also be a scenario of finding an immobile entity where the mine is the entity whose location is fixed [1].

In all the above cases, the aim is to find a target object. There are existing approaches which consume a lot of computational cost, hence we explore a simple yet effective method to search an immobile entity in a minimized time.

1.3 Objective

The objective of the project is to develop a search strategy as to minimize the expected search time to find the entity present in the network provided the entity is uniformly distributed. The expected search time is minimum or optimal if we traverse the network via Eulerian tour during the search process. Thus our main objective is to obtain an Eulerian tour.

There are two possible scenarios:

- Graph is Eulerian- we obtain an Eulerian tour that gives the optimal solution
- Graph is not Eulerian- we change the graph into Eulerian graph following a strategy for obtaining the optimal solution.

There is no restriction on the length of the path to be traversed. Only objective is to minimize the search time without taking into account the path length during the search process.

1.4 Organization of the thesis

The thesis is organized as follows.

In chapter 2, we have given the literature survey that is focused on different works that has already been undertaken in the field of finding an immobile entity on the network.

In chapter 3, we have proposed our search strategy to find the immobile entity on the network.

In chapter 4, we give the implementation details of the algorithm and the result.

In chapter 5, we give the concluding remarks with a focus on future research directions that could be undertaken.

Chapter 2

Literature Survey

The amount of work that has been undertaken till date on the field for searching an immobile entity on a network is very few.

2.1 Review of Jotshi et.al. Work

Arun Jotshi and Rajan Batta proposed an algorithm to search an immobile entity on a network [1].

2.1.1 Search Theory

During World War II rigorous work was carried out in the field of Operation Research taking search theory as a discipline. After the World War II was over, the work in respect to Search theory took a new form. After World War II, Search theory developed as a separate field of interest to many.

Depending on the motives of the target, there are two possible scenarios:

1. The target has motive of either being found out or not being found at all.
2. The target is either static, hidden according to some known distribution or the target is mobile and its motion can be determined stochastically by some known rules.

Our case of consideration is that of an immobile hider.

2.1.2 Search Games

Search games are two person games. Such type of games follow the rule that neither the searcher nor the hider have any idea regarding the movement of one another unless and until both of them are in close proximity of one another or their distance of separation is according to some pre-defined conditions. The aim of the searcher is to capture the target in minimum time by travelling with maximum permissible speed.

Gal [3] proved that the value of a search game is equal to half the sum of edge lengths of a network G(V,E) for an immobile hider having a specified start point

if and only if the Graph G is Eulerian. Some modified version of Chinese Postman Problem (CPP) strategy [4] has to be used if the graph is not Eulerian.

2.1.3 Definitions

Definition 1:

For a given network G(V,E), P denotes a path that covers all the edges of the network at least once. The sequence of edges on path P is represented by $E^{'}$(P). Some edges might be covered more than once so as to cover the whole network. The probability of finding an entity on an edge that has been traversed more than once is zero [5]. The reason is if we are not able to find the entity on our first visit there is no chance that we will find it next traversals.

Definition 2:

An *Eulerian tour* is an Eulerian trail that starts and ends at the same vertex. An Eulerian tour is a cycle that uses each edge exactly once. Any Eulerian tour gives an optimal solution if the graph is Eulerian [1].

A graph G(V, E) is Eulerian if all its vertices have even degree, and weakly Eulerian when the graph consists of Eulerian components connected by a tree structure.

An *Euler path* is defined as a walk on the edges of a graph that uses each edge exactly once. A connected graph has an Eulerian trail if and only if it has at most two vertices of odd degree. These two vertices are the start and end vertices of the trail. For non-Eulerian graphs that have odd degree vertices greater than two, we first convert the graph into Eulerian and then find an Eulerian path. The conversion is done by adding edges between certain pairs of odd degree vertices so that their degree becomes even and the number of odd degree vertices remains two [1].

2.1.4 Minimum Weight Perfect Matching

A non-Eulerian graph is converted into an Eulerian by performing a minimum weight perfect matching. A matching M is defined as a set of edges of a graph

such that no two of them share a common vertex. This M is added to the graph G(V,E) to convert it into $G^{'}(V^{'}, E^{'})$, which is Eulerian. The given graph becomes Eulerian in the sense that on addition of these extra edge the degree of even degree vertices is increased by two and those of odd degree vertices is increased exactly by one making the odd degree vertices as even. But sometimes the minimum expected search time obtained due to minimum weight perfect matching is not always optimal.

2.1.5 Solution Strategy

Step 1:

Check for the graph to be Eulerian or not. If it is Eulerian, any Eulerian tour results in optimal solution.
Step 2:

If the graph is not Eulerian apply the following procedure [1]:

1. Obtain the graph $G^{'}(V^{'}, E^{'})$ from G(V,E) defined over the set of odd degree vertices. Initialize the matching M and then go to 2.
2. Apply kruskals algorithm [6] to obtain minimum spanning tree (MST). If the MST obtained is a Hamiltonian Path, then go to 5 else go to 3.
3. Delete all vertices with degree one from the set $V^{'}$. Obtain the Hamiltonian path.
4. Obtain the Open Eulerian Graph $G^{''}$ by adding matched edges to the original graph G. Go to 5.
5. Remove the edges belonging to the Hamiltonian path from the Open Eulerian graph $G^{''}$ to obtain graph $G^{'''}$ which is also an Open Eulerian Graph. Find an Eulerian trail on the generated graph which gives the optimal solution.

Chapter 3

Proposed Methodology

3.1 Approach

We represent the network as a graph and let that graph is G(V, E) where the entity is likely to be present. Here V represents the number of nods and E represents the number of edges in the graph. The searcher travels through a vertex and start the search process. When it finds the entity (target node) somewhere on network the search process stops.

The selection of start vertex (or node) to begin the search process in graph is polynomial, because the network has finite number of nodes (| V |). But the possible number of tours is not polynomial, even after we have a specific start vertex. The total number of possible tours for this abstract *problem*[1] goes very high (non-polynomial) if we count the tours for every possible start vertex (or for every instance of the problem). Hence we must limit the choice of start and also end vertex to decrease set of problem instances. Generally the choice of the possible start vertices is clear.

Our aim is to find the Eulerian trail because the Eulerian tour will be optimal. If the given graph is Eulerian then any vertex can be chosen as the start vertex, possibly choose vertex which is closest to the searcher. If the graph is open *Eulerian*[2], then the start vertex should be chosen an odd degree vertex. After selecting a proper start vertex we find a path/trail that traverses all the edges of network at least once (Eulerian trail).
We apply the following strategy or heuristic method.

[1] *Abstract Problem is a relation I X S where I denote the set of problem instances and S denotes set of solutions. For example while finding a shortest path in a given graph, then different graphs constitutes one instance of the problem. That is a particular graph from the given graph is an instance problem I and its corresponding solution is put in S.*

[2] *If the Eulerian graph has exactly two odd degree vertices then it is called an Open Eulerian graph. If all vertices in graph are even degree vertices then this graph is called a Close Eulerian graph or simply an Eulerian graph.*

3.2 Strategic Steps

Step 1:

First of all check the given graph whether it is Eulerian or not. If the graph is Eulerian then apply the Hierholzers Algorithm [7] presented in Section 3.3 to find the Eulerian trail which is optimal for this case.

If the graph is Open Eulerian then pick any of the two odd degree vertices as start vertex and find Eulerian trail by applying the above mentioned algorithm.

If the graph is not Eulerian then go to Step 2.

Step 2:

For non-Eulerian graph, apply the heuristic presented in Section 3.4 to obtain the Eulerian trail. This heuristic involves some sequencing steps which convert the non-Eulerian graph into the Eulerian one. Then go to step 3.

Step 3:

After getting the corresponding Eulerian graph by using step 3 we again go to Step 1 to find the Eulerian trail.

3.3 Hierholzers Algorithm when the graph is Eulerian

Step 1:

Take stack and an array called circuit and initialize them empty (Eulerian path).

- If the graph is closed Eulerian i.e. all vertices are even degree vertices then choose any one out of all nodes as start (current) vertex.
- If the graph is open Eulerian i.e. there are exactly 2 odd degree vertices then choose one of them as start vertex.
- Else there is no Euler circuit or trail exists.

Step 2:

- If all the edges of graph have not been traversed and the current vertex has no neighbors then
  - Add the current node to circuit.
  - Remove the last vertex from the stack and set it as the current vertex.
- Else (if the current node has adjacent vertices) then
  - Add this current vertex to the stack
  - Take any neighbor of this current vertex
  - Remove the edge between selected neighbor (adjacent vertex) and that vertex and set that neighbor as the current vertex.

Step 3:

Repeat step 2 until the current vertex has no adjacent vertices (connected nodes) and the stack is empty.

Note: The obtained circuit will be in reverse order i.e. it will be from end vertex to start vertex.
Complexity: O(M+N)
Where M is Number of edges and N is the Number of vertices.

3.3.1 Illustrative Example

This example explains the working of the Hierholzers algorithm on an Eulerian graph. We have a ten node sample network represented as an undirected graph as shown in the figure3.1. The graph in the figure is a closed Eulerian graph because each node is an even degree node (vertex). Hence Hierholzers algorithm is applicable on this graph.

Because this is a closed Eulerian graph any node can be selected as start point (vertex). Suppose we take vertex 8 as start point then the Eulerian path is:

Circuit: 8 10 9 7 8 6 7 2 1 6 5 4 3 5
Eulerian Path: 8 5 3 4 5 6 1 2 7 6 8 7 9 10

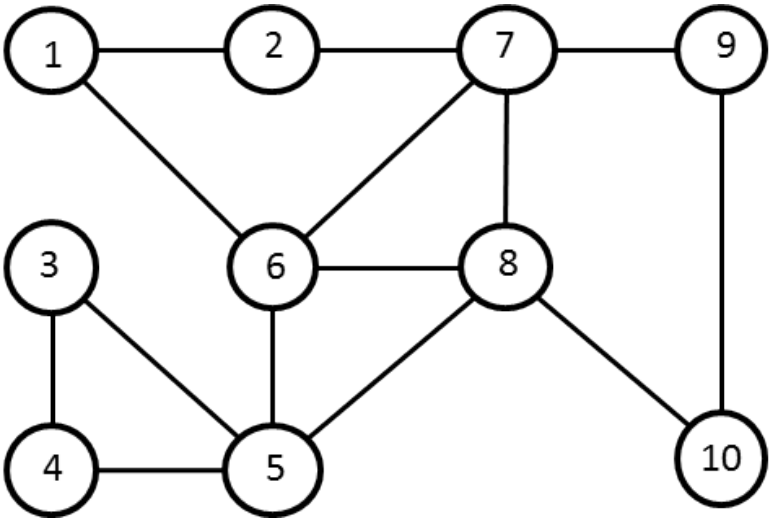

FIGURE 3.1: G(V, E)

A detailed step-by-step execution of the algorithm is shown below. Initially in the start state Stack and Circuit are empty and no vertex is chosen as current location. As the start point is selected vertex 8 the current location is updated as 8 and correspondingly the stack updates.

Stack: Empty
Current Location: None
Circuit: Empty

Stack: Empty
Current Location: 8
Circuit: Empty

Stack: 8
Current Location: 5
Circuit: Empty

Stack: 8 5
Current Location: 3
Circuit: Empty

Stack: 8 5 3
Current Location: 4
Circuit: Empty

Stack: 8 5 3 4
Current Location: 5
Circuit: Empty

Stack: 8 5 3 4 5
Current Location: 6
Circuit: Empty

Stack: 8 5 3 4 5 6
Current Location: 1
Circuit: Empty

Stack: 8 5 3 4 5 6 1
Current Location: 2
Circuit: Empty

Stack: 8 5 3 4 5 6 1 2
Current Location: 7
Circuit: Empty

Stack: 8 5 3 4 5 6 1 2 7
Current Location: 6
Circuit: Empty

Stack: 8 5 3 4 5 6 1 2 7 6
Current Location: 8
Circuit: Empty

Stack: 8 5 3 4 5 6 1 2 7 6 8
Current Location: 7
Circuit: Empty

Stack: 8 5 3 4 5 6 1 2 7 6 8 7
Current Location: 9
Circuit: Empty

Stack: 8 5 3 4 5 6 1 2 7 6 8 7 9
Current Location: 10
Circuit: Empty

Stack: 8 5 3 4 5 6 1 2 7 6 8 7 9 10
Current Location: None
Circuit: Empty

Either there are all vertex left isolated or the searcher follow the path which lead to current location as isolated vertex before traversing all the vertices in the graph. Then the Circuit updates its value from the stack as shown below.

Stack: 8 5 3 4 5 6 1 2 7 6 8 7 9
Current Location: 10
Circuit: 8

Stack: 8 5 3 4 5 6 1 2 7 6 8 7
Current Location: 9
Circuit: 8 10

Finally Stack becomes empty and the traversing vertices are stored in the array circuit in reverse order. The Eulerian trial is got from the circuits elements in reverse order.

Stack: Empty
Current Location: None
Circuit: 8 10 9 7 8 6 7 2 1 6 5 4 3 5

Each edge has been traversed exactly once but the nodes can be traversed more that once. The length of the trail is 13 to start travelling from any arbitrary vertex to its adjacent vertex by traversing all the edges exactly once. But the length of the complete tour is 14, which means the searcher reaches its initial vertex after traversing each edge exactly once throughout the complete graph.

3.4 Heuristic when input graph is non-Eulerian

We have an arbitrary network as input which is represented as graph G(V, E). This graph certainly has more than 3 odd degree nodes that is $\mid V \mid \geq 4$ where $V^{'} \subseteq V$.

The final result (output) consist an Eulerian trail and also proper selection of start vertex and end vertex.

Procedure:

Step 0:

Derive the graph $G^{'}(V^{'}, E^{'})$ from G(V, E) defined over the set of odd degree vertices $V^{'} \subseteq V$. Follow the below steps to find $G^{'}(V^{'}, E^{'})$

- Count all vertices with their number of adjacent vertices (neighbors)
- Apply Warshalls Algorithm [8] for shortest path evaluation
- Reduce the matrix size to the total number of odd degree vertices ($\mid V \mid$)

Now the graph $G^{'}(V^{'}, E^{'})$ is represented as weighted Adjacent matrix.

Step 1:

- Apply Kruskals Algorithm on graph$G^{'}(V^{'}, E^{'})$ to find Minimum Spanning Tree (MST).
- Let the edges of Minimum Spanning Tree be denoted by the set $E^{''}$ and nodes $V^{'}$.
- Now check whether the MST is Hamiltonian path or not?

If MST has vertices with degree greater than two then the path is not Hamiltonian. Else the path is Hamiltonian.

- If *Hamiltonian* then MST and modified MST are same and go to the last step 3.
- If *not Hamiltonian* then go to next step 2.

Step 2:

In MST mark all nodes with degree greater than two as V_1,V_2, ... ,V_m which leaves a set $V^{'''}$. Here m is the number of nodes having degree greater than two and $m < \mid V^{'} \mid$. Follow the below steps until the total cost of the tree is less than equal to the Minimum cost of the MST.

- Now delete the edges between a node from $V^{'''}$ and the node from $V^{'}$ having degree greater than equal to two.Also mark these nodes used from $V^{'}$.
- Insert the edges between the nodes from$V^{'}$ and the nodes adjacent to node of $V^{'''}$ without forming any cycle.
- If the degree of any node of $V^{'''}$ reduces less than three then remove that node from $V^{'''}$ and put that into $V^{'}$.
- Repeat steps until there is no node having the degree greater than two i.e. $\mid V^{'''} \mid = 0$

This step 2 results in a Hamiltonian path having all the vertices with degree less than equal to two over the set of vertices $V^{'}$. Thus we get modified MST. Go to step 3.

Step 3:

Insert extra edges in between suitable nodes as well as choose start and end vertices.

- In modified MST take one degree vertices and its adjacent vertices from the set $V^{'}$'. There will be exact two such pairs containing four vertices and two edges.
- Now compare the weights (cost) of edges between one degree vertex and its adjacent vertex.
- Mark the nodes as entry vertices (start and end vertices) which are connected with largest weighted edge.
- Among rest of the vertices in modified MST insert the extra edges in such a way that each nodes degree is incremented by one.

Step 4:

Update the graph G(V, E) according to the modified MST. By inserting the extra edges, the degree of each odd degree vertex of graph G(V, E) increments by one and as a result it is converted into even degree vertices. That makes the graph an Eulerian graph. Now apply Hierholzers Algorithm on the Eulerian graph described in the section 3.3 to obtain the optimal path.

3.4.1 Illustrative Example

General graph

We take a general graph to explain step-by-step implementation of the heuristic. An arbitrary graph with total 15 nodes is taken to perform the heuristic as shown in figure3.2. The edges of this graph are considered unit and equal weighted. We go through the various steps to obtain our potential result.

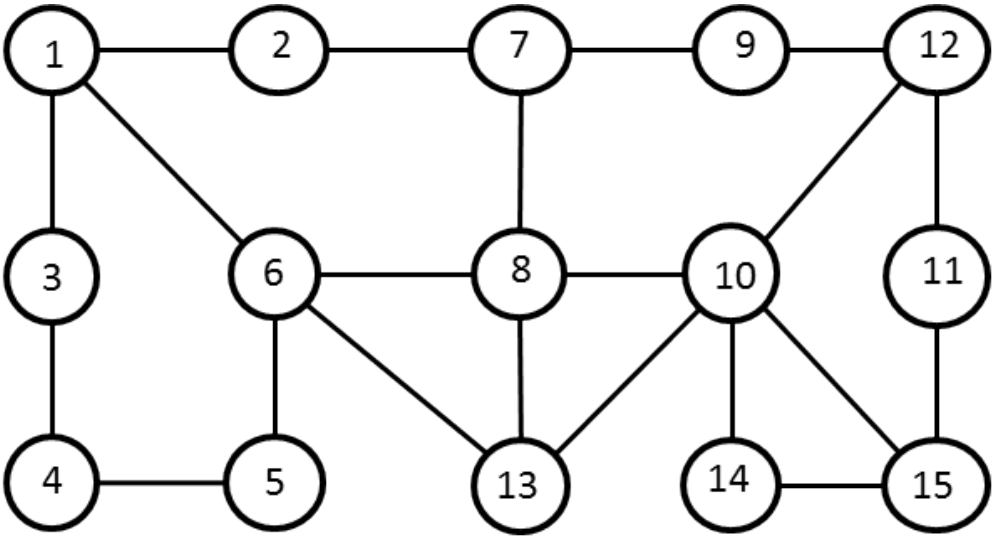

FIGURE 3.2: A general graph G(V, E) of 15 nodes

Step 0:

First extract the graph $G^{'}(V^{'}, E^{'})$ from above graph G(V, E) which is defined over odd degree vertices. Graph G(V, E) contains 6 odd degree vertices that is $\mid V^{'} \mid = 6$ and the odd degree vertices are $\mid V^{'} \mid = \{1,7,10,12,13,15\}$. Also the edges of the graph $G^{'}(V^{'}, E^{'})$ show the costs from every vertex to other vertex in $V^{'}$. The extracted graph $G^{'}(V^{'}, E^{'})$ is shown in the figure3.3.

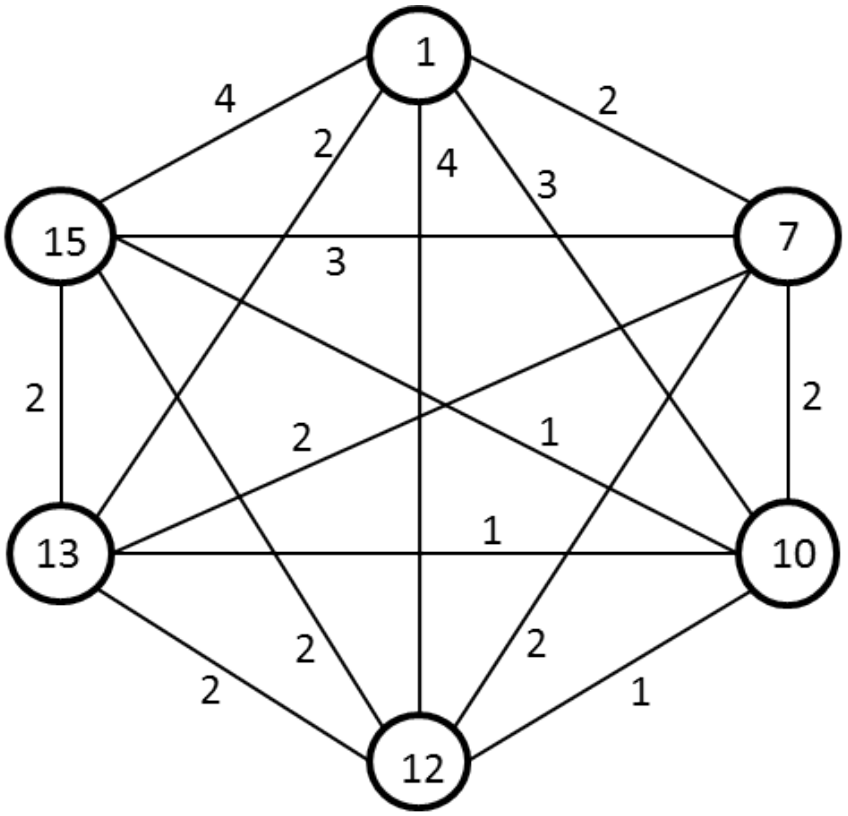

FIGURE 3.3: $G'(V', E')$ graphical form

Step 1:

Now derive the Minimum Spanning Tree (MST) from the graph $G'(V', E')$. The edges of the MST are represented as Tree as shown in the figure3.4. The minimum cost of the tree is 7. There is a vertex 10 which has degree greater than two. Hence we go to step 2 because the MST is not a Hamiltonian path. This has to be converted into Hamiltonian path by following the next step.

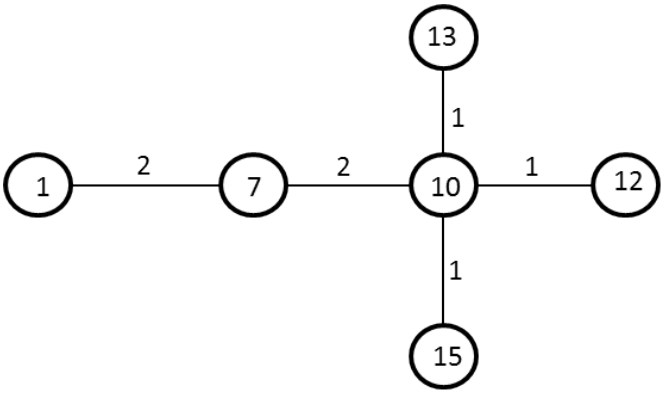

FIGURE 3.4: Minimum Spanning Tree (MST)

Step 2:

The vertex 10 (which has degree greater than 2) and its adjacent vertex having degree greater than one (vertex 7) are disconnected by removing the edge in between. The edge is inserted between the vertex 7 and the adjacent vertex of vertex 10. At last each vertex in MST has the degree less than equal to two in such a way that no cycle is formed. The resultant modified MST is shown in the figure3.5. Go to step 3.

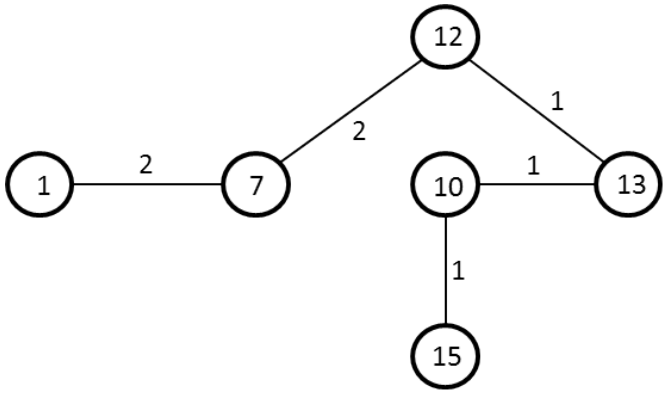

FIGURE 3.5: Modified MST (Hamiltonian path)

Step 3:

The vertex 1 and 7 are selected as entry vertices (start and end nodes) because the cost of edge between vertices 1 and 7 is greater than the cost of edge between vertices 10 and 15. And extra edges are inserted among rest of the vertices by keeping total cost of the tree less than equal to minimum cost (7). The insertion of extra edges among appropriate vertices is shown in figure3.6.

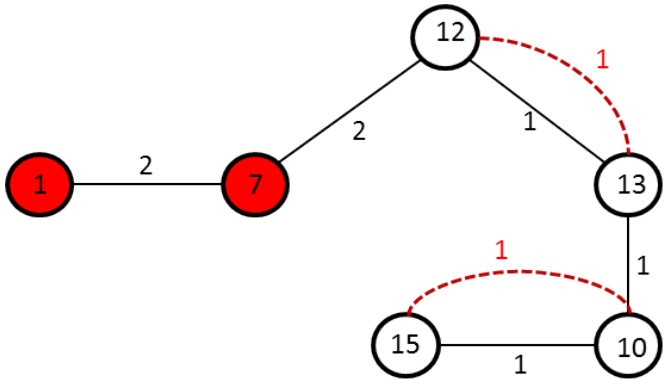

FIGURE 3.6: Extra edges insertion in modified MST

Step 4:

The above three steps have finalized the entry vertices (shown as red vertices) and the proper insertion of extra edges (red edges) into the graph to make it Eulerian. Because the forward and backward both the weights are equal, hence any vertex from the entry vertices can be chosen as start point. The length of the trail from both the sides is equal.

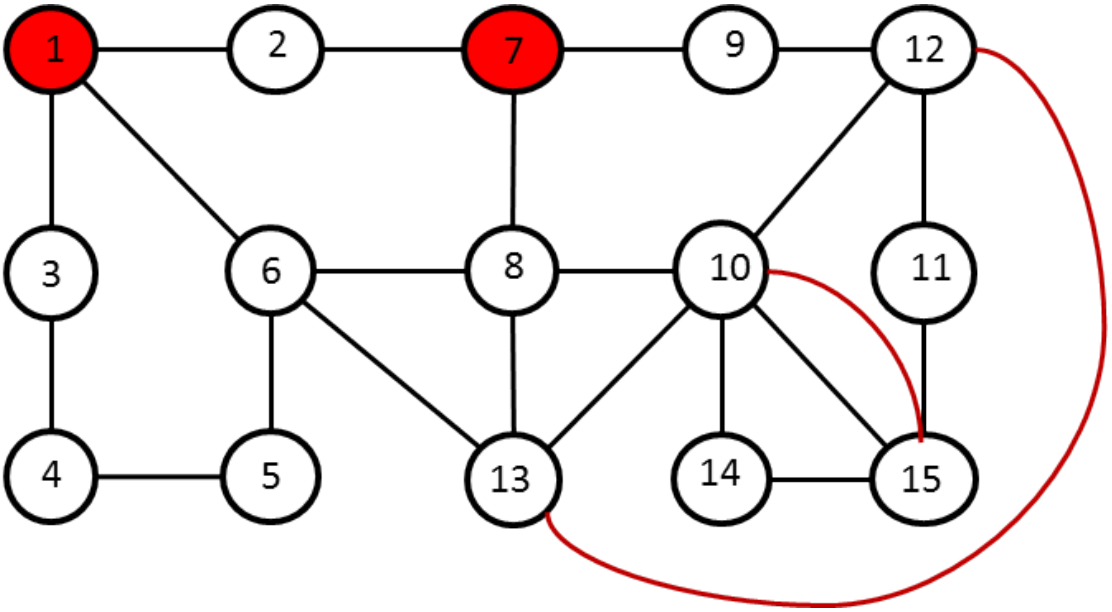

FIGURE 3.7: Modified G(V, E) after insertion of extra edges

The original graph G(V, E) had total 21 edges. After having inserted two extra edges the total length of the trail becomes 23. We have only two choices to select the start point, either vertex 1 or vertex 7. In program if we input a start node other than these two nodes then a wrong selection message prints. Hence we should select any one vertex from these two vertices so that we get our result that is an Eulerian trail. Also the length of the Eulerian trail from forward and backward directions is same. Hence the extra edges have been inserted which converted the graph into Eulerian.

The above mentioned steps denote our methodology to obtain the optimal solution.

Chapter 4

Implementation and Results

4.1 Setup

The implementation was done on a Lenovo Ideapad with core i5 processor having clock speed of 2.53Ghz and 4GB of Ram. The programs were written in *C* language which were run on Dev C++ program that uses GCC compiler, the GNU compiler collection

4.2 Input

The input was taken specifying the edges present between two nodes.

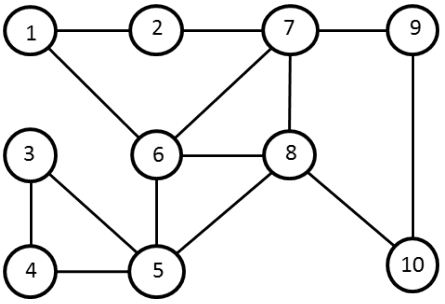

FIGURE 4.1: G(V, E)

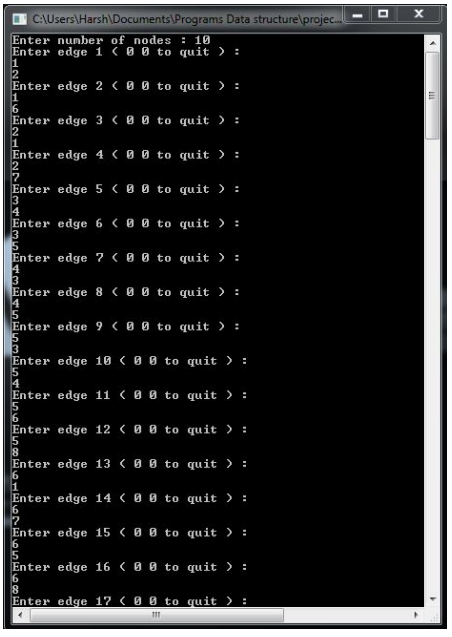

FIGURE 4.2: Sample Input for 10 nodes

The figure 4.2 shows the input for the graph shown in figure 4.1

This input was used to develop the adjacency matrix which served as the base input for carrying out the further computations.

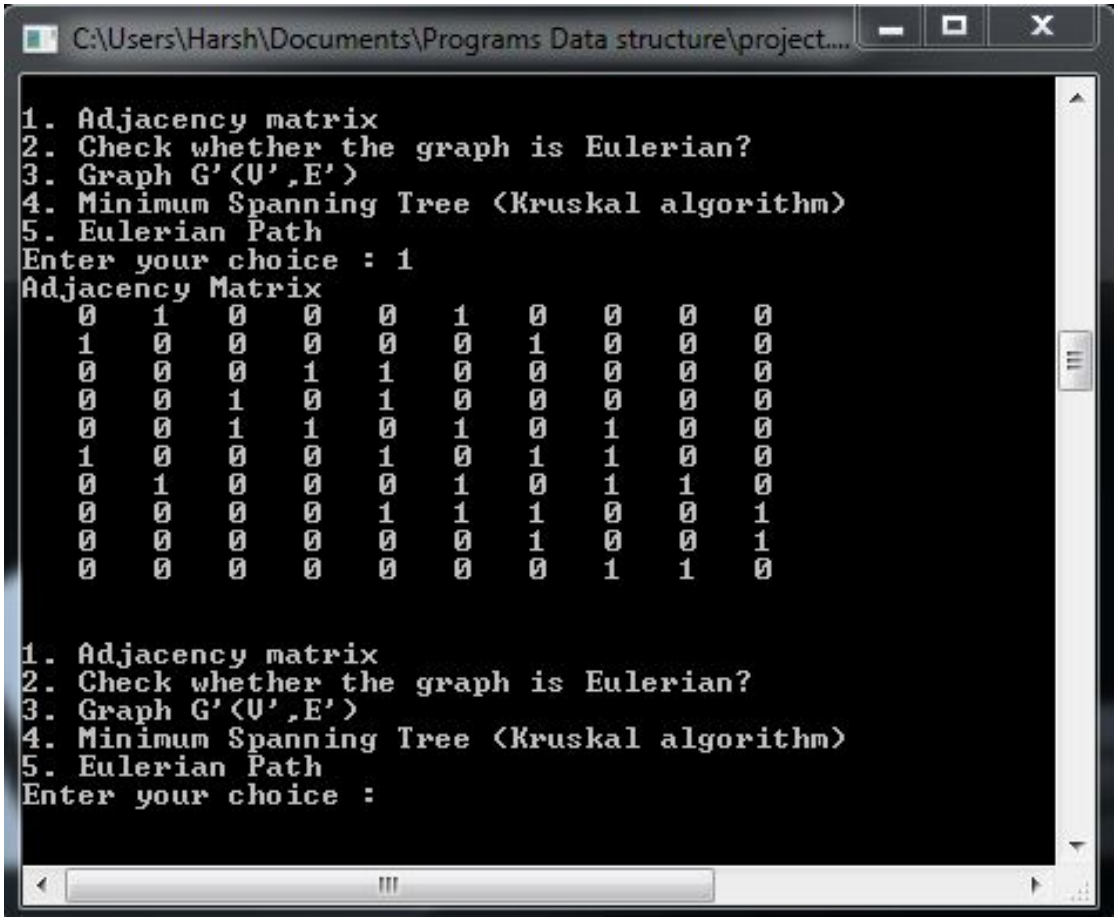

FIGURE 4.3: Adjacency Matrix

The snapshot above shows the adjacency matrix for the 10 node graph.

4.3 Results

We obtain the computation result for 2 sample outputs, one being an Eulerian and other a non-Eulerian one.
Sample Output 1:

The first input graph consists of 10 nodes and is Eulerian. The below snapshot shows the Eulerian trail for the graph.
For Intput 2:

The second input graph is non-Eulerain and consists of 15 nodes.

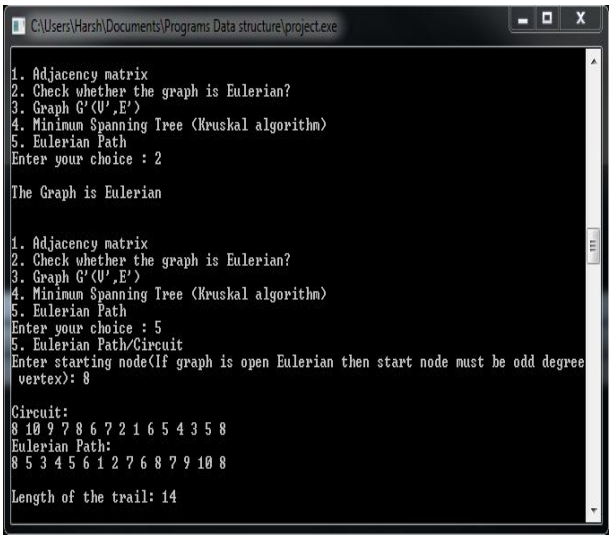

FIGURE 4.4: Ouput for Graph with 10 nodes (Eulerian Graph)

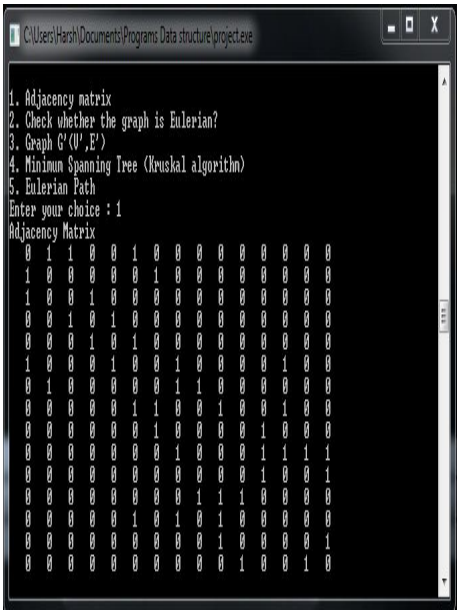

FIGURE 4.5: Adjacency matrix for graph with 15 nodes

Ontput 2:

The different stages for obtaining the final optimal path are shown.

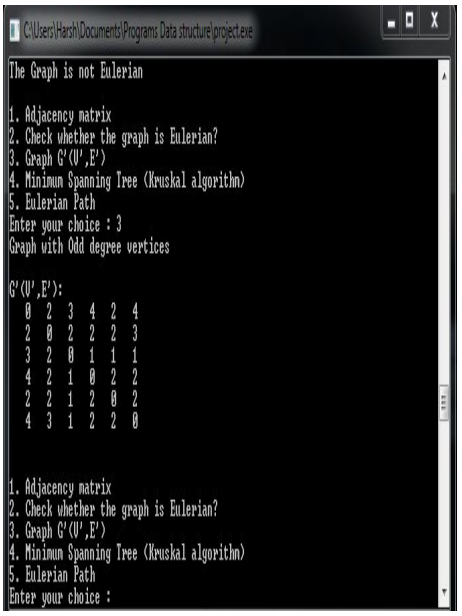

FIGURE 4.6: Checking whether the graph is Eulerian or not and obtaining $G^{'}(V^{'}, E^{'})$

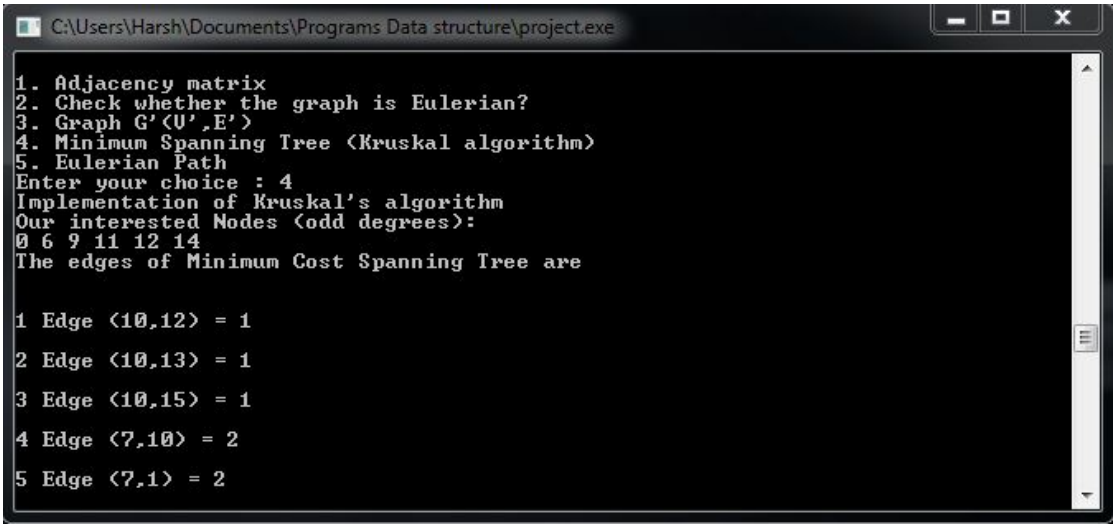

FIGURE 4.7: Minimum Spanning Tree for graph $G^{'}$

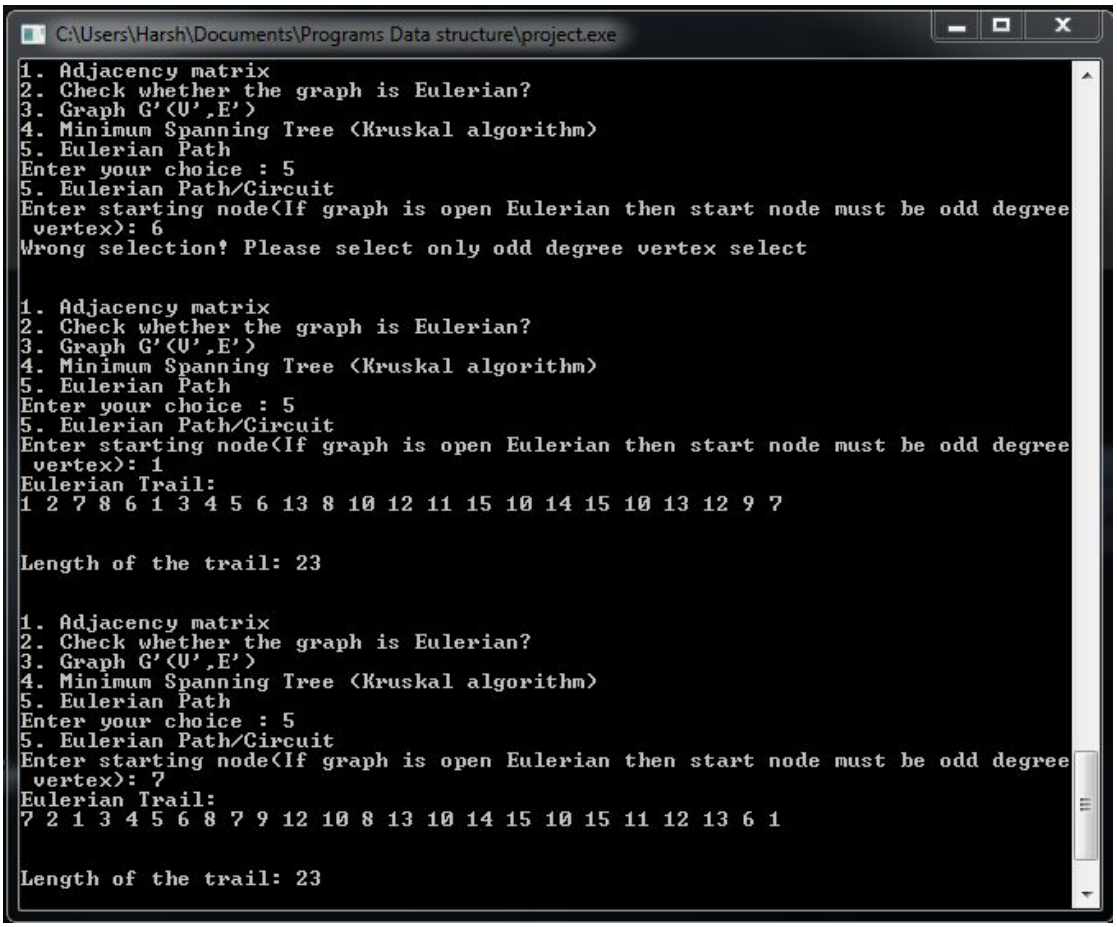

FIGURE 4.8: Final Optimal solution

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We proposed a methodology to detect a stationary target on a network. The immobile entity is uniformly distributed in the network which we have represented as an undirected graph. There are two possible cases. First if the graph is Eulerian, than any Eulerian tour gives an optimal solution. Second if the graph is non-Eulerain, we convert the graph into Eulerian using our proposed strategy so that the graph becomes Eulerian. Once the graph is Eulerain, the Eulerain tour is obtained that result in optimal solution. We have shown the implementation results in the form snapshots. We obtain the output in the form of an Eulerian trail and we can perform the search with either of the entry vertex. So we conclude that the length of the forward as well as the backward Eulerian trail is same and preferences would be given on choosing the start node on the basis of the location of the target node i.e. we choose the vertex as start point which is closest to the target.

5.2 Future Work

In future, research can be done so as to minimize the repeated use of Hierholzers algorithm for obtaining the Eulerian trail. Further in case of large networks the problem becomes complex to search the target as the graph contains too many vertices. Moreover there may be some vertices with too many neighbors (very large degree node). The search process can go on repeatedly on this vertex because the heuristic traverses each edge once but the vertex can be traversed multiple times. Thus the search process will omit the target vertex. In such situation the graph can be broken down into sub-regions in such a way that the neighbors of the very large degrees can be reduced. Then on these regions the search strategy should be implemented. Also the potential regions (where the entity is expected to be present) can be specified. This could result in less computation as well as the expected search time will be minimized as the number of repeated edges required to traverse decreases.

The methodology is used in military operations to rescue hosts and to defuse the bombs. When hosts are kidnapped by terrorists in a building then an automatic robot (searcher) is designed to search the exact location of terrorists in

minimum possible time. Also the planted bomb must have been detected as soon as possible[1].

The Eulerian graph property is applied in generation of layouts of CMOS circuits for VLSI design [9].

Eulerian path approach is used for assembling the DNA fragment. The reduction of fragment assembly is done using Eulerian path problem that allows one to generate accurate solutions of large-scale sequencing problems [10].

References

[1] A. Jotshi and R. Batta, *Search for an immobile entity on a network*, European Journal of Operational Research, 2008, Vol. 191, pp. 347-359

[2] A. Jotshi, Q. Gong and R. Batta, *Dispatching and Routing of Emergency Vehicles in Disaster Mitigation Using Data Fusion*, Socio-Economic Planning Sciences, 2009, Vol. 43, Issue 1, pp. 1-24

[3] S. Gal, *Search games with mobile and immobile hider*, SIAM Journal on Control and Optimization, 1979, Vol. 17, Issue 1, pp. 99-122

[4] S. Gal, *On the Optimality of a Simple Strategy for Searching Graphs*, International Journal of Game Theory, 2001, Vol. 29, pp. 533-542

[5] S. Alpern and S. Gal, *The Theory of Search Games and Rendezvous*, Springer, 2003

[6] J.B. Kruskal, *On the shortest spanning subtree and the traveling salesman problem*, Proceedings of the American

Mathematical Society, 1956, Vol. 7, Issue 1, pp. 48-50

[7] C. Hierholzer, *ber die Mglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren*, Mathematische Annalen, 1873, Vol. 6, pp. 30-32

[8] S. Warshall, *A theorem on boolean matrices*, Journal of the ACM, 1962, Vol. 9, Issue 1, pp. 11-12

[9] B.S. Carlson, C.Y.R. Chan and D.S. Meliksetian, *Circuits snd Systems*, Proceedings of ISCAS, 1992, Vol. 5, pp. 2248-2251

[10] P.A. Pevzner, H. Tang, and M.S. Waterman, *An Eulerian path approach to DNA fragment assembly*, Proceedings of National Academy of Sciences of the United States of the America, 2001, Vol. 98, Issue 17, pp. 9748-9753