B.Sc. Engg. Thesis

Design of a Surveillance System for Dhaka City

by

Md. Lutfar Rahman
Student No. 0705064

Eshita Zaman
Student No. 0705065

Fahim Tahmid Chowdhury
Student No. 0705074

Submitted to
Department of Computer Science and Engineering

in partial fulfilment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000

Certificate

This is to certify that the work presented in this thesis entitled "Design of a Surveillance System for Dhaka City" is the outcome of the investigation carried out by us under the supervision of Professor Dr. Md. Saidur Rahman in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka. It is also declared that neither this thesis nor any part thereof has been submitted or is being currently submitted anywhere else for the award of any degree or diploma.

(Supervisor)

.....

Dr. Md. Saidur Rahman

Professor

Department of Computer Science and Engineering

BUET, Dhaka-1000.

(Authors)

.....

Lutfar Rahman

Student No. 0705064

.....

Eshita Zaman

Student No. 0705065

.....

Fahim Tahmid Chowdhury

Student No. 0705074

Contents

List of Figures

Acknowledgments

First of all, we would like to thank almighty **ALLAH** for giving us the patience and capability to complete our thesis work. We would like to express our deep gratitude to our thesis supervisor Professor **Dr. Md. Saidur Rahman** for not a few reasons. He not only introduced us to this wonderful and fascinating world of Graph Theory but also provided us with numerous resources and helpful directions. Without his constant supervision and words of encouragement, this thesis would not have been possible at all.

This thesis work has been done in the *Graph Drawing and Information Visualization Lab*, Department of CSE,BUET. This lab is well equipped and full of resources. It is our great pleasure to work in such a wonderful environment.

We would also thank all the members of our research group for their valuable suggestions, continual encouragements and critic decisions.

Finally, we thank our parents for supporting and encouraging us throughout all our studies and research at university.

Abstract

Surveillance system is a way to monitor and control a particular state or group of situations that depict a state. In this thesis, we have studied and analyzed the current state of the road and transportation of Dhaka city. The main purpose of our thesis is to design and simulate an integrated system to monitor and control the traffic system of the city and navigate through the roads from one place to another. For navigation through the city from one place to another, we have analyzed and simulated *Dijkstra's Algorithm*. During this task, we have discovered different strategies of applying *Dijkstra's Algorithm* on big set of data efficiently and compared the simulation results. Moreover, we have studied about draining the traffic flow towards an alternative way by automating the signaling in some special situations. Another target of our thesis was to work on security measures of the Dhaka city. *2-Approximation Vertex Cover Algorithm* was used to cover the city's prominent road-crossings by police-boxes so that maximum security can be assured. This sort of analysis is in an initial stage in Bangladesh. We think this system should be developed in our country as soon as possible to enhance the overall security and help to navigate through different places.

Chapter 1

Introduction

Different types of real world situations can be described and explained by a figure of a set of points together with lines joining certain pairs of these points. A mathematical abstraction of such situation arises the concept of graph.

A *graph* consists of a collection of vertices called *nodes* and a collection of edges that connect pairs of vertices.

Planar graph is a graph that can be embedded in the plane. A planar graph drawn in the plane without edge intersections is called a *plane graph*.

Map is a diagrammatic representation of an area of land or sea showing physical features, cities and roads. We can model the road map as planar graph where the nodes represent intersections and the edges represent road segments between intersections, and edge weights represent road distances.

There may be several paths between a specific source and destination place but our job is to find the *shortest path*. If any node of the shortest path is disconnected for any reason then we need to find second best pathway from

source to destination.

In another part of our thesis we supposed to set police box in important junction of roads for ensuring road safety all over the Dhaka city. To set minimum number of police box we used *2-Approximation Vertex Cover Algorithm*.

1.1 Source to Destination Shortest Path

The path having minimum weight from source to destination is the shortest path. There are several algorithms for computing shortest paths between source and destination. *Dijkstra's Single Source Shortest Path Algorithm* gives shortest path between all pairs of vertices for weighted graph.

1.2 Previous Results

In many countries of the world different car navigation system is developed. But in Bangladesh it is a new concept. For that reason we had to face difficulties in collecting data, making database and deciding where to start from. There's no well structured database for all the roads and highways in Bangladesh. Moreover some of the information is classified and is not accessible at all. Then we decided to use Openstreet Map database. But it is a worldwide system. Several operations were needed to customize it according to our need. As the survey is not done in our country we can not ensure the latest update in the database.

1.3 Scope of this Thesis

In this section, we mention the starting approach; we have analyzed how to deal with the problem of working with a very large plane graph. At the end, we list the results obtained by us in this thesis.

Main objective of our thesis is to develop an automated surveillance system for the traffic system of our country that will not only monitor the state of the road and traffic but also give dynamic solution of some problems. When a route is affected by traffic jam or violence the automated traffic system will drain vehicles to a different shortest route. Locating minimum number of police boxes around the city efficiently and effectively so that the appropriate police box can be notified about any mishap. We have approached by calculating shortest path from source to destination by Dijkstra's algorithm for directed, weighted graph. Then we applied our algorithm on the same graph and compared the time required for calculating the shortest path. After that we used 2-approximation vertex cover algorithm to detect nodes representing the police boxes.

The rest of this thesis is organized as follows. In Chapter 2, we give some basic terminology of graph theory and algorithmic theory. In Chapter 3, we give some algorithms for computing shortest from specific source to destination. In Chapter 4, we give 2-approximation vertex cover algorithm. In Chapter 5, we give the definition of clustering and the properties of clustering used here to manipulate graph with large number of vertices and edges. In chapter 6, we give our algorithm for computing shortest path between two places given in the map graph. It also shows the road juntioncs where

police boxes are to be built to ensure road safety and also shows second best shortest path if any junction is blocked for any kind of mishap.

Chapter 2

Preliminaries

In this chapter, we define some basic terminologies of graph theory and algorithm theory, that we will use throughout the rest of this thesis. Definitions which are not included in this chapter will be introduced as they are needed. We review, in Section 2.1, some definitions of standard graph-theoretical terms. In Section 2.3, we discuss about some special operations on graphs that are important for the ideas and concepts used in the later parts of this thesis. We devote Section 2.2 to define terms related to planar graphs. Finally, we introduce the definition of time complexity in Section 2.4.

2.1 Basic Terminology

In this section we give some definitions of terms of standard graph theory.

2.1.1 Graphs

A Graph G is a triple consisting of a vertex set $V(G)$, an edge set $E(G)$. Vertices are also known as nodes or points and edges are also known as lines or links. We can draw a graph on paper by placing each vertex at a point and representing each edge by a curve joining the location of it's endpoints. As an example, the graph depicted in Figure 1 has vertex set $V = \{$a,b,c,d,e.f$\}$ and edge set $E = \{$(a,b),(b,c),(c,d),(c,e),(d,e),(e,f)$\}$.

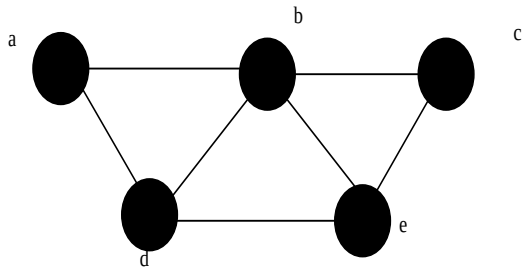


Figure 2.1: A graph with five vertices and five edges.

Fig. 2.1 depicts a graph $G = (V, E)$ where each vertex in $V = \{v_1, v_2, \ldots, v_5\}$ is drawn as a small circle and each edge in $E = \{e_1, e_2, \ldots, e_7\}$ is drawn by a line segment.

2.1.2 Simple Graphs and Multigraphs

A *loop* is an edge which starts and ends on the same vertex. An edge with two different ends is called a *link*. *Multiple edges* are edges having the same pair of endpoints. *Multigraph* is a graph which has multiple edges and sometimes loops. *Simple graph* is a graph having no loops or multiple edges between

two different vertices. In a simple graph, the edges of the graph create set and each edge is pair of distinct vertices. In a simple graph with n vertices, every vertex has a degree less than n.

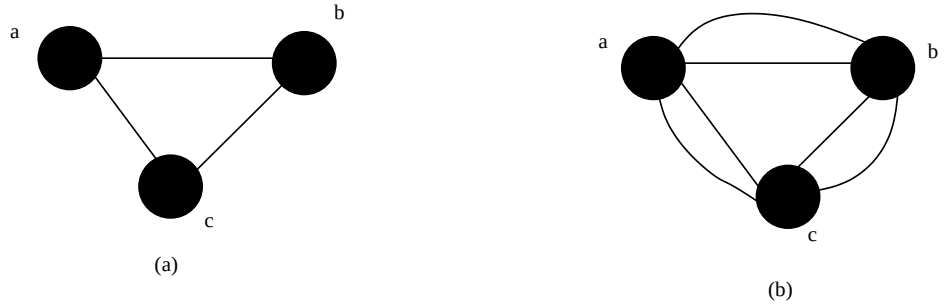


Figure 2.2: (a) Simple graph (b) Multi graph.

2.1.3 Subgraph

A *subgraph* of a graph G is a graph whose vertex and edge sets are subsets of those of G. On the contrary, a supergraph of a graph G is a graph that contains G as a subgraph.

For example, H will be the subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ where $V(G)$ is the vertex set of graph G and $E(G)$ is the edge set of graph G.

In figure 2.3, subgraph of figure 2.1 has been illustrated.

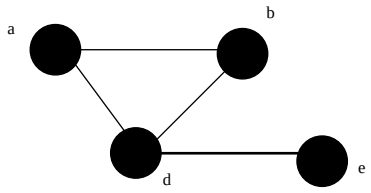


Figure 2.3: A subgraph of the graph in Figure 2.1.

2.1.4 Walk, Trail, Paths and Cycles

A *walk* in a graph is a sequence of vertices and edges alternately interconnected with each other. A *trail* is a special type of walk which is allowed to traverse only once for each edge. However, a trail is still allowed to traverse multiple times for the same vertex. A *path* is a walk with no repeated vertices. A path is disallowed to an edge or a vertex to be traversed multiple times. A *cycle* is a closed trail with at least one edge and with no repeated vertices except that the initial vertex.

2.1.5 Shortest Path

In a shortest path problem,we are given a weighted, *directed graph* $G = (V, E)$, with *weight function* w:E $\rightarrow$ R mapping edges to real valued weights. The weight of path $P = \{v_1, v_2, \ldots, v_k\}$ is the sum of the weight of its constituent edges: w(p)=$\sum_{i=1}^{k} w(v_{i-1}, v_i)$. We define shortest path weight from u to v by

$$\delta(u,v) = \begin{cases} min\{w(p) : u \rightarrow v\} \text{if there is a path from u to v} \\ \infty \end{cases}$$

A shortest path from vertex u to to vertex v is then defined as any path

p with weight $w(p) = \delta(u, v)$.

2.1.6 Connectivity

A *separating set* or *vertex cut* of a graph G is a set S$\subseteq V(G)$ such that G-S has more than one component. The *connectivity* of G denoted as, $k(G)$, is the minimum set of a vertex set S such that G-S is disconnected or has only one vertex. A Graph G is *k-connected* if its connectivity is at least k.

The minimum number of edges that would need to be removed from G in order to make it disconnected is the *edge connectivity* of that graph.

2.2 Planar Graphs

In this section we give some definitions related to planar graphs that is used in the remainder of the thesis. For readers interested in planar graphs

2.2.1 Planar Graphs and Plane Graphs

A *planar graph* is a graph that can be embedded in the plane. It can be drawn on the plane in such a way that its edges intersect only at their endpoints. In other words, a planar graph exists, if the graph can be drawn in such a way that there is no edge crossing.

A planar graph already drawn in the plane without edge intersections is called a *plane graph* or *planar embedding of the graph.Face* of a plane graph

divides the plane into connected regions. A finite plane graph G has one unbounded face and it is called the *outer face* of G.

2.3 Operations on Graph

To make the graph found from the map useful we need to do some operations on it. They are described in this section.

2.3.1 2-Approximation Vertex Cover

A vertex cover of a graph G is a set $Q \subseteq V(G)$ that contains at least one endpoint of every edge. The vertices in Q cover $E(G)$. The vertex cover problem is to find e vertex cover of minimum size in a given undirected graph. We shall call such a vertex cover an optimal vertex cover. Even though it is difficult to find an optimal vertex cover in a graph G, it is not too hard to find a vertex cover near optimal. The algorithm we used here takes as input an undirected graph G and returns a vertex cover whose size is guaranteed to be no more than twice the size of an optimal vertex cover. We refer to [Cormen01].

2.3.2 Dijkstra's Single Source Shortest Path

Dijkstras algorithm solves single source shortest paths problem on a weighted, directed graph $G=(V,E)$ for the case in which all the edges are nonnegative.

Therefore, we assume that w(u,v)>0 for each edge (u,v) $\in E$. Dijkstras algorithm maintains a set S of vertices whose final shortest path from source s have already been determined. The algorithm repeatedly selects the vertex u $\in V - S$ with the minimum shortest path estimate, adds u to S and relaxes all edges leaving u. We refer to [Cormen01].

2.3.3 Static Clustering

Graph *clustering* is the task of grouping the vertices of the graph into clusters taking into consideration the edge structure of the graph in such a way that there should be many edges within each cluster and relatively few between the clusters. We refer to [Flake04].

2.3.4 Monotone Path

Let p be a point in the plane and l a half-line starting at p. The *slope* of l, denoted by *slope*(l), is the angle spanned by a counter-clockwise rotation that brings a horizontal half-line starting at p and directed towards increasing x-coordinates to coincide with l. We consider slopes that are equivalent modulo 2π as the same slope (e.g., $\frac{3}{2}\pi$ is regarded as the same slope as - $\frac{\pi}{2}$). Let T be a drawing of a graph G and let (u, v) be an edge of G. The half-line starting at u and passing through v, denoted by d(u, v), is the direction of (u, v). The *slope* of edge (u, v), denoted by slope(u, v), is the *slope* of d(u, v). The direction and the slope of an edge (u, v) are depicted in Fig. 1(a). Observe that slope(u, v) = slope(v, u)π. When comparing directions

and their slopes,we assume that they are applied at the origin of the axes. An edge (u, v) is monotone with respect to a half-line l if it has a *positive projection* on l, i.e., if slope(l) $\frac{\pi}{2}$ ¡ slope(u, v) ¡ slope(l) + $\frac{\pi}{2}$. A path $P(u_1, u_n) = (u_1, \ldots, u_n)$is monotone with respect to a half-line l if $(u_i, u_i + 1)$ is monotone with respect to l, for each $i = 1, \ldots, n1$; $P(u_1, u_n)$ is a monotone path if there exists a half-line l such that $P(u_1, u_n)$ is monotone with respect to l. Fig. 1(b) shows a path that is monotone with respect to a half-line. Observe that, as shown in the figure, if a path $P(u_1, u_n) = (u_1, \ldots, u_n)$ is monotone with respect to l, then the orthogonal projections on l of $u1, \ldots, un$ appear in this order along l. A drawing T of a graph G is monotone if, for each pair of vertices u and v in G, there exists a monotone path P(u, v) in T. Observe that monotonicity implies connectivity. Let p be a point in the plane and l a half-line starting at p. The slope of l, denoted by slope(l), is the angle spanned by a counter-clockwise rotation that brings a horizontal half-line starting at p and directed towards increasing x- coordinates to coincide with l. We consider slopes that are equivalent modulo 2 as the same slope (e.g., $\frac{3}{2}\pi$ is regarded as the same slope as -$\frac{\pi}{2}$). Let T be a drawing of a graph G and let (u, v) be an edge of G. The half- line starting at u and passing through v, denoted by d(u, v), is the direction of (u, v). The slope of edge (u, v), denoted by slope(u, v), is the slope of d(u, v). The direction and the slope of an edge (u, v) are depicted in Fig. 1(a). Observe that slope(u, v) = slope(v, u)π. When comparing directions and their slopes, we assume that they are applied at the origin of the axes. An edge (u, v) is monotone with respect to a half-line l if it has a positive projection on l, i.e., if slope(l) $\frac{\pi}{2}$ ¡ slope(u, v) ¡ slope(l) + $\frac{\pi}{2}$. A path $P(u_1, u_n) = (u_1, \ldots, u_n)$ is monotone

with respect to a half-line l if (ui, ui+1) is monotone with respect to l, for each $i = 1, \ldots, n1$; $P(u_1, u_n)$ is a monotone path if there exists a half-line l such that $P(u_1, u_n)$ is monotone with respect tol. Fig. 1(b) shows a path that is monotone with respect to a half-line. Observe that, as shown in the figure, if a path $P(u_1, u_n) = (u_1, \ldots, u_n)$ is monotone with respect to l, then the orthogonal projections on l of $u1, \ldots, u_n$ appear in this order along l. A drawing T of a graph G is monotone if, for each pair of vertices u and v in G, there exists a monotone path $P(u, v)$ in T. Observe that monotonicity implies connectivity. We refer this to [S05].

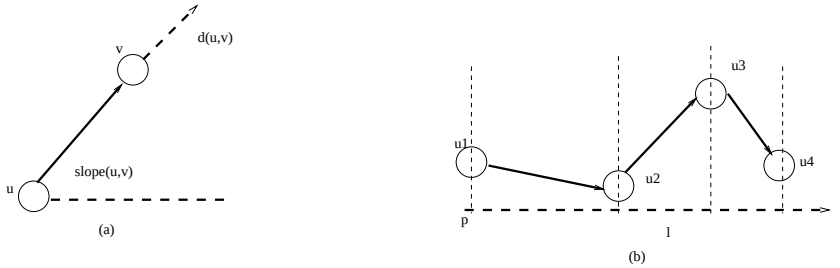


Figure 2.4: (a) The direction d(u, v) of an edge (u, v) and its slope slope(u, v). (b) A path P (u1 , u4) that is monotone with respect to a half-line l.

In figure 2.4, a monotone path with respect to horizontal line has been illustrated.

2.4 Complexity of Algorithms

In this section we briefly introduce some terminologies related to complexity of algorithms. It is very convenient to classify algorithms based on the relative

amount of time or relative amount of space they require and specify the growth of time or space requirements as a function of the input size.

Running time of the program as a function of the size of input is called the *time complexity*. Amount of computer memory required during the program execution, as a function of the input size is called the *space complexity*.

2.4.1 The Notation $O(n)$

$O(n)$ notation is used to express the worst-case order of growth of an algorithm. That is, how the algorithm's worst-case performance changes as the size of the data set it operates on increases. $O(n)$ means the algorithm's performance is directly proportional to the size of the data set being processed. The time complexitiy of $O(n)$ algorithm is linear.

2.4.2 Polynomial Algorithms

An algorithm is said to be solvable in polynomial time if the number of steps required to complete the algorithm for a given input is $O(n^k)$ for some non-negative integer k, where n is the complexity of the input. Polynomial-time algorithms are said to be "fast". Most familiar mathematical operations such as addition, subtraction, multiplication, and division, as well as computing square roots, powers, and logarithms, can be performed in polynomial time. Computing the digits of most interesting mathematical constants, including pi and e, can also be done in polynomial time.

Chapter 3

Shortest Path Algorithms

3.1 Dijkstra's Shortest Path Algorithm

Single source shortest path can be solved using the *Bellman Ford Algorithm* and *Dijkstra's Algorithm*. But in the *Bellman Ford Algorithm* weight of edges can be negative. So, *Dijkstra's Algorithm* is the most convenient for us.

Dijkstra's Algorithm-

```
Pseudo code:
S = {};
d[s] = 0;
d[v] = infinity for v != s;
while (S != V)
```

```
{
   choose v in V-S with minimum d[v];
   add v to S;
   for each  w in the neighborhood of v
       d[w] = min(d[w], d[v] + c(v, w));
}
return S;
```

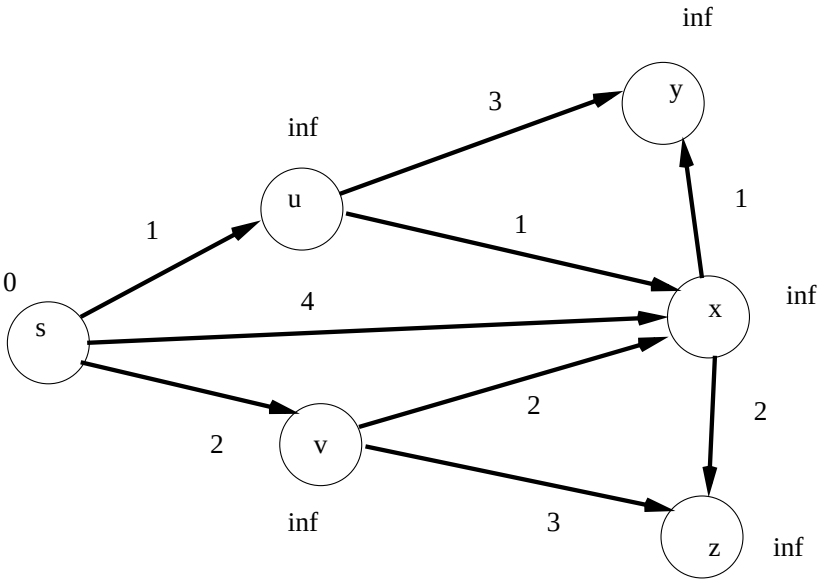
Dijkstra's algorithm gives correct results but it performs very slowly when the number of nodes is very large. To speed up Dijkstra's algorithm we can use *Bidirectional Search*. The algorithm for *Bidirectional Search* is given below-

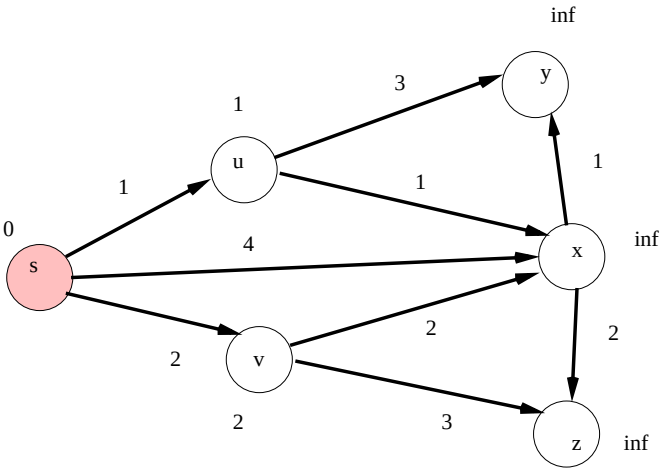
Bidirectional Search-

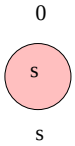
```
Pseudo code:
While(!same node visited from both directions)
{
   Execute Dijkstra's Algorithm from both source
   and Destination
}
Shortest path can be derived from the information
already gathered
```

3.2 Simulation of Dijkstra's Shortest Path Algorithm

Simulation of Shortest Path with Dijkstra's Algorithm is given below:







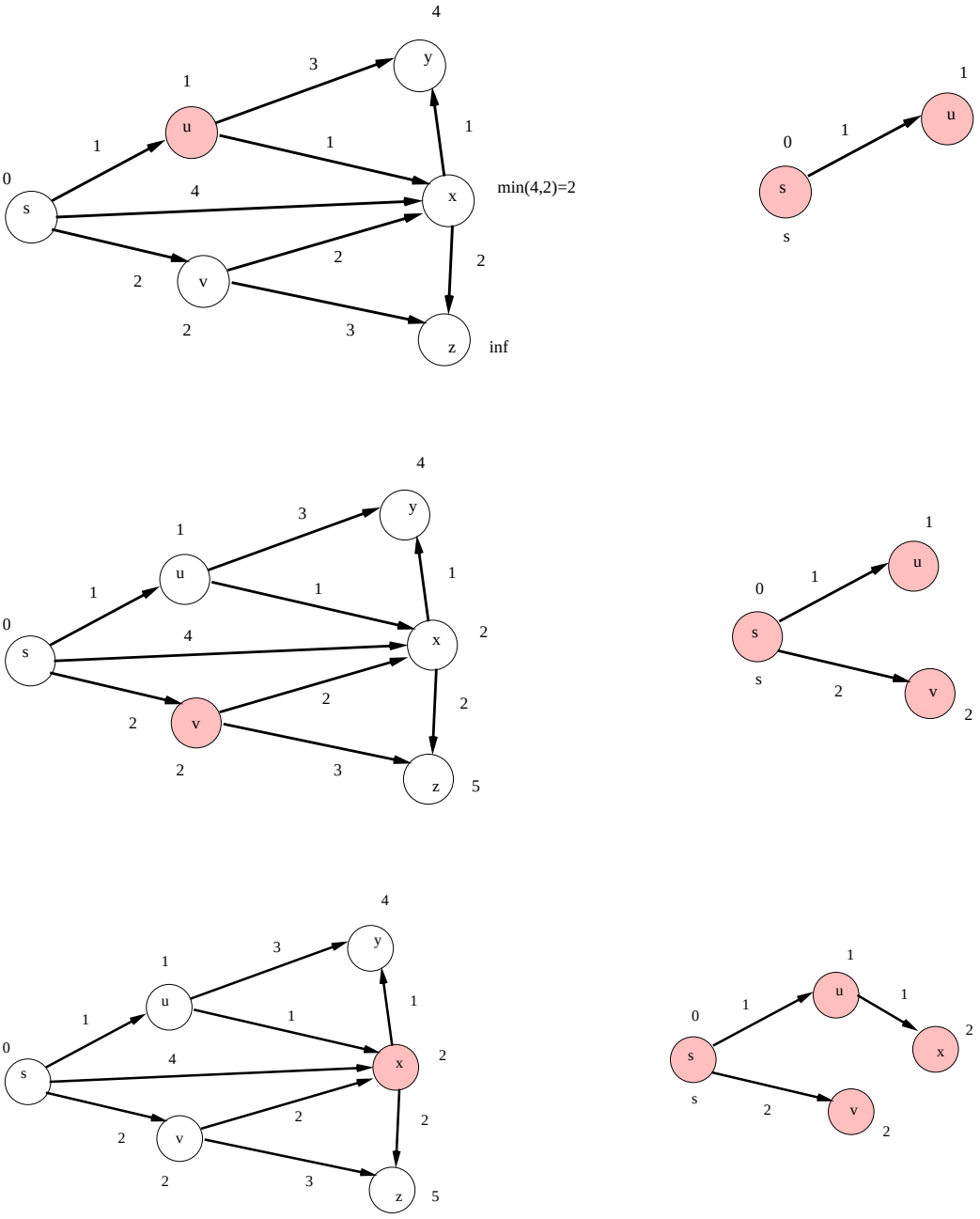


Figure 3.1: Simulation of Dijkstra's Shortest Path Algorithm.

3.3 Conclusion

In this chapter, we have reviewed some important algorithms for computing shortest path between source and destination. The approach of these algorithms are very often followed by the researchers in this field. In particular, we have used Dijkstra's algorithm to obtain results to compare it with the result found using our algorithm.

Chapter 4

2-Approximation Vertex Cover Algorithm

A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if (u,v) is an edge of G, then either u$\in V$ or $v \in V$(or both). The size of the vertex cover is the number of vertices in it. We refer it to [Cormen01].

APPROX-VERTEX-COVER(G)

$C \leftarrow \phi$

$E \leftarrow E[G]$

While $E \neq \phi$

do let (u, v) be an arbitrary edge of E

$C \leftarrow C \cup \{u, v\}$

remove from E every edge incident to either u or v

return C

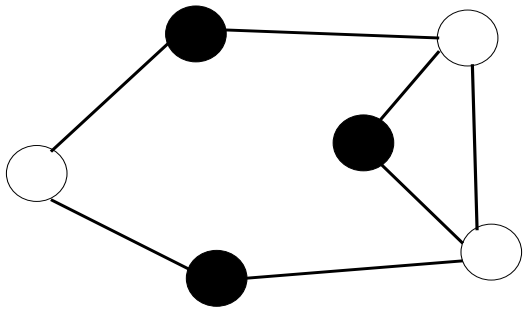


Figure 4.1: Vertex cover of the graph.

For the graph of Figure 4.1, if we take the vertices marked with squares, it will cover all the edges of that graph. So, the size of vertex cover in this graph is 3.

Chapter 5

Clustering

We have studied some basic clustering techniques and come up with a decision that the conventional clustering are not sufficient to handle the data representing the city of Dhaka. Moreover the connectivity among the clusters is also an important factor that we must be aware of. So eventually we have suggested a clustering technique to match our requirements and serve the purpose of our algorithm described later.

The total graph is clustered in such a manner that the clustering holds the following properties:

1. There are some common vertices (one or more than one) between the clusters named GATEWAYS.

2. The number of nodes in a cluster cannot exceed a maximum value (i.e. 20).

3. The gateways between two clusters cannot be connected to each other.

After determining the clusters according to the above algorithm, we then represent the clustering by a new graph. In this new graph, the heuristic of each node is determined by the average heuristic of all the nodes in a particular cluster. And the common nodes between two clusters are represented by edges in the new graph. The weight of individual edge will be the heuristic of the common node that makes the edge. We refer this to [pregel].

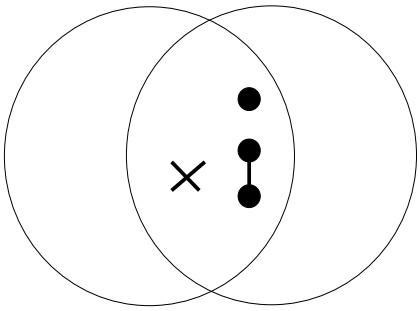


Figure 5.1: The common nodes between two clusters will not be connected to each other

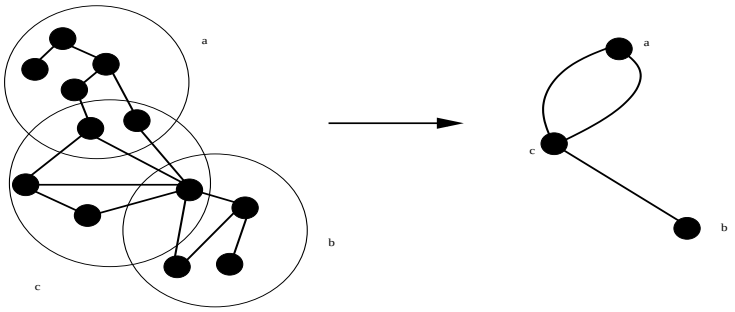


Figure 5.2: The clusters are repesented as a graph

Chapter 6

Our Algorithms

6.1 Shortest Path Algorithm with Clustering and Flooding

While determining the Shortest Path for the navigation of the traffic in our thesis, we came out with an intuitive idea to manipulate large set of data. Initially, we proposed a strategy with clustering discussed in Chapter 5 and flooding in monotone path discussed in Sub-section 2.3.4 of Chapter 2 to find the shortest path between a source and a destination.

6.1.1 Introduction

In this section we will describe our algorithm in details. This section has the followings:

1. Data Structure
2. Heuristic
3. Algorithm with Clustering and Heuristic
4. Advantages and Drawbacks

6.1.2 Data Structure

Database we used is collected from OpenstreetMap.org. The database is in xml file format. The xml file covers a particular rectangular area. Xml file contains all significant nodes and its details like its latitude, longitude, name (if it has) etc. Xml further contains all the ways in that rectangular area. In the way tag, there are some child tags that represent the nodes building the way. We can find the id of the node from the "ref" attribute in the child tag "nd" inside tag "way". However, The Xml looks like following:

$<?xmlversion = "1.0"encoding = "UTF-8"?>$
$<osmversion = "0.6"generator = "CGImap0.0.2">$
$<boundsminlat = "23.6670000"minlon = "90.3000000"maxlat = "23.9000000"maxlon = "90.4500000"/>$
$<nodeid = "60916927"lat = "23.7441220"lon = "90.4142066"user = "Xaman"uid = "334443"visible = "true"version = "8"changeset = "11657750"timestamp =$

$$"2012-05-21T01:37:08Z" >$$
$$< tagk = "highway" v = "traffic\_signals" / >$$
$$< tagk = "name" v = "MalibaghCircle" / >$$
$$< /node >$$
$$\vdots$$
$$< wayid = "167571575" user = "lytron" uid = "406374" visible = "true" version =$$
$$"1" changeset = "11912092" timestamp = "2012-06-16T08:37:43Z" >$$
$$< ndref = "1789664920" / >$$
$$< ndref = "1789664848" / >$$
$$< ndref = "1789664847" / >$$
$$< ndref = "1789664885" / >$$
$$< ndref = "1789664882" / >$$
$$< ndref = "1789664916" / >$$
$$< ndref = "1789664920" / >$$
$$< tagk = "building" v = "yes" / >$$
$$< tagk = "source" v = "bing" / >$$
$$< /way >$$
$$\vdots$$
$$< /xml >$$

From this basic data structure we have processed some more data structures for our purpose. The connectivity among the nodes is determined from the "way" tag in the map.osm file. By the connectivity we acquired the adjacency matrix and write another xml for our purpose. Each $< item >$ tag contains a connectivity between a pair of nodes $(prev\_id, current\_id)$ and the

weight is calculated as the Euclidean distance of their global position(latitude and longitude). The xml for adjacency matrix looks like following:

$<?xmlversion = "1.0"? >$

$< graph >$

$< itemprev_id = "1789664920" current_id = "1789664848" prev_lon = "90.4103533" prev_lat =$
"23.8588913" $current_lon$ = "90.4103511" $current_lat$ = "23.8587782" $distance =$
"0.0001131213949702 2" $cluster_id_prev$ = "0" $cluster_id_current$ = "0" / >

$\vdots$

$< /graph >$

$< /xml >$

For the shortest path simulation we load the adjacency list from this xml file containing adjacency matrix. This adjacency list is supplied to the Dijkstra's algorithm for computing shortest path.

For the vertex cover simulation part, we searched that way node that connects three different ways. We write those nodes into another xml called common nodes. Then we needed the connectivity of those significant nodes. For this purpose we computed shortest path between every pair of nodes. From those paths our connectivity is defined by the shortest paths containing only significant nodes at starting and ending position. After a long computing effort we have our adjacency matrix of significant nodes.

6.1.3 Heuristic

Heuristic of the Node: $f(n) = g(n) + h(n)$

$g(n)$: The distance of a node from the source

$h(n)$: The straight-line distance of a node to the target

Heuristic of the Cluster: When a clustered graph is converted to a newly constructed graph, the node representing a cluster has a heuristic that has the following value:

```
fc(n) = Average of the heuristics of all the nodes in that cluster.
```

6.1.4 Algorithm with Clustering and Heuristic

```
Algorithm:
Dijkstra's Algorithm with Flooding and Heuristic(s,t)
{
  1. Find the straight-line distance between Source and Target
  2. Avoid expanding the nodes that have bigger straight-line
     distance than the distance determined earlier.
     Thus, the FLOODING is implemented.
  3. f(n) = g(n)+h(n) = real distance from the source
     + straight-line distance from the target
     While expanding the in Dijkstra's Algorithm,
           take the node with the smallest f(n) value
  4. Satisfying the above constraints,
           apply Dijkstra's Algorithm for finding the shortest path

}
Algorithm:
Shortest Path Algorithm(s,t)
{
  if(s and t are in the same cluster)
  {
```

```
    Dijkstra's Algorithm with Flooding and Heuristic(s,t);
  }
  else
  {
    1. Convert the total graph into a new graph representing the clusters as
       nodes and  the GATEWAYS as edges.
      a. Weights of the edges are the heuristics of the gateway nodes
      b. The heuristic of the nodes representing the cluster is the average
         of the heuristics of the nodes inside the cluster that were
         determined initially
    2. Apply Dijkstra's Algorithm with Flooding and Heuristic(sc,tc)
       in the newly created graph to find a sequence of clusters.
    3. Apply Dijkstra's Algorithm with Flooding and Heuristic(s,t)
       recursively in every cluster in that sequence to find the final path.
  }
}
```

6.1.5 Advantages and Drawbacks

Advantages: The advantages of this algorithm are given below-

1. Immensely decreases the space and time complexity.

2. The use of flooding and heuristics saves the amount of computations per manipulation.

3. Recursive Definition provides a chance of applying Dynamic Programming.

Drawbacks: the drawbacks are-

1. The avoidance of expanding some vertices creates a risk of the algorithm not being complete or optimal. But in the case of Geographical maps, in most of the cases, the assumptions we have made are correct.

2. The initialization is costly in this case. The costly tasks like making clusters and determining the heuristics of the vertices can be done in the initialization stage of the system.

6.1.6 Conclusion

The algorithm discussed in this section was proposed by us, but it was not implemented in our application.

6.2 Shortest Path Algorithm in a Particular Region

When we were implementing the application, we came out with another idea to find the shortest path between two nodes in a graph that represents a real-world map. We found out that, in real-world map navigation, we generally do not have to go outside a circular region having the source and destination on its circumference. So, we decided to implement this intuitive idea in our application.

6.2.1 Introduction

In this section we will describe this idea in details. This section has the followings:

1. Data Structure
2. Heuristic
3. Algorithm with Heuristic
4. Advantages and Drawbacks

6.2.2 Data Structure

In this algorithm we have used the data structure discussed in Sub-section 6.1.2 of Section 6.1 of this Chapter.

6.2.3 Heuristic

In this algorithm, we have applied the following heuristics:

1. Heuristic of Circualar Region
2. Heuristic of Band Region

Heuristic of Circular Region

We have selected particular portion of the map circularly around a point. This point is the middle point of the source and destination. The radius is the distance between middle point and source or destination. In the Figure 6.1 we denote s as the source node and t as the destination or target node. The middle point is m and the radius of the circle is r. We run *Dijkstra's Algorithm* for the nodes and edges inside the circular region.

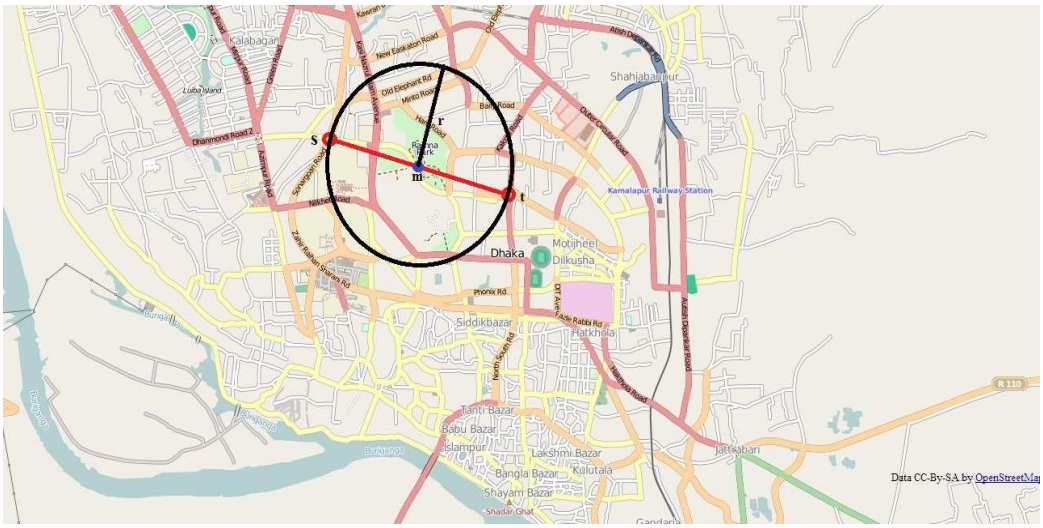


Figure 6.1: Circular Heuristic

Heuristic of Band Region

We have considered a band region along a line directly connecting source and destination. A node is considered as inside the band region if the perpendicular distance from the line directly connecting source and destination is less then the width of the band. In the Figure 6.2 we denoted source node as s and destination or target node as t. Width of the band is denoted by w.

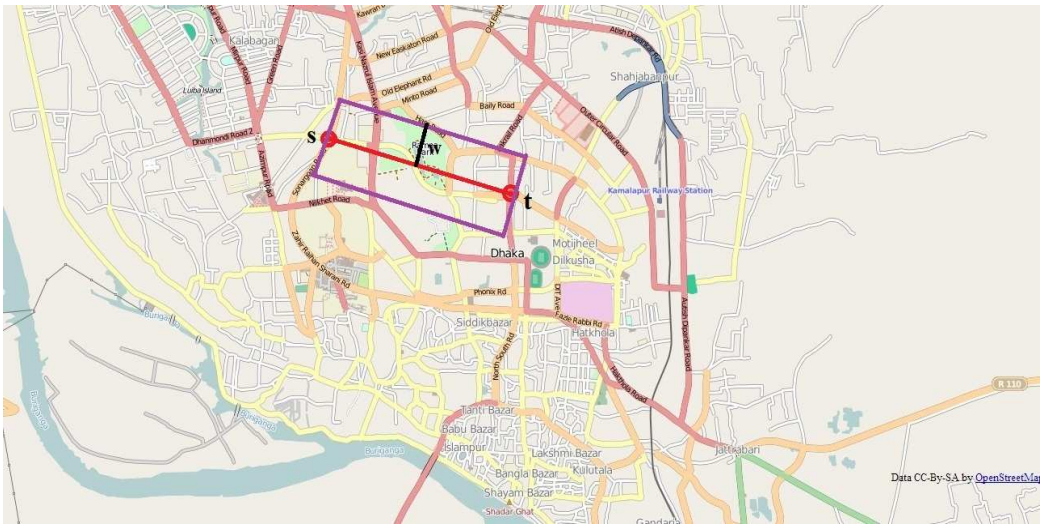


Figure 6.2: Band Heuristic

6.2.4 Algorithm with Heuristic

```
Algorithm:

Dijkstra's Algorithm in a Circular Area(source, destination, Graph)

{

  1. mid-point = Cartessian mid-point of source and destination

  2. r = Euclidean distance between mid-point and source or
     destination.
     b = a particular width of the band having the straight line
     connecting source and destination as the centre

  3. Graph' = The subset of Graph inside the circular region with
     the centre at the mid-point and with radius r
```

```
    OR
    The subset of Graph inside the band region with width b
  4. Dijkstra(source, destination, Graph')
}
```

Dijkstra's Algorithm is discussed in Section 3.1 of Chapter 3.

6.2.5 Advantages and Drawbacks

Advantages: The advantages of this algorithm are given below-

1. Decreases the space and time complexity.
2. The use of heuristics saves the amount of computations per manipulation.

Drawbacks: In some exceptional cases, it can show wrong result. But in real-life highways this sort of exception is unlikely to happen.

6.2.6 Conclusion

The simulation for this algorithm in our application is discussed in 7.2.2 of Chapter 7

Chapter 7

Our Results

7.1 Introduction

Our thesis has two major parts. First part is developing an automated and intelligent traffic system for Dhaka city. For this we needed such an automated system that can dynamically understand present roads' situations and shows the best navigation way for the user all over Dhaka city. For this navigation system, we needed to find the shortest path between two places.

Second part is finding minimum number of police boxes and their locations necessary for Dhaka city covering all areas. For this part we used vertex cover algorithm for finding minimum vertex cover set similar to police box. Then we were asked to develop a surveillance system with the police box dynamically.

7.2 Our Findings

We searched for a proper map database of Dhaka city from the government agencies. But here in Bangladesh we couldn't access the road database as it is a classified database, not publicly available. So, we have collected OSM (openstreetmap.org)'s open source database. The database is in xml file format which contains all significant node and roads definition. Detailed structure of this xml file format is described in Chapter 6 on the Section 6.1.2.

7.2.1 Map Display

We have two approaches to show the map. Number one, Save all the map images locally and use them. This approach can be called offline approach to show the map, as there is no internet connection needed. Number two, get all the map images from a dedicated server which will on request returns map images.We call this approach as online approach,as we need internet connection to access the server. The detailed discussion on these two approach is given below:

Offline Approach

Our first attempt was to develop an offline version for simulating shortest path on Dhaka city map. The platform we chose was $C\#.NET$. The map initially collected from the Bagnladesh Army is shown in Figure 7.1.

Figure 7.1: Map from Bangladesh Army

Then we moved on to a better map drom Openstreet Map shown in Figure 7.2. The main advatage of this map in Figure 7.2 is we could directly use the data from Openstreet Map for our calculations.

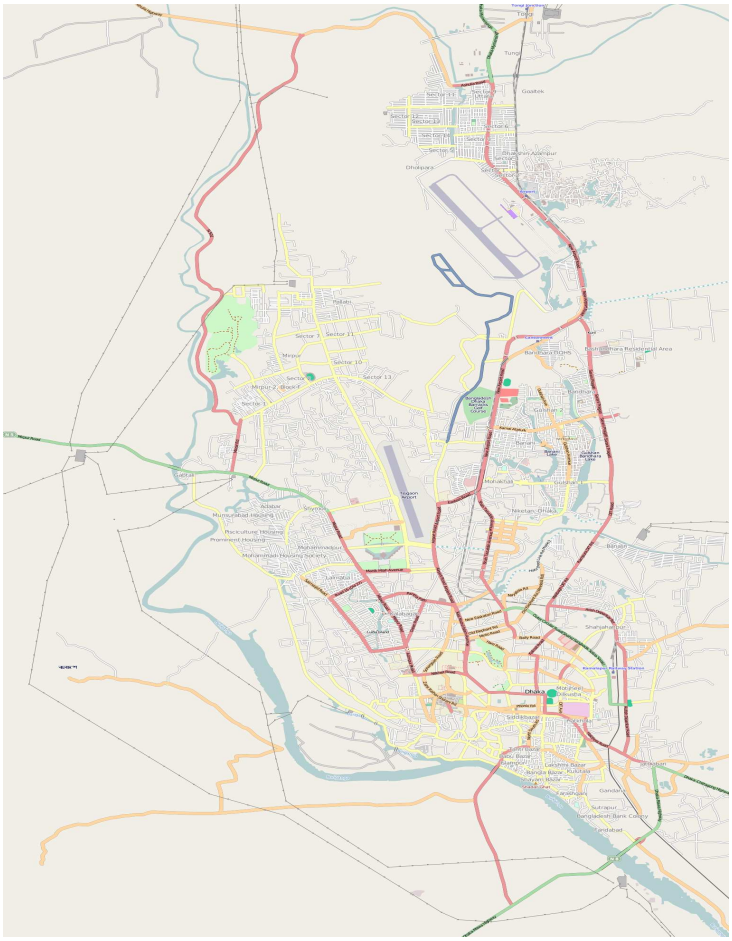


Figure 7.2: Map from Openstreet Map

For the visualization part, we imported an image file with a high resolution. So that zoom-in and zoom-out works correctly without pixel distortion. On backend a point is defined by its latitude and longitude but for the visualization part it is defined by the pixel position on the map image.

We wrote converter function to get the pixel position from latitude and longitude. We acquired relations among the (longitude,latitude)and (pixelX,pixelY) shown in Figure 7.3

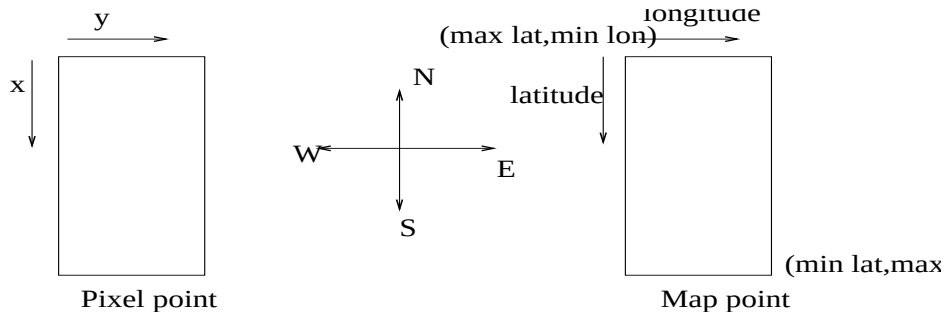


Figure 7.3: Map to Pixel conversion

As both of them are the definition of 2D Cartesian point, so they must have the same ratio.So,

$PixelX/(TotalPixelAlongX-axis) = (Lon-Minlon)/(TotalLonCoveredByTheMapImage)$

$PixelY/(TotalPixelAlongY-axis) = (TotalLatCoveredByTheMapImage-Maxlat+Lat)/(TotalLatCoveredByTheMapImage)$

From these relations we can convert pixel to lonlat and lonlat to pixel.

We faced synchronization problem with the pixel position on the map image and real latitude and longitude of a certain point. We could not find a precise map image that has perfectly distributed pixel with latitude and

longitude. So, the drawn point doesn't perfectly match the point on earth. But it is close enough to consider.

For the shortest path simulation we tried to load the entire map in *C#.NET*. Now we will see the *C#.NET* codes for the task:

1. Our Code starts from Program.cs . It includes all the GUI and backend codes. See codes in Appendix A.

2. Program.cs declares a form from MainForm.cs that is our GUI design. See codes in Appendix B.

3. MainForm.cs includes imageBox.cs. which manipulates and shows our map image.imagBox has all the events like mouseclick,mousemove,doubleclick events on map. When we click, the map zoom in,when right click,the map zoom out. See codes in Appendix C.

4. We wrote a class file for the dijkstra's shortest path. It can read a graph with adjacency list and returns shortest path between any two connected points. It uses a minheap in dijkstra. See codes in Appendix D.

5. We have our graph reader class to read graph from the xml file and load it to variables. See codes in Appendix E.

When we load the program it shows all the points reading its latitude and longitude and converts it to proper pixel point. The snapshot of this situation is depicted in Figure 7.4

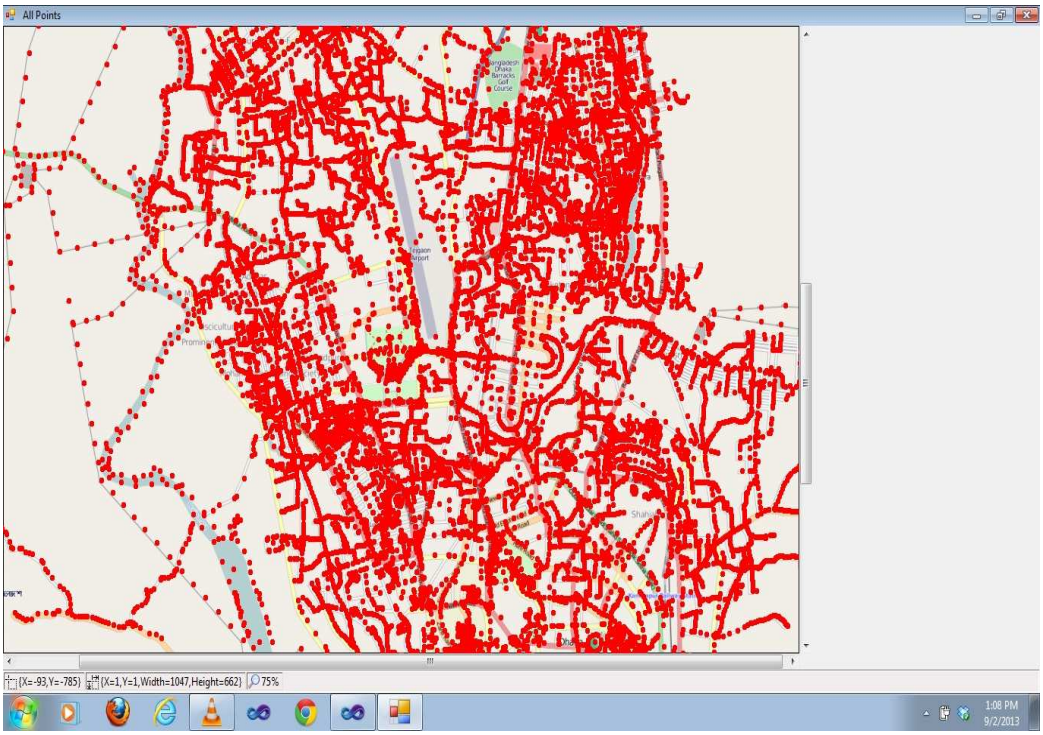


Figure 7.4: After plotting all the points

Problems In Offline Approach

C#.NET does not support huge array data structure. We needed huge array to hold the map for the further calculation. So, We got stuck in simulating small map data with *C#.NET*.

Online Approach

We decided to develop an online version as it supports huge data variable we needed and has mobility. We preferred PHP as the core language of development and OSM API as the development tool. Using PHP we got rid of the memory management problem. We implemented all our desired simulations in PHP. Our User Interface is designed in a way to make the application very easy to run. We implemented clicking option set the source and destination. We obtained the database of nodes for the Dhaka city from openstreetmap.org that is map.osm. Then this original xml file was converted to adjacency matrix. We have kept the adjacency matrix in an xml file name mapgraph.xml. In this file the data structure followed is an item tag that is like this:

```
<item prev_id="1789664920" current_id="1789664848" prev_lon="90.4103533"
prev_lat="23.8588913" current_lon="90.4103511" current_lat="23.8587782"
distance="0.00011312139497022" cluster_id_prev="0" cluster_id_current="0"/>
```

See codes for converting map.osm to our desired format in Appendix F.

This OSM API has made our graph loading easy and handy. We could easily show the whole graph and get the exact position of each nodes from map.osm that is imported from openstreetmap.org.

When we loaded the graph with OSM API we got a picture that is shown in Figure 7.5.

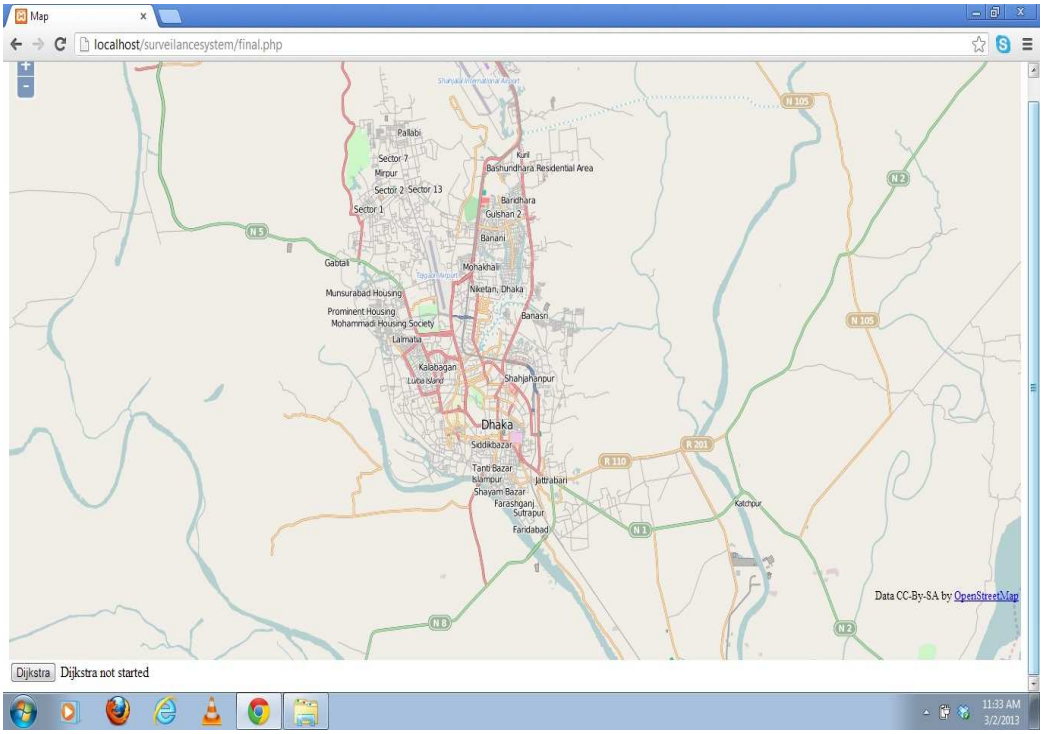


Figure 7.5: Initial display of the Map

The application we developed became very user friendly as we could use some default functionalities in the API. Moreover, huge data storing capability of PHP helped us to further calculate the simulations.
When we zoomed into a centain region we could see a picture shown in Figure 7.6.

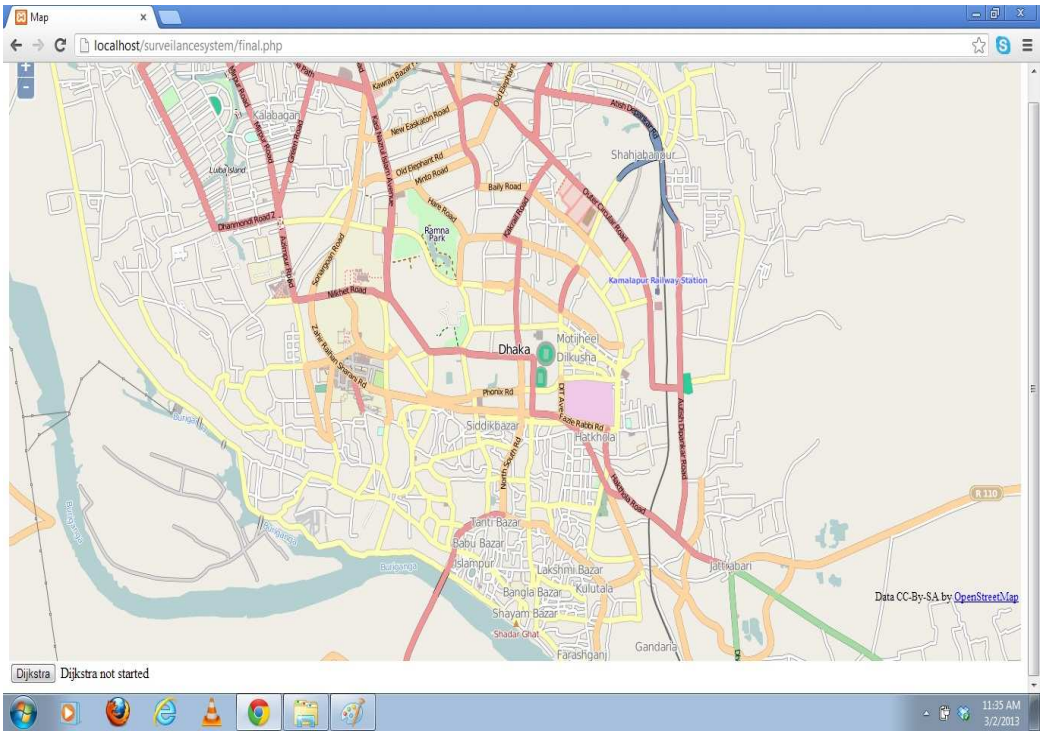


Figure 7.6: Zoomed display of the Map

7.2.2 Navigation through Shortest Path

One of the basic aims of our thesis is to find a navigation path for the traffics from one place to another in the Dhaka city. For serving this purpose,

we had to analyze the Shortest Path Search Algorithms. We decided to analyze and simulate *Dijkstra's Algorithm for single source all pairs Shortest Path*. In this Subsection we will discuss the tasks we have done in order to pursuing the simulation of shortest path algorithm. Afterwards we did some improvizations on the execution time of our application.

Simulation of Basic Dijkstra's Algorithm

We simulated the fundamental Dijkstra's all pairs shortest path algorithm to find out the shortest navigation path between two points on the map of Dhaka city. In this portion, we followed the strategy to feed the whole map.osm mentioned earlier without any pre-calculation to the system and run the Dijkstra's Algorithm on the whole map. It takes around 110 seconds to run while using a Computer with Intel(R) Core(TM) i3-2310M CPU@ 2.10GHz, 2.00GB RAM and with 32-bit Operating System of Windows 7. We used Google Chrome browser for running the system.

This simulation was done by the following four modules:

1. In *PriorityQueue.php* we have developed a data structure to maintain the min heap to make the basic Dijkstra's Algorithm faster. The code for this task is at Appendix G.

2. In *Dijkstra.php* we have implemented the basic Dijkstra's Algorithm using the priority queue. The code goes at Appendix H.

3. *RunTest.php* is the file where the graph is fed to and source and destination points are set for the Dijkstra.php module. RunTest_init.php,

Dijkstra.php and PriorityQueue.php makes the total module of implementing the Dijkstra's Algorithm. The code for RunTest_init.php is at Appendix I.

4. *final_init.php* is the file that contains the user Interface of the system. We have used the osmapi to show the map and implemented the click inputs in it. Then we applied an intuitive idea to find the nearest node on the way from the clicked point. The codes for this portion is given at Appendix J.

Snapshots of the Simulation of Basic Dijkstra's Algorithm

The step-by-step snapshots of the module to simulate *Basic Dijkstra's Algorithm* we developed in our application are given below:

1. The initial state of the graph shown in our application is pictured in Figure 7.7.

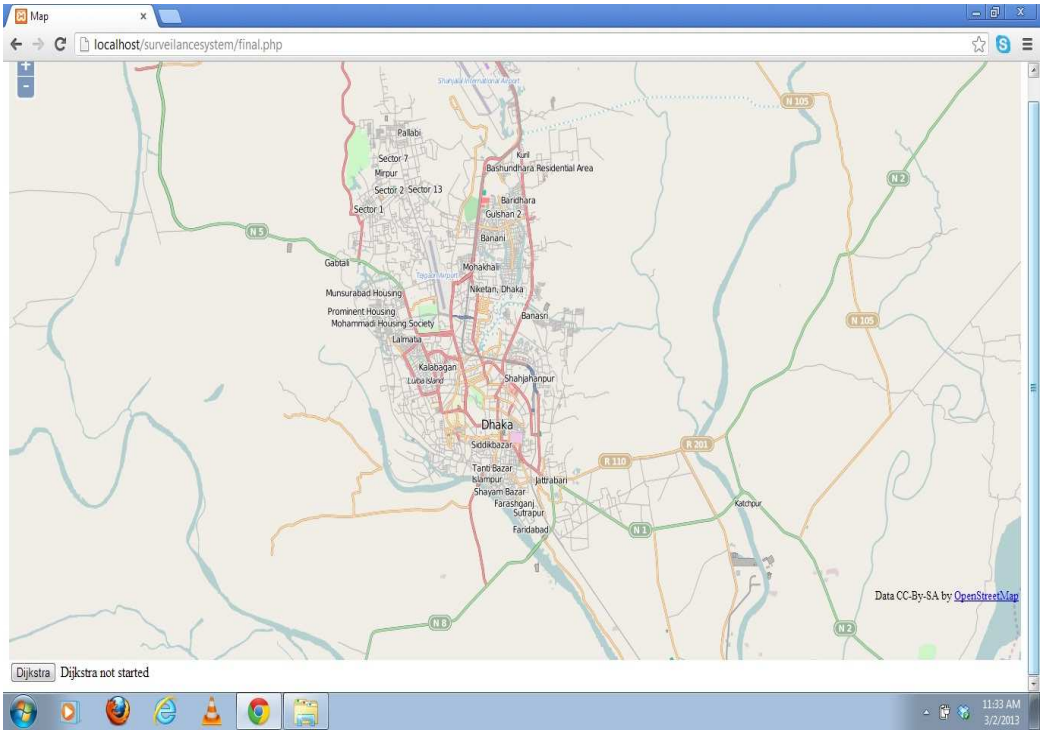


Figure 7.7: Initial display of the Map

2. If we zoom to a particular region we get a picture that is shown in Figure 7.8.

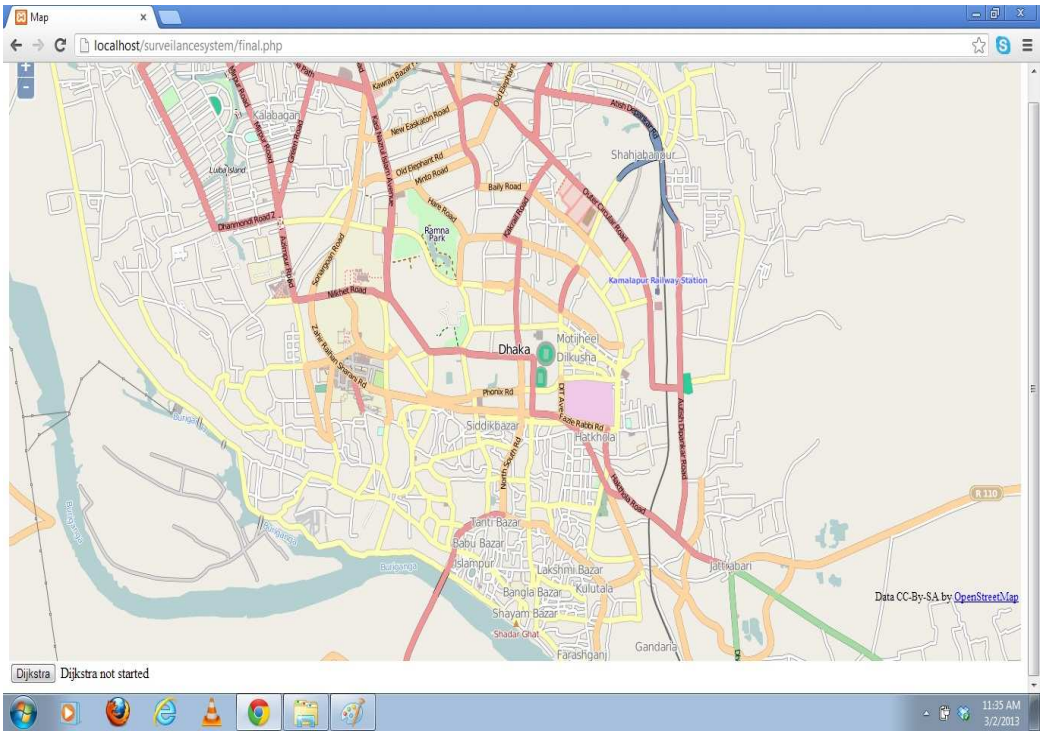


Figure 7.8: Zoomed display of the Map

3. After setting the source and destination points we get a picture that is shown in Figure 7.9.

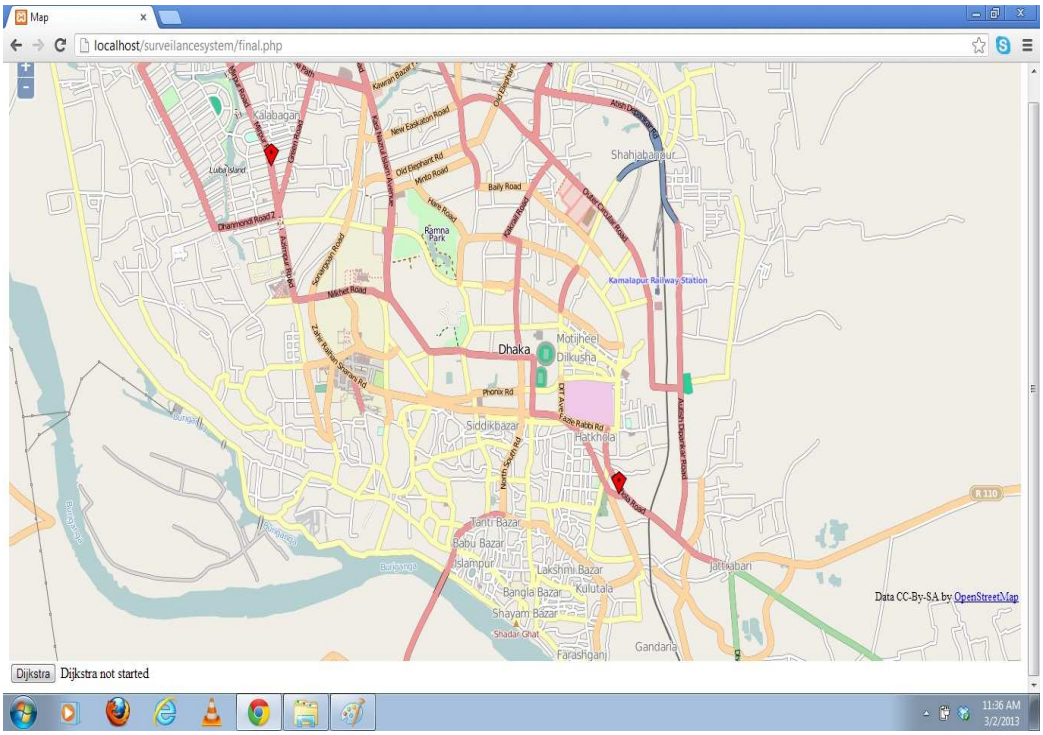


Figure 7.9: Setting source and destination on the Map

4. After somewhile we got a path as the result. In Figure 7.10 the path in blue is the result of Basic Dijkstra's Algorithm. We can see the

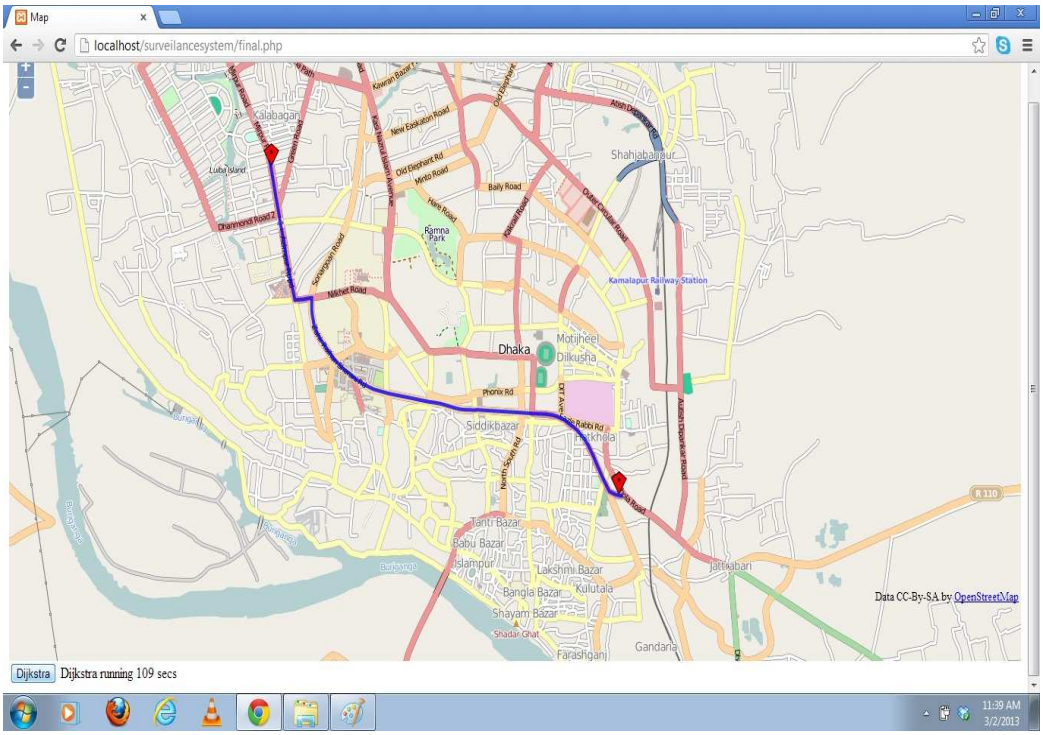


Figure 7.10: Displaying the result

results for Basic Dijkstra's Algorithm and the time is also shown in Figure 7.10. It takes on an average 110.6 seconds (from Figure 7.22), in this case it took 109 seconds to get the path.

Simulation of Basic Dijkstra's Algorithm with Pre-Calculation

We moved on to simulate the fundamental shortest path searching algorithm with some essential pre-calculations. In this case, we loaded the graph once when the system is started and kept the data in $\$\_SESSION$ variable in PHP and used the stored data to run the simulation. The session exists for 24 minutes as the default session expiration duration of PHP is so. In this case it takes 103 seconds (from Figure 7.22) to load the graph on an average and the simulation take only 10 seconds (from Figure 7.22) on an average each time.

This simulation was done by the following five modules:

1. In *LoadGraph.php* we have read the required information from the adjacency matrix and stored them in the $\$\_SESSION$ variable. See code at Appendix K.

2. *PriorityQueue.php* is the same file as mentioned earlier in the previous version. See code at Appendix G.

3. *Dijkstra.php* is the same file as depicted in the previous version. See codes in Appendix H.

4. *RunTest.php* has some slight changes. It gets the data stored in the session and feeds them to Dijkstra.php to get the shortest path from source to the destination input by the user. The code is at Appendix L.

5. *final.php* is the file that handles the User Interface. In this file there is a provision to load the graph initially. See code in Appendix M.

Snapshots of the Simulation of Basic Dijkstra's Algorithm with Pre-Calculation

The step-by-step snapshots of the module to simulate *Basic Dijkstra's Algorithm with Pre-Calculation* we developed in our application are given below:

1. Initially our system has the condition that is shown in Figure 7.11.

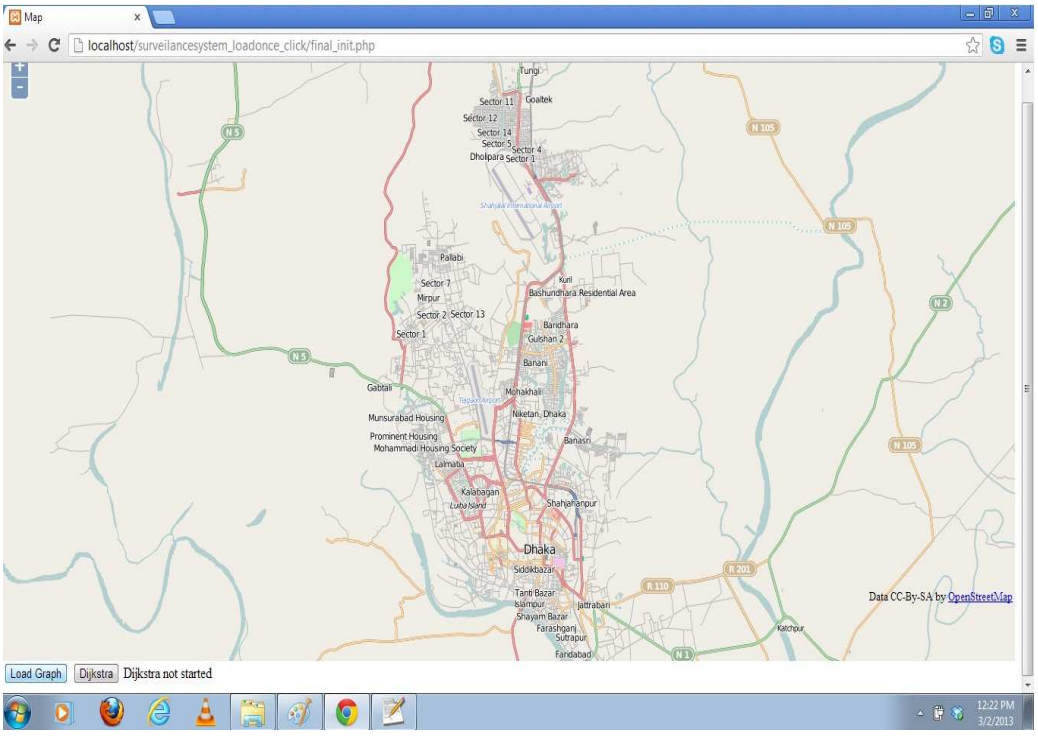


Figure 7.11: Initial display of the Map

2. Then we have to click on the bottom-left "Load Graph" button and after around 103 seconds, we get a picture that is shown in Figure 7.12.

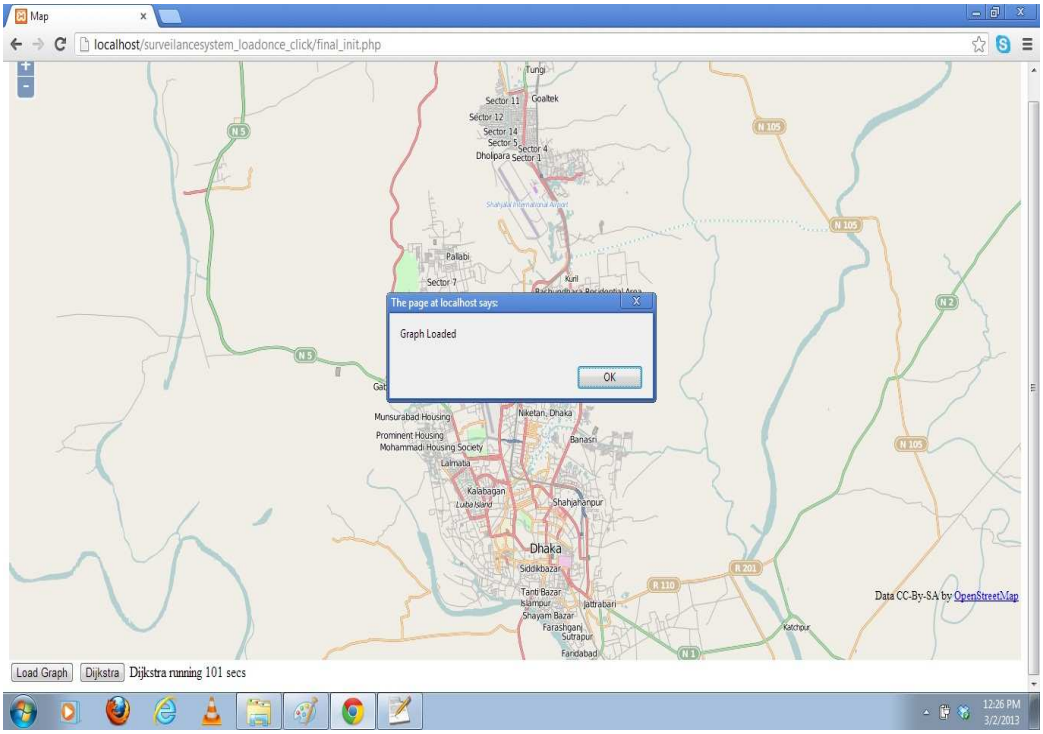


Figure 7.12: Graph is loaded

3. Now we can start the calculation of the shortest path. We begin with setting the source and destination. In Figure 7.13 the state of the application after setting the source and destination is shown.

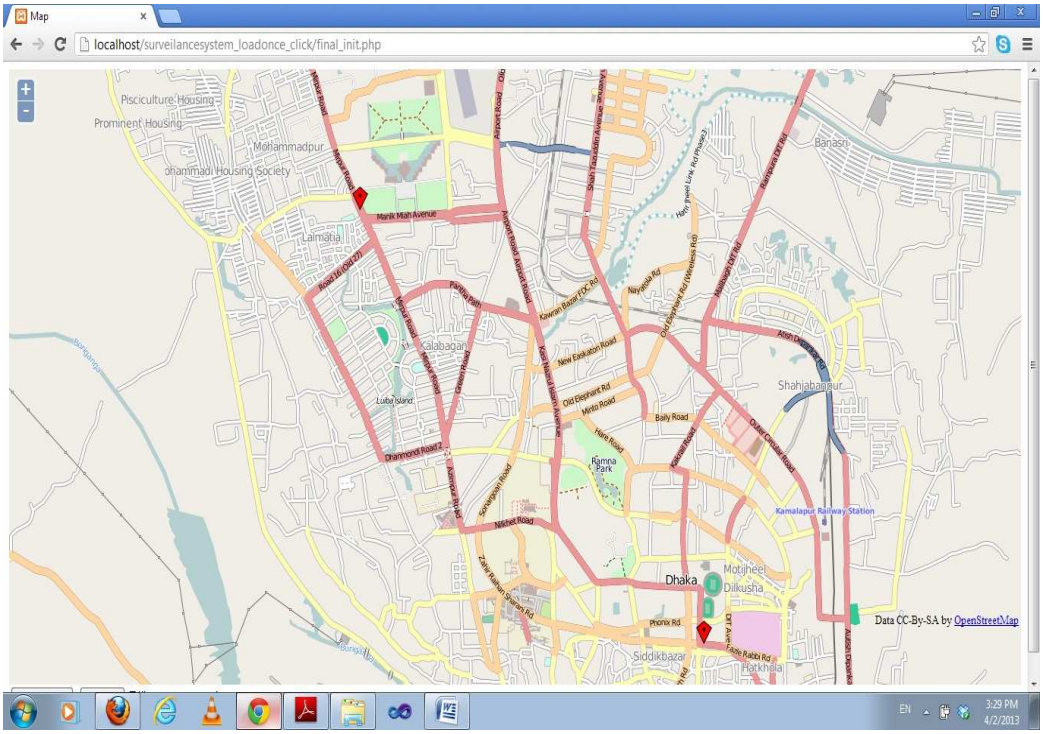


Figure 7.13: Setting the source and the destination

4. Then if we click the button "Dijkstra" the result is shown followed by a prompt that the calculation is ended. The prompt that the calculation is ended is shown in Figure 7.14 and the result is shown in Figure 7.15. In Figure 7.15 the path in blue color is the resulting path.

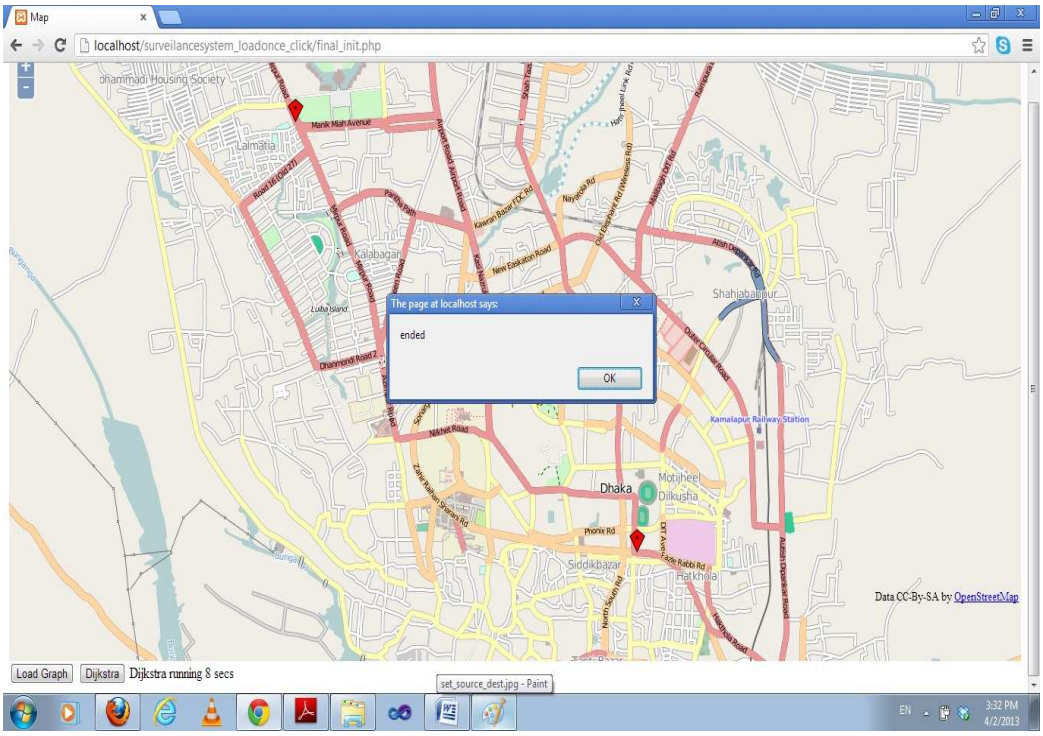


Figure 7.14: Prompt for the end of calculation

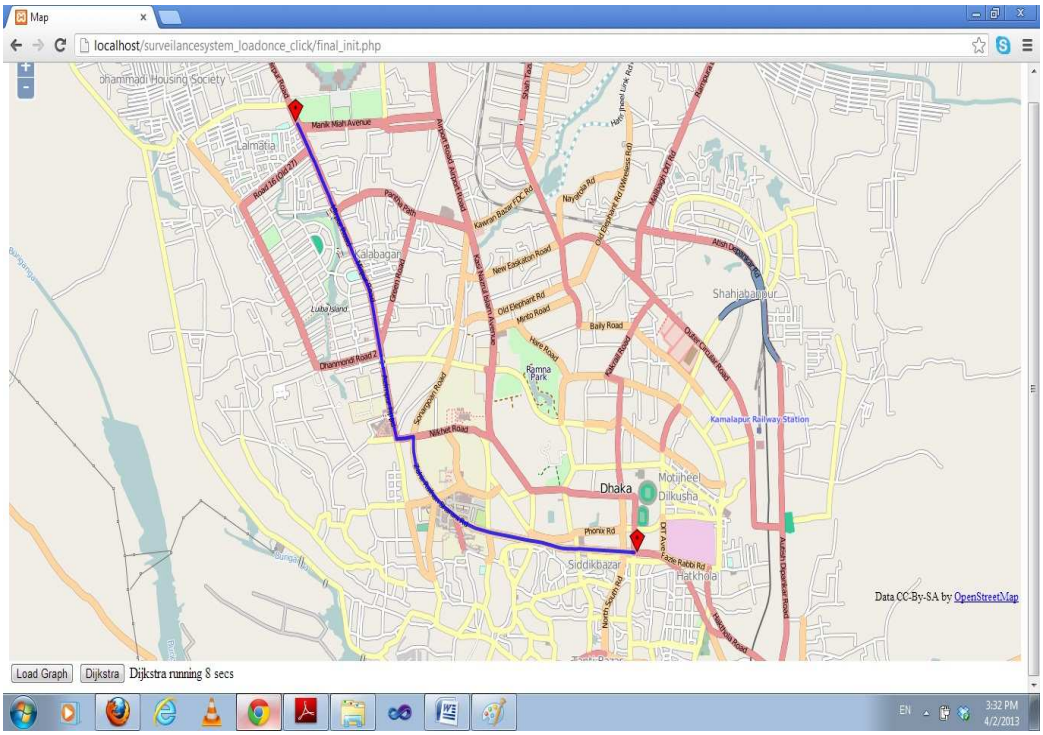


Figure 7.15: Displaying Result

In Figure 7.15 we can see that the time needed is around 10 seconds on an average; 8 seconds in this case.

5. If there is no path, the application shows an output that is shown in Figure 7.16.

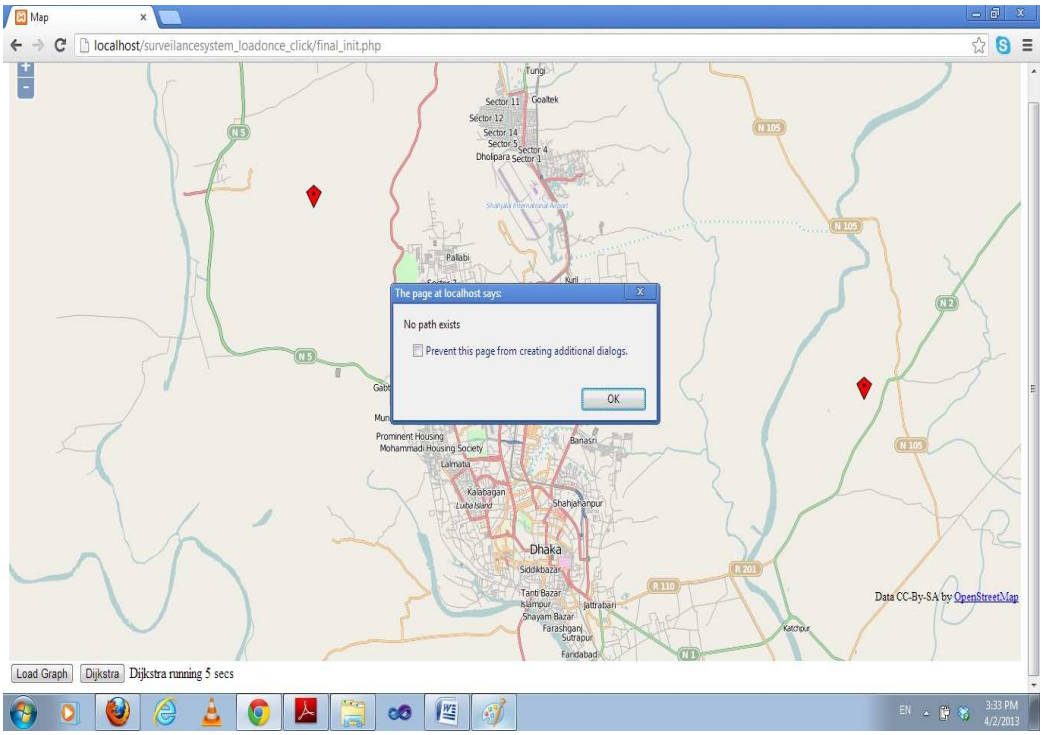


Figure 7.16: Display for no path

Simulation of Improved Dijkstra's Algorithm with Pre-Calculation

We started the work for simulating our own algorithm mentioned earlier in Chapter 6. In this regard, found out another handy process to do the task. We loaded the whole graph in the session variable in this case at the initialization of the application. Then, we took the clicks as input for the source and destination points which is a bit time consuming task. And we run Dijkstra's Algorithm only on a particular circular area of the graph stored in the session. We determined the longitude and latitude of the mid-point of the straight line joining source and destination. Then we took the radius by calculating the Euclidean distance between the mid-point and the source point. Then we fed the system with the graph data only covering the circular area about the mid-point of the pre-determined radius. We fine-tuned the radius by increasing it .001 to get the best and accurate result. In this process, the result is found only in 6 seconds on an average instead of 10 seconds as mentioned in the previous case (from Figure 7.22).
This simulation was done by the following five modules:

1. *LoadGraph.php* is the same as the previous version. See codes in Appendix K.

2. *PriorityQueue.php* is the same as the previous version. See codes in Appendix G.

3. *Dijkstra.php* is the same as the previous version. See codes in Appendix H.

4. *ImprovedRunTest.php* is the file that filters the graph to feed the Dijk-

stra.php in this case. It excludes all the nodes outside of the circular area of our consideration. The code is in Appendix N.

5. The file *Improvedfinal.php* is almost the as the final.php file discussed earlier. The only difference is that, Improvedfinal.php calls the file ImprovedRunTest.php. See codes in Appendix O.

Snapshots of the Simulation of Improved Dijkstra's Algorithm with Pre-Calculation

The step-by-step snapshots of the module to simulate *Improved Dijkstra's Algorithm with Pre-Calculation* we developed in our application are given below:

1. In this case we have to initially load the graph as mentioned earlier in the previous version by clicking the "Load Graph" button. Initial state is shown in Figure 7.17 and the application shows the prompt shown in Figure 7.18 after the graph is loaded.

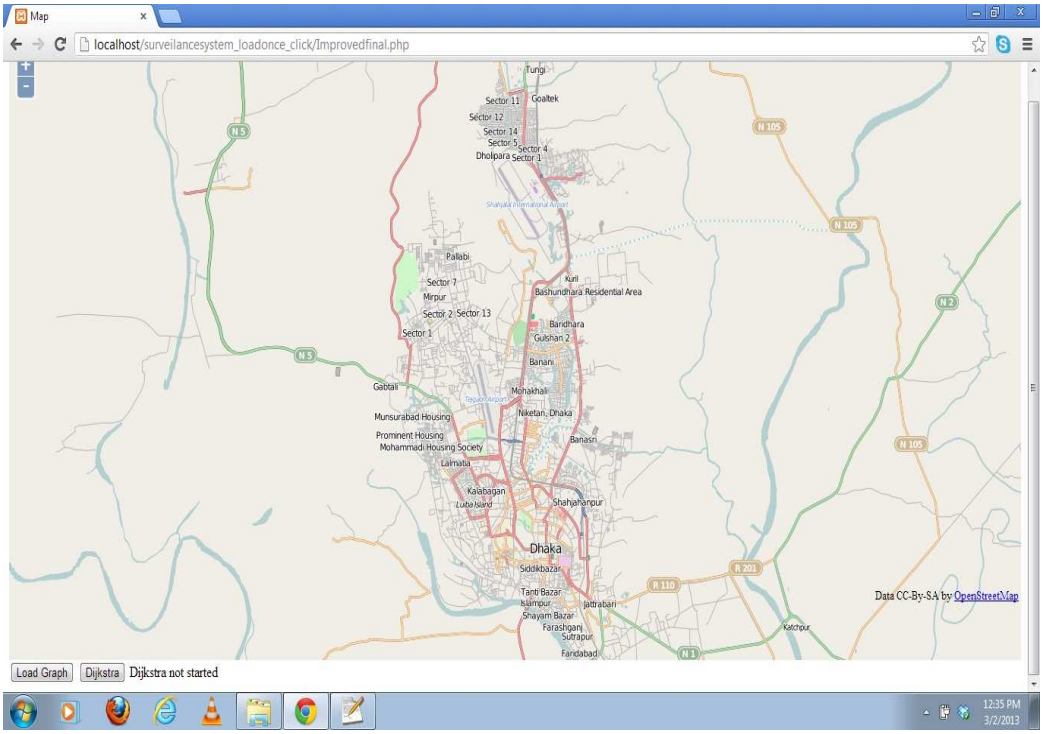


Figure 7.17: Initial display of the Map

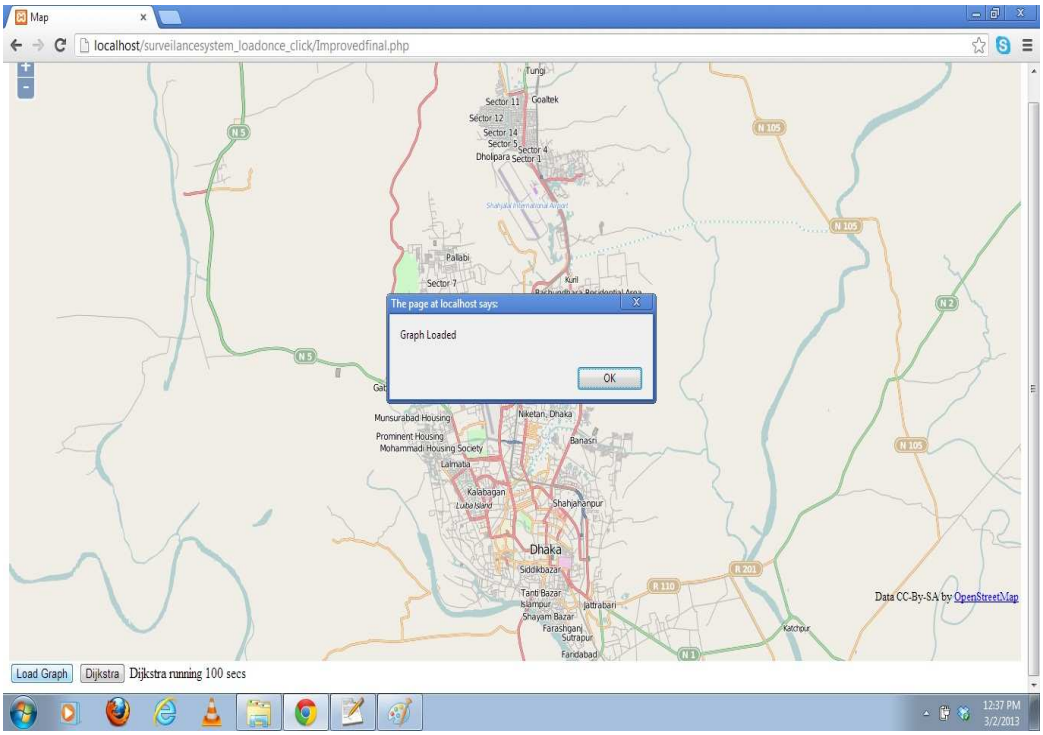


Figure 7.18: Graph is loaded

2. Then After setting the source and destination we get the result by clicking the “Dijkstra” button. The state of the application after setting the source and destination is shown in Figure 7.19. Figure 7.20 shows the prompt that appears after the calculation is done.

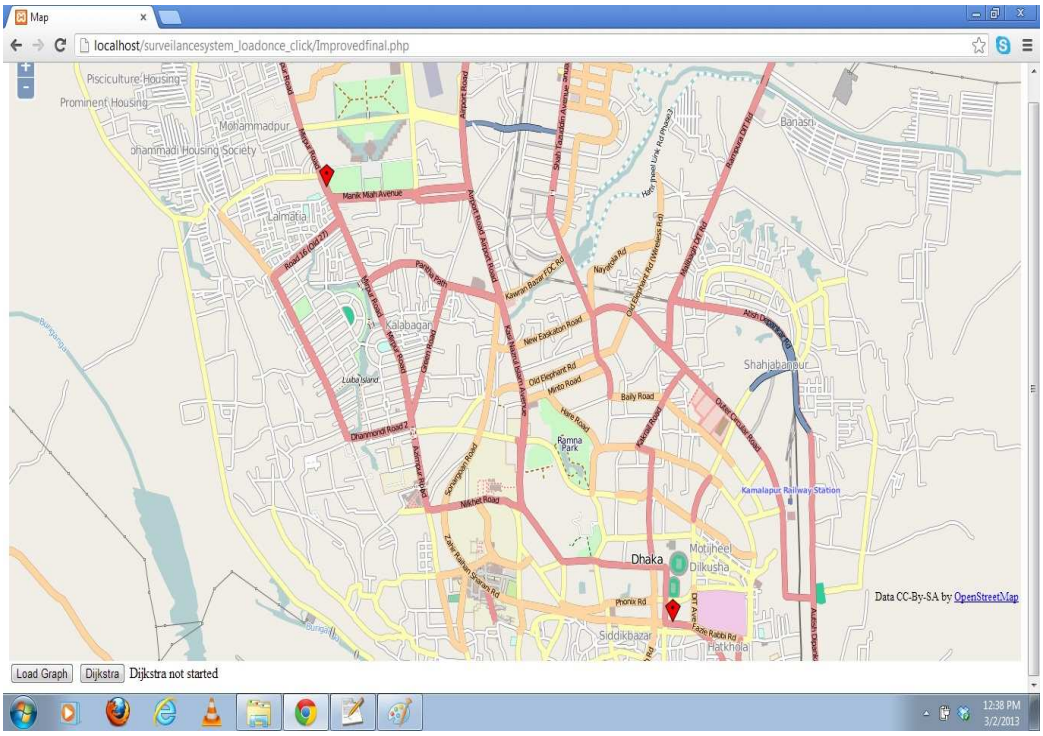


Figure 7.19: Setting source and Destination

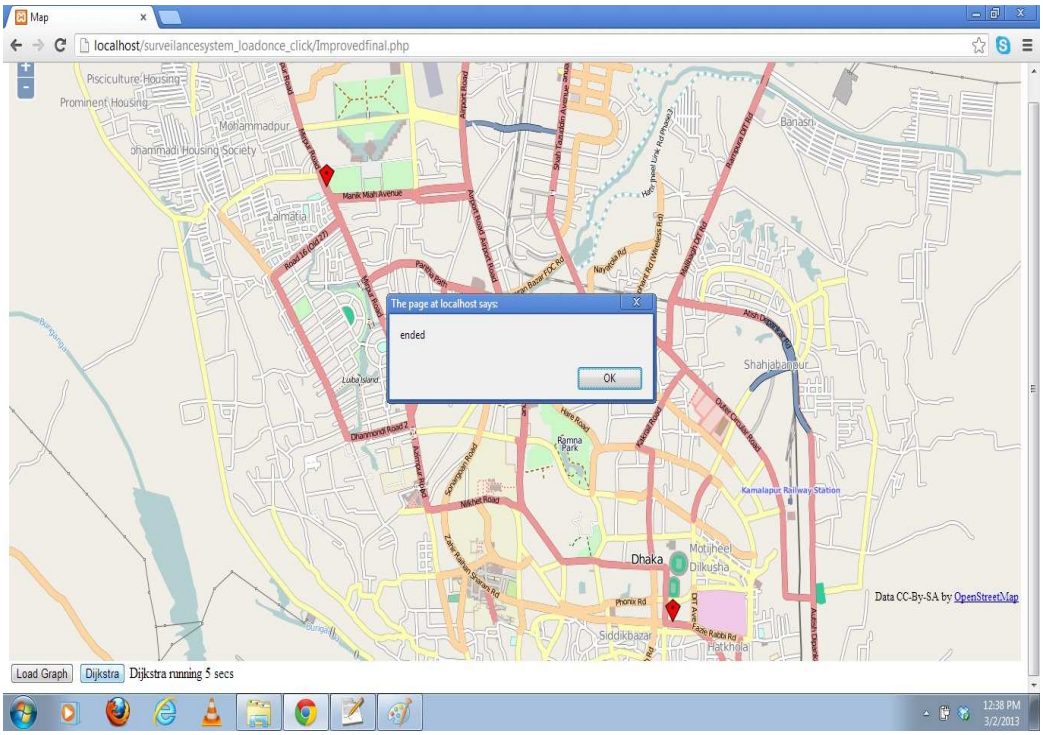


Figure 7.20: Prompt for the end of calculation

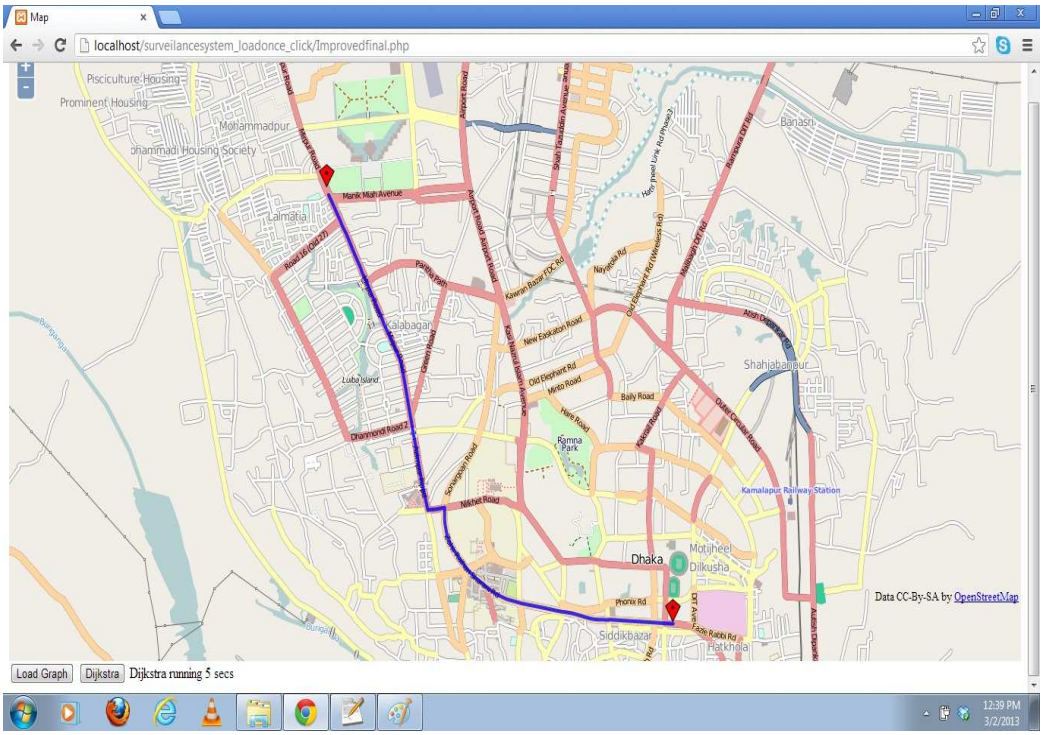


Figure 7.21: Displaying the result

In Figure 7.21 we can see the result path in blue. Figure 7.21 shows that it takes only 5 seconds when we apply this improvization.

Performance Analysis

We have analyzed these three versions and found comparative results for different strategies. Figure 7.22 shows the comparative analysis of defferent strategies with respect to execution time in seconds for 10 different executions of the application. We can see in Figure 7.22 that executing plain Dijkstra's Algorithm we can get the result in 110 seconds on an average, where if we load the graph in 103 seconds initially, we can get the same result in around 10 seconds for the whole graph. In the fourth column in Figure 7.22 we can see the improvement in execution time after applying the improvement of taking only a circular region around the source and destination into consideration.

Normal Dijkstra (in seconds)	Loading Graph (in seconds)	Improvement with Loading (in seconds)	Improvement with loading and circular area (in seconds)
110	103	9	7
113	103	9	6
111	103	11	7
116	103	9	6
106	103	11	6
108	103	14	6
110	103	10	5
109	103	8	5
111	103	9	6
112	103	10	6

Figure 7.22: Comparison table

If we demonstrate a graphical representation of the analysis in Figure 7.22 we can easily understand the improvement shown in Figure 7.23. In Figure 7.23 we have represented each execution by horizontal axis and the execution time is shown in the vertical axis. Figure 7.23 shows that the Plain Dijkstra's Algorithm takes the highest execution time shown by the blue line, it takes 103 seconds to load the graph once shown by the brown line, the green line shows the improved execution time of the application. Finally when we applied the improvement, the application takes very small time to execute that is shown by the purple line in Figure 7.23.

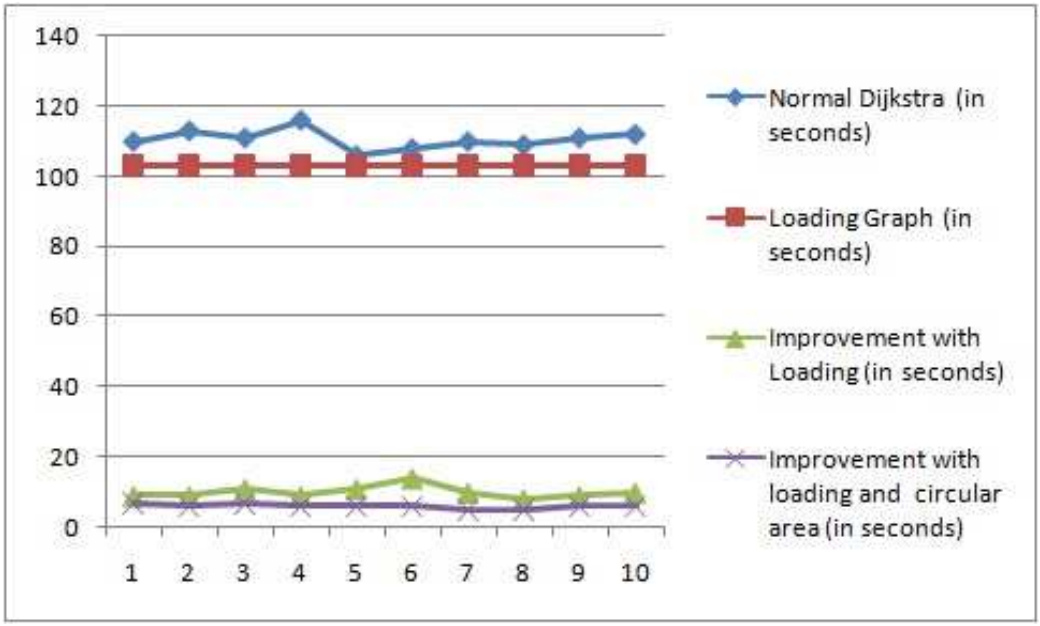


Figure 7.23: Comparison graph

7.2.3 Finding Location for Police-Boxes

We tried another strategy of taking only the 3 or more road crossings from the graph. It took 17 seconds per calculation on an average including the loading of the graph. As in this case, the graph size decreases to only 255 nodes from 42727 nodes. Then we moved on to implement the vertex cover algorithm for finding and demonstrating the positions for the police-boxes. We have run the 2-approxiamtion vertex cover algorithm on the graph data and found 200 nodes that represent the positions for the police-boxes. We constructed a new xml file named common.xml. In this case we have filtered the nodes involved in the residential areas, parks, buildings, tracks, landuses and water-bodies. We built a new adjacency matrix from the 255 nodes found to run the vertex cover algorithm. And in this case, we have used an intuitive idea to find the connectivity between pairs of nodes of common.xml. From map.osm, we found the connectivity between two nodes in map.osm. Using this information we have found the connectivity between the nodes in common.xml. Our strategy is like this:

```
For each pair (x,y) of vertices in common.xml
{
      Run Dijkstra's Algorithm for (x,y) pair on map.osm.
         If no path exists{
              x and y are not connected
         }
         Else{
```

```
                For each node z in common.xml
                {
                       If (x!=z and y!=z){
                           If only z is in the path{
                                x is connected to z and y is connected to z
                           }
                           else if no z in the path{
                                  x and y are connected
                           }
                           else{
                                  no result deducted from this iteration
                           }
                       }
                 }
            }
}
```

This exhaustive method took a lot of time to execute. Nevertheless, our purpose was served and it had to be done only once. Finally we got the common_edgelist.xml file as the adjacency matrix.
The vertex cover part of our system consists of the following files:

1. In *RunTestVertexCover.php*, we have determined the connectivity ma-

trix of the common.xml. The code is given in Appendix P.

2. In *RunVertexCover.php* we have executed the *2-Approximation Vertex Cover Algorithm* on the data from files *common.xml* and *common_edgelist.xml* and constructed the *police_box.xml* that shows the 200 nodes for situating the police-boxes. The code is given in Appendix Q

3. *vertexcoverbefore.php* displays the positions of the nodes representing 3-road crossings by marker layer in osm api. The code is given in Appendix R.

4. *vertexcoverafter.php* shows the positions of the police-boxes by markers. The code is given in See codes in Appendix S.

Snapshots of the Simulation of Vertex Cover

The step-by-step snapshots of the module to simulate *2-Approximation Vertex Cover Algorithm* we developed in our application are given below:

1. We have shown all the vertices in the map by the markers in osmapi. Figure 7.24 is the snapshot showing all the nodes in our application.

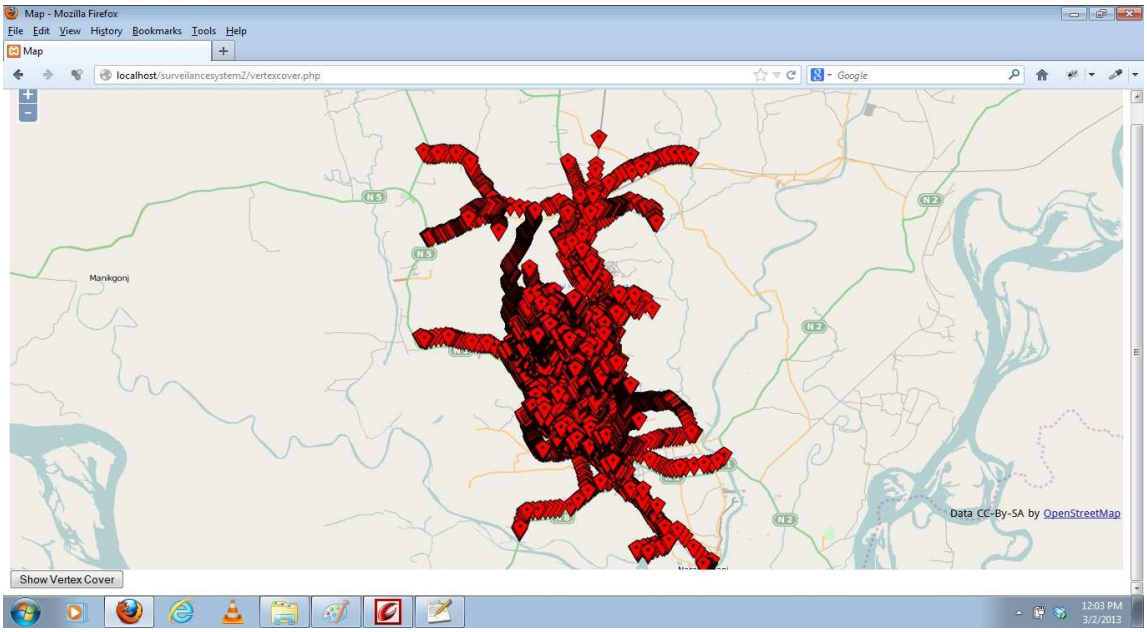


Figure 7.24: All Nodes in Dhaka City

2. After running the code for determining the 3-road crossings' nodes we get a picture like in Figure 7.25. In Figure 7.25 we can see the decreasing the number of nodes to consider compared to that in Figure 7.24.

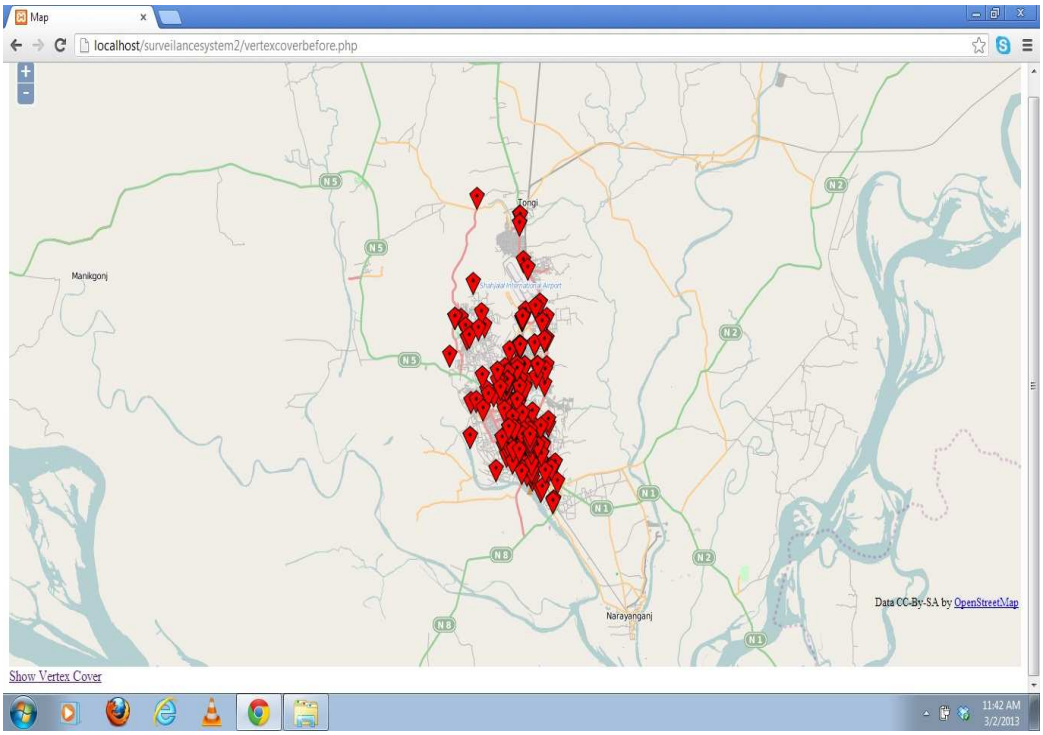


Figure 7.25: 3-road crossing in Dhaka City

3. If we zoom-in the picture to a particular area we get a snapshot that is stated in Figure 7.26.

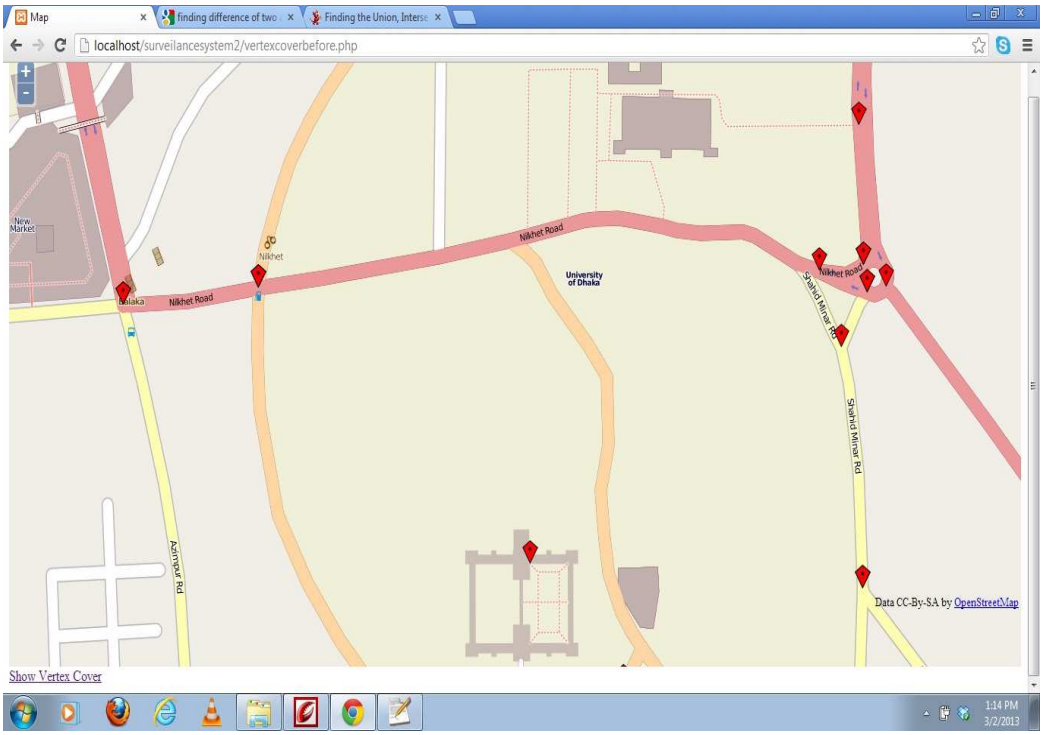


Figure 7.26: After zooming to a particular region

4. Then if we click on Show Vertex Cover link, we get a picture of the area that is shown in Figure 7.27.

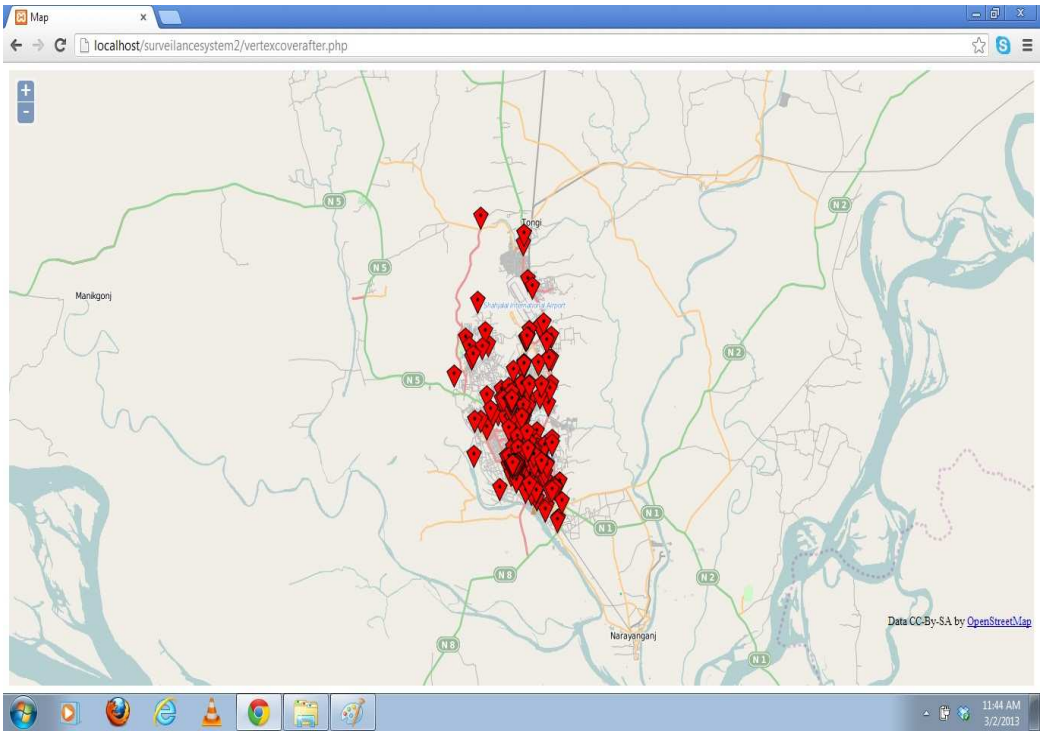


Figure 7.27: Result of Vertex Cover on the map

5. Then zooming in to that particular region shown in Figure 7.26 we get a picture like Figure 7.28 Here in Figure 7.28 we can see a node has

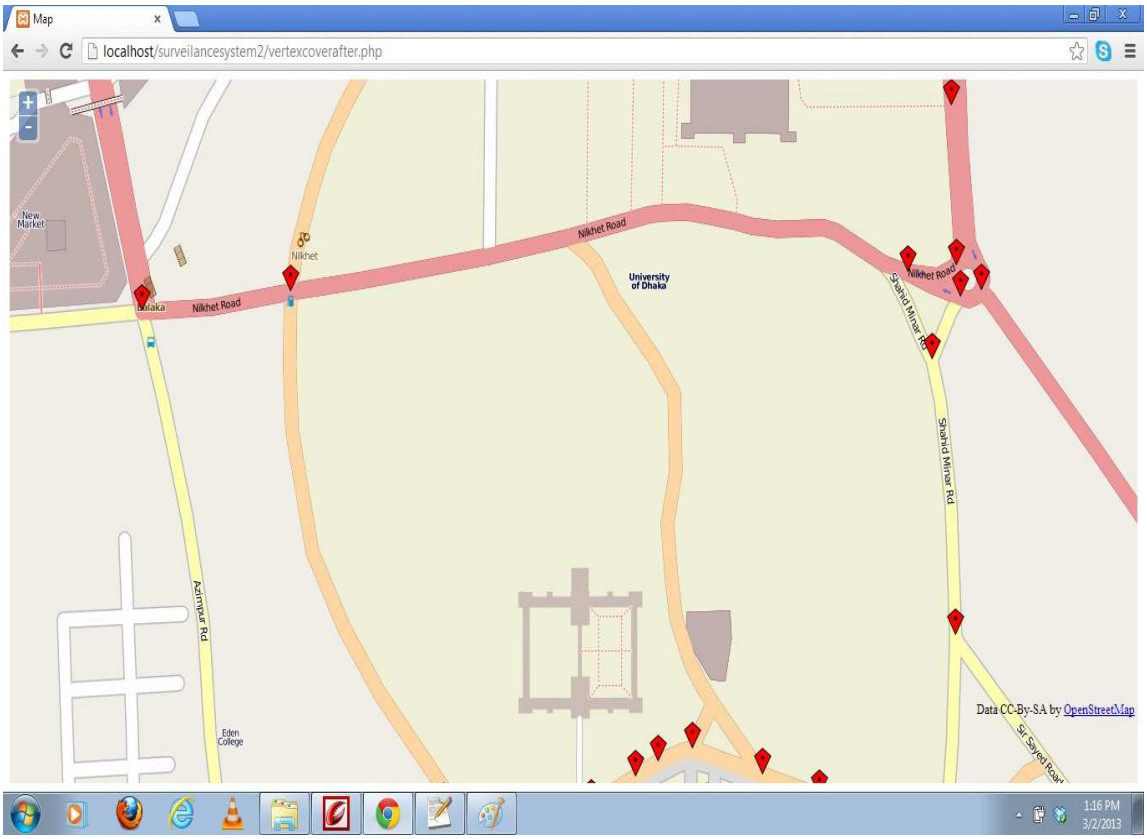


Figure 7.28: Zooming to vertex cover to particular region

been dismissed from the original map in Figure 7.26 by 2-approximate vertex cover algorithm. The markers shown in this case represent the tentative points in the map to situate the police-boxes.

6. Again in this case if we set the source and destinations to run Dijkstra's Algorithm on 255 3-road crossing nodes we get Figure 7.29.

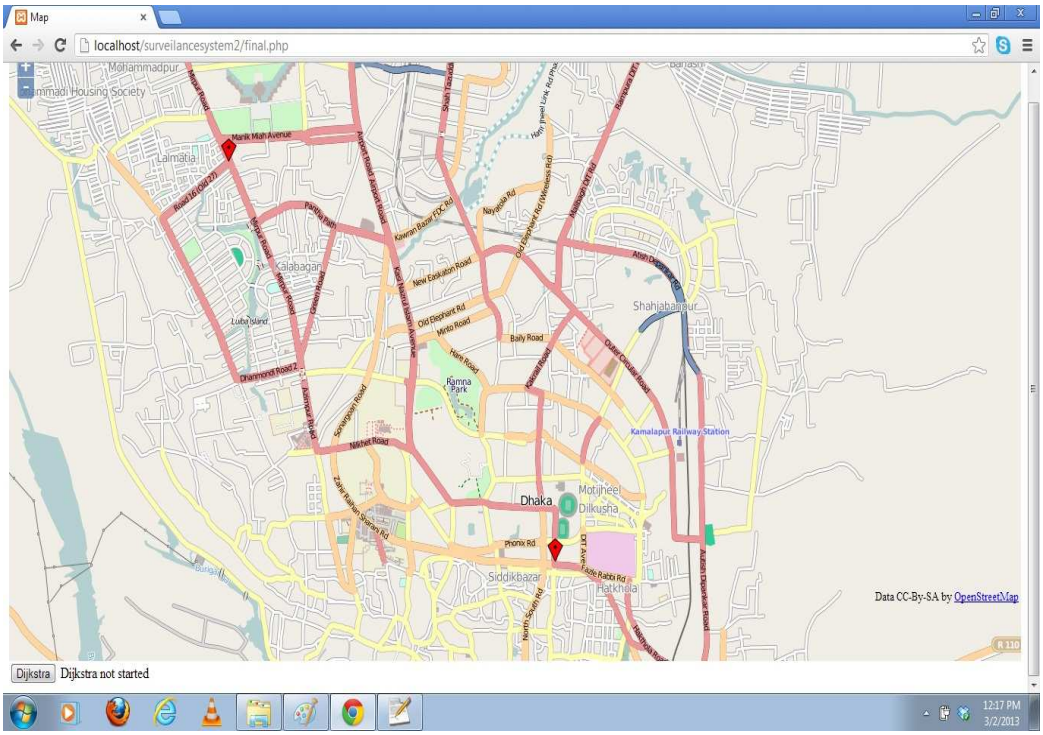


Figure 7.29: Setting the source and destination ons the Map

7. We can see the result including the time for execution in Figure 7.30.

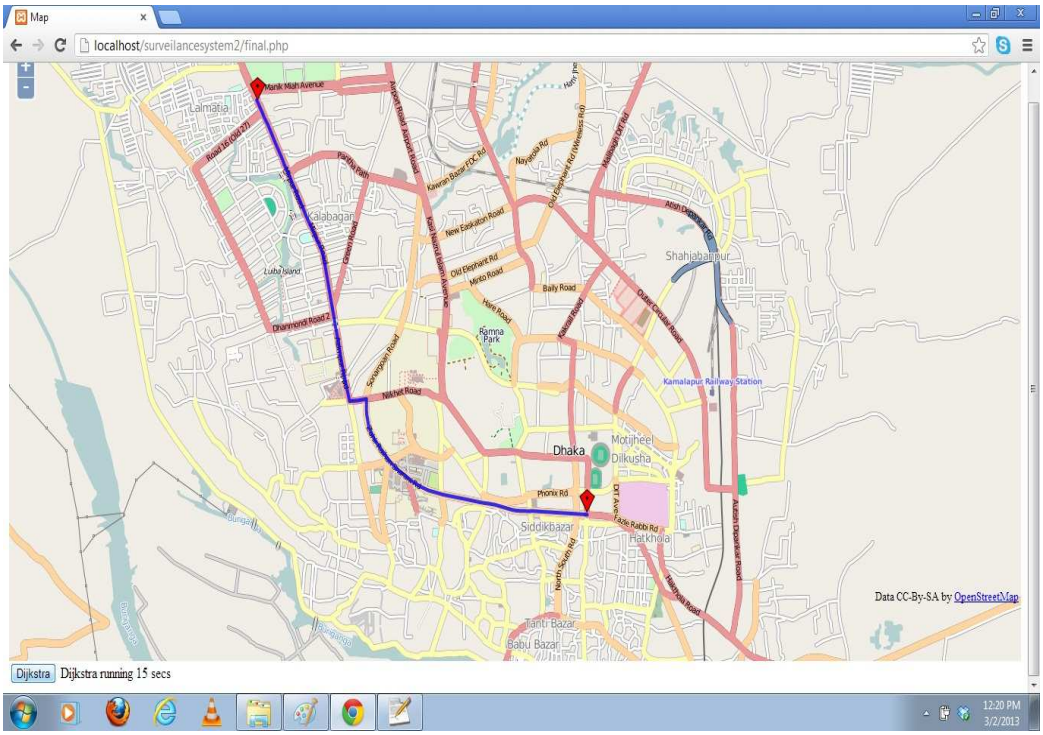


Figure 7.30: Displaying Result

7.2.4 Navigation through Alternative Path

In real world there can be some situations when a road or a path becomes blocked and cannot be used to travel. In this situation we have to use alternative way to reach a destination from a source. Keeping this reality in mind, we developed a module in our thesis to help people navigate through an alternative shortest path when a particular location is blocked.
The following files are present in this module:

1. In LoadGraph.php we have read the required information from the adjacency matrix and stored them in the $\$\_SESSION$ variable. See code at Appendix K.

2. In *PriorityQueue.php* we have developed a data structure to maintain the min heap to make the basic Dijkstra's Algorithm faster. The code for this task is at Appendix G.

3. In *Dijkstra.php* we have implemented the basic Dijkstra's Algorithm using the priority queue. The code goes at Appendix H.

4. *AltRunTest.php* gets the data stored in the session and feeds them to Dijkstra.php to get the shortest path from source to the destination input by the user. Moreover, when a certain node's ID is sent to this file it removes the node from the whole graph and finds the alternative shortest path. The code is at Appendix T.

5. *Altfinal.php* is the file that handles the User Interface. The code for this file is at Appendix U.

Snapshots of the Simulation of Finding Alternative Path

The step-by-step snapshots of the module of *Finding Alternative Path* we developed in our application are given below:

1. Initially our system has the condition that is shown in Figure 7.31.

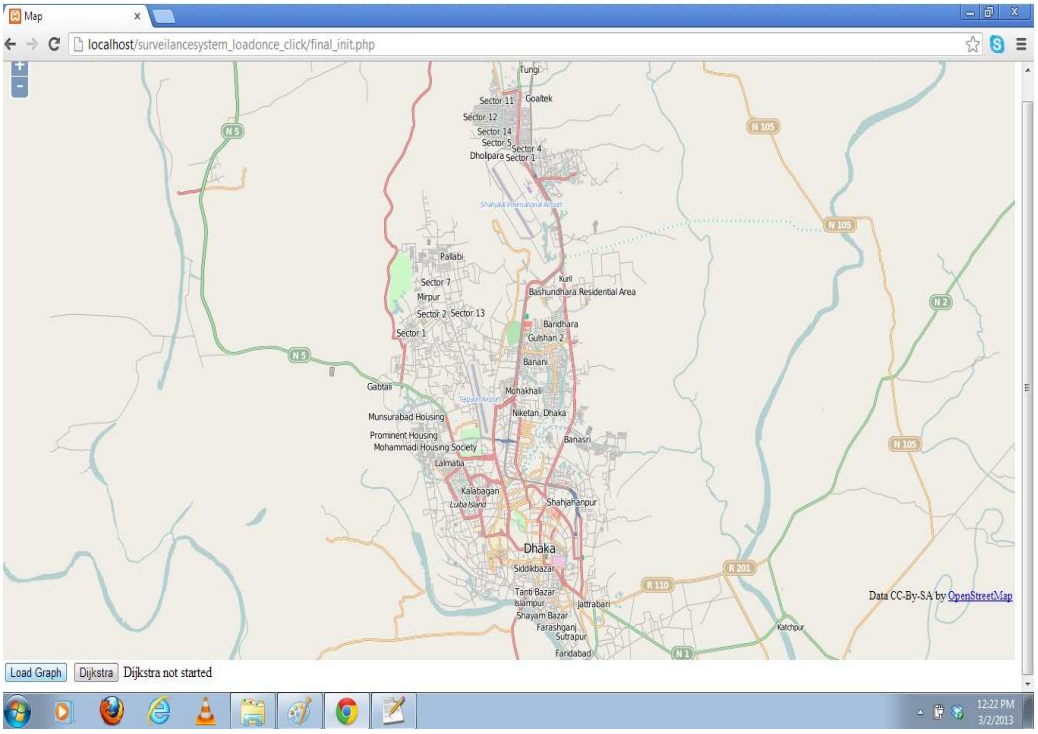


Figure 7.31: Initial display of the Map

2. Then we have to click on the bottom-left "Load Graph" button and after around 103 seconds, we get a picture that is shown in Figure 7.32.

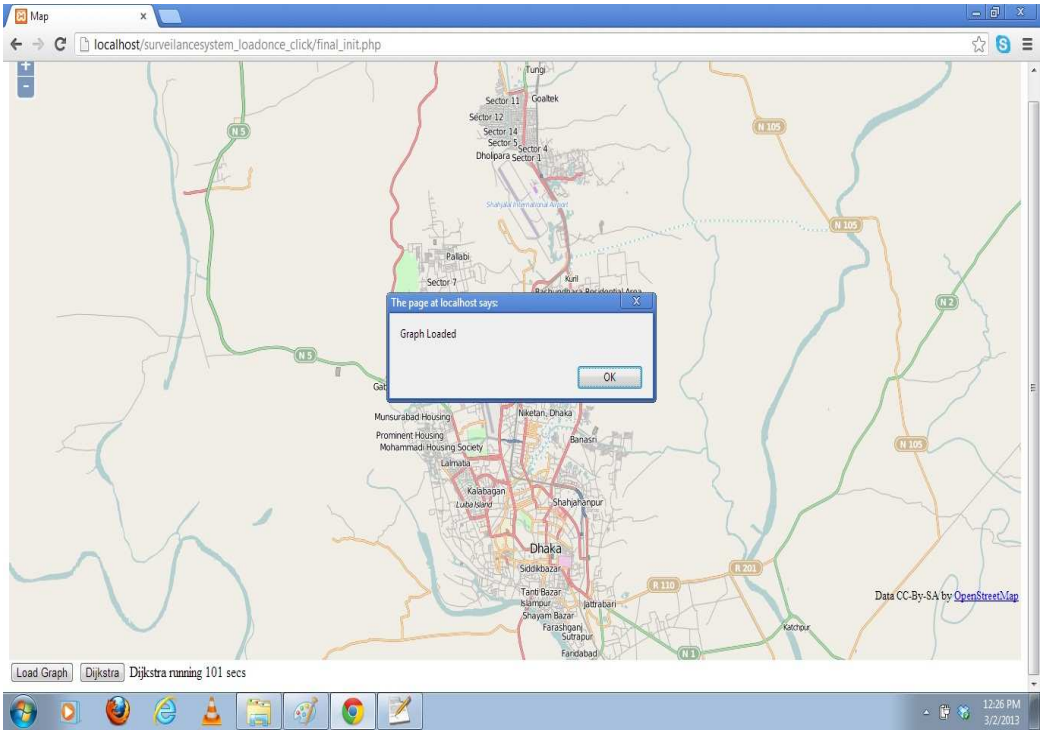


Figure 7.32: Graph is loaded

3. Then if we click the button “Dijkstra” after setting the source and destination the resulted path is shown in blue in Figure 7.33.

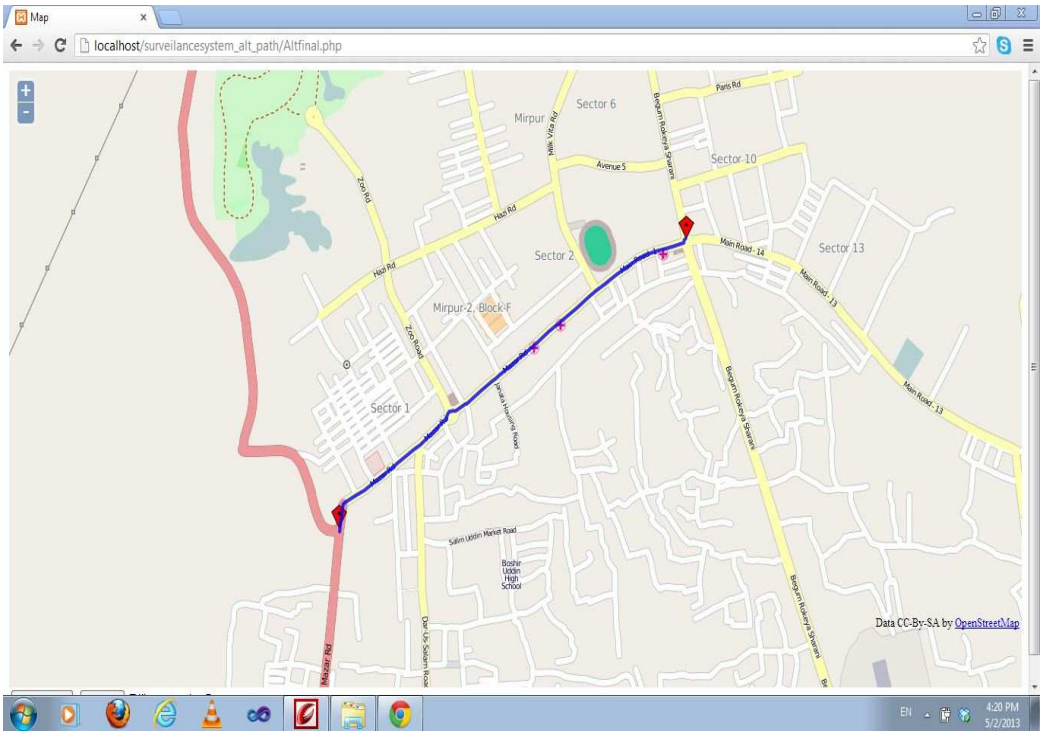


Figure 7.33: Shortest Path between source and destination

4. Now we have to select a node by clicking on the map as shown in Figure 7.34. This operation will take the closest node from the added marker and consider that node as a blocked location.

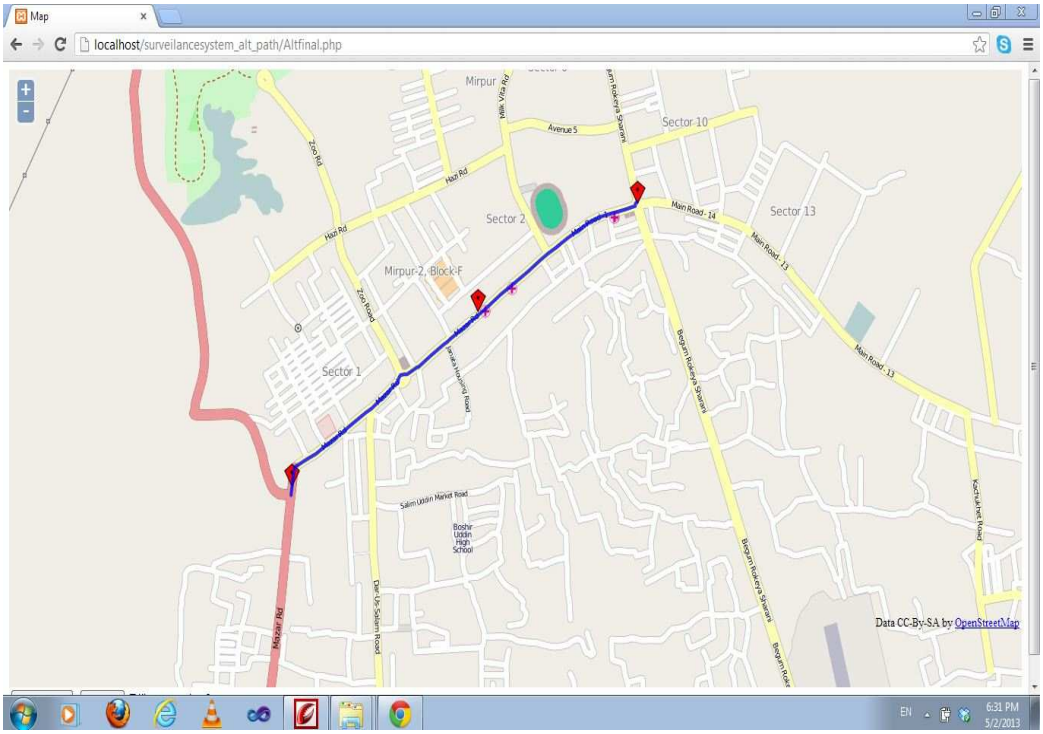


Figure 7.34: Setting the blocked location

5. Then if we click the button "Dijkstra" the application will show the alternative path when a node is blocked. The picture of this resutl is shown in Figure 7.35.

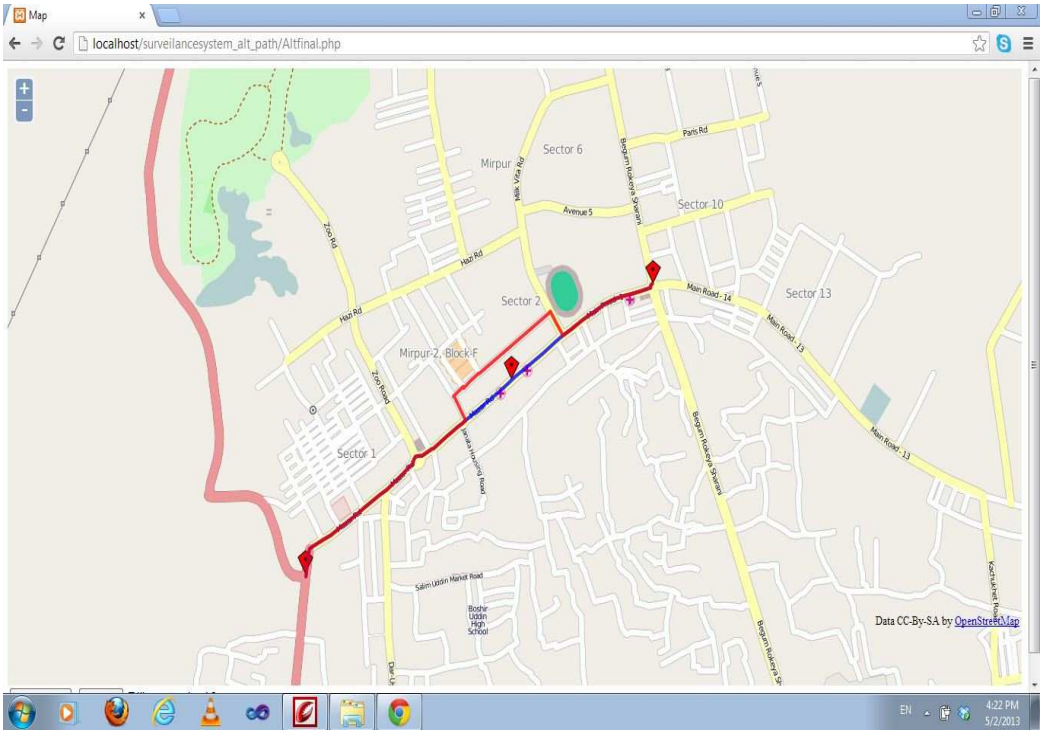


Figure 7.35: Alternate Path between source and destination

Chapter 8

Conclusion

In this thesis, we have studied the map of Dhaka city and performed some analyses on how to represent the map data as graph and depict them in a software application. For the overall surveillance of the city we have developed a simulation of 2-vertex cover algorithm and shown the tentative positions of the police-boxes for covering the whole map. We have two versions of our software i.e. offline and online. The online version is fully developed and the offline version is partially done. However, finally we came out with some ideas on different data structures for keeping the data. And some ideas, strategies and algorithms for shortest path search are generated and simulated in the thesis. Finally we got some remarkable improvements by performing some pre-computation. The comparative study on different strategies is also included. Some techniques are used that are relevant to the special situations that arise in the garbled map of Dhaka city. In Chapter 1 , we have

discussed about the Graph Theory and given a brief description on our thesis. In Chapter 2, we have given some preliminary ideas and definitions on Graph Theory that are used throughout this thesis. In Chapter 3, we have described different techniques for finding the shortest path between pairs of nodes in a graph emphasizing on the Dijkstras Algorithm. In Chapter 4, 2-approximation vertex cover algorithm is discussed in brief. In Chapter 5, basic clustering techniques are depicted and finally we suggested a clustering idea that is relevant to the given situation and preserves connectivity amongst the clusters. In Chapter 6, we have given the details on our algorithm for finding the shortest path between a pair of source and destination nodes of a large graph of 42797 nodes and 46406 edges. It also contains the simulations we developed followed by advantages and drawbacks. In Chapter 7, we have compiled our results and shown the comparative descriptions among them. The following problems are still unsolved by us:

1. Handling this big data structure in .NET framework.
2. Implementing *Algorithm with Clustering and Heuristic* stated in Subsection 6.1.4 of Chapter 6
3. Applying *1.5-Approximation Vertex Cover Algorithm* in finding the position of the police-boxes.
4. Automating the traffic signaling according to the situation.

The thesis is still in an initial stage now, lots of works to be done on this. We intend to do more work on it and find out much better results on this in future.

Bibliography

[Flake04] Gary William Flake and Robert E. Tarjan and Kostas Tsioutsiouliklis, *Graph clustering and minimum cut trees*, journal on Internet Mathematics, 1, pp. 385-408, 2004.

[pregel] Grzegorz Malewicz and Matthew H. Austern and Aart J. C. Bik and James C. Dehnert and Ilan Horn and Naty Leiser and Grzegorz Czajkowski, *Pregel: a system for large-scale graph processing*, 2009.

[S05] Peter Sanders and Dominik Schultes, *Highway Hierarchies Hasten Exact Shortest Path Queries*, Lecture Notes in Computer Science, 3669,Springer, pp. 568-579, 2005.

[D06] Daniel Delling and Martin Holzer and Kirill Mller and Frank Schulz and Dorothea Wagner, *High-performance multi-level graphs*, Proc. of 9th DIMACS IMPLEMENTATION CHALLENGE, pp. 52-65, 2006.

[P07] Sebastian Knopp and Dorothea Wagner and et al., *Computing Many-to-Many Shortest Paths Using Highway Hierarchies*, 2009.

[SCH08] Dominik Schultes, *Route Planning in Road Networks*, 2008.

[Cormen01] Cormen, Thomas H. and Stein, Clifford and Rivest, Ronald L. and Leiserson, Charles E, *Introduction to Algorithms*, Books of Computer Science and Engineering,2nd,McGraw-Hill Higher Education, pp. 580-584,1024-1030, 2001.

Appendix A

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace ImageBoxSample
{
  static class Program
  {
    [STAThread]
    static void Main()
    {
      Application.EnableVisualStyles();
```

```
      Application.SetCompatibleTextRenderingDefault(false);
      Application.Run(new MainForm());
    }
  }
}
```

Appendix B

```
using System;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;
using Thesis.Windows.Forms;

namespace ImageBoxSample
{
  public partial class MainForm : Form
  {
    #region  Public Constructors

    public MainForm()
```

```
    {
      InitializeComponent();

      this.UpdateStatusBar();
    }

    #endregion  Public Constructors

    #region  Event Handlers

    private void imageBox_Paint(object sender, PaintEventArgs e)
    {
     // highlight the image
     // if (showImageRegionToolStripButton.Checked)
     //this.DrawBox(e.Graphics, Color.CornflowerBlue,
//((ImageBox)sender).GetImageViewPort());

     // show the region that will be drawn from the source image
     //if (showSourceImageRegionToolStripButton.Checked)
     //this.DrawBox(e.Graphics, Color.Firebrick, new Rectangle
//(((ImageBox)sender).GetImageViewPort().Location,
```

```
//((ImageBox)sender).GetSourceImageRegion().Size));
    }

    private void imageBox_Scroll(object sender, ScrollEventArgs e)
    {
      this.UpdateStatusBar();
    }

    private void showImageRegionToolStripButton_Click(object sender, EventArgs e)
    {
      imageBox.Invalidate();
    }

    #endregion  Event Handlers

    #region  Private Methods

    private void DrawBox(Graphics graphics, Color color, Rectangle rectangle)
    {
      int offset;
      int penWidth;
```

```
      offset = 9;
      penWidth = 2;

      using (SolidBrush brush = new SolidBrush(Color.FromArgb(64, color)))
        graphics.FillRectangle(brush, rectangle);

      using (Pen pen = new Pen(color, penWidth))
      {
        pen.DashStyle = DashStyle.Dot;
        graphics.DrawLine(pen, rectangle.Left, rectangle.Top - offset, rectangle.
Left, rectangle.Bottom + offset);
        graphics.DrawLine(pen, rectangle.Left + rectangle.Width, rectangle.Top
- offset, rectangle.Left + rectangle.Width, rectangle.Bottom + offset);
        graphics.DrawLine(pen, rectangle.Left - offset, rectangle.Top, rectangle.
Right + offset, rectangle.Top);
        graphics.DrawLine(pen, rectangle.Left - offset, rectangle.Bottom,
rectangle.Right + offset, rectangle.Bottom);
      }
    }
```

```
    #endregion  Private Methods

    private void UpdateStatusBar()
    {
      positionToolStripStatusLabel.Text = imageBox.AutoScrollPosition.ToString();
      imageSizeToolStripStatusLabel.Text = imageBox.GetImageViewPort().ToString();
      zoomToolStripStatusLabel.Text = string.Format("{0}%", imageBox.Zoom);
    }

    private void imageBox_ZoomChanged(object sender, EventArgs e)
    {
      this.UpdateStatusBar();
    }

    private void imageBox_Resize(object sender, EventArgs e)
    {
      this.UpdateStatusBar();
    }
  }
}
```

Appendix C

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;
using System.Xml;
using System.Xml.XPath;

namespace Thesis.Windows.Forms
{
  [DefaultProperty("Image"), ToolboxBitmap(typeof(ImageBox))]
  public partial class ImageBox : ScrollableControl
  {
```

```
#region  Private Class Member Declarations

  private static readonly int MinZoom = 38;
  private static readonly int MaxZoom = 3500;

#endregion  Private Class Member Declarations

#region  Private Member Declarations

  private bool _autoCenter;
  private bool _autoPan;
  private BorderStyle _borderStyle;
  private int _gridCellSize;
  private Color _gridColor;
  private Color _gridColorAlternate;
  [Category("Property Changed")]
  private ImageBoxGridDisplayMode _gridDisplayMode;
  private ImageBoxGridScale _gridScale;
  private Bitmap _gridTile;
  private System.Drawing.Image _image;
  private InterpolationMode _interpolationMode;
```

```
    private bool _invertMouse;
    private bool _isPanning;
    private bool _sizeToFit;
    private Point _startMousePosition;
    private Point _startScrollPosition;
    private TextureBrush _texture;
    private int _zoom;
    private int _zoomIncrement;
    private int draw_x;
    private int draw_y;
    private System.IO.StreamWriter sw1;
    private int id;
  #endregion  Private Member Declarations

  #region  Public Constructors

    public ImageBox()
    {
      InitializeComponent();

      this.SetStyle(ControlStyles.AllPaintingInWmPaint | ControlStyles.UserPaint
```

```
| ControlStyles.OptimizedDoubleBuffer | ControlStyles.ResizeRedraw, true);
    this.SetStyle(ControlStyles.StandardDoubleClick, false);
    this.UpdateStyles();

    this.BackColor = Color.White;
    this.AutoSize = true;
    this.GridScale = ImageBoxGridScale.Small;
    this.GridDisplayMode = ImageBoxGridDisplayMode.Client;
    this.GridColor = Color.Gainsboro;
    this.GridColorAlternate = Color.White;
    this.GridCellSize = 8;
    this.BorderStyle = BorderStyle.FixedSingle;
    this.AutoPan = true;
    this.Zoom = 100;
    this.ZoomIncrement = 20;
    this.InterpolationMode = InterpolationMode.Default;
    this.AutoCenter = true;
    this.draw_x = 1104;
    this.draw_y = 3219;
    String applicationPath = System.IO.Path.GetDirectoryName
(Application.ExecutablePath);
```

```
      ///this.sw1 = new System.IO.StreamWriter(applicationPath + "\\mapXY.txt");
      this.id = 1;
    }

  #endregion  Public Constructors

  #region  Events

    [Category("Property Changed")]
    public event EventHandler AutoCenterChanged;

    [Category("Property Changed")]
    public event EventHandler AutoPanChanged;

    [Category("Property Changed")]
    public event EventHandler BorderStyleChanged;

    [Category("Property Changed")]
    public event EventHandler GridCellSizeChanged;

    [Category("Property Changed")]
```

```
public event EventHandler GridColorAlternateChanged;

[Category("Property Changed")]
public event EventHandler GridColorChanged;

[Category("Property Changed")]
public event EventHandler GridDisplayModeChanged;

[Category("Property Changed")]
public event EventHandler GridScaleChanged;

[Category("Property Changed")]
public event EventHandler ImageChanged;

[Category("Property Changed")]
public event EventHandler InterpolationModeChanged;

[Category("Property Changed")]
public event EventHandler InvertMouseChanged;

[Category("Property Changed")]
```

```
    public event EventHandler PanEnd;

    [Category("Property Changed")]
    public event EventHandler PanStart;

    [Category("Property Changed")]
    public event EventHandler SizeToFitChanged;

    [Category("Property Changed")]
    public event EventHandler ZoomChanged;

    [Category("Property Changed")]
    public event EventHandler ZoomIncrementChanged;

  #endregion  Events

  #region  Overriden Properties

    [Browsable(true), EditorBrowsable(EditorBrowsableState.Always),
DesignerSerializationVisibility(DesignerSerializationVisibility.Visible),
DefaultValue(true)]
```

```
public override bool AutoSize
{
  get { return base.AutoSize; }
  set
  {
    if (base.AutoSize != value)
    {
      base.AutoSize = value;
      this.AdjustLayout();
    }
  }
}

[DefaultValue(typeof(Color), "White")]
public override Color BackColor
{
  get { return base.BackColor; }
  set { base.BackColor = value; }
}

[Browsable(false), EditorBrowsable(EditorBrowsableState.Never),
```

```
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public override Image BackgroundImage
    {
      get { return base.BackgroundImage; }
      set { base.BackgroundImage = value; }
    }

    [Browsable(false), EditorBrowsable(EditorBrowsableState.Never),
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public override ImageLayout BackgroundImageLayout
    {
      get { return base.BackgroundImageLayout; }
      set { base.BackgroundImageLayout = value; }
    }

    [Browsable(false), EditorBrowsable(EditorBrowsableState.Never),
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public override Font Font
    {
      get { return base.Font; }
      set { base.Font = value; }
```

```
    }

    [Browsable(false), EditorBrowsable(EditorBrowsableState.Never),
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public override string Text
    {
      get { return base.Text; }
      set { base.Text = value; }
    }

  #endregion  Overriden Properties

  #region  Public Overridden Methods

    public override Size GetPreferredSize(Size proposedSize)
    {
      Size size;

      if (this.Image != null)
      {
        int width;
```

```
        int height;

        // get the size of the image
        width = this.ScaledImageWidth;
        height = this.ScaledImageHeight;

        // add an offset based on padding
        width += this.Padding.Horizontal;
        height += this.Padding.Vertical;

        // add an offset based on the border style
        width += this.GetBorderOffset();
        height += this.GetBorderOffset();

        size = new Size(width, height);
      }
      else
        size = base.GetPreferredSize(proposedSize);

      return size;
    }
```

```
#endregion  Public Overridden Methods

#region  Protected Overridden Methods

  protected override void Dispose(bool disposing)
  {
    if (disposing)
    {
      if (components != null)
        components.Dispose();

      if (_texture != null)
      {
        _texture.Dispose();
        _texture = null;
      }

      if (_gridTile != null)
      {
        _gridTile.Dispose();
```

```
          _gridTile = null;
        }
      }
      base.Dispose(disposing);
    }

    protected override bool IsInputKey(Keys keyData)
    {
      bool result;

      if ((keyData & Keys.Right) == Keys.Right
  | (keyData & Keys.Left) == Keys.Left
  | (keyData & Keys.Up) == Keys.Up | (keyData & Keys.Down) == Keys.Down)
        result = true;
      else
        result = base.IsInputKey(keyData);

      return result;
    }

    protected override void OnBackColorChanged(EventArgs e)
```

```
{
  base.OnBackColorChanged(e);

  this.Invalidate();
}

protected override void OnDockChanged(EventArgs e)
{
  base.OnDockChanged(e);

  if (this.Dock != DockStyle.None)
    this.AutoSize = false;
}

protected override void OnKeyDown(KeyEventArgs e)
{
  base.OnKeyDown(e);

  switch (e.KeyCode)
  {
    case Keys.Left:
```

```
        this.AdjustScroll(-(e.Modifiers == Keys.None ?
  this.HorizontalScroll.SmallChange : this.HorizontalScroll.LargeChange),
  0);
        break;
      case Keys.Right:
        this.AdjustScroll(e.Modifiers == Keys.None ?
  this.HorizontalScroll.SmallChange : this.HorizontalScroll.LargeChange,
  0);
        break;
      case Keys.Up:
        this.AdjustScroll(0, -(e.Modifiers == Keys.None ?
  this.VerticalScroll.SmallChange : this.VerticalScroll.LargeChange));
        break;
      case Keys.Down:
        this.AdjustScroll(0, e.Modifiers == Keys.None ?
  this.VerticalScroll.SmallChange : this.VerticalScroll.LargeChange);
        break;
    }
  }

  protected override void OnMouseClick(MouseEventArgs e)
```

```
{/*
    if (e.Button == MouseButtons.Left)
    {
        double tx = e.X - this.AutoScrollPosition.X;
        double ty = e.Y - this.AutoScrollPosition.Y;
        this.sw1.WriteLine(this.id + ":" + tx + "," + ty);
    }
    else if (e.Button == MouseButtons.Right)
    {
        sw1.Close();
    }
    this.id++;*/
  //Console.WriteLine(this.AutoScrollPosition.X);
  //double x = e.X-this.AutoScrollPosition.X;
  //double y = e.Y-this.AutoScrollPosition.Y;

  double x = this.draw_x;
  double y = this.draw_y;
  double prev_size_x = 577 + 577*(this.Zoom-10)/10;
  double prev_size_y = 933 + 933 * (this.Zoom - 10) / 10;
```

```
      if (!this.IsPanning && !this.SizeToFit)
      {
        if (e.Button == MouseButtons.Left && Control.ModifierKeys == Keys.None)
        {

          if (this.Zoom >= 100)
            this.Zoom = (int)Math.Round((double)(this.Zoom + 100) / 100) * 100;
          else if (this.Zoom >= 75)
            this.Zoom = 100;
          else
            this.Zoom = (int)(this.Zoom / 0.75F);
        }
        else if (e.Button == MouseButtons.Right
|| (e.Button == MouseButtons.Left && Control.ModifierKeys != Keys.None))
        {
          if (this.Zoom > 100 && this.Zoom <= 125)
            this.Zoom = 100;
          else if (this.Zoom > 100)
            this.Zoom = (int)Math.Round((double)(this.Zoom - 100) / 100) * 100;
          else
            this.Zoom = (int)(this.Zoom * 0.75F);
```

```
      }
    }
    double new_size_x = 577 + 577 * (this.Zoom - 10) / 10;
    double new_size_y = 933 + 933 * (this.Zoom - 10) / 10;
    double x_new = x * (new_size_x / prev_size_x);
    double y_new = y * (new_size_y / prev_size_y);
    this.draw_x = (int)x_new;
    this.draw_y = (int)y_new;
    Console.WriteLine(x_new+" "+y_new);
    //this.AdjustScroll((int)x_new+284, (int)y_new+211);

    base.OnMouseClick(e);
  }

  protected override void OnMouseDown(MouseEventArgs e)
  {
    base.OnMouseDown(e);

    if (!this.Focused)
      this.Focus();
  }
```

```
    protected override void OnMouseMove(MouseEventArgs e)
    {
      base.OnMouseMove(e);

      if (e.Button == MouseButtons.Left && this.AutoPan && this.Image != null)
      {
        if (!this.IsPanning)
        {
          _startMousePosition = e.Location;
          this.IsPanning = true;
        }

        if (this.IsPanning)
        {
          int x;
          int y;
          Point position;

  if (!this.InvertMouse)
  {
```

```
x = -_startScrollPosition.X + (_startMousePosition.X - e.Location.X);
y = -_startScrollPosition.Y + (_startMousePosition.Y - e.Location.Y);
  }
  else
  {
x=-(_startScrollPosition.X + (_startMousePosition.X - e.Location.X));
y=-(_startScrollPosition.Y + (_startMousePosition.Y - e.Location.Y));
  }

          position = new Point(x, y);

          this.UpdateScrollPosition(position);
        }
      }
    }

    protected override void OnMouseUp(MouseEventArgs e)
    {
      base.OnMouseUp(e);

      if (this.IsPanning)
```

```
        this.IsPanning = false;
    }

    protected override void OnMouseWheel(MouseEventArgs e)
    {
      if (!this.SizeToFit)
      {
        int increment;

        if (Control.ModifierKeys == Keys.None)
          increment = this.ZoomIncrement;
        else
          increment = this.ZoomIncrement * 5;

        if (e.Delta < 0)
          increment = -increment;

        this.Zoom += increment;
      }
    }
```

```
    protected override void OnPaddingChanged(System.EventArgs e)
    {
      base.OnPaddingChanged(e);
      this.AdjustLayout();
    }

    protected override void OnPaint(PaintEventArgs e)
    {
      Rectangle innerRectangle;

      // draw the borders
      switch (this.BorderStyle)
      {
        case BorderStyle.FixedSingle:
          ControlPaint.DrawBorder(e.Graphics, this.ClientRectangle,
  this.ForeColor, ButtonBorderStyle.Solid);
          break;
        case BorderStyle.Fixed3D:
          ControlPaint.DrawBorder3D(e.Graphics, this.ClientRectangle,
  Border3DStyle.Sunken);
          break;
```

```
}

innerRectangle = this.GetInsideViewPort();

// draw the background
using (SolidBrush brush = new SolidBrush(this.BackColor))
  e.Graphics.FillRectangle(brush, innerRectangle);

if (_texture!=null && this.GridDisplayMode!=ImageBoxGridDisplayMode.None)
{
  switch (this.GridDisplayMode)
  {
    case ImageBoxGridDisplayMode.Image:
      Rectangle fillRectangle;

      fillRectangle = this.GetImageViewPort();
      e.Graphics.FillRectangle(_texture, fillRectangle);

      if (!fillRectangle.Equals(innerRectangle))
      {
        fillRectangle.Inflate(1, 1);
```

```
            ControlPaint.DrawBorder(e.Graphics, fillRectangle, this.ForeColor,
  ButtonBorderStyle.Solid);
          }
          break;
        case ImageBoxGridDisplayMode.Client:
          e.Graphics.FillRectangle(_texture, innerRectangle);
          break;
      }
    }

    // draw the image
    if (this.Image != null)
      this.DrawImage(e.Graphics);
   // Pen pen1 = new Pen(Color.Red, 2.0f);
    Brush brush1 = Brushes.Red;
    //Brush brush2 = Brushes.Blue;
    Graphics surface = this.CreateGraphics();
    Pen pen1 = new Pen(Color.Blue, 2.0f);//for drawing the path
    String applicationPath = System.IO.Path.GetDirectoryName
  (Application.ExecutablePath);
```

```
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load(applicationPath + "\\map_1.osm");
XmlDocument xmlDoc1 = new XmlDocument();
xmlDoc1.Load(applicationPath + "\\mapGraphXml.xml");
//draw any way region start
/*XmlNodeList xmlnodelist1 = xmlDoc.GetElementsByTagName("way");
for (int i = 0; i < xmlnodelist1.Count; i++)
{
    if (xmlnodelist1[i].Attributes["id"].Value == "33793112")
    {
        //XmlDocument wayDoc = new XmlDocument(xmlnodelist1[i].Name);
        XmlNodeList way = xmlnodelist1[i].ChildNodes;
        double x_new_prev=0,y_new_prev=0;
        for (int j = 0; j < way.Count; j++)
        {
            if (way[j].Name == "nd") {
                XmlNodeList xmlnodelist2 = xmlDoc1.GetElementsByTagName
("node"+way[j].Attributes["ref"].Value);

                double x = Double.Parse(xmlnodelist2[0].Attributes["lat"]
.Value);
```

```
                    double y = Double.Parse(xmlnodelist2[0].Attributes["lon"]
  .Value);
                    Console.WriteLine(x+" "+y);
                    double prev_size_x = 1529;
                    double prev_size_y = 2595;

                    double new_size_x = 1529 + 152 * (this.Zoom - 100) / 10;
                    double new_size_y = 2595 + 260 * (this.Zoom - 100) / 10;
                    double x_new = x * (new_size_x / prev_size_x);
                    double y_new = y * (new_size_y / prev_size_y);
                    //Console.WriteLine(x_new + " " + y_new);
                    //
                    if(j==0)
                      surface.DrawLine(pen1, (int)x_new + this.AutoScrollPositio
(int)y_new + this.AutoScrollPosition.Y,(int)x_new +
this.AutoScrollPosition.X, (int)y_new +
this.AutoScrollPosition.Y);
                    else
                        surface.DrawLine(pen1, (int)x_new_prev +
  this.AutoScrollPosition.X, (int)y_new_prev +
  this.AutoScrollPosition.Y, (int)x_new +
```

```
this.AutoScrollPosition.X, (int)y_new +
this.AutoScrollPosition.Y);


                    //
                    e.Graphics.FillEllipse(brush1,
(int)x_new + this.AutoScrollPosition.X,
(int)y_new + this.AutoScrollPosition.Y,
(int)(10 * new_size_x / prev_size_x),
(int)(10 * new_size_y / prev_size_y));
                    x_new_prev = x_new;
                    y_new_prev = y_new;
                };
            }


        }
    }
    Console.WriteLine("Finish");*/
    System.IO.StreamReader sr2 = new System.IO.StreamReader
(applicationPath + "\\mapXY_new.txt");
    String vertexInfo;
      //**********
```

```
    while ((vertexInfo = sr2.ReadLine()) != null)
    {
        String[] splitter1, splitter2;
        splitter1 = vertexInfo.Split(':');

        splitter2 = splitter1[1].Split(',');
        //int i = Int16.Parse(splitter1[0]);
        double x = Double.Parse(splitter2[0]);
        double y = Double.Parse(splitter2[1]);
        double prev_size_x = 1529;
        double prev_size_y = 2595;

        double new_size_x = 1529 + 152 * (this.Zoom - 100) / 10;
        double new_size_y = 2595 + 260 * (this.Zoom - 100) / 10;
        double x_new = x * (new_size_x / prev_size_x);
        double y_new = y * (new_size_y / prev_size_y);
        //Console.WriteLine(x_new + " " + y_new);
        e.Graphics.FillEllipse(brush1, (int)x_new + this.AutoScrollPosition.X,
(int)y_new + this.AutoScrollPosition.Y,
(int)(10 * new_size_x / prev_size_x),
(int)(10 * new_size_y / prev_size_y));
```

```
      }
      //draw the whole graph from textfile end
        //******
    //draw the result path start
      /*System.IO.StreamReader resultReader =
  new System.IO.StreamReader(applicationPath + "\\result.txt");
      String node = "";
      Double x_new_prev=0, y_new_prev=0;
      int delay_counter = 0;
      int j = 0;
      while ((node = resultReader.ReadLine()) != null)
      {
          XmlNodeList xmlnodelist2 = xmlDoc1.GetElementsByTagName("node" + node);

          double x = Double.Parse(xmlnodelist2[0].Attributes["lat"].Value);
          double y = Double.Parse(xmlnodelist2[0].Attributes["lon"].Value);

          double prev_size_x = 1529;
          double prev_size_y = 2595;

          double new_size_x = 1529 + 152 * (this.Zoom - 100) / 10;
```

```
            double new_size_y = 2595 + 260 * (this.Zoom - 100) / 10;
            double x_new = x * (new_size_x / prev_size_x);
            double y_new = y * (new_size_y / prev_size_y);
            //Console.WriteLine(x_new + " " + y_new);
            if(j>0)
                surface.DrawLine(pen1, (int)x_new_prev + this.AutoScrollPosition.X,
  (int)y_new_prev + this.AutoScrollPosition.Y,
  (int)x_new + this.AutoScrollPosition.X, (int)y_new +
  this.AutoScrollPosition.Y);
            //surface.Dispose();
            //
            x_new_prev = x_new;
            y_new_prev = y_new;
            j++;

        } */
      //draw the result path end


        base.OnPaint(e);
      }
```

```
protected override void OnParentChanged(System.EventArgs e)
{
  base.OnParentChanged(e);
  this.AdjustLayout();
}

protected override void OnResize(EventArgs e)
{
  this.AdjustLayout();

  base.OnResize(e);
}

protected override void OnScroll(ScrollEventArgs se)
{
  this.Invalidate();

  base.OnScroll(se);
}
```

```
  #endregion  Protected Overridden Methods


  #region  Public Methods


    public virtual Rectangle GetImageViewPort()
    {
      Rectangle viewPort;


      if (this.Image != null)
      {
        Rectangle innerRectangle;
        Point offset;


        innerRectangle = this.GetInsideViewPort();


        if (this.AutoCenter)
        {
          int x;
          int y;


          x = !this.HScroll ? (innerRectangle.Width -
```

```
  (this.ScaledImageWidth + this.Padding.Horizontal)) / 2 : 0;
          y = !this.VScroll ? (innerRectangle.Height -
  (this.ScaledImageHeight + this.Padding.Vertical)) / 2 : 0;

          offset = new Point(x, y);
        }
        else
          offset = Point.Empty;

        viewPort = new Rectangle(offset.X + innerRectangle.Left +
this.Padding.Left, offset.Y + innerRectangle.Top + this.Padding.Top,
innerRectangle.Width - (this.Padding.Horizontal + (offset.X * 2)),
innerRectangle.Height - (this.Padding.Vertical + (offset.Y * 2)));
      }
      else
        viewPort = Rectangle.Empty;

      return viewPort;
    }

    public Rectangle GetInsideViewPort()
```

```
{
  return this.GetInsideViewPort(false);
}

public virtual Rectangle GetInsideViewPort(bool includePadding)
{
  int left;
  int top;
  int width;
  int height;
  int borderOffset;

  borderOffset = this.GetBorderOffset();
  left = borderOffset;
  top = borderOffset;
  width = this.ClientSize.Width - (borderOffset * 2);
  height = this.ClientSize.Height - (borderOffset * 2);

  if (includePadding)
  {
    left += this.Padding.Left;
```

```
      top += this.Padding.Top;
      width -= this.Padding.Horizontal;
      height -= this.Padding.Vertical;
    }

    return new Rectangle(left, top, width, height);
  }

  public virtual Rectangle GetSourceImageRegion()
  {
    int sourceLeft;
    int sourceTop;
    int sourceWidth;
    int sourceHeight;
    Rectangle viewPort;
    Rectangle region;

    if (this.Image != null)
    {
      viewPort = this.GetImageViewPort();
      sourceLeft = (int)(-this.AutoScrollPosition.X / this.ZoomFactor);
```

```
        sourceTop = (int)(-this.AutoScrollPosition.Y / this.ZoomFactor);
        sourceWidth = (int)(viewPort.Width / this.ZoomFactor);
        sourceHeight = (int)(viewPort.Height / this.ZoomFactor);

        region = new Rectangle(sourceLeft, sourceTop, sourceWidth, sourceHeight);
      }
      else
        region = Rectangle.Empty;

      return region;
    }

    public virtual void ZoomToFit()
    {
      if (this.Image != null)
      {
        Rectangle innerRectangle;
        double zoom;
        double aspectRatio;

        this.AutoScrollMinSize = Size.Empty;
```

```
        innerRectangle = this.GetInsideViewPort(true);

        if (this.Image.Width > this.Image.Height)
        {
          aspectRatio = ((double)innerRectangle.Width) /
  ((double)this.Image.Width);
          zoom = aspectRatio * 100.0;

          if (innerRectangle.Height < ((this.Image.Height * zoom) / 100.0))
          {
            aspectRatio = ((double)innerRectangle.Height) /
((double)this.Image.Height);
            zoom = aspectRatio * 100.0;
          }
        }
        else
        {
          aspectRatio = ((double)innerRectangle.Height) /
  ((double)this.Image.Height);
          zoom = aspectRatio * 100.0;
```

```
          if (innerRectangle.Width < ((this.Image.Width * zoom) / 100.0))
          {
            aspectRatio = ((double)innerRectangle.Width) /
((double)this.Image.Width);
            zoom = aspectRatio * 100.0;
          }
        }

        this.Zoom = (int)Math.Round(Math.Floor(zoom));
      }
    }

  #endregion  Public Methods

  #region  Public Properties

    [DefaultValue(true), Category("Appearance")]
    public bool AutoCenter
    {
      get { return _autoCenter; }
```

```
    set
    {
      if (_autoCenter != value)
      {
        _autoCenter = value;
        this.OnAutoCenterChanged(EventArgs.Empty);
      }
    }
  }

  [DefaultValue(true), Category("Behavior")]
  public bool AutoPan
  {
    get { return _autoPan; }
    set
    {
      if (_autoPan != value)
      {
        _autoPan = value;
        this.OnAutoPanChanged(EventArgs.Empty);
```

```
          if (value)
            this.SizeToFit = false;
        }
      }
    }

    [Browsable(false), EditorBrowsable(EditorBrowsableState.Never),
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public new Size AutoScrollMinSize
    {
      get { return base.AutoScrollMinSize; }
      set { base.AutoScrollMinSize = value; }
    }

    [Category("Appearance"), DefaultValue(typeof(BorderStyle), "FixedSingle")]
    public BorderStyle BorderStyle
    {
      get { return _borderStyle; }
      set
      {
        if (_borderStyle != value)
```

```
      {
        _borderStyle = value;
        this.OnBorderStyleChanged(EventArgs.Empty);
      }
    }
  }

  [Category("Appearance"), DefaultValue(8)]
  public int GridCellSize
  {
    get { return _gridCellSize; }
    set
    {
      if (_gridCellSize != value)
      {
        _gridCellSize = value;
        this.OnGridCellSizeChanged(EventArgs.Empty);
      }
    }
  }
```

```
[Category("Appearance"), DefaultValue(typeof(Color), "Gainsboro")]
public Color GridColor
{
  get { return _gridColor; }
  set
  {
    if (_gridColor != value)
    {
      _gridColor = value;
      this.OnGridColorChanged(EventArgs.Empty);
    }
  }
}

[Category("Appearance"), DefaultValue(typeof(Color), "White")]
public Color GridColorAlternate
{
  get { return _gridColorAlternate; }
  set
  {
    if (_gridColorAlternate != value)
```

```
        {
          _gridColorAlternate = value;
          this.OnGridColorAlternateChanged(EventArgs.Empty);
        }
      }
    }

    [DefaultValue(ImageBoxGridDisplayMode.Client), Category("Appearance")]
    public ImageBoxGridDisplayMode GridDisplayMode
    {
      get { return _gridDisplayMode; }
      set
      {
        if (_gridDisplayMode != value)
        {
          _gridDisplayMode = value;
          this.OnGridDisplayModeChanged(EventArgs.Empty);
        }
      }
    }
```

```
[DefaultValue(typeof(ImageBoxGridScale), "Small"), Category("Appearance")]
public ImageBoxGridScale GridScale
{
  get { return _gridScale; }
  set
  {
    if (_gridScale != value)
    {
      _gridScale = value;
      this.OnGridScaleChanged(EventArgs.Empty);
    }
  }
}

[Category("Appearance"), DefaultValue(null)]
public virtual Image Image
{
  get { return _image; }
  set
  {
    if (_image != value)
```

```
      {
        _image = value;
        this.OnImageChanged(EventArgs.Empty);
      }
    }
  }

  [DefaultValue(InterpolationMode.Default), Category("Appearance")]
  public InterpolationMode InterpolationMode
  {
    get { return _interpolationMode; }
    set
    {
      if (value == InterpolationMode.Invalid)
        value = InterpolationMode.Default;

      if (_interpolationMode != value)
      {
        _interpolationMode = value;
        this.OnInterpolationModeChanged(EventArgs.Empty);
      }
```

```
      }
    }

    [DefaultValue(false), Category("Behavior")]
    public bool InvertMouse
    {
      get { return _invertMouse; }
      set
      {
        if (_invertMouse != value)
        {
          _invertMouse = value;
          this.OnInvertMouseChanged(EventArgs.Empty);
        }
      }
    }

    [DefaultValue(false), DesignerSerializationVisibility
(DesignerSerializationVisibility.Hidden), Browsable(false)]
    public bool IsPanning
    {
```

```
    get { return _isPanning; }
    protected set
    {
      if (_isPanning != value)
      {
        _isPanning = value;
        _startScrollPosition = this.AutoScrollPosition;

        if (value)
        {
          this.Cursor = Cursors.SizeAll;
          this.OnPanStart(EventArgs.Empty);
        }
        else
        {
          this.Cursor = Cursors.Default;
          this.OnPanEnd(EventArgs.Empty);
        }
      }
    }
  }
```

```
[DefaultValue(false), Category("Appearance")]
public bool SizeToFit
{
  get { return _sizeToFit; }
  set
  {
    if (_sizeToFit != value)
    {
      _sizeToFit = value;
      this.OnSizeToFitChanged(EventArgs.Empty);

      if (value)
        this.AutoPan = false;
    }
  }
}

[DefaultValue(100), Category("Appearance")]
public int Zoom
{
```

```
    get { return _zoom; }
    set
    {
      if (value < ImageBox.MinZoom)
        value = ImageBox.MinZoom;
      else if (value > ImageBox.MaxZoom)
        value = ImageBox.MaxZoom;

      if (_zoom != value)
      {
        _zoom = value;
        this.OnZoomChanged(EventArgs.Empty);
      }
    }
  }

  [DefaultValue(20), Category("Behavior")]
  public int ZoomIncrement
  {
    get { return _zoomIncrement; }
    set
```

```
      {
        if (_zoomIncrement != value)
        {
          _zoomIncrement = value;
          this.OnZoomIncrementChanged(EventArgs.Empty);
        }
      }
    }

  #endregion  Public Properties

  #region  Private Methods

    private int GetBorderOffset()
    {
      int offset;

      switch (this.BorderStyle)
      {
        case BorderStyle.Fixed3D:
          offset = 2;
```

```
        break;
      case BorderStyle.FixedSingle:
        offset = 1;
        break;
      default:
        offset = 0;
        break;
    }

    return offset;
  }

  private void InitializeGridTile()
  {
    if (_texture != null)
      _texture.Dispose();

    if (_gridTile != null)
      _gridTile.Dispose();

    if (this.GridDisplayMode != ImageBoxGridDisplayMode.None
```

```
  && this.GridCellSize != 0)
      {
        _gridTile = this.CreateGridTileImage(this.GridCellSize,
this.GridColor, this.GridColorAlternate);
        _texture = new TextureBrush(_gridTile);
      }

      this.Invalidate();
    }

  #endregion  Private Methods

  #region  Protected Properties

    protected virtual int ScaledImageHeight
    { get { return this.Image != null ?
(int)(this.Image.Size.Height * this.ZoomFactor) : 0; } }

    protected virtual int ScaledImageWidth
    { get { return this.Image != null ?
(int)(this.Image.Size.Width * this.ZoomFactor) : 0; } }
```

```
    protected virtual double ZoomFactor
    { get { return (double)this.Zoom / 100; } }

  #endregion  Protected Properties

  #region  Protected Methods

    protected virtual void AdjustLayout()
    {
      if (this.AutoSize)
        this.AdjustSize();
      else if (this.SizeToFit)
        this.ZoomToFit();
      else if (this.AutoScroll)
        this.AdjustViewPort();
      this.Invalidate();
    }

    public virtual void AdjustScroll(int x, int y)
    {
```

```
      Point scrollPosition;

      scrollPosition = new Point(this.HorizontalScroll.Value
  + x, this.VerticalScroll.Value + y);

      this.UpdateScrollPosition(scrollPosition);
    }

    protected virtual void AdjustSize()
    {
      if (this.AutoSize && this.Dock == DockStyle.None)
        base.Size = base.PreferredSize;
    }

    protected virtual void AdjustViewPort()
    {
      if (this.AutoScroll && this.Image != null)
        this.AutoScrollMinSize = new Size(this.ScaledImageWidth
+ this.Padding.Horizontal, this.ScaledImageHeight
+ this.Padding.Vertical);
    }
```

```
    protected virtual Bitmap
CreateGridTileImage(int cellSize, Color firstColor, Color secondColor)
    {
      Bitmap result;
      int width;
      int height;
      float scale;

      // rescale the cell size
      switch (this.GridScale)
      {
        case ImageBoxGridScale.Medium:
          scale = 1.5F;
          break;
        case ImageBoxGridScale.Large:
          scale = 2;
          break;
        default:
          scale = 1;
          break;
```

```
      }

      cellSize = (int)(cellSize * scale);

      // draw the tile
      width = cellSize * 2;
      height = cellSize * 2;
      result = new Bitmap(width, height);
      using (Graphics g = Graphics.FromImage(result))
      {
        using (SolidBrush brush = new SolidBrush(firstColor))
          g.FillRectangle(brush, new Rectangle(0, 0, width, height));

        using (SolidBrush brush = new SolidBrush(secondColor))
        {
          g.FillRectangle(brush, new Rectangle(0, 0, cellSize, cellSize));
          g.FillRectangle(brush,
  new Rectangle(cellSize, cellSize, cellSize, cellSize));
        }
      }
```

```
      return result;
    }

    protected virtual void DrawImage(Graphics g)
    {
      g.InterpolationMode = this.InterpolationMode;
      g.DrawImage(this.Image, this.GetImageViewPort(),
  this.GetSourceImageRegion(), GraphicsUnit.Pixel);
    }

    protected virtual void OnAutoCenterChanged(EventArgs e)
    {
      this.Invalidate();

      if (this.AutoCenterChanged != null)
        this.AutoCenterChanged(this, e);
    }

    protected virtual void OnAutoPanChanged(EventArgs e)
    {
      if (this.AutoPanChanged != null)
```

```
    this.AutoPanChanged(this, e);
}

protected virtual void OnBorderStyleChanged(EventArgs e)
{
  this.AdjustLayout();

  if (this.BorderStyleChanged != null)
    this.BorderStyleChanged(this, e);
}

protected virtual void OnGridCellSizeChanged(EventArgs e)
{
  this.InitializeGridTile();

  if (this.GridCellSizeChanged != null)
    this.GridCellSizeChanged(this, e);
}

protected virtual void OnGridColorAlternateChanged(EventArgs e)
{
```

```
    this.InitializeGridTile();

    if (this.GridColorAlternateChanged != null)
      this.GridColorAlternateChanged(this, e);
  }

  protected virtual void OnGridColorChanged(EventArgs e)
  {
    this.InitializeGridTile();

    if (this.GridColorChanged != null)
      this.GridColorChanged(this, e);

  }

  protected virtual void OnGridDisplayModeChanged(EventArgs e)
  {
    this.InitializeGridTile();
    this.Invalidate();

    if (this.GridDisplayModeChanged != null)
```

```
        this.GridDisplayModeChanged(this, e);
    }

    protected virtual void OnGridScaleChanged(EventArgs e)
    {
      this.InitializeGridTile();

      if (this.GridScaleChanged != null)
        this.GridScaleChanged(this, e);
    }

    protected virtual void OnImageChanged(EventArgs e)
    {
      this.AdjustLayout();

      if (this.ImageChanged != null)
        this.ImageChanged(this, e);
    }

    protected virtual void OnInterpolationModeChanged(EventArgs e)
    {
```

```
      this.Invalidate();

      if (this.InterpolationModeChanged != null)
        this.InterpolationModeChanged(this, e);
    }

    protected virtual void OnInvertMouseChanged(EventArgs e)
    {
      if (this.InvertMouseChanged != null)
        this.InvertMouseChanged(this, e);
    }

    protected virtual void OnPanEnd(EventArgs e)
    {
      if (this.PanEnd != null)
        this.PanEnd(this, e);
    }

    protected virtual void OnPanStart(EventArgs e)
    {
      if (this.PanStart != null)
```

```
        this.PanStart(this, e);
    }

    protected virtual void OnSizeToFitChanged(EventArgs e)
    {
      this.AdjustLayout();

      if (this.SizeToFitChanged != null)
        this.SizeToFitChanged(this, e);
    }

    protected virtual void OnZoomChanged(EventArgs e)
    {
      this.AdjustLayout();

      if (this.ZoomChanged != null)
        this.ZoomChanged(this, e);
    }

    protected virtual void OnZoomIncrementChanged(EventArgs e)
    {
```

```
      if (this.ZoomIncrementChanged != null)
        this.ZoomIncrementChanged(this, e);
    }

    protected virtual void UpdateScrollPosition(Point position)
    {
      this.AutoScrollPosition = position;
      this.Invalidate();
      this.OnScroll(new ScrollEventArgs(ScrollEventType.ThumbPosition, 0));
    }

  #endregion  Protected Methods
  }
}
```

Appendix D

Shortestpath.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;


namespace Thesis
{
```

```
public class shortestPath
{
    Int16[] a = new Int16[100];
    Int16 heapsize=0;
    float[] d = new float[100];
    Int16 n,m;
    Int16[,] adj_list =new Int16[100,20];
    Int16[] node_degree = new Int16[100];
    float[,] weight = new float[100,100];
    Int16[] S = new Int16[100];
    Int16[] p = new Int16[100];
    float INF = 10000;



    public void init()
    {
        for (Int16 i = 0; i < 100; i++)
        {
            d[i] = INF;
            //node_degree[i] = 0;
```

```
                S[i] = 0;
            }
        }
        public void read_graph()
        {

            Int16 u=0, v=0;
            float w = 0;

            System.IO.StreamReader file =
new System.IO.StreamReader(System.IO.Path.GetDirectoryName
(Application.ExecutablePath) + "\\graph.txt");

            String graphSize = file.ReadLine();
            String[] nm = graphSize.Split(' ');
            n = Int16.Parse(nm[0]);
            m = Int16.Parse(nm[1]);

           // scanf("%d %d", &n, &m);
            String edgeInfo;
            while((edgeInfo = file.ReadLine()) != null)
```

```
{
   // scanf("%d %d %d", &u, &v, &w);

    String[] uvw = edgeInfo.Split(' ');
    u = Int16.Parse(uvw[0]);
    v = Int16.Parse(uvw[1]);
    w = float.Parse(uvw[2]);
    adj_list[u,node_degree[u]] = v;
    adj_list[v,node_degree[v]] = u;
    //sw1.WriteLine(adj_list[u, node_degree[u]]);
    //sw1.WriteLine(adj_list[v, node_degree[v]]);


    weight[u,v] = w;
    weight[v,u] = w;

    node_degree[u] = (Int16)(node_degree[u] + 1);
    //sw1.WriteLine(node_degree[u]);
    node_degree[v] = (Int16)(node_degree[v] + 1);
    //sw1.WriteLine(node_degree[v]);
    //Console.Write(adj_list);
```

```
            //sw1.WriteLine(adj_list.GetValue(0).ToString());
            //sw1.Close();
        }
        //sw1.Close();
        file.Close();
    }
    public void Min_Heapify(Int16 i)
    {
        Int16 l = (Int16)(2 * i);
        Int16 r = (Int16)(2 * i + 1);
        Int16 lowest;
        if (l <= heapsize && d[a[l]] < d[a[i]])
            lowest = l;
        else
            lowest = i;

        if (r <= heapsize && d[a[r]] < d[a[lowest]])
            lowest = r;
        if (lowest != i)
        {
            Int16 temp = a[i];
```

```
            a[i] = a[lowest];
            a[lowest] = temp;
            Min_Heapify(lowest);
        }
    }
    public void Enqeue(Int16 key)
    {
        heapsize++;
        a[heapsize] = 10000;

        Int16 i = heapsize;
        a[i] = key;

        Int16 parent = (Int16)(i / 2);
        while ((i > 1) && (d[a[parent]] > d[a[i]]))
        {
            Int16 temp = a[parent];
            a[parent] = a[i];
            a[i] = temp;
```

```
            i = parent;
            parent = (Int16)(i / 2);
        }
    }
    public Int16 Deqeue()
    {
        Int16 min = a[1];
        a[1] = a[heapsize];
        heapsize--;
        Min_Heapify(1);
        return min;

    }
    public void shortest_path(Int16 source)
    {
        Int16 i, u;
        Enqeue(source);
        d[source] = 0;

        while (heapsize > 0)
        {
```

```
u = Deqeue();



S[u] = 1;
//sw1.WriteLine(node_degree[u]);
for (i = 0; i < node_degree[u]; i++)
{

    if (S[adj_list[u,i]] == 0)
    {
        if (weight[u,adj_list[u,i]] < d[adj_list[u,i]])
        {
            d[adj_list[u,i]] = weight[u,adj_list[u,i]];
            p[adj_list[u,i]] = u;
            //sw1.WriteLine(p[adj_list[u,i]]);
            //sw1.WriteLine("OK");
        }

        Enqeue(adj_list[u,i]);
```

```
                }
            }

        }

        //sw1.Close();
    }
    public void show_edges(Int16 source)
    {
        Int16 i;
        System.IO.StreamWriter sw1 =
new System.IO.StreamWriter(System.IO.Path.GetDirectoryName
(Application.ExecutablePath) + "\\TextFile2.txt");

        for (i = 1; i <= n; i++)
        {
            if (i != source)
            {
               // prInt16f("(%d,%d)::%d\n", i, p[i], weight[i][p[i]]);
```

```
                    sw1.WriteLine(i);
                    sw1.WriteLine(p[i]);
                    sw1.WriteLine(weight[i,p[i]]);
                    sw1.WriteLine("next");

                }

            }
            sw1.Close();
        }
        public void show_tree(Int16 source, Int16 dest)
        {
            Int16 i = dest;
            //Int16[] path = new Int16[100];
            System.IO.StreamWriter sw = new
System.IO.StreamWriter(System.IO.Path.GetDirectoryName
(Application.ExecutablePath) + "\\TextFile1.txt");
            while (i!=0 )
            {
                //prInt16f("\n%d\n", i);
                sw.WriteLine(i.ToString());
```

```
            i = p[i];

            //sw.WriteLine(i.ToString());
        }

        //path[j] = -1;
        sw.Close();
        //return path;
    }
    public void show_heap()
    {

        Int16 i;
        for (i = 1; i <= heapsize; i++)
        {

           // prInt16f("%d ", a[i]);


        }
```

```
            // prInt16f("\n");


        }


    }
}
```

Appendix E

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using System.Xml;
using System.Xml.XPath;

namespace Thesis
{
    class XMLtoText
```

```
    {
        public void readXml()
        {
            String applicationPath
= System.IO.Path.GetDirectoryName(Application.ExecutablePath);
            System.IO.StreamWriter sw1
= new System.IO.StreamWriter(applicationPath + "\\mapXY.txt");

            XmlTextReader textReader
= new XmlTextReader(applicationPath+"\\map_1.osm");
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load(applicationPath + "\\map_1.osm");

            XmlNodeList xmlnodelist = xmlDoc.GetElementsByTagName("node");

            for (int i = 0; i < xmlnodelist.Count; i++)
            {
                String id = xmlnodelist[i].Attributes["id"].Value;
                String lon = xmlnodelist[i].Attributes["lon"].Value;
                String lat = xmlnodelist[i].Attributes["lat"].Value;
```

```
                Double lonD = Double.Parse(lon);
                Double latD = Double.Parse(lat);

                Double sx,sy,dx,dy;
                sx = 90 + (18.0 / 60);
                sy = 23 + (54.0 / 60);
                dx = 90 + (27.0 / 60);
                dy = 23 + (40.0 / 60);
                Double X = (lonD - sx) * 60 * 333;
                Double Y = ((sy - dy) - (latD - dy)) * 60 * 333;
                sw1.WriteLine(id+":"+X+","+Y);
            }

            sw1.Close();
        }
    }
}
```

Appendix F

```
<?php

set_time_limit(1000);

//read os

$doc_read = new DOMDocument();

$doc_read->load( 'map.osm' );

$doc_write = new DOMDocument();

$doc_write->formatOutput = true;

$nodes = $doc_read->getElementsByTagName( "node'' );

$nodes_array = array();

foreach($nodes as $node)

{

$nodes_array[$node->getAttribute("id")] = $node;

}
```

```
$ways = $doc_read->getElementsByTagName("way");

$root = $doc_write->createElement("graph");

$doc_write->appendChild($root);

$edge_count = 0;

foreach ($ways as $way)

{

$nds = $way->getElementsByTagName("nd");

$index = 0;

$prev_node = 0;

$current_node = 0;

foreach($nds as $nd)

{

$current_node_id = $nd->getAttribute("ref");

if($index==0) $prev_node_id = $current_node_id;


$prev_node = $nodes_array[$prev_node_id];

$current_node = $nodes_array[$current_node_id];


$prev_lon = $prev_node->getAttribute("lon");

$prev_lat = $prev_node->getAttribute("lat");
```

```
$current_lon = $current_node->getAttribute("lon");
$current_lat = $current_node->getAttribute("lat");

//echo $prev_lon."and".$prev_lat;
//echo "Connected to ";
//echo $current_lon."and".$current_lat;

//now we have to determine the euclidean distance between the prev and cur nodes
$square_lon = ($current_lon-$prev_lon) * ($current_lon-$prev_lon);
$square_lat = ($current_lat - $prev_lat) * ($current_lat - $prev_lat);

$distance = sqrt($square_lon + $square_lat);
//$distance_rounded = sprintf ("%.2f", $distance);



$if($prev_node_id != $current_node_id)
{
// create child element
$item = $doc_write->createElement("item");
$root->appendChild($item);
```

```
// create attribute node
$prev_id_xml = $doc_write->createAttribute("prev_id");
$item->appendChild($prev_id_xml);

// create attribute value node
$prev_id_xml_text = $doc_write->createTextNode($prev_node_id);
$prev_id_xml->appendChild($prev_id_xml_text);

// create attribute node
$current_id_xml = $doc_write->createAttribute("current_id");
$item->appendChild($current_id_xml);

// create attribute value node
$current_id_xml_text = $doc_write->createTextNode($current_node_id);
$current_id_xml->appendChild($current_id_xml_text);

// create attribute node
$prev_lon_xml = $doc_write->createAttribute("prev_lon");
$item->appendChild($prev_lon_xml);

// create attribute value node
```

```
$prev_lon_xml_text = $doc_write->createTextNode($prev_lon);
$prev_lon_xml->appendChild($prev_lon_xml_text);

// create attribute node
$prev_lat_xml = $doc_write->createAttribute("prev_lat");
$item->appendChild($prev_lat_xml);

// create attribute value node
$prev_lat_xml_text = $doc_write->createTextNode($prev_lat);
$prev_lat_xml->appendChild($prev_lat_xml_text);

// create attribute node
$current_lon_xml = $doc_write->createAttribute("current_lon");
$item->appendChild($current_lon_xml);

// create attribute value node
$current_lon_xml_text = $doc_write->createTextNode($current_lon);
$current_lon_xml->appendChild($current_lon_xml_text);

// create attribute node
$current_lat_xml = $doc_write->createAttribute("current_lat");
```

```
$item->appendChild($current_lat_xml);

// create attribute value node
$current_lat_xml_text = $doc_write->createTextNode($current_lat);
$current_lat_xml->appendChild($current_lat_xml_text);

// create attribute node
$distance_xml = $doc_write->createAttribute("distance");
$item->appendChild($distance_xml);

// create attribute value node
$distance_xml_text = $doc_write->createTextNode($distance);
$distance_xml->appendChild($distance_xml_text);

// create attribute node
$cluster_id_prev_xml = $doc_write->createAttribute("cluster_id_prev");
$item->appendChild($cluster_id_prev_xml);

// create attribute value node
$cluster_id_prev_xml_text = $doc_write->createTextNode("0");
$cluster_id_prev_xml->appendChild($cluster_id_prev_xml_text);
```

```
// create attribute node
$cluster_id_current_xml = $doc_write->createAttribute("cluster_id_current");
$item->appendChild($cluster_id_current_xml);

// create attribute value node
$cluster_id_current_xml_text = $doc_write->createTextNode("0");
$cluster_id_current_xml->appendChild($cluster_id_current_xml_text);

$edge_count++;
}

$prev_node_id = $current_node_id;
$index++;
//NEW$node = $doc_read->getElementsByAttribute("id",$nd->getAttribute("ref"));
//var_dump($nodes->getElementById($node_id));
//var_dump($node);
}
}
// create child element
$init = $doc_write->createElement("init");
```

```
$root->appendChild($init);

// create attribute node

$number_nodes = $doc_write->createAttribute("node");

$init->appendChild($number_nodes);

// create attribute value node

$number_nodes_text = $doc_write->createTextNode(count($nodes_array));

$number_nodes->appendChild($number_nodes_text);

// create attribute node

$number_edges = $doc_write->createAttribute("edge");

$init->appendChild($number_edges);

// create attribute value node

$number_edges_text = $doc_write->createTextNode($edge_count);

$number_edges->appendChild($number_edges_text);

//read osm ends

echo $doc_write->saveXML();

$doc_write->save("mapgraph_new.xml");

$?>
```

Appendix G

```
<?php
class PriorityList {
public $next;
public $data;
function __construct($data) {
$this->next = null;
$this->data = $data;
}
}

class PriorityQueue {
$private $size;
private $liststart;
```

```
private $comparator;

$function __construct($comparator) {

$this->size = 0;

$this->liststart = null;

$this->listend = null;

$this->comparator = $comparator;

}

$function add($x) {

$this->size = $this->size + 1;


$if($this->liststart == null) {

$this->liststart = new PriorityList($x);

} else {

$node = $this->liststart;

$comparator = $this->comparator;

$newnode = new PriorityList($x);

$lastnode = null;

$added = false;

while($node) {

if ($comparator($newnode, $node) < 0) {

$newnode->next = $node;
```

```
if ($lastnode == null) {
$this->liststart = $newnode;
} else {
$lastnode->next = $newnode;
}
$added = true;
break;
}
$lastnode = $node;
$node = $node->next;
}
if (!$added) {
$lastnode->next = $newnode;
}
}
}

$function debug() {
$node = $this->liststart;
$i = 0;
if (!$node) {
```

```
print "<< No nodes >>\n";

return;

}

while($node) {

print "[$i]=" . $node->data[1] . " (" . $node->data[0] . ")\n";

$node = $node->next;

$i++;

}

}


$function size() {

return $this->size;

}


$function peak() {

return $this->liststart->data;

}


$function remove() {

$x = $this->peak();

$this->size = $this->size - 1;
```

```
$this->liststart = $this->liststart->next;

return $x;

}

}

?>
```

Appendix H

```
<?php

require_once("PriorityQueue.php");


error_reporting(0);


class Edge {


$public $start;

public $end;

public $weight;


$public function __construct($start, $end, $weight) {

$this->start = $start;
```

```
$this->end = $end;
$this->weight = $weight;
}
}

class Graph {

$public $nodes = array();

$public function addedge($start, $end, $weight = 0) {
if (!isset($this->nodes[$start])) {
$this->nodes[$start] = array();
}
array_push($this->nodes[$start], new Edge($start, $end, $weight));
}

$public function removenode($index) {
array_splice($this->nodes, $index, 1);
}
```

```
$public function paths_from($from) {

$dist = array();

$dist[$from] = 0;


$visited = array();


$previous = array();


$queue = array();

$Q = new PriorityQueue("compareWeights");

$Q->add(array($dist[$from], $from));


$nodes = $this->nodes;


$while($Q->size() > 0) {

list($distance, $u) = $Q->remove();


$if (isset($visited[$u])) {

continue;

}

$visited[$u] = True;
```

```
$if (!isset($nodes[$u])) {
//print "WARNING: '$u' is not found in the node list\n";
}

$foreach($nodes[$u] as $edge) {

$alt = $dist[$u] + $edge->weight;
$end = $edge->end;
if (!isset($dist[$end]) || $alt < $dist[$end]) {
$previous[$end] = $u;
$dist[$end] = $alt;
$Q->add(array($dist[$end], $end));
}
}
}
return array($dist, $previous);
}

$public function paths_to($node_dsts, $tonode) {
// unwind the previous nodes for the specific destination node
```

```
$current = $tonode;
$path = array();

$if (isset($node_dsts[$current])) { // only add if there is a path to node
array_push($path, $tonode);
}
while(isset($node_dsts[$current])) {
$nextnode = $node_dsts[$current];

$array_push($path, $nextnode);

$current = $nextnode;
}

$return array_reverse($path);

}

$public function getpath($from, $to) {
list($distances, $prev) = $this->paths_from($from);
```

```
return $this->paths_to($prev, $to);
}

}

function compareWeights($a, $b) {
return $a->data[0] - $b->data[0];
}
?>
```

Appendix I

```
<?php
set_time_limit(1000);

$s_lat=$_GET["s_lat"];
$s_lon=$_GET["s_lon"];
$d_lat=$_GET["d_lat"];
$d_lon=$_GET["d_lon"];

require("Dijkstra.php");

function runTest($s_lat,$s_lon,$d_lat,$d_lon) {
//read xml
$result_array = array();
```

```
$result_array[0] = array();//path

$result_array[1] = array();//lon

$result_array[2] = array();//lat

$doc_read = new DOMDocument();

$doc_read->load( 'mapgraph_new.xml' );

$items = $doc_read->getElementsByTagName("item");

$g = new Graph();


$slon=100;

$slat=100;

$dlon=100;

$dlat=100;

//LOOP

foreach ($items as $item)

{

$prev_id = $item->getAttribute("prev_id");

$prev_lon = $item->getAttribute("prev_lon");

$result_array[1][$prev_id] = $prev_lon;

$prev_lat = $item->getAttribute("prev_lat");

$result_array[2][$prev_id] = $prev_lat;
```

```
$current_id = $item->getAttribute("current_id");

$current_lon = $item->getAttribute("current_lon");

$result_array[1][$current_id] = $current_lon;

$current_lat = $item->getAttribute("current_lat");

$result_array[2][$current_id] = $current_lat;


$distance = $item->getAttribute("distance");


$g->addedge($prev_id, $current_id, $distance);

$g->addedge($current_id, $prev_id, $distance);



//now we have to determine the euclidean distance between the

//prev and current nodes

$square_lon = ($current_lon-$s_lon) * ($current_lon-$s_lon);

$square_lat = ($current_lat - $s_lat) * ($current_lat - $s_lat);


$sd = sqrt($square_lon + $square_lat);

//$distance_rounded = sprintf ("%.2f", $distance);


if($sd<$slon)
```

```
{
        $slon = $sd;
$source_id = $current_id ;

}

//now we have to determine the euclidean distance between the prev
//and current nodes
$square_lon = ($current_lon-$d_lon) * ($current_lon-$d_lon);
$square_lat = ($current_lat - $d_lat) * ($current_lat - $d_lat);

$dd = sqrt($square_lon + $square_lat);
//$distance_rounded = sprintf ("%.2f", $distance);

if($dd<$dlon)
{
        $dlon = $dd;
$destination_id = $current_id ;

}
}
```

```
//LOOP

list($distances, $prev) = $g->paths_from($source_id);

$path = $g->paths_to($prev, $destination_id);

$result_array[0] = $path;

if(count($path)==0){
 print_r("error");
 die();
}

print_r(json_encode($result_array));

}

runTest($s_lat,$s_lon,$d_lat,$d_lon);

?>
```

Appendix J

```
<html>
    <head>
        <title>Map</title>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
</script>
<script type="text/javascript"
src="jquery.svg.package/jquery.svg.js"></script>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script src="json2.js"></script>
    </head>
    <body>
```

```
        <div id="ourMap"></div>
        <script>
map = new OpenLayers.Map("ourMap");
 map.addLayer(new OpenLayers.Layer.OSM());
 var markers = new OpenLayers.Layer.Markers( "Markers" );
         map.addLayer(markers);
//read the osm file
var id_points = new Array();
var lat_points = new Array();
var lon_points = new Array();
var map_points = new Array();
var lonlat;
var index=0;
var timer_start=0;
var count=0;
var s_lat;
var s_lon;
var d_lat;
var d_lon;
var select_source=1;
var select_dest=0;
```

```
var vector;


    var counter=setInterval(timer, 1000); //1000 will  run it every 1 sec
function timer()
{
if(timer_start==1)
{
    count=count+1;


  //Do code for showing the number of seconds here
document.getElementById("timer").innerHTML= "Dijkstra running "
+ count + " secs";
}
}
$(function() {
$("#dijkstra").click(function(){



timer_start=1;
```

```
var source_id = $("#source").val();

var destination_id = $("#destination").val();


$.ajax({

      method: 'get',

      url: 'RunTest_init.php',

      data: {

's_lat': s_lat,

's_lon': s_lon,

'd_lat': d_lat,

'd_lon': d_lon,

        'ajax': true

      },

      success: function(data_json) {

   if(data_json=="error"){

alert("No path exists");

}

else{

    var data = new Array();

 data = JSON.parse(data_json);

 var i=0;
```

```
 var length = data[0].length;
 for(;i<length;i++)
 {
map_points[i] =
new OpenLayers.Geometry.Point
(data[1][data[0][i]],data[2][data[0][i]]);

 }

 var style = {
strokeColor: '#0000ff',
strokeOpacity: 0.8,
strokeWidth: 4
};

 vector = new OpenLayers.Layer.Vector();
   var featureVector=new OpenLayers.Feature.Vector
 (new OpenLayers.Geometry.LineString(map_points)
 .transform(new OpenLayers.Projection("EPSG:4326"),
 new OpenLayers.Projection("EPSG:900913"))
 ,null,style);
```

```
   vector.addFeatures([featureVector]);
   map.addLayers([vector]);

 timer_start=0;
       }
  }
    });
});
     }); //close $(
//read the osm file end
//marker drawing ends

ourpoint1 = new OpenLayers.LonLat(90.3989,23.7937);
ourpoint1.transform(new OpenLayers.Projection("EPSG:4326")
,map.getProjectionObject());
map.setCenter(ourpoint1,12);

/// get position from click
 OpenLayers.Control.Click = OpenLayers.Class(OpenLayers.Control, {
                defaultHandlerOptions: {
                    'single': true,
```

```
        'double': false,
        'pixelTolerance': 0,
        'stopSingle': false,
        'stopDouble': false
    },

    initialize: function(options) {
        this.handlerOptions = OpenLayers.Util.extend(
            {}, this.defaultHandlerOptions
        );
        OpenLayers.Control.prototype.initialize.apply(
            this, arguments
        );
        this.handler = new OpenLayers.Handler.Click(
            this, {
                'click': this.trigger
            }, this.handlerOptions
        );
    },

    trigger: function(e) {
```

```
                    lonlat = map.getLonLatFromPixel(e.xy);
lonlat.transform(new OpenLayers.Projection("EPSG:900913")
, new OpenLayers.Projection("EPSG:4326"));


                    if(select_source==1)
    {
s_lat = lonlat.lat;
s_lon = lonlat.lon;


 var ourpoint1=newOpenLayers.LonLat(s_lon,s_lat)
                         ourpoint1.transform
 (newOpenLayers.Projection("EPSG:4326"),
 map.getProjectionObject());


 markers.addMarker(new OpenLayers.Marker(ourpoint1));


 select_dest=1;
 select_source=0;
}


else if(select_dest==1)
```

```
{
d_lat = lonlat.lat;
d_lon = lonlat.lon;

var ourpoint2=newOpenLayers.LonLat(d_lon,d_lat)
                          ourpoint2.transform
 (newOpenLayers.Projection("EPSG:4326"),
 map.getProjectionObject());

 markers.addMarker(new OpenLayers.Marker(ourpoint2));
 select_dest=0;
}
else
{
    select_dest=0;
select_source=1;
markers.clearMarkers();
}
                }

            });
```

```
    var click = new OpenLayers.Control.Click();
            map.addControl(click);
            click.activate();









        </script>
<button type="button" id="dijkstra" name="dijkstra">Dijkstra</button>
<label id="timer">Dijkstra not started<label/>
    </body>


</html>
```

Appendix K

```
<?php

set_time_limit(1000);

require("Dijkstra.php");

function loadGraph() {
//read xml
$lonlat = array();
$lonlat[0] = array();//lon
$lonlat[1] = array();//lat

$doc_read = new DOMDocument();
```

```
$doc_read->load( 'mapgraph_new.xml' );

$items = $doc_read->getElementsByTagName("item");

$g = new Graph();

//LOOP

foreach ($items as $item)

{

$prev_id = $item->getAttribute("prev_id");

$prev_lon = $item->getAttribute("prev_lon");

$lonlat[0][$prev_id] = $prev_lon;

$prev_lat = $item->getAttribute("prev_lat");

$lonlat[1][$prev_id] = $prev_lat;


$current_id = $item->getAttribute("current_id");

$current_lon = $item->getAttribute("current_lon");

$lonlat[0][$current_id] = $current_lon;

$current_lat = $item->getAttribute("current_lat");

$lonlat[1][$current_id] = $current_lat;


$distance = $item->getAttribute("distance");


$g->addedge($prev_id, $current_id, $distance);
```

```
$g->addedge($current_id, $prev_id, $distance);
}
session_start();
$_SESSION["graph"] = serialize($g);
$_SESSION["lonlat"] = serialize($lonlat);
echo "Loaded Ok";
}
loadGraph();
?>
```

Appendix L

```
<?php

error_reporting(0);

set_time_limit(1000);

session_start();

require("Dijkstra.php");


$graph = unserialize($_SESSION["graph"]);

$lonlat = unserialize($_SESSION["lonlat"]);


$s_lat=$_GET["s_lat"];

$s_lon=$_GET["s_lon"];

$d_lat=$_GET["d_lat"];

$d_lon=$_GET["d_lon"];
```

```
function find_nearest_node($s_lon,$s_lat,$d_lon,$d_lat,$lonlat)
{
   $slon=100;
   $slat=100;
   $dlon=100;
   $dlat=100;
   if (is_array($lonlat[0])) {
   foreach($lonlat[0] as $id=>$lon)
   {
//now we have to determine the euclidean distance between
//the prev and current nodes
$square_lon = ($lon-$s_lon) * ($lon-$s_lon);
$square_lat = ($lonlat[1][$id] - $s_lat) * ($lonlat[1][$id] - $s_lat);

$sd = $square_lon + $square_lat;
//$distance_rounded = sprintf ("%.2f", $distance);

$if($sd<$slon)
{
$slon = $sd;
```

```
$source_id = $id ;
}

//now we have to determine the euclidean distance between
//the prev and current nodes
$square_lon = ($lon-$d_lon) * ($lon-$d_lon);
$square_lat = ($lonlat[1][$id] - $d_lat) * ($lonlat[1][$id] - $d_lat);

//$distance_rounded = sprintf ("%.2f", $distance);
$dd = $square_lon + $square_lat;
if($dd<$dlon)
{
$dlon = $dd;
$destination_id = $id ;

}
   }
   }
   $clicked_node = array();
   $clicked_node[0] = $source_id;
   $clicked_node[1] = $destination_id;
```

```
   return $clicked_node;
}



function runTest($s_lon,$s_lat,$d_lon,$d_lat,$graph,$lonlat) {
//read xml
$result_array = array();
$result_array[0] = array();//path
$result_array[1] = $lonlat[0];//lon
$result_array[2] = $lonlat[1];//lat
$g = new Graph();
$g = $graph;
$clicked_node = array();


        ///finding source_id and destination_id
        $clicked_node = find_nearest_node($s_lon,$s_lat,$d_lon,$d_lat,$lonlat);
        $source_id = $clicked_node[0];
        $destination_id = $clicked_node[1];
list($distances, $prev) = $g->paths_from($source_id);
$path = $g->paths_to($prev, $destination_id);
```

```
$result_array[0] = $path;

$if(count($path)==0){

 print_r("error");

 die();

}

print_r(json_encode($result_array));

}

runTest($s_lon,$s_lat,$d_lon,$d_lat,$graph,$lonlat);

?>
```

Appendix M

```
<html>
    <head>
        <title>Map</title>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
</script>
<script type="text/javascript" src="jquery.svg.package/jquery.svg.js">
</script>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script src="json2.js"></script>
    </head>
    <body>
        <div id="ourMap"></div>
```

```
<script>

map = new OpenLayers.Map("ourMap");

map.addLayer(new OpenLayers.Layer.OSM());

var markers = new OpenLayers.Layer.Markers( "Markers" );

map.addLayer(markers);

//read the osm file

var id_points = new Array();

var lat_points = new Array();

var lon_points = new Array();

var map_points = new Array();

var lonlat;

var index=0;

var timer_start=0;

var count=0;

var s_lat;

var s_lon;

var d_lat;

var d_lon;

var select_source=1;

var select_dest=0;

var data;
```

```
var vector;
var counter=setInterval(timer, 1000); //1000 will  run it every 1 second
function timer()
{
if(timer_start==1)
{
count=count+1;

//Do code for showing the number of seconds here
document.getElementById("timer").innerHTML= "Dijkstra running "+count+" secs";
}
}
$(function() {
$("#load").click(function(){



timer_start=1;

var source_id = $("#source").val();
var destination_id = $("#destination").val();
```

```
$.ajax({
method: 'get',
url: 'LoadGraph.php',
success: function(data_json) {
timer_start=0;
alert("Graph Loaded");
}

});
});
//broken
$("#dijkstra").click(function(){
timer_start=1;

var source_id = $("#source").val();
var destination_id = $("#destination").val();

$.ajax({
method: 'get',
url: 'RunTest.php',
data: {
```

```
's_lon': s_lon,
's_lat': s_lat,
'd_lon': d_lon,
'd_lat': d_lat,
'ajax': true
},
success: function(data_json) {
alert("ended");
if(data_json=="error"){
alert("No path exists");
}
else{
var data = new Array();
//data[0] = $path, $data[1] = $lon, $data[1] = $lat;
data = JSON.parse(data_json);
var i=0;
var length = data[0].length;
for(;i<length;i++)
{
map_points[i] = new OpenLayers.Geometry.Point(data[1][data[0][i]],
data[2][data[0][i]]);
```

```
}

var style = {
strokeColor: '#0000ff',
strokeOpacity: 0.8,
strokeWidth: 4
};

vector = new OpenLayers.Layer.Vector();
var featureVector = new OpenLayers.Feature.Vector
(new OpenLayers.Geometry.LineString(map_points)
.transform(new OpenLayers.Projection("EPSG:4326"),
new OpenLayers.Projection("EPSG:900913")),null,style);
vector.addFeatures([featureVector]);
$map.addLayers([vector]);

timer_start=0;
}
}
});
});
```

```
}); //close $(
//read the osm file end
//marker drawing ends

ourpoint1 = new OpenLayers.LonLat(90.3989,23.7937);
ourpoint1.transform(new OpenLayers.Projection("EPSG:4326"),
map.getProjectionObject());
map.setCenter(ourpoint1,12);

/// get position from click
OpenLayers.Control.Click = OpenLayers.Class(OpenLayers.Control, {
defaultHandlerOptions: {
'single': true,
'double': false,
'pixelTolerance': 0,
'stopSingle': false,
'stopDouble': false
},

initialize: function(options) {
this.handlerOptions = OpenLayers.Util.extend(
```

```
{}, this.defaultHandlerOptions
);
OpenLayers.Control.prototype.initialize.apply(
this, arguments
);
this.handler = new OpenLayers.Handler.Click(
this, {
'click': this.trigger
}, this.handlerOptions
);
},

trigger: function(e) {
lonlat = map.getLonLatFromPixel(e.xy);
lonlat.transform(new OpenLayers.Projection("EPSG:900913"),
new OpenLayers.Projection("EPSG:4326"));

if(select_source==1)
{
s_lat = lonlat.lat;
s_lon = lonlat.lon;
```

```
var ourpoint1 = new OpenLayers.LonLat(s_lon,s_lat)
ourpoint1.transform(new OpenLayers.Projection("EPSG:4326" ),
map.getProjectionObject());

markers.addMarker(new OpenLayers.Marker(ourpoint1));

select_dest=1;
select_source=0;
}

else if(select_dest==1)
{
d_lat = lonlat.lat;
d_lon = lonlat.lon;

var ourpoint2 = new OpenLayers.LonLat(d_lon,d_lat)
ourpoint2.transform(new OpenLayers.Projection("EPSG:4326" ),
map.getProjectionObject());

markers.addMarker(new OpenLayers.Marker(ourpoint2));
```

```
// alert("You clicked near " + d_lat + " N, " +
// + d_lon+ " E");

select_dest=0;
//select_source=1;
}
else
{
select_dest=0;
select_source=1;
markers.clearMarkers();



}
}

});

var click = new OpenLayers.Control.Click();
map.addControl(click);
```

```
click.activate();

</script>

<button type="button" id="load" name="load">Load Graph</button>

<button type="button" id="dijkstra" name="dijkstra">Dijkstra</button>

<label id="timer">Dijkstra not started<label/>

</body>

</html>
```

Appendix N

```
<?php

error_reporting(0);

set_time_limit(1000);

session_start();

require("Dijkstra.php");


$graph = unserialize($_SESSION["graph"]);

$lonlat = unserialize($_SESSION["lonlat"]);


//$s_lat=23.7249233;  //1778185456

//$s_lon=90.4119726;

//$d_lat=23.7253504; //1778185464

//$d_lon=90.4107634;
```

```
//var_dump($lonlat);
$s_lat=$_GET["s_lat"];
$s_lon=$_GET["s_lon"];
$d_lat=$_GET["d_lat"];
$d_lon=$_GET["d_lon"];

$mid_lon = ($s_lon+$d_lon)/2;
$mid_lat = ($s_lat+$d_lat)/2;

$radius = ($mid_lon-$s_lon)*($mid_lon-$s_lon)
 +($mid_lat-$s_lat)*($mid_lat-$s_lat)+.001;

function find_nearest_node($s_lon,$s_lat,$d_lon,$d_lat,
$lonlat,$mid_lon,$mid_lat,$radius,$g)
{
   $clicked_node = array();
   $clicked_node[2] = array();//the graph will be here
   $slon=100;
   $slat=100;
   $dlon=100;
```

```
   $dlat=100;
   if (is_array($lonlat[0])) {
   foreach($lonlat[0] as $id=>$lon)
   {
    $distance_check = ($lon-$mid_lon)*($lon-$mid_lon)
+($lonlat[1][$id]-$mid_lat)*($lonlat[1][$id]-$mid_lat);
    if($distance_check>$radius)
{
$unset($g->nodes[$id]);
continue;
}
//now we have to determine the euclidean distance between
//the prev and current nodes
$square_lon = ($lon-$s_lon) * ($lon-$s_lon);
$square_lat = ($lonlat[1][$id] - $s_lat) * ($lonlat[1][$id] - $s_lat);

$sd = $square_lon + $square_lat;

$if($sd<$slon)
{
$slon = $sd;
```

```
$source_id = $id ;

}

$square_lon = ($lon-$d_lon) * ($lon-$d_lon);

$square_lat = ($lonlat[1][$id] - $d_lat) * ($lonlat[1][$id] - $d_lat);


$dd = $square_lon + $square_lat;

if($dd<$dlon)

{

$dlon = $dd;

$destination_id = $id;

}

   }

   }

   $clicked_node[0] = $source_id;

   $clicked_node[1] = $destination_id;

   $clicked_node[2] = $g;


  $return $clicked_node;

}
```

```
function runTest($s_lon,$s_lat,$d_lon,$d_lat,$graph,$lonlat,
$mid_lon,$mid_lat,$radius) {
//read xml
$result_array = array();
$result_array[0] = array();//path
$result_array[1] = $lonlat[0];//lon
$result_array[2] = $lonlat[1];//lat
$g = new Graph();
$g = $graph;
        //var_dump($graph);
$clicked_node = array();

        ///finding source_id and destination_id
        $clicked_node = find_nearest_node($s_lon,$s_lat,$d_lon,$d_lat,
$lonlat,$mid_lon,$mid_lat,$radius,$g);


        $source_id = $clicked_node[0];
        $destination_id = $clicked_node[1];
$g = $clicked_node[2];
```

```
list($distances, $prev) = $g->paths_from($source_id);


$path = $g->paths_to($prev, $destination_id);


$result_array[0] = $path;


$if(count($path)==0){
 print_r("error");
 die();
}


        //echo "paths $path";
//var_dump($path);
print_r(json_encode($result_array));


}



runTest($s_lon,$s_lat,$d_lon,$d_lat,$graph,$lonlat,$mid_lon,$mid_lat,$radius);
?>
```

Appendix O

```
<html>
    <head>
        <title>Map</title>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
</script>
<script type="text/javascript" src="jquery.svg.package/jquery.svg.js">
</script>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script src="json2.js"></script>
    </head>
    <body>
        <div id="ourMap"></div>
```

```
<script>

map = new OpenLayers.Map("ourMap");

map.addLayer(new OpenLayers.Layer.OSM());

var markers = new OpenLayers.Layer.Markers( "Markers" );

map.addLayer(markers);

//read the osm file

var id_points = new Array();

var lat_points = new Array();

var lon_points = new Array();

var map_points = new Array();

var lonlat;

var index=0;

var timer_start=0;

var count=0;

var s_lat;

var s_lon;

var d_lat;

var d_lon;

var select_source=1;

var select_dest=0;

var data;
```

```
var vector;
var counter=setInterval(timer, 1000); //1000 will  run it every 1 second
function timer()
{
if(timer_start==1)
{
count=count+1;

//Do code for showing the number of seconds here
document.getElementById("timer").innerHTML= "Dijkstra running "+count+" secs";
}
}
$(function() {
$("#load").click(function(){
timer_start=1;
var source_id = $("#source").val();
var destination_id = $("#destination").val();

$.ajax({
method: 'get',
url: 'LoadGraph.php',
```

```
success: function(data_json) {
timer_start=0;
alert("Graph Loaded");
}

});
});
//broken
$("#dijkstra").click(function(){
timer_start=1;

var source_id = $("#source").val();
var destination_id = $("#destination").val();

$.ajax({
method: 'get',
url: 'ImprovedRunTest.php',
data: {
's_lon': s_lon,
's_lat': s_lat,
'd_lon': d_lon,
```

```
'd_lat': d_lat,
'ajax': true
},
success: function(data_json) {
alert("ended");
if(data_json=="error"){
alert("No path exists");
}
else{
var data = new Array();
//data[0] = $path, $data[1] = $lon, $data[1] = $lat;
data = JSON.parse(data_json);
var i=0;
var length = data[0].length;
for(;i<length;i++)
{
map_points[i] = new OpenLayers.Geometry.Point(data[1][data[0][i]],
data[2][data[0][i]]);
}

var style = {
```

```
strokeColor: '#0000ff',
strokeOpacity: 0.8,
strokeWidth: 4
};

vector = new OpenLayers.Layer.Vector();
var featureVector = new OpenLayers.Feature.Vector
(new OpenLayers.Geometry.LineString(map_points)
.transform(new OpenLayers.Projection("EPSG:4326"),
new OpenLayers.Projection("EPSG:900913")),null,style);
vector.addFeatures([featureVector]);
map.addLayers([vector]);

timer_start=0;
}
}
});
});
//broken

}); //close $(
```

```
//read the osm file end
//marker drawing ends

ourpoint1 = new OpenLayers.LonLat(90.3989,23.7937);
ourpoint1.transform(new OpenLayers.Projection("EPSG:4326"),
map.getProjectionObject());
map.setCenter(ourpoint1,12);

/// get position from click
OpenLayers.Control.Click = OpenLayers.Class(OpenLayers.Control, {
defaultHandlerOptions: {
'single': true,
'double': false,
'pixelTolerance': 0,
'stopSingle': false,
'stopDouble': false
},

initialize: function(options) {
this.handlerOptions = OpenLayers.Util.extend(
{}, this.defaultHandlerOptions
```

```
);
OpenLayers.Control.prototype.initialize.apply(
this, arguments
);
this.handler = new OpenLayers.Handler.Click(
this, {
'click': this.trigger
}, this.handlerOptions
);
},

trigger: function(e) {
lonlat = map.getLonLatFromPixel(e.xy);
lonlat.transform(new OpenLayers.Projection("EPSG:900913"),
new OpenLayers.Projection("EPSG:4326"));

if(select_source==1)
{
s_lat = lonlat.lat;
s_lon = lonlat.lon;
```

```
var ourpoint1 = new OpenLayers.LonLat(s_lon,s_lat)
ourpoint1.transform(new OpenLayers.Projection("EPSG:4326" ),
map.getProjectionObject());

markers.addMarker(new OpenLayers.Marker(ourpoint1));

select_dest=1;
select_source=0;
}

else if(select_dest==1)
{
d_lat = lonlat.lat;
d_lon = lonlat.lon;

var ourpoint2 = new OpenLayers.LonLat(d_lon,d_lat)
ourpoint2.transform(new OpenLayers.Projection("EPSG:4326" ),
map.getProjectionObject());

markers.addMarker(new OpenLayers.Marker(ourpoint2));
```

```
// alert("You clicked near " + d_lat + " N, " +

// + d_lon+ " E");


select_dest=0;

//select_source=1;

}

else

{

select_dest=0;

select_source=1;

markers.clearMarkers();




}

}


});


var click = new OpenLayers.Control.Click();

map.addControl(click);

click.activate();
```

```
</script>
<button type="button" id="load" name="load">Load Graph</button>
<button type="button" id="dijkstra" name="dijkstra">Dijkstra</button>
<label id="timer">Dijkstra not started<label/>
    </body>
</html>
```

Appendix P

```
<?php
set_time_limit(-1);




require("Dijkstra.php");


function runTestVertexCover() {
//read xml
$doc_write = new DOMDocument();
   $doc_write->formatOutput = true;
   $root = $doc_write->createElement("graph");
$doc_write->appendChild($root);
```

```
$doc_read = new DOMDocument();

$doc_read->load( 'mapgraph_vertexcover_new.xml' );

$items = $doc_read->getElementsByTagName("item");

$g = new Graph();

$array_vertex_cover = array();

$slon=100;

$slat=100;

$dlon=100;

$dlat=100;

//LOOP

foreach ($items as $item)

{

$prev_id = $item->getAttribute("prev_id");

$prev_lon = $item->getAttribute("prev_lon");

$result_array[1][$prev_id] = $prev_lon;

$prev_lat = $item->getAttribute("prev_lat");

$result_array[2][$prev_id] = $prev_lat;


$current_id = $item->getAttribute("current_id");

$current_lon = $item->getAttribute("current_lon");

$result_array[1][$current_id] = $current_lon;
```

```
$current_lat = $item->getAttribute("current_lat");
$result_array[2][$current_id] = $current_lat;


$distance = $item->getAttribute("distance");


$g->addedge($prev_id, $current_id, $distance);
$g->addedge($current_id, $prev_id, $distance);
}
//LOOP
$doc_read_common = new DOMDocument();
$doc_read_common->load( 'common.xml' );
$items_common = $doc_read_common->getElementsByTagName("item");
$index=0;
foreach($items_common as $item)
{
$array_vertex_cover[$index] = $item->getAttribute("id");
$index++;
}
//Loop
$index1=0;
$result = array();
```

```
for(;$index1<count($array_vertex_cover);$index1++)
{
$for($index2=$index1+1;$index2<count($array_vertex_cover);$index2++)
{
$list($distances, $prev) = $g->paths_from($array_vertex_cover[$index1]);
$path = $g->paths_to($prev, $array_vertex_cover[$index2]);
$result = array_intersect($path, $array_vertex_cover);
if(count($result)==2) {
// create child element
$item = $doc_write->createElement("edge");
$root->appendChild($item);

// create attribute node
$prev_id_xml = $doc_write->createAttribute("id1");
$item->appendChild($prev_id_xml);

// create attribute value node
$prev_id_xml_text = $doc_write->createTextNode
($array_vertex_cover[$index1]);
$prev_id_xml->appendChild($prev_id_xml_text);
```

```
// create attribute node
$current_id_xml = $doc_write->createAttribute("id2");
$item->appendChild($current_id_xml);

// create attribute value node
$current_id_xml_text = $doc_write->createTextNode
($array_vertex_cover[$index2]);
$current_id_xml->appendChild($current_id_xml_text);
}
}
}
//LoopEnd
echo $doc_write->saveXML();
   $doc_write->save("common_edgelist.xml");
}



runTestVertexCover();


?>
```

Appendix Q

```
<?php
set_time_limit(1000);


function runVertexTest() {
//read xml
$vertices = array();
        $lons = array();
        $lats = array();
$vertex_read = new DOMDocument();
$vertex_read->load( 'common.xml' );
$items = $vertex_read->getElementsByTagName("item");


         echo "starting";
```

```
        //reading vertices
$i=0;
foreach ($items as $item)
{
$id = $item->getAttribute("id");
$lon = $item->getAttribute("lon");
$lat = $item->getAttribute("lat");

$vertices[$i++] = $id;
$lons[$id] = $lon;
        $lats[$id] = $lat;

}
        $total_vertices=$i;

echo "ended $i";
       //reading edges

        $edges = array();
$edge_read = new DOMDocument();
        $edge_read->load('common_edgelist.xml');
```

```
$items = $edge_read->getElementsByTagName("edge");


        $i=0;
foreach ($items as $item)
{
$id1 = $item->getAttribute("id1");
$id2 = $item->getAttribute("id2");


                $edges[$i] = array();
                $edges[$i][0] = $id1;
$edges[$i][1] = $id2;
                $edges[$i][2] = 0;


$i++;


             //echo "$i";


}
        $total_edges=$i;
```

```
      ///vertex cover algo


    $c = array();
     $E = $edges;


    $i=-1;
    while(count($E)>0)
    {
       $i++;
       if(!array_key_exists($i,$E)) continue;


      $c = array_unique(array_merge($c, array($E[$i][0],$E[$i][1])));
     $k=0;
        foreach($E as $e)
{
$if($E[$i][0]==$e[0]&&$E[$i][1]==$e[1]) continue;
if($E[$i][0]==$e[0]||$E[$i][0]==$e[1]
||$E[$i][1]==$e[0]||$E[$i][1]==$e[1])
{
$unset($e);
}
```

```
}
unset($E[$i]);



    }
    $doc_write = new DOMDocument();
    $doc_write->formatOutput = true;
    $root = $doc_write->createElement("graph");
    $doc_write->appendChild($root);

    foreach($c as $item_c)
    {
$item = $doc_write->createElement("item");
$root->appendChild($item);

// create attribute node
$current_id_xml = $doc_write->createAttribute("id");
$item->appendChild($current_id_xml);

// create attribute value node
$current_id_xml_text = $doc_write->createTextNode($item_c);
```

```
$current_id_xml->appendChild($current_id_xml_text);


// create attribute node
$current_lon_xml = $doc_write->createAttribute("lon");
$item->appendChild($current_lon_xml);


// create attribute value node
$current_lon_xml_text = $doc_write->createTextNode($lons[$item_c]);
$current_lon_xml->appendChild($current_lon_xml_text);


// create attribute node
$current_lat_xml = $doc_write->createAttribute("lat");
$item->appendChild($current_lat_xml);


// create attribute value node
$current_lat_xml_text = $doc_write->createTextNode($lats[$item_c]);
$current_lat_xml->appendChild($current_lat_xml_text);
//new end
     }


 $echo $doc_write->saveXML();
```

```
  $doc_write->save("police_box.xml");



}

runVertexTest();

?>
```

Appendix R

```
<html>
    <head>
        <title>Map</title>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
</script>
<script type="text/javascript" src="jquery.svg.package/jquery.svg.js">
</script>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script src="json2.js"></script>
    </head>
    <body>
```

```
       <div id="ourMap"></div>
<script>
map = new OpenLayers.Map("ourMap");
map.addLayer(new OpenLayers.Layer.OSM());
var markers = new OpenLayers.Layer.Markers( "Markers" );
map.addLayer(markers);

var con=0;
//read the osm file
$(function() {
$.ajax({
type: "GET",
url: "common.xml",
dataType: "xml",
success: function(xml) {
$(xml).find('item').each(function(){
var id_text = $(this).attr('id');
var lat_text = $(this).attr('lat');
var lon_text = $(this).attr('lon');
var lonLat = new OpenLayers.LonLat( lon_text ,lat_text )
.transform(new OpenLayers.Projection("EPSG:4326"),
```

```
map.getProjectionObject());
markers.addMarker(new OpenLayers.Marker(lonLat));

con++;
//if(con>2) return false;

});



}
}); //close $.ajax(
}); //close $(
//read the osm file end
//marker drawing ends

var ourpoint1 = new OpenLayers.LonLat(90.3989,23.7937);
ourpoint1.transform(new OpenLayers.Projection("EPSG:4326"),
map.getProjectionObject());
map.setCenter(ourpoint1,12);
</script>
    <a id="vertex_cover" name="vertex_cover" href="vertexcoverafter.php">
```

```
Show Vertex Cover
</a>
   </body>
</html>
```

Appendix S

```
<html>
    <head>
        <title>Map</title>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
</script>
<script type="text/javascript" src="jquery.svg.package/jquery.svg.js">
</script>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script src="json2.js"></script>
    </head>
    <body>
```

```
       <div id="ourMap"></div>
<script>
map = new OpenLayers.Map("ourMap");
map.addLayer(new OpenLayers.Layer.OSM());
var markers = new OpenLayers.Layer.Markers( "Markers" );
map.addLayer(markers);

var con=0;
//read the osm file
$(function() {
$.ajax({
type: "GET",
url: "police_box.xml",
dataType: "xml",
success: function(xml) {
$(xml).find('item').each(function(){
var id_text = $(this).attr('id');
var lat_text = $(this).attr('lat');
var lon_text = $(this).attr('lon');
var lonLat = new OpenLayers.LonLat( lon_text ,lat_text )
.transform(new OpenLayers.Projection("EPSG:4326"),
```

```
map.getProjectionObject());
markers.addMarker(new OpenLayers.Marker(lonLat));

con++;

});



}
}); //close $.ajax(
}); //close $(
//read the osm file end
//marker drawing ends

var ourpoint1 = new OpenLayers.LonLat(90.3989,23.7937);
ourpoint1.transform(new OpenLayers.Projection("EPSG:4326"),
map.getProjectionObject());
map.setCenter(ourpoint1,12);
</script>
    </body>
</html>
```

Appendix T

```
<?php

error_reporting(0);

set_time_limit(1000);

session_start();

require("Dijkstra.php");


$graph = unserialize($_SESSION["graph"]);

$lonlat = unserialize($_SESSION["lonlat"]);

$latest_path = unserialize($_SESSION["path"]);


$s_lat=$_GET["s_lat"];

$s_lon=$_GET["s_lon"];

$d_lat=$_GET["d_lat"];
```

```
$d_lon=$_GET["d_lon"];

$a_lat=$_GET["a_lat"];

$a_lon=$_GET["a_lon"];



$mid_lon = ($s_lon+$d_lon)/2;

$mid_lat = ($s_lat+$d_lat)/2;


$radius=($mid_lon-$s_lon)*($mid_lon-$s_lon)+($mid_lat-$s_lat)*($mid_lat-$s_lat)+.(


function find_nearest_node($s_lon,$s_lat,$d_lon,$d_lat,$a_lon,$a_lat,$lonlat,

$mid_lon,$mid_lat,$radius,$g,$latest_path)

{

   $clicked_node = array();

   $clicked_node[2] = array();//the graph will be here

   $slon=100;

   $dlon=100;

   $alon=100;

   if (is_array($lonlat[0])) {

   foreach($lonlat[0] as $id=>$lon)

   {
```

```
$distance_check = ($lon-$mid_lon)*($lon-$mid_lon)
+($lonlat[1][$id]-$mid_lat)*($lonlat[1][$id]-$mid_lat);
    if($distance_check>$radius)
{
//unset($g->nodes[$id]);
//continue;
}
//now we have to determine the euclidean distance between
//the prev and curr nodes
$square_lon = ($lon-$s_lon) * ($lon-$s_lon);
$square_lat = ($lonlat[1][$id] - $s_lat) * ($lonlat[1][$id] - $s_lat);

$sd = $square_lon + $square_lat;
//$distance_rounded = sprintf ("%.2f", $distance);

if($sd<$slon)
{
$slon = $sd;
$source_id = $id ;
}
```

```
//now we have to determine the euclidean distance between
//the prev and current nodes
$square_lon = ($lon-$d_lon) * ($lon-$d_lon);
$square_lat = ($lonlat[1][$id] - $d_lat) * ($lonlat[1][$id] - $d_lat);

//$distance_rounded = sprintf ("%.2f", $distance);
$dd = $square_lon + $square_lat;
if($dd<$dlon)
{
$dlon = $dd;
$destination_id = $id;
}

   }
   }

   if(is_array($latest_path)){


foreach($latest_path as $id){
```

```
$square_lon = ($lon-$a_lon) * ($lon-$a_lon);

$square_lat

= ($lonlat[1][$id] - $a_lat) * ($lonlat[1][$id] - $a_lat);


$ad = $square_lon + $square_lat;

if($ad<$alon)

{

$alon = $ad;

$alter_id = $id;

}

   }


    unset($g->nodes[$alter_id]);

}




   $clicked_node[0] = $source_id;

   $clicked_node[1] = $destination_id;

   $clicked_node[2] = $g;
```

```
   return $clicked_node;
}



function runTest($s_lon,$s_lat,$d_lon,$d_lat,$a_lon,$a_lat,
$graph,$lonlat,$mid_lon,$mid_lat,$radius,$latest_path) {
//read xml
$result_array = array();
$result_array[0] = array();//path
$result_array[1] = $lonlat[0];//lon
$result_array[2] = $lonlat[1];//lat
$g = new Graph();
$g = $graph;
        //var_dump($graph);
$clicked_node = array();


        ///finding source_id and destination_id
     $clicked_node = find_nearest_node($s_lon,$s_lat,$d_lon,$d_lat,
$a_lon,$a_lat,$lonlat,$mid_lon,$mid_lat,$radius,$g,$latest_path);
        $source_id = $clicked_node[0];
        $destination_id = $clicked_node[1];
```

```
$g = $clicked_node[2];
list($distances, $prev) = $g->paths_from($source_id);
$path = $g->paths_to($prev, $destination_id);
$result_array[0] = $path;
$_SESSION["path"] = serialize($path);


if(count($path)==0){
 print_r("error");
 die();
}
        //echo "paths $path";
//var_dump($path);
print_r(json_encode($result_array));
}
runTest($s_lon,$s_lat,$d_lon,$d_lat,$a_lon,$a_lat,$graph,
$lonlat,$mid_lon,$mid_lat,$radius,$latest_path);
?>
```

Appendix U

```
<html>
    <head>
        <title>Map</title>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
</script>
<script type="text/javascript" src="jquery.svg.package/jquery.svg.js">
</script>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script src="json2.js"></script>
    </head>
    <body>
        <div id="ourMap"></div>
```

```
<script>

map = new OpenLayers.Map("ourMap");

map.addLayer(new OpenLayers.Layer.OSM());

var markers = new OpenLayers.Layer.Markers( "Markers" );

map.addLayer(markers);

//read the osm file

var id_points = new Array();

var lat_points = new Array();

var lon_points = new Array();

var map_points = new Array();

var lonlat;

var index=0;

var timer_start=0;

var count=0;

var s_lat;

var s_lon;

var d_lat;

var d_lon;

var a_lat=0;

var a_lon=0;

var select_source=1;
```

```
var select_dest=0;

var select_alter=0;

var data;

var vector;

var style = {

strokeColor: '#0000ff',

strokeOpacity: 0.8,

strokeWidth: 4

};

var counter=setInterval(timer, 1000);

function timer()

{

if(timer_start==1)

{

count=count+1;


//Do code for showing the number of seconds here

document.getElementById("timer").innerHTML=

"Dijkstra running " + count + " secs";

}

}
```

```
$(function() {
$("#load").click(function(){



timer_start=1;


var source_id = $("#source").val();
var destination_id = $("#destination").val();


$.ajax({
method: 'get',
url: 'LoadGraph.php',
success: function(data_json) {
timer_start=0;
alert("Graph Loaded");
}


});
});
//broken
$("#dijkstra").click(function(){
```

```
timer_start=1;


var source_id = $("#source").val();
var destination_id = $("#destination").val();


$.ajax({
method: 'get',
url: 'AltRunTest.php',
data: {
's_lon': s_lon,
's_lat': s_lat,
'd_lon': d_lon,
'd_lat': d_lat,
'a_lon': a_lon,
'a_lat': a_lat,
'ajax': true
},
success: function(data_json) {
//alert("ended");
if(data_json=="error"){
alert("No path exists");
```

```
}
else{
var data = new Array();
//data[0] = $path, $data[1] = $lon, $data[1] = $lat;
data = JSON.parse(data_json);
var i=0;
var length = data[0].length;
for(;i<length;i++)
{
map_points[i]=new OpenLayers.Geometry.Point
(data[1][data[0][i]],data[2][data[0][i]]);
}

vector = new OpenLayers.Layer.Vector();
var featureVector = new OpenLayers.Feature.Vector
(new OpenLayers.Geometry.LineString(map_points)
.transform(new OpenLayers.Projection("EPSG:4326"),
new OpenLayers.Projection("EPSG:900913")),null,style);
vector.addFeatures([featureVector]);
map.addLayers([vector]);
```

```
timer_start=0;
}
}
});
});
}); //close $(
//read the osm file end
//marker drawing ends

ourpoint1 = new OpenLayers.LonLat(90.3989,23.7937);
ourpoint1.transform(new OpenLayers.Projection("EPSG:4326"),
map.getProjectionObject());
map.setCenter(ourpoint1,12);

/// get position from click
OpenLayers.Control.Click = OpenLayers.Class(OpenLayers.Control, {
defaultHandlerOptions: {
'single': true,
'double': false,
'pixelTolerance': 0,
'stopSingle': false,
```

```
'stopDouble': false
},

initialize: function(options) {
this.handlerOptions = OpenLayers.Util.extend(
{}, this.defaultHandlerOptions
);
OpenLayers.Control.prototype.initialize.apply(
this, arguments
);
this.handler = new OpenLayers.Handler.Click(
this, {
'click': this.trigger
}, this.handlerOptions
);
},

trigger: function(e) {
lonlat = map.getLonLatFromPixel(e.xy);
lonlat.transform(new OpenLayers.Projection("EPSG:900913"),
new OpenLayers.Projection("EPSG:4326"));
```

```
if(select_source==1)
{
s_lat = lonlat.lat;
s_lon = lonlat.lon;

var ourpoint1=newOpenLayers.LonLat(s_lon,s_lat)
ourpoint1.transform(newOpenLayers.Projection("EPSG:4326" ),
map.getProjectionObject());

markers.addMarker(new OpenLayers.Marker(ourpoint1));

select_dest=1;
select_source=0;
}

else if(select_dest==1)
{
d_lat = lonlat.lat;
d_lon = lonlat.lon;

var ourpoint2=newOpenLayers.LonLat(d_lon,d_lat)
```

```
ourpoint2.transform(newOpenLayers.Projection("EPSG:4326" ),
map.getProjectionObject());

markers.addMarker(new OpenLayers.Marker(ourpoint2));

select_dest=0;
select_alter=1;
}
else if(select_alter==1)
{
a_lat = lonlat.lat;
a_lon = lonlat.lon;

var ourpoint3=newOpenLayers.LonLat(a_lon,a_lat)
ourpoint3.transform(newOpenLayers.Projection("EPSG:4326" ),
map.getProjectionObject());

markers.addMarker(new OpenLayers.Marker(ourpoint3));

style = {
strokeColor: '#ff0000',
```

```
strokeOpacity: 0.8,
strokeWidth: 4
};

select_alter=0;

}
}

});

var click = new OpenLayers.Control.Click();
map.addControl(click);
click.activate();
</script>
<button type="button" id="load" name="load">Load Graph</button>
<button type="button" id="dijkstra" name="dijkstra">Dijkstra</button>
<label id="timer">Dijkstra not started<label/>
</body>
</html>
```

Index