# A PROJECT REPORT ON

# ”HIERARCHICAL OPTIMIZATION OF OPTIMAL PATH FINDING IN TRANSPORTATION APPLICATIONS”

**Under The Guidance of**
**Prof. U.R.GODASE**

**BY**


**ASHWINI MORE(B8238586)** **PUNAM NERKAR(B8238589)**
**RAMOLA NIKAM(B8238591)** **SHRADHA PIDHEKAR(B8238605)**


IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE

OF

**BACHELOR OF ENGINEERING**
**(INFORMATION TECHONOLOGY)**

**AT**

SINHGAD TECHNICAL EDUCATION SOCIETY


**DEPARTMENT OF INFORMATION TECHONOLGY**

**SINHGAD COLLEGE OF ENGINEERING,PUNE 411041**


**Affilated to**

**UNIVERSITY OF PUNE**

**MAY 2012**

# CERTIFICATE

This is to certify that the project report entitles

**"HIERARCHICAL OPTIMIZATION OF OPTIMAL PATH FINDING IN TRANSPORTATION APPLICATIONS"**

Submitted by

ASHWINI MORE (B8238586)
PUNAM NERKAR (B8238589)
RAMOLA NIKAM (B8238591)
SHRADHA PIDHEKAR(B8238605)

is a bonafide work carried out by them under the supervision of Prof.U.R.Godase and it is approved for the fulfilment of the requirement of University of Pune, Pune for the award of the degree of Bachelor of Engineering (Information Technology)

This project work has not been earlier submitted to any other Institute or University for the award of any degree or diploma.

Prof. U. R. Godase
Guide
Department of Information Technology

Prof. A.W. Rohankar
Head
Department of Information Technology

Prof. S. D. Lokhande
Principal
Sinhgad College of Engineering

**Examination**
**Examiner 1:.....**
**Examiner 2:.....**

# Abstract

Efficient path query processing is a key requirement for advanced database applications including GIS (Geographic Information Systems) and ITS (Intelligent Transportation Systems). We study the problem in the context of automobile navigation systems where a large number of path requests can be submitted over the transportation network within a short period of time. To guarantee efficient response for path queries, we employ a path view materialization strategy for precomputing the best paths. We tackle the following three issues: (1) memory-resident solutions quickly exceed current computer storage capacity for networks of thousands of nodes, (2) disk based solutions have been found inefficient to meet the stringent performance requirements, and (3) path views become too costly to update for large graphs. Proposed HEPV(Hierarchical Encoded Path View) approach addresses these problems while guaranteeing the optimality of path retrieval.

# Acknowledgement

It is our privilege to acknowledge with deep sense of gratitude towards all, who gave us the possibility of completing this project. We would also like to express our sincere gratitude to our project guide,Prof.U.R.Godase for her valuable suggestions and guidance throughout our course of study and timely help given for completing project.

This project would not have been possible without the support of our H.O.D. Prof. A. W. Rohankar and our principal Prof. S.D.Lokhande. Their knowledge and assistance has always been supportive and helpful throughout the course of our Bachelor of Engineering.

Special thanks to our colleagues for providing the support in developing the project, who backed our interest by giving useful and timely suggestions and for sharing the literature and invaluable assistance.

We wish to express love and gratitude to our families; for their understanding and moral support.

Date: 17/06/2012

Place: Pune

# List of Figures

# Contents

# Chapter 1

## INTRODUCTION

## 1.1 BACKGROUND

In early days,it was much difficult for an individual to travel from one place to another.To reach a particular destination,while travelling at every turn he had to ask other person for the right path.This was quite time consuming and a reason for unnecessary extra fare at times if he go in a wrong way.

So to overcome these problems and to reach the particular destination in lesser time with optimal path,this idea emerged.By using this HEPV algorithm,we can get the most optimum path between any two places.

### 1.1.1 Basic Concepts

**a) FlatGraph :**

The graph of nodes and links where nodes represents the places and links represents the path is called the flat graph.

**b) Local Node :**

The node which is present in only one fragment is called as local node.

**c) Border Node :**

The node which is present in more than one fragment is called as border node.

**d) Fragments :**

The flat graph is divided into smaller graphs to create a hierarchy of a graph are called as fragments.

In our project we are taking flat graph of 30 nodes. The nodes are as follows:

1. Pune Station
2. Sanchetee Hospital
3. Shivaji Nagar

4. Manapa
5. Shaniwar Wada
6. Swargate
7. Hirabaug Bus stop
8. Bharti Vidyapith
9. Saras Baug
10. Katraj
11. Balgandharv
12. Crompton Greaves
13. Aundh
14. FC
15. Garware Chowk
16. Dandekar Pool
17. Nal Stop
18. Mhatre Pool
19. Karnataka School
20. Tathawade Udyan
21. Rajaram Pool
22. Tukai Nagar
23. Kothrud Depo
24. Karve Putala
25. Kothrud Stand
26. Karve Nagar
27. Meenakshi Puram
28. SCOE
29. Vadgaon BK
30. Aanand Nagar

The flat graph looks like as given in next figure

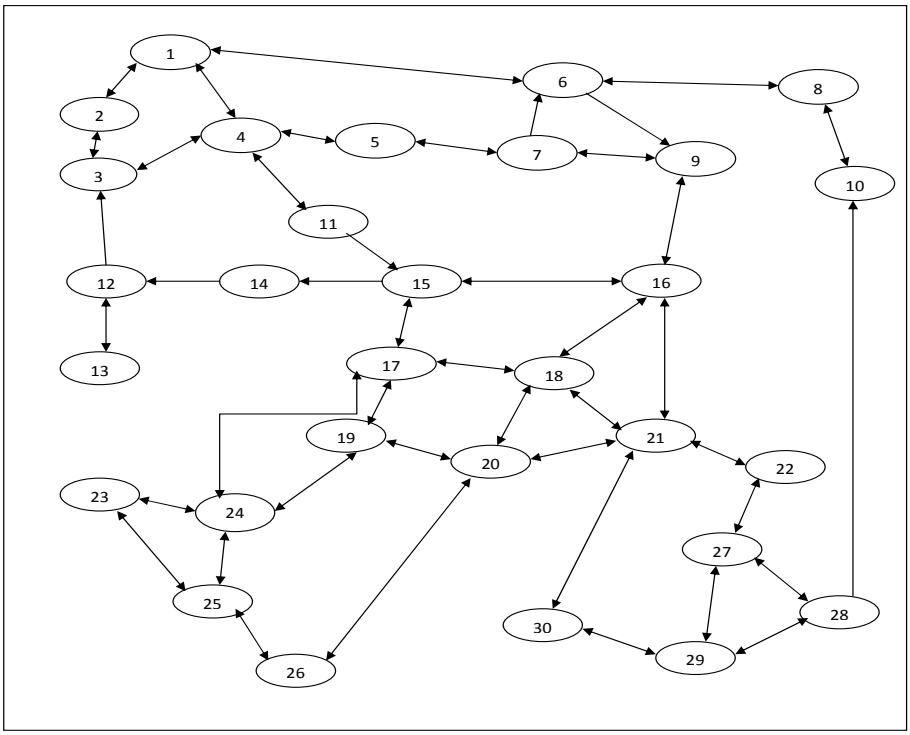



Figure 1.1: Flatgraph

Above figure shows the graphical representation of selected area. It is called as flat graph. The flat graph is divided into partitions. In our project we have only 1 partition and 4 fragments of that partition. The figures of fragments are as shown below.

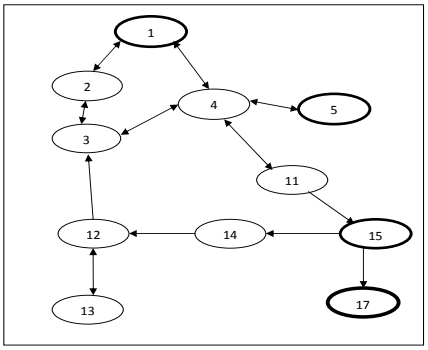



Figure 1.2: Fragment1

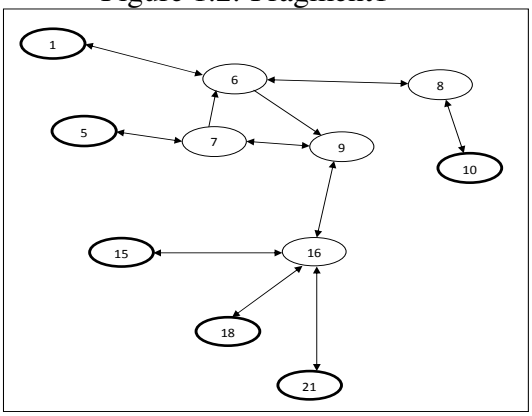



Figure 1.3: Fragment2

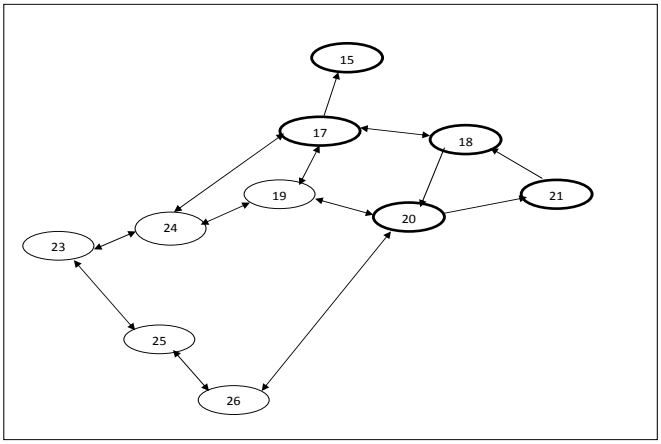



Figure 1.4: Fragment3

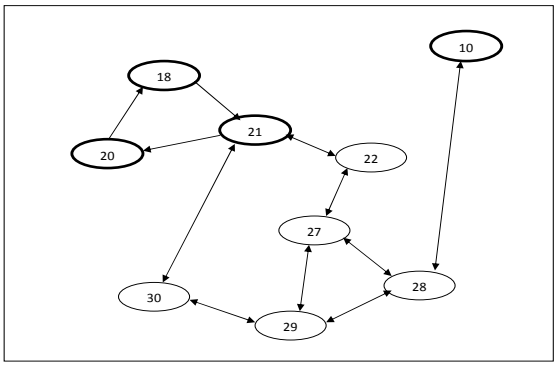



Figure 1.5: Fragment4

The nodes which are highlighted are border nodes and remaining are local nodes. First we divide the flat graph into no. of small graphs and then by taking a border nodes we create a super graph, which is used to find the optimal path between source and destination. Thus we create a hierarchy of graph.

## 1.2 LITERATURE SURVEY

Dijikstra algorithm is used to compute the shortest path between two nodes. It calculates the path on demand. It does not precompute the path. It does not use the concept of fragments. It does not guarantee the optimal solution. It can not handle the large graphs. Thus it is not efficient algorithm for shortest path finding.

Transitive closure algorithm[2,3] precomputes the paths between source and destination. It does not use fragment also it does not guarantee optimality of a solution. Theoretically it can handle the huge graph but while much research has been conducted by both the theory and database communities on path finding, many suggested transitive closure algorithms do not effectively handle path computation on large graphs in real-time application domains such as ITS (Intelligent Transportation Systems) systems. With new computer architectures and networks, the traditional algorithms were adapted to parallel and distributed transitive closure algorithms[5].

As an alternative approach, precompute all-pair shortest paths and store them in an encoded path view structure. Queries can be efficiently answered by performing simple lookups on the precomputed path view. This approach is a trade-off between computing paths from scratch and precomputing all paths. While this approach has been proven to be promising for relatively small map data sets, it was determined that it exceeds the memory capacity of typical computer systems for large graphs (e.g., graphs of 3,600 nodes or larger) and the performance of computing the path view deteriorates quickly (e.g., it takes 4 minutes for a graph with 3,600 nodes on a Sun SPARC-20).

The hierarchical abstraction has been proposed by some researchers to overcome this problem[5]. for example divided a relation into fragments and introduced the notion of high-speed fragment. Unfortunately, the formation of high speed fragments are very sensitive to the update of the underlying base relation. Therefore the authors recommended this approach only for rather stable base relations. Some researches have presented a hierarchical graph model which classifies links according to road types and pushes up high speed roads such as highways to the next higher level of hierarchy. However, the paths retrieved from such hierarchical graphs are not guaranteed to be optimal.

A hierarchical A* algorithm for road navigation systems, which while more efficient than flat A* does not guarantee optimality either. It can handle large graph. It uses both the techniques of precomputation and on demand computation.

Similarly, some other researchers proposed another hierarchical multigraph model by dividing graph into subgraphs and pushing up the precomputed paths as well as links between the boundary nodes. The paper did not discuss how the optimality can be achieved. Hierarchical graph refreshing essential for road navigation to reflect the dynamic traffic condition was also not handled.

## 1.3 PROBLEM DEFINITION

### 1.3.1 Software Requirement Specification

**Problem Statement:**

To find an optimal path for an individual to travel in an alien place such that intermediate nodes are revealed. The capability of computing path queries is an essential feature for advanced database applications. Our goal is to explore solutions for path finding in general with particular focus on addressing the problems inherent to navigation systems applications.

**Intended Audience and Reading Suggestions:**

This SRS is used by developers, project managers, marketing staff, users, testers and documentation writers for different purposes like developer refers this SRS for coding, testers to build test plans, marketing staff for deciding marketing strategies. This SRS also contains scope of the project, references, project features etc.

**Project Scope:**

This system is developed under windows environment in JAVA.. Simple database are used and also simple program files are used. There are separate files for each transaction.

1. We are concerned with providing answers to path queries issued by a potentially large number of concurrent requests (e.g., during peak rush hour periods).

2. Our solution must handle the dynamic nature of transportation networks, i.e., it must provide up-to-date query results even when the underlying transportation networks change frequently.

3. Our solution must provide response at a near real-time level of performance (i.e., within seconds).

4. Based on the requirements of instruction based navigation systems, we are interested in efficiently determining the next link for the desired path rather than necessarily retrieving the complete path all at once.

**Product Perspective:**

The product is standalone application and It is neither a distributed nor a client-server system.

**Product Features:**

1. Optimal Path

2. Intermediate nodes

**User Classes and Characteristics:**

1. User-requires some knowledge about computers. How to use the system.

2. Employee should be well trained to handle the application. He/she must know all the features and functions of this software,so he can update the database.

**Operating Environment :**

Windows operating system

**Design and Implementation Constraints:**

Databases to be used : Oracle 10g

Language requirements : JAVA

**User Documentation:**

User manuals, help ,CD and tutorials

**Assumptions and Dependencies:**

The graph of interconnected places within city is predefined .This graph can be modified depending on requirement. The computer system on which our product is installed should have sufficient resources to store this information.

We assume that our product does not provide networking capabilities. It is a standalone application.

### 1.3.2 Project Plan

# Chapter 2

## PROJECT PLANNING AND MANAGEMENT

### 2.1 PROJECT DEVELOPMENT LIFECYCLE

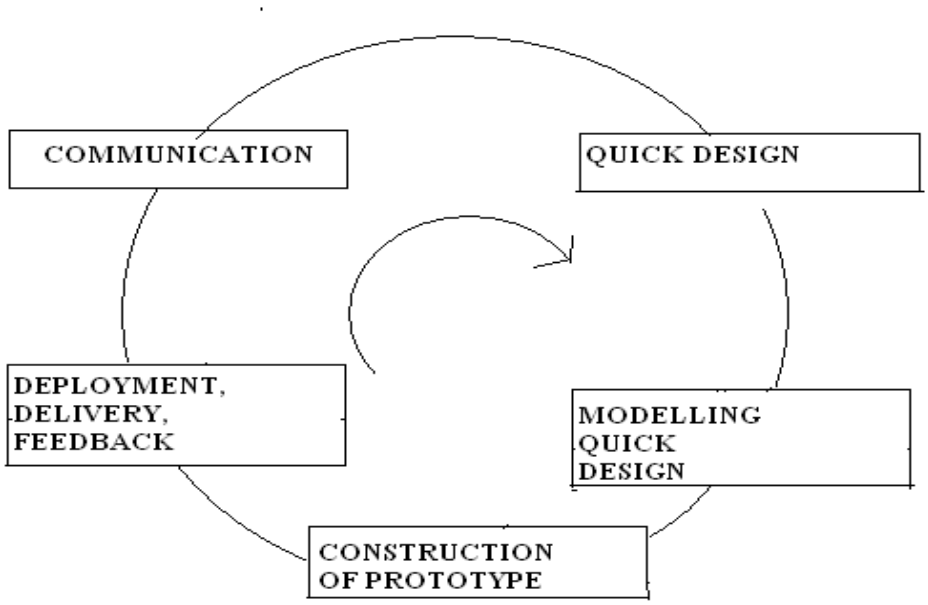



Figure 2.1: Project Devlopment Life Cycle

**1. Requirement Gathering:**

The project development lifecycle starts with understanding all the requirements of the system. It involves the communication between the user and the developer. Based upon the functionalities of the system and the points stated by the end user the requirements of the project are decided. It also involves generating the Software Requirement Specification (SRS).

Requirement of project is

1. Flatgraph with numbered nodes, weight
2. Database that stores the flatgraph, partition and supergraph
3. GUI that makes path viewing easier
4. Authentication of user for security
5. Platform Independent language to be used for coding

**2.Planning:**

The planning phase includes complete estimation and scheduling of the activities to be performed in the project. It includes the distribution of work, the estimated time to complete the project, the estimated cost, job scheduling and tracking.

Information gathering, study of IEEE papers and estimation of resources required is completed before beginning of documentation and designing.

After completion of documentation and design project work will be evenly distributed in team. Team will first study the project flow. Database being less complex will be designed precisely by one team member. Then another team member will work on the gui design. The other two members will begin with coding part. All these sections have to start simultaneously.

Project will be divided roughly into 3 phases. Phase 1 has to be completed till mid December. Phase 2 has to be completed till January end and Phase 3 has to be completed till February end.

**3.Modeling and Design:**

It includes detail requirement analysis and project design. It also involves development of algorithms and various architectural concepts like use case diagrams, flowcharts etc.

Use Case Diagram, Class Diagram, DFD, ERD, Activity Diagram, Sequence Diagram and State Transition Diagram are designed before 2nd Review. Complete detailing and flow of project is demonstrated by these diagrams.

**Project is divided in 3 phases**

**Phase 1-**

Algorithms will be thoroughly studied before beginning of database design. Flatgraph

will be designed and module will be designed to encode the flatgraph. The encoded flatgraph will be stored in database.

Gui will be developed. Authentication for system security will be designed.

**Phase 2-**

After encoding the flatgraph into database coding for partition and supergraph must begin. Database design for storing partition and supergraph will be done.

Gui for creating, encoding partition and supergraph will be designed simultaneously.

**Phase 3-**

During coding of path retrieval normalization of database and perfection of gui will be carried out. Development of Path Retrieval module will be time consuming thus in this phase focus will be mainly on path retrieval module.

**4.Construction:**

It includes coding and testing steps:

1. Coding: Design details are implemented using appropriate programming language.
2. Testing: Testing is carried out to check whether flow of coding is correct and to point out the errors of the program.

Coding will be done in following steps

1. Encoding flatgraph
2. Authentication of user
3. Create and encode partition
4. Develop gui for the same
5. Create and encode supergraph
6. Develop gui for the same
7. Develop path retrieval module
8. Develop gui for usage of customer

Testing will be done with overall gui and algorithms. Testing will be performed timely and before deployment of project.

**5.Deployment:**

It includes software delivery , support and feedback from user. If the user suggests some corrections, or demands additional capabilities then changes are required for such corrections or enhancements.

Deployment will be done after complete testing of the project.

## 2.2 PROJECT DELIVERABLES

Deliverables for our project will include:

1.**Project report**

It includes Software Requirement Specification, All design details, technical diagrams, schedule, cost, testing details etc..

2.**CD**

It contains the Hierarchical Optimization Of Optimal Path Finding System GUI set up.

3.**Manual**

Provides all procedures required while using the system. Also contains the Help.

## 2.3 TASKS AND MILESTONES

### 2.3.1 Tasks

**The following tasks have to be performed:**

1. Database with tables required for flat-graph, partitions and supergraph will be created.
2. Authentication mechanism will be developed.
   Admin only is allowed to create, encode partitions and supergraph.
3. Modules to create and encode partitions and supergraph will be developed.
4. Optimal Path Retrieving module will be developed.

### 2.3.2 Milestones

In general the following major milestones can be considered ”completed” when the criteria noted have been met.

1) Development of different modules: It is necessary to divide the project in different modules and work on those modules individually.

2) Testing of modules individually: The modules which are developed are then tested one at a time. All the elements related to a module are tested and if needed modification is done at that time.

3) Module Integration : After all the modules are tested individually they are combined to form the whole project.

4) Testing Project as a Whole : After integrating the modules the project is tested as a whole to see if any bugs are left. They are rectified then and the project is then ready for

Deployment.

5) Follow up of Guidelines : It is seen that all Guidelines given are maintained and all the expectations of the user from the application are fulfilled. Also the application should be user friendly.

**Hierarchical Optimization of Optimal Path Finding System will have following milestones:**

1. Authentication of Admin
2. Creation and encoding of Partitions from flatgraph
3. Creation and encoding of Supergraph from partitions
4. Maintenance of Hierarchical Encoded Path View(HEPV)
5. Optimal Path Retrival using Hierarchical Encoded Path View(HEPV)

## 2.4 COST AND EFFORT ESTIMATION

The constructive Cost Model is generally used for estimation measure of project cost, project duration, manpower etc.

Like all estimation models, the COCOMO model requires sizing information .This information can be specified in terms of:

- Object Point(OP)
- Function Point(FP)
- Lines Of Source Code(KLOC)

For our project we use sizing information in the form of line of source code(KLOC).

- Total lines of code for our Project,KLOC = 3.6k(approx)
- Cost of each person per month Cp = Rs.1500/-(approx)

**Efforts**

Equation for calculation of efforts in person-months for COCOMO model is:

$$E = a*(KLOC)^{b}$$

Where,

a=3.0

b=1.12

E=Efforts in person-months

$$E=3.0*(3.6)^{1.12}$$

E=12.59 person-months

Total efforts of 12.59 person-months are required to complete the project successfully

**Duration of Project:**

Equation for calculation of Duration of projects in months for COCOMO model is:

$$D = A * (E)^b$$

Where,

a=2.5

b=0.32

D=Duration of Project in months.

$$D=2.5*(12.59)^{0.32}$$

The approximate duration of project is 5.63 months

**Number of team members:**

The equation for calculation of Number of team members required for completion of project, using COCOMO model is:

$$N = E / D$$

Where,

N=number of team members required.

E=Efforts in person-months.

D=Duration of project in months.

$$N=12.59/5.63$$

$$N=3 \text{ (approx)}$$

**Cost of project:**

Equation for calculation of cost of project, using COCOMO model is:

$$C = D * Cp$$

Where,

C=Cost of Project.

D=Duration of Project in Months.

Cp=Cost incurred per person-month

$$C=5.63 * 6400$$

Therefore total cost of project is Rs. 36032 /-(approx)

## 2.5 RISKS

### 2.5.1 Technical Risks:

No technical risks as such is existing since the Operating System being used is Windows, which is well established as well as accepted. Maintenance problem could exist in system.

### 2.5.2 Product Size Risks:

The size estimate in LOC of our project may be significantly low. This may affect the schedule as well as upset the estimate and total cost of our project.

### 2.5.3 Business Risks:

The top five business risks are

1) Building an excellent product or system that nobody really wants (market risk).

2) Building a product that no longer fits into the overall business strategy for the company (strategy risk).

3) Building a project that the sales force does not understand how to sell.

4) Losing the support of senior management due to a change in focus or change in people (management risk).

5) Losing budgetary or personnel commitment (budget risk).

### 2.5.4 System Failure

If Database crash during the path retrieval then it is tedius task to recover the database.

# Chapter 3

## PROJECT ANALYSIS

## 3.1 USE CASE MODEL WITH SCENARIOS

A Use case diagram captures use case and actor interactions.There are two level of use case diagram one context level and other is requirement level.

### 3.1.1 Use Case level 0

Here is the use case model level 0

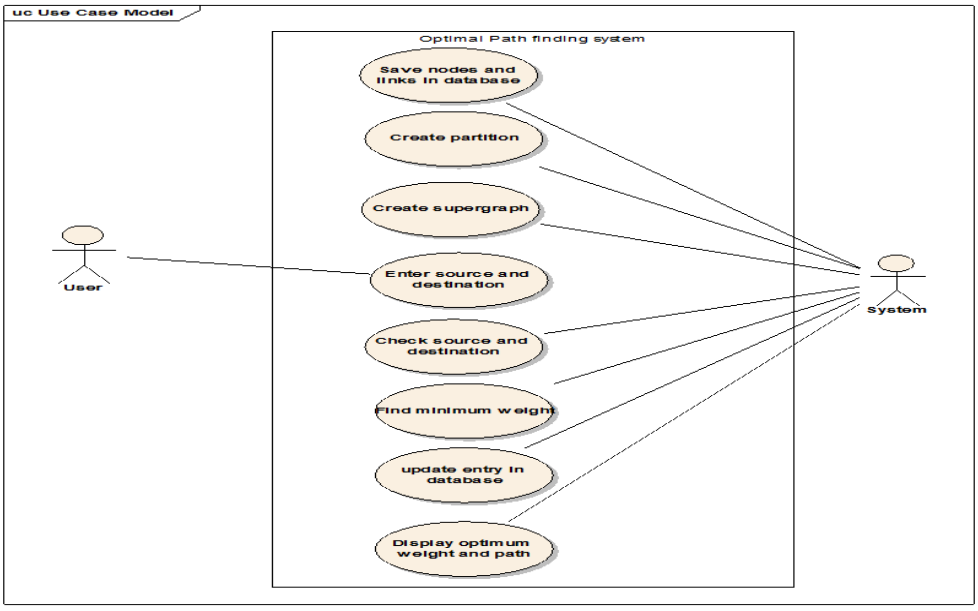



Figure 3.1: Use Case Level 0

### 3.1.2 Use Case level 1

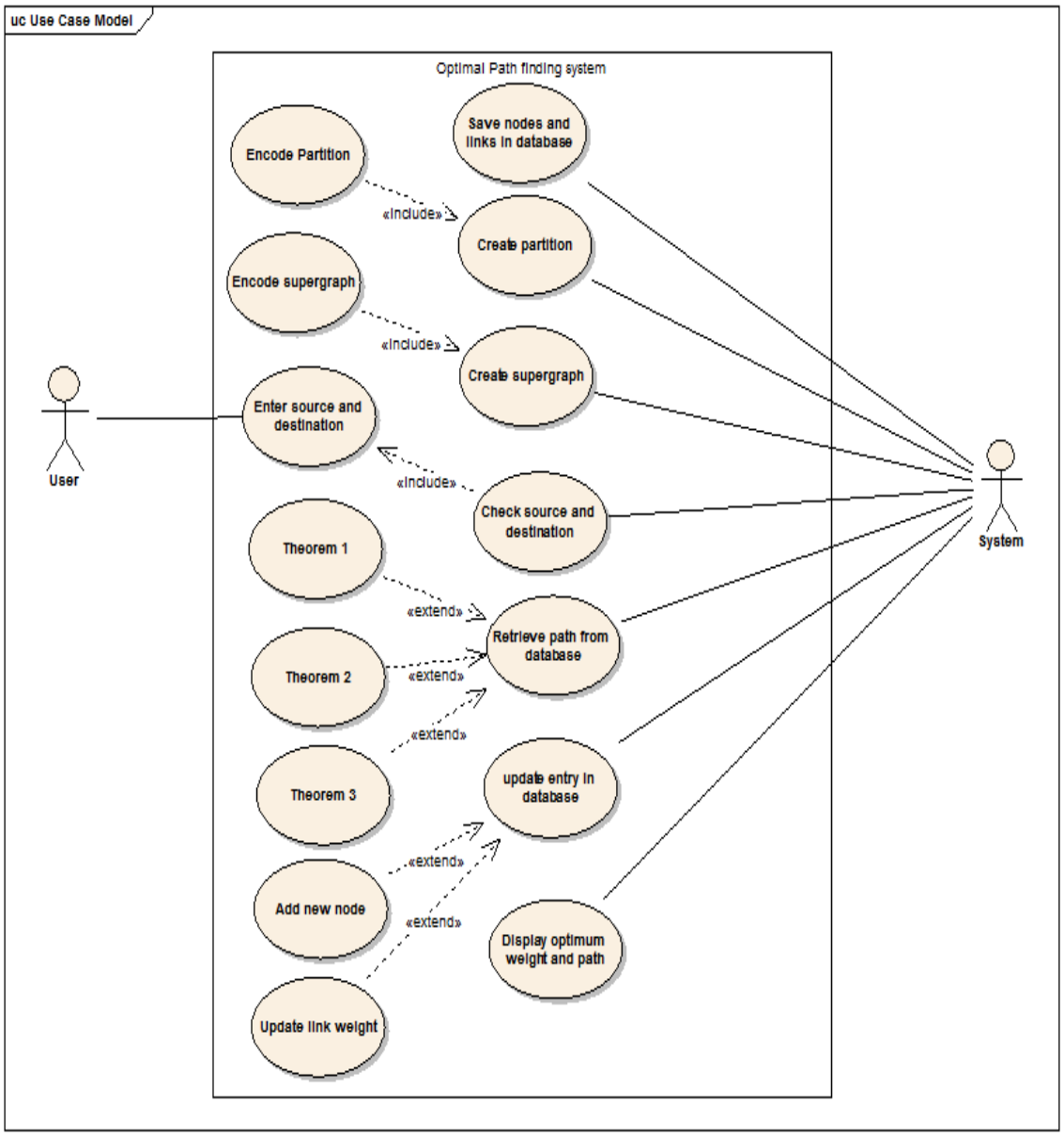



Figure 3.2: Use Case level 1

## 3.2 ACTIVITY DIAGRAM

Activity diagrams are used to model the behaviours of a system, and the way in which these behaviours are related in an overall flow of the system. The logical paths a process follows, based on various conditions, concurrent processing, data access, interruptions and other logical path distinctions, are all used to construct a process, system or procedure.

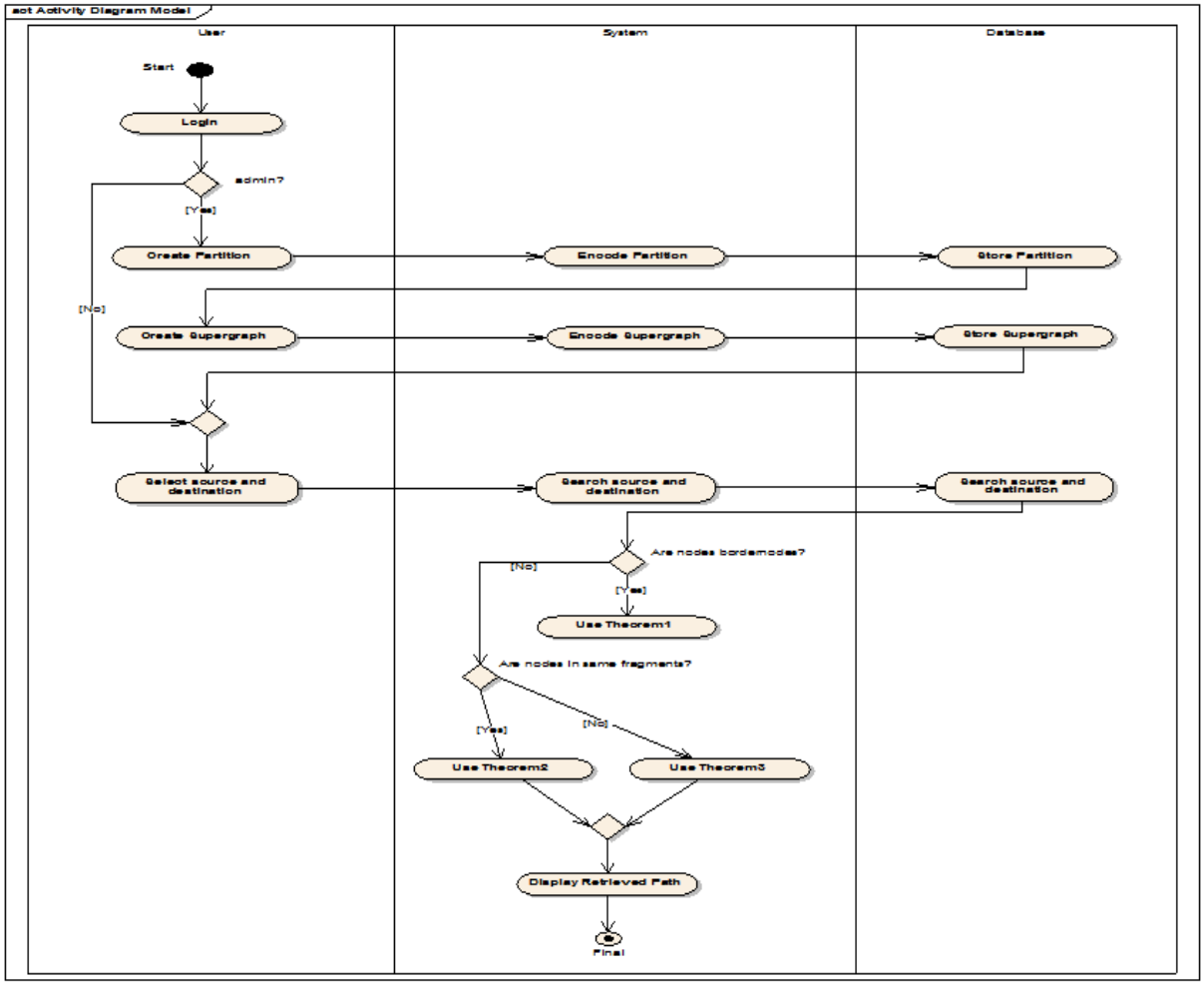



Figure 3.3: Activity Diagram

# Chapter 4

## PROJECT DESIGN

### 4.1 DATAFLOW DIAGRAM LEVEL 0

A data flow diagram (DFD) is a graphical representation of the ”flow” of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design.
DFD shows interaction between external entities and processes and process and data store.

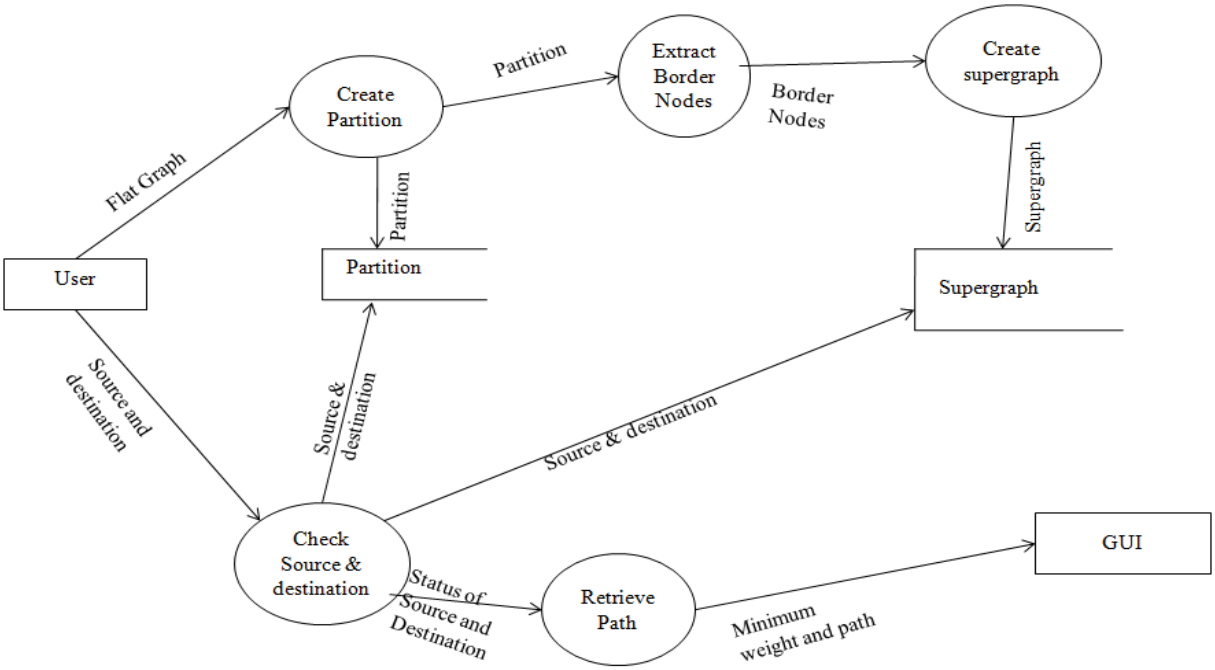



Figure 4.1: Data Flow Diagram

## 4.2 ENTITY-RELATIONSHIP DIAGRAM

Following is the Entity Relationship Diagram which shows relationship between different entities.

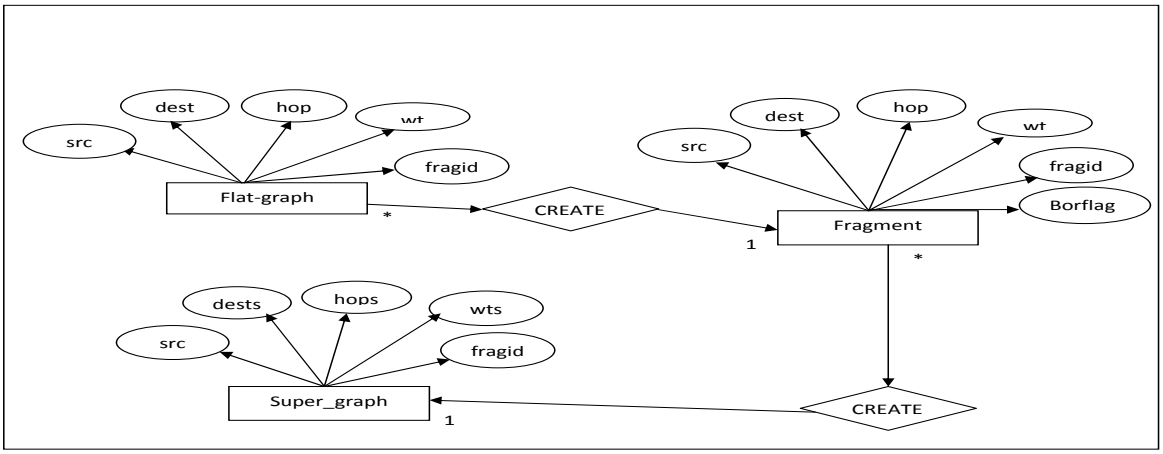



Figure 4.2: Entity-Relationship Diagram

## 4.3 CLASS DIAGRAM

The below diagram depicts the class diagram for image processing. System will comprise of various classes such as image, cartoon, imagesketch etc. The class diagram shows interdependency between the classes and how they communicate between them.

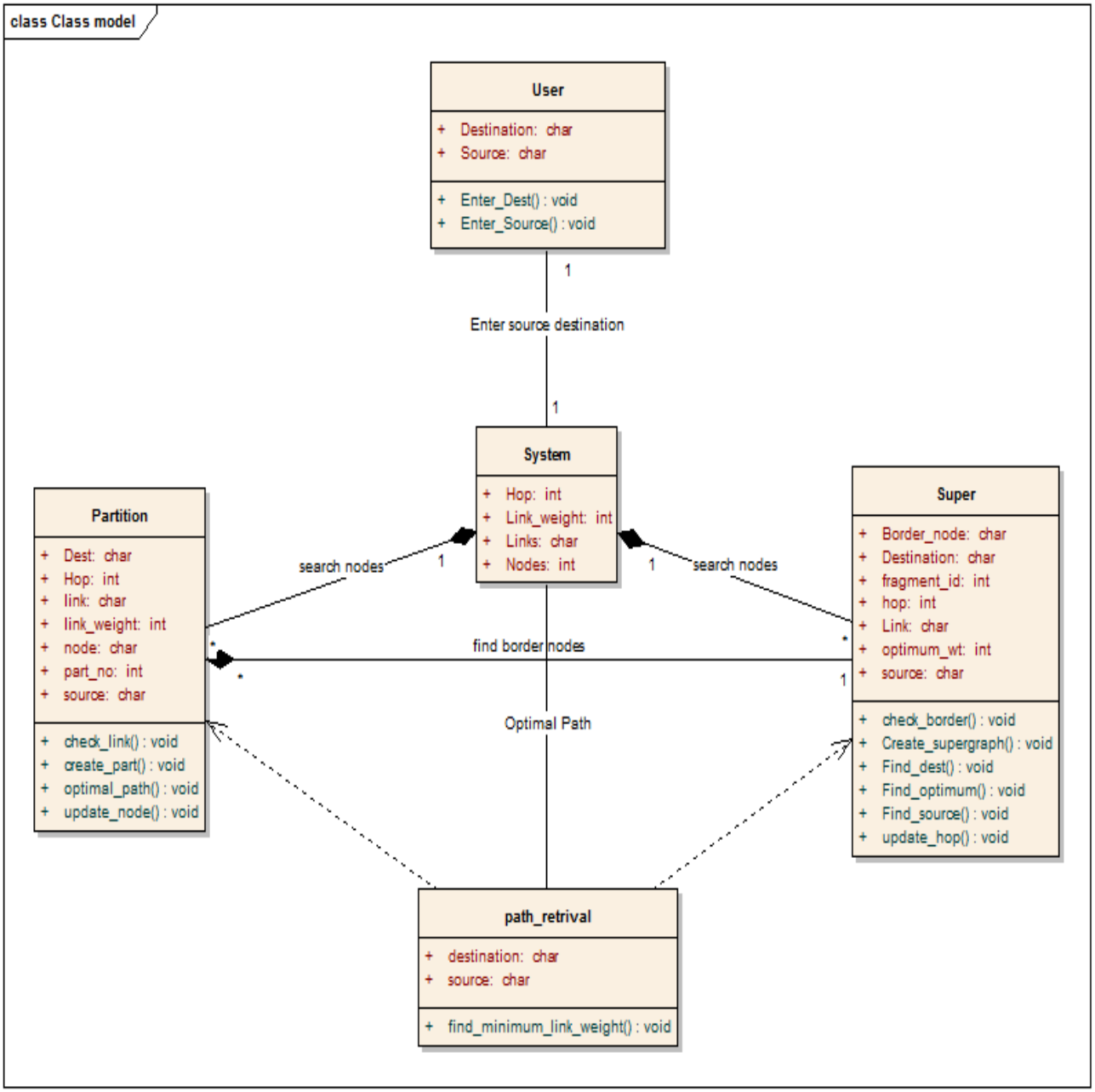



Figure 4.3: Class Diagram

## 4.4 SEQUENCE DIAGRAM

Sequence diagram represents the dynamic behaviour of the system. A Sequence diagram is a structured representation of behaviour as a series of sequential steps over time. The applications sequence diagram shows how the elements of the application pass messages to each other during actual execution.

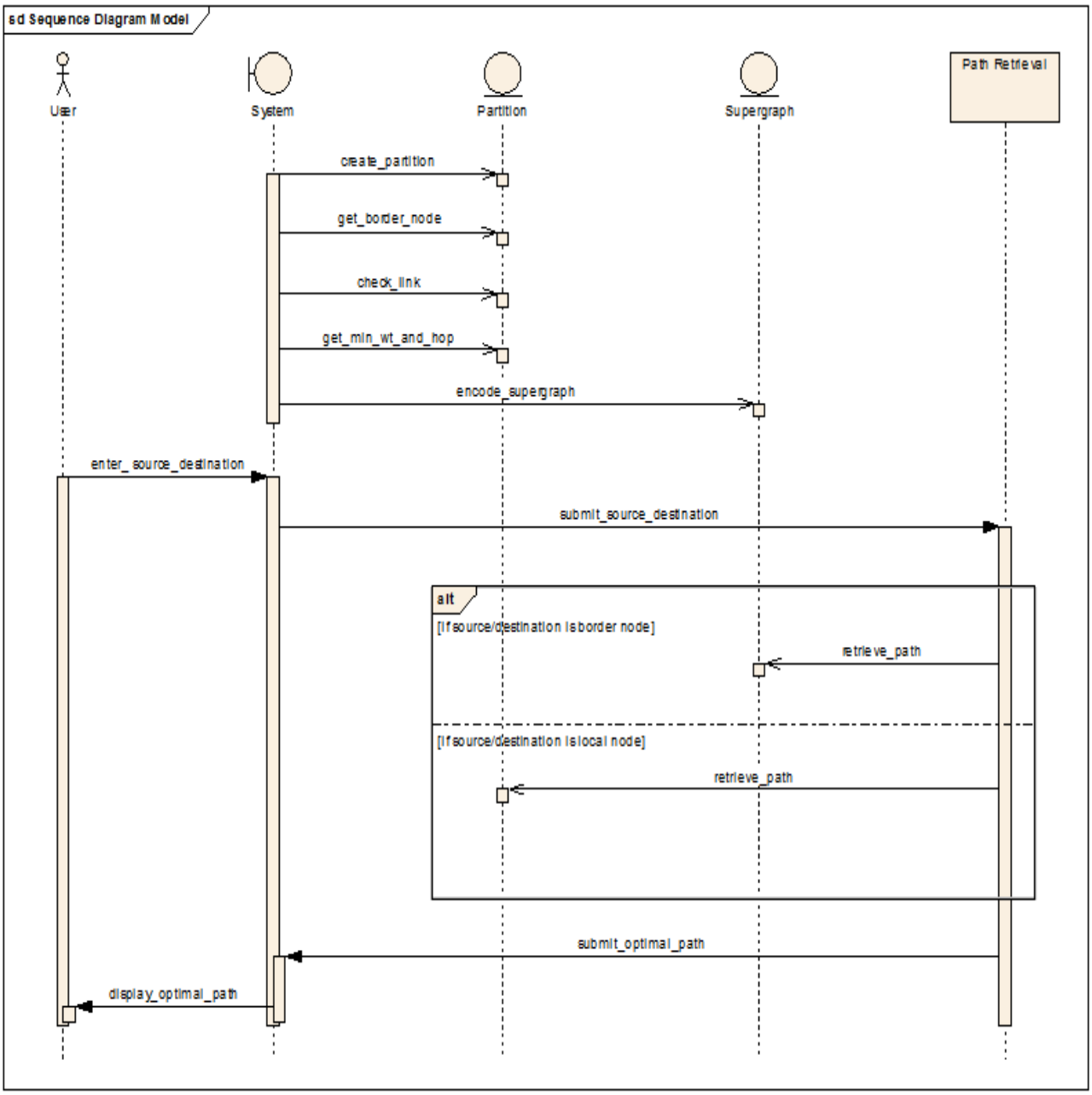



Figure 4.4: Sequence Diagram

## 4.5 COLLABORATION DIAGRAM

Following is the Collaboration diagram which shows the communication between different entities.

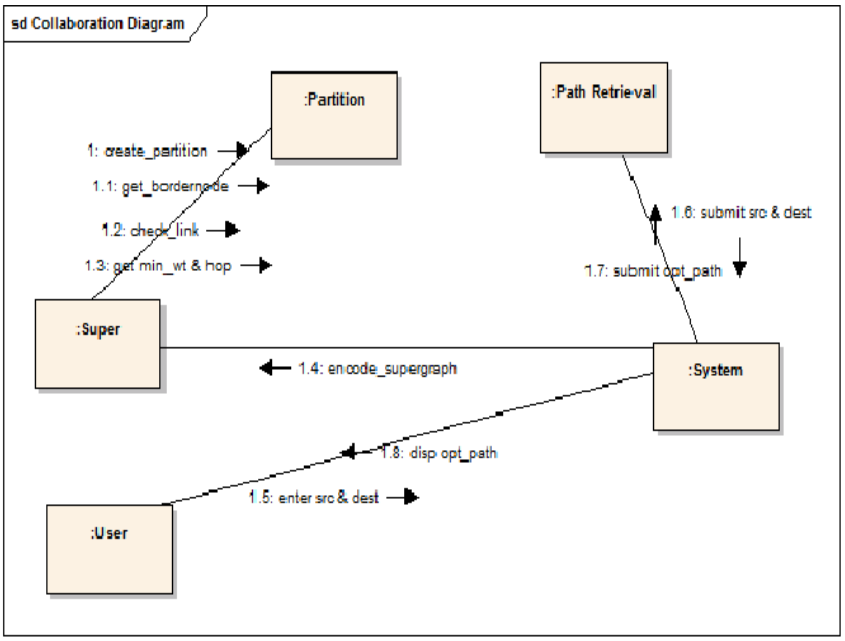



Figure 4.5: Collaboration Diagram

## 4.6 STATE TRANSITION

This diagram is a way of describing the time-dependent behavior of a system. State diagrams are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system.

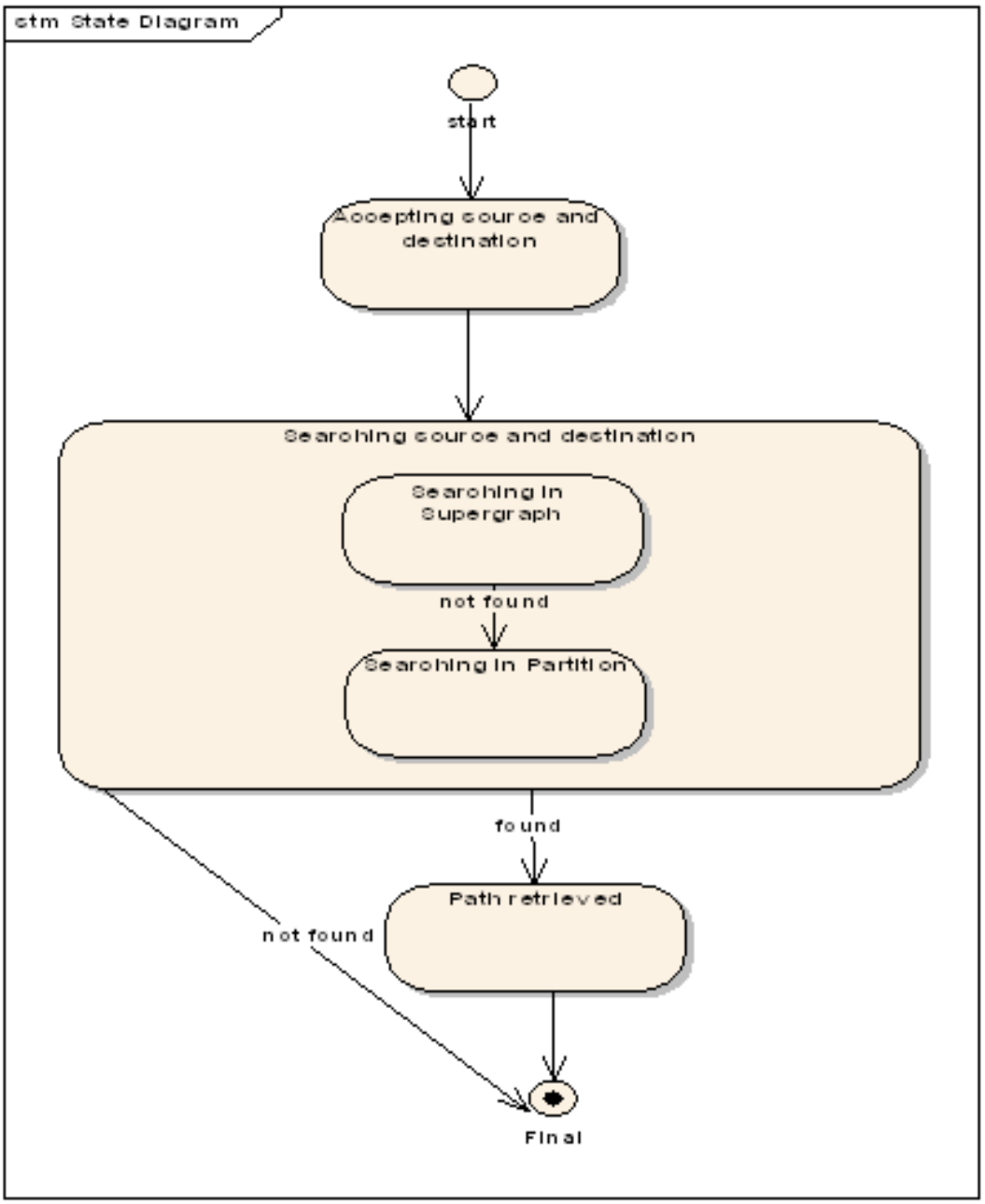



Figure 4.6: State Transition Diagram

## 4.7 DEPLOYMENT DIAGRAM

A deployment diagram in the Unified Modeling Language models the physical deployment of artifacts on nodes.

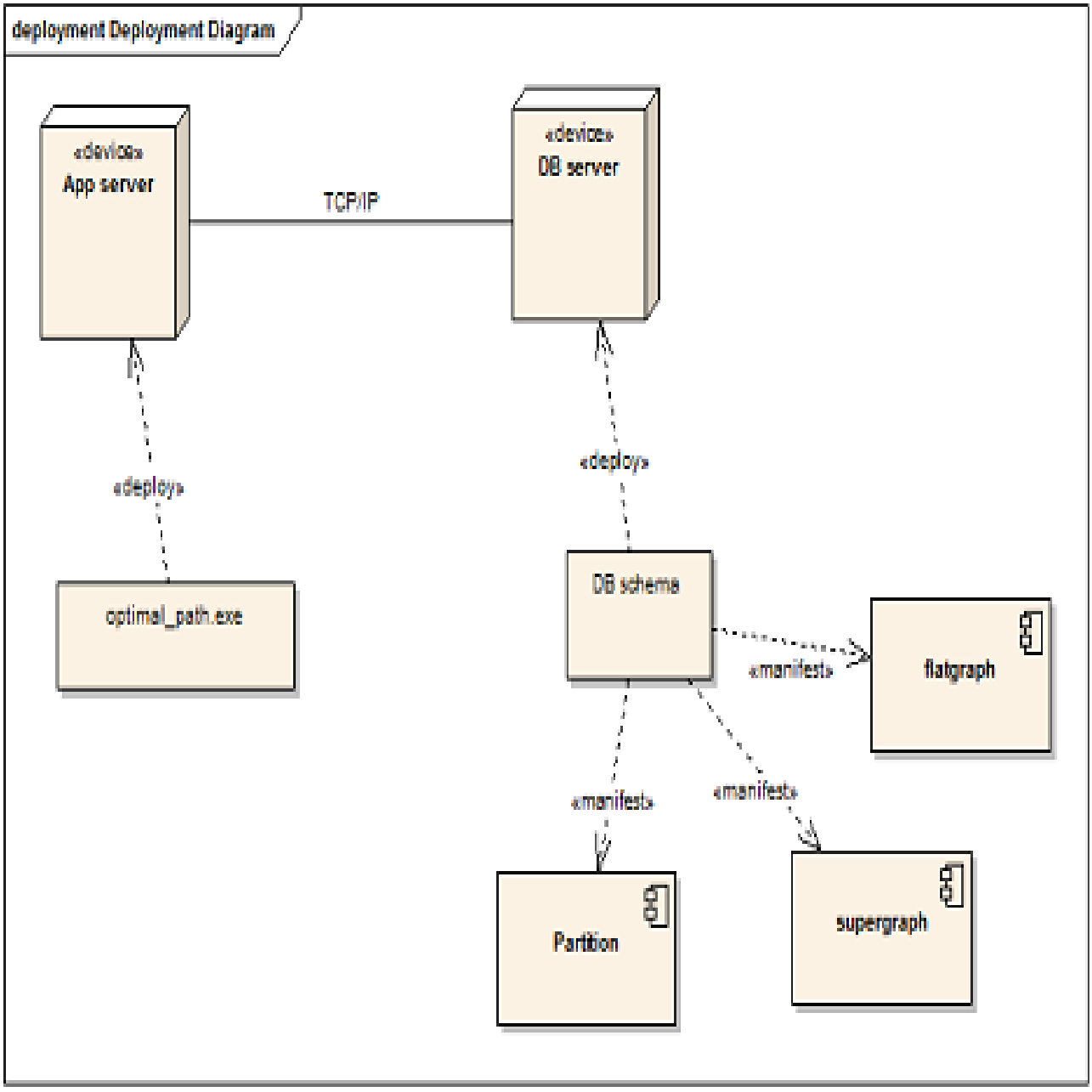



Figure 4.7: Deployment Diagram

# Chapter 5

## PROJECT IMPLEMENTATION AND CODING

### 5.1 DATABASE IMPLEMENTATION

The database used in this project is as given below. It gives information about the tables and their attributes.

**1.Source**

src - varchar2 - Primary Key

**2.Map**

src - varchar2 - Foreign Key

src name - varchar2

**3.Flatgraph**

src -varchar2 - Foreign Key

dest - varchar2

wt -float

fragid - number

**4.frag**

fragid - number

src - varchar2 - Foreign Key

hop - varchar2

wt - float

bor - number

**5.Supergraph**

src - varchar2 - Foreign Key

dests - varchar2

hops - varchar2

wts - float

fragid - number

**6.Login**

Username -varchar2- Primary Key

Password - varchar2

adminflag - number

## 5.2 GUI IMPLEMENTATION

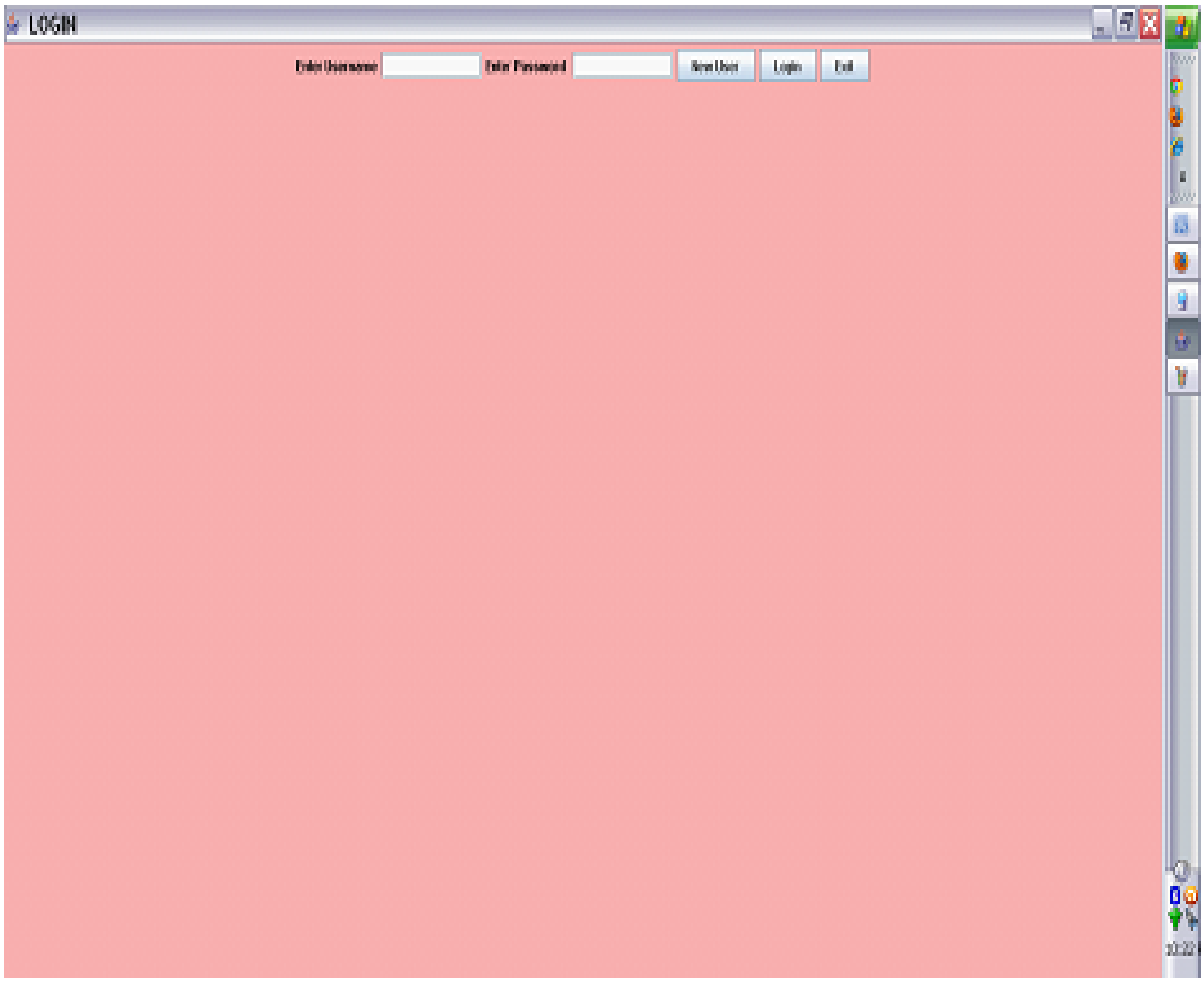



Figure 5.1: Login Form

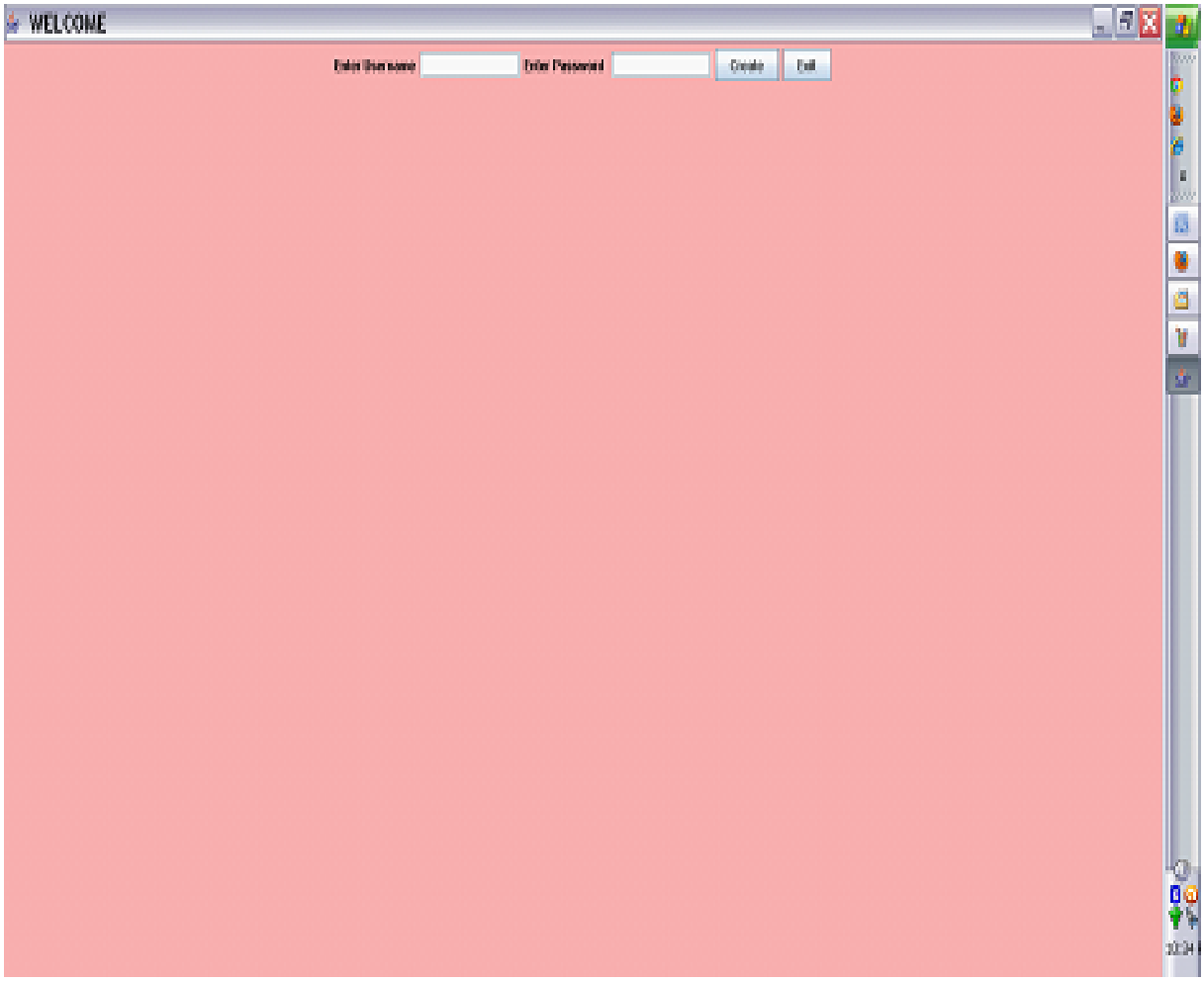



Figure 5.2: New user form

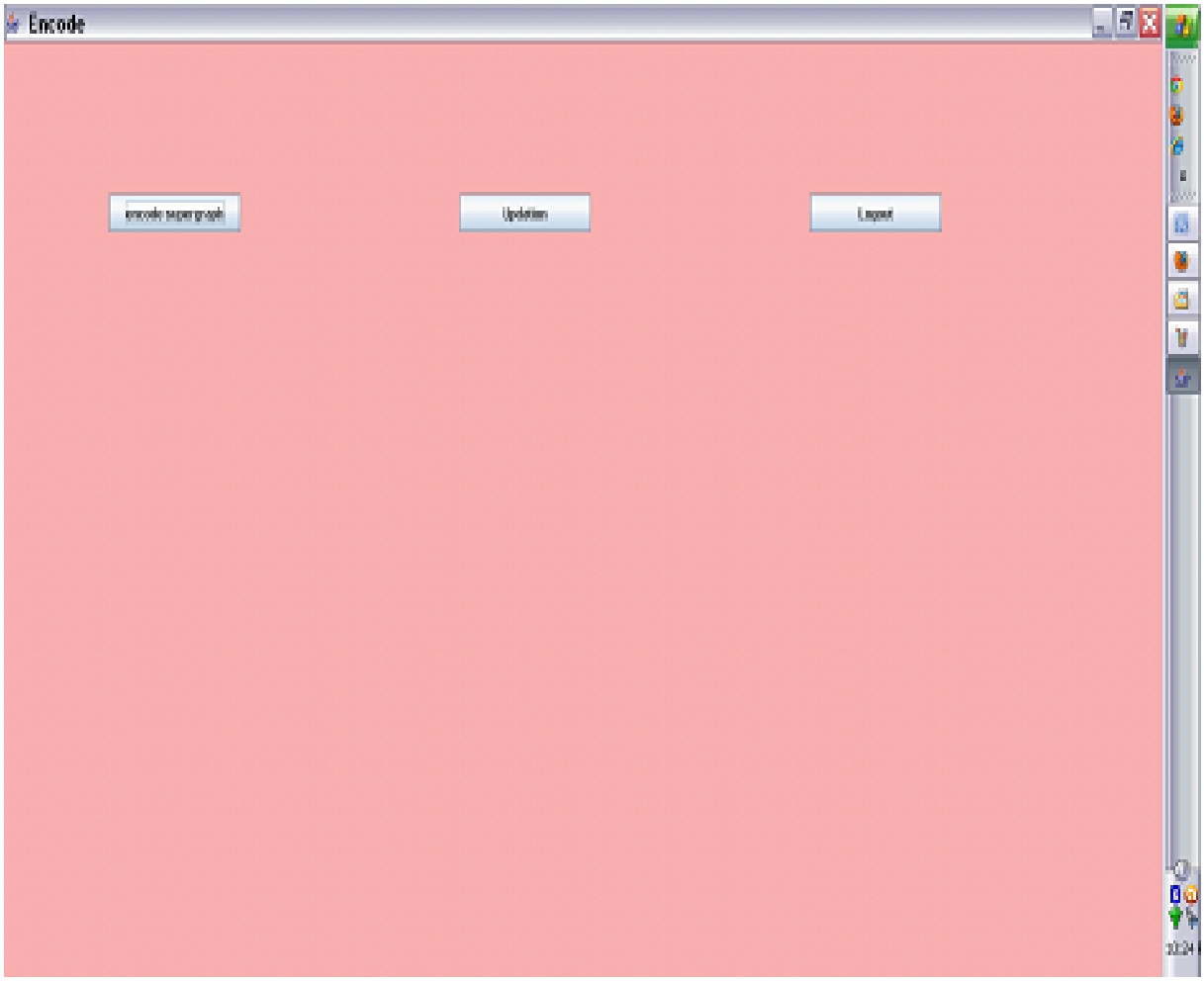



Figure 5.3: Admin form

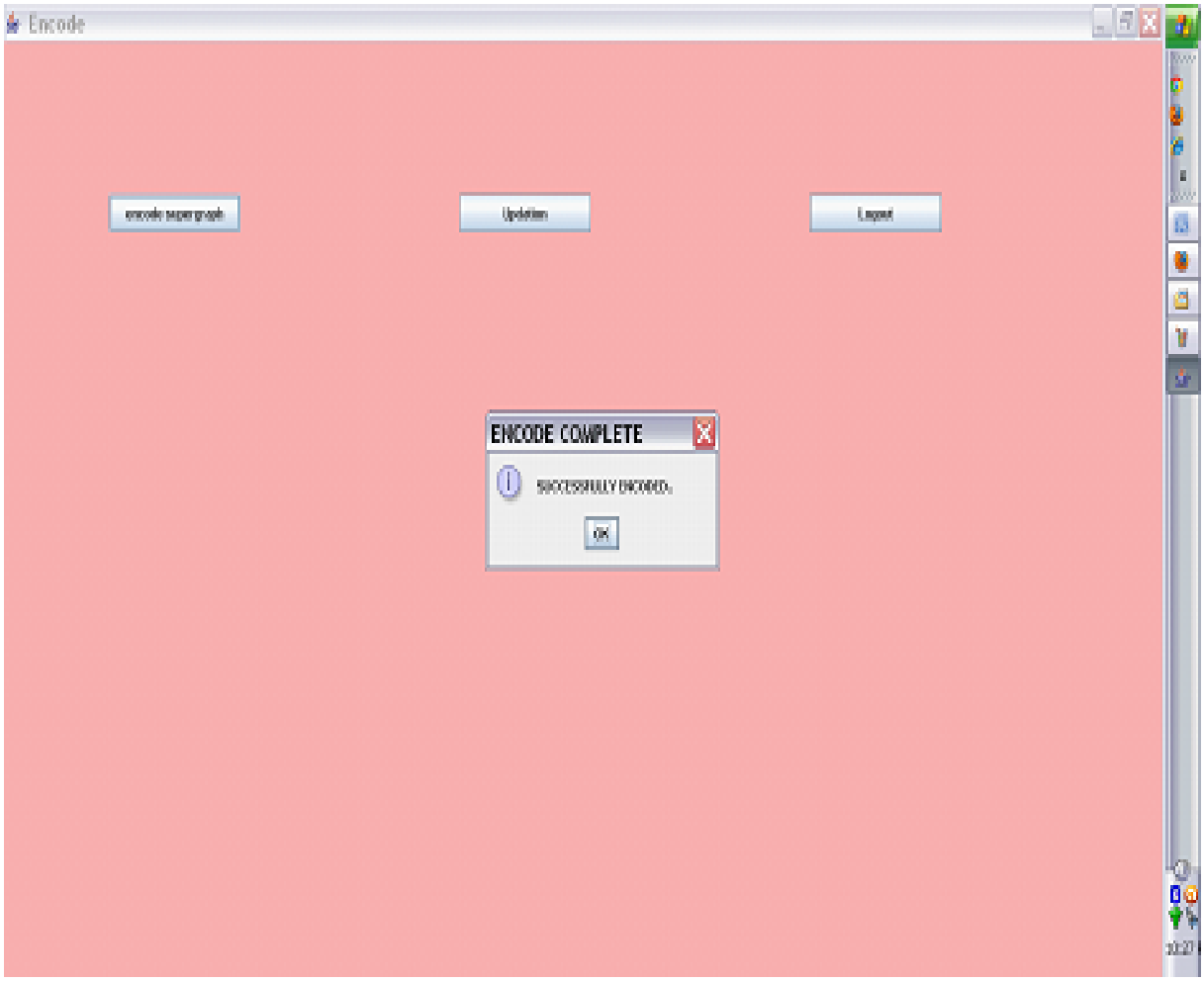



Figure 5.4: Encode Form

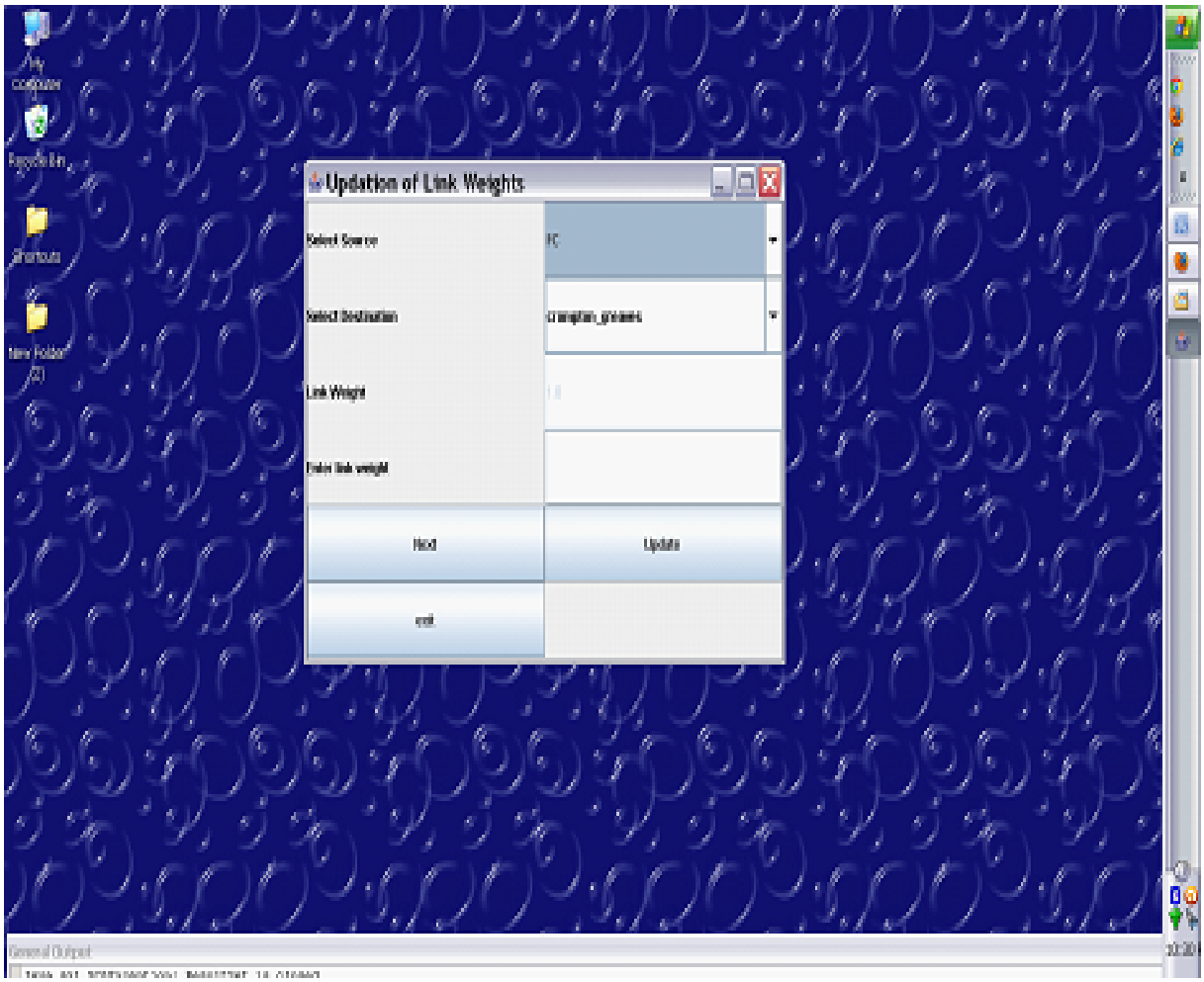



Figure 5.5: Update Form

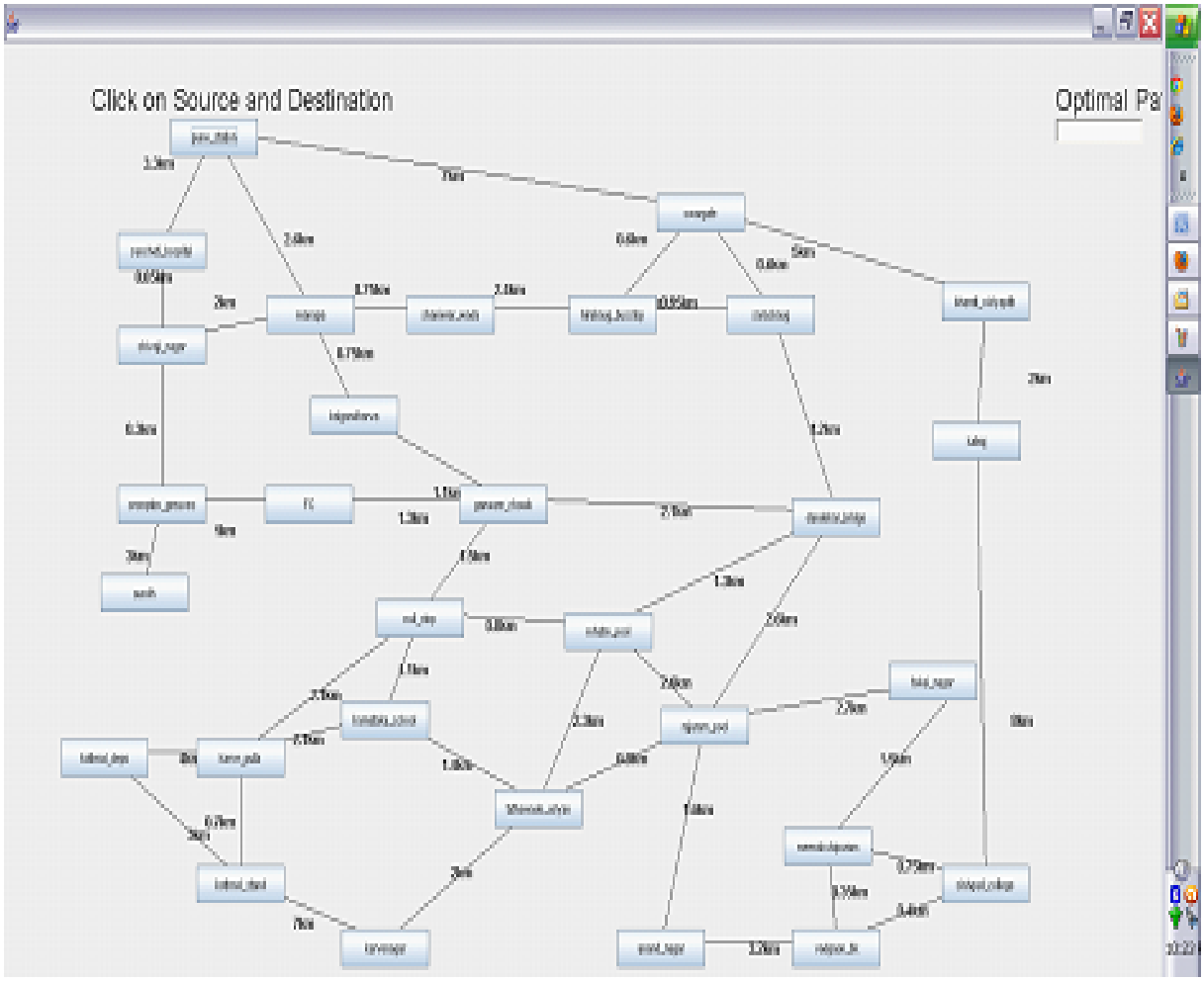
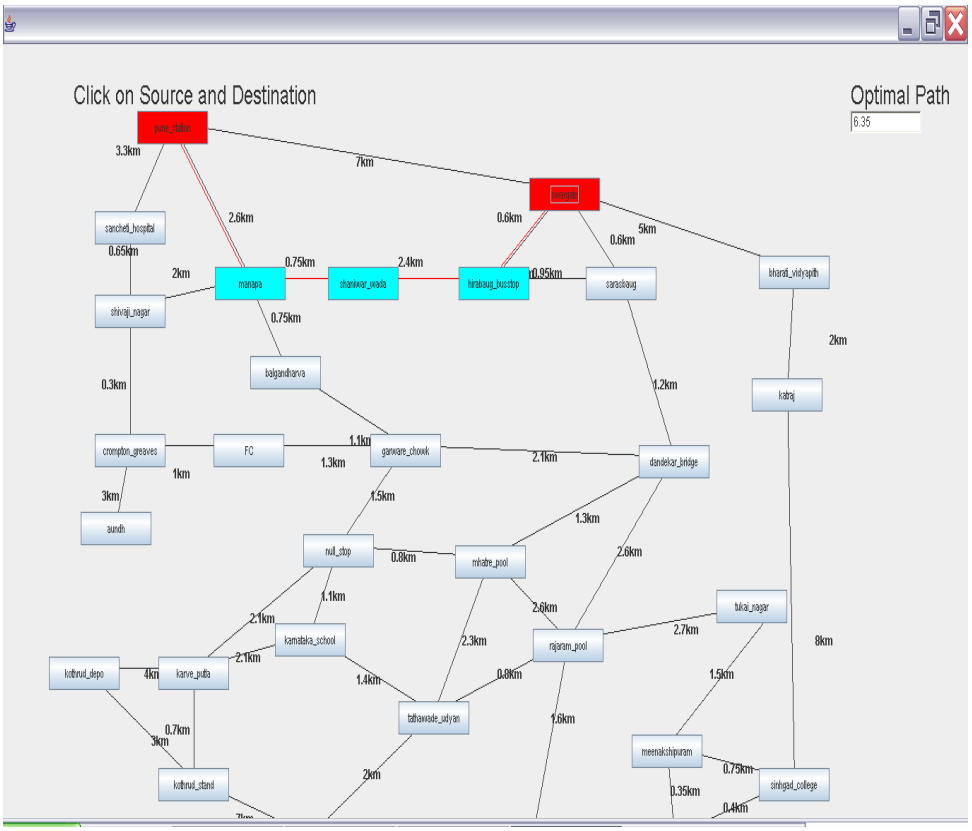



Figure 5.6: Map Form


Click on Source and Destination
Optimal Path
6.35
pune_station
swargate
sancheti_hospital
mandai
shaniwar_wada
hirabaug_busstop
sarasbaug
bharati_vidyapith
shivaji_nagar
balgandharva
katraj
crompton_greaves
FC
ganware_chowk
dandekar_bridge
aundh
mhatre_pool
null_stop
tukai_nagar
kamataka_school
rajaram_pool
kothrud_depo
karve_putla
tathawade_udyan
meenakshipuram
kothrud_stand
sinhgad_college
3.3km
7km
2.6km
0.6km
5km
0.6km
0.65km
2km
0.75km
2.4km
0.95km
0.75km
2km
0.3km
1.2km
1.1km
1.3km
2.1km
1km
3km
1.5km
1.3km
0.8km
2.6km
1.1km
2.1km
2.6km
2.7km
8km
2.3km
2.1km
4km
1.4km
0.8km
1.5km
0.7km
3km
1.6km
2km
0.75km
0.35km
0.4km

## 5.3 CODE DETAILS

```
import java.sql.*;
import java.util.*;
import java.io.*;
import javax.swing.*;

class HEPV10
{
private String superborder[], border[];
int n,p,l;

public HEPV10()
{
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:HEPV","system","system");
Statement st = con.createStatement();
ResultSet rs = null;

rs = st.executeQuery("select count(src) from source"); // src is primary key ...

if(rs.next())
n = rs.getInt("count(src)");
rs.close();

rs =st.executeQuery("select count(distinct fragid) from flat_graph");
if(rs.next())
p = rs.getInt("count(distinctfragid)");
rs.close();

rs = st.executeQuery("select count(*) from flat_graph");
if(rs.next())
l = rs.getInt("count(*)");
rs.close();

st.close();
```

```
con.close();
}
catch(Exception e)
{
e.printStackTrace();
}
superborder = new String[15];
border = new String[10];
}

public static void main(String[] args)
{
JFrame frame = new Map1("Map");
frame.setVisible(true);
frame.setExtendedState(frame.getExtendedState()|JFrame.MAXIMIZED_BOTH);
}

public void createHEPV()
{
Fragment frag = new Fragment();
Supergraph superg = new Supergraph();

borderSetting();

for(int k = 1; k <= p; k++) // k = fragid
{
encode(k);

for(int c = 0;border[c] != null ;c++)
border[c] = null;

bordern(border,k);

for(int i = 0;border[i]!=null;i++) // border[i] = src
{
for(int j = 0;border[j]!=null;j++) // border[j] = dest
{
if(i != j)
```

```
{
frag.set(border[i], border[j], k);
superg.set(border[i], border[j],k);

float swt, fwt;
String fhop;

swt = superg.getWt();
fwt = frag.getWt();
fhop = frag.getHop();

if(fwt < 999.00f)
{
if(swt < 999.00f)
{
if(swt > fwt)
{
superg.update(fhop, fwt);
System.out.println(" update");
}
}
else
{
superg.insert(fhop, fwt);
System.out.println(" insert");
}
}
}
}
}
}
encode(0); // encode supergraph
}

public float pathRetrieval(String src,String dest, Map1 frame)
{
float SPW = 999.00f;
int flags = 0, flagd = 0;
```

```
Fragment frag = new Fragment();
Supergraph superg = new Supergraph();

if(!src.equals(dest))
{
if(superborder[0] == null)
{
fillSuperborder();
}
for(int i = 0; superborder[i] != null; i++)
{
if(superborder[i].equals(src))
{
flags = 1;  // src is a border node (in supergraph)
}
else if(superborder[i].equals(dest))
{
flagd = 1; // dest is a border node (in supergraph)
}
}
if(flags == 1 && flagd == 1) // Case 1: Both src and dest nodes
//are border nodes (in supergraph)
{
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = null;
ResultSet rs = null;
ResultSet rs1 = null;
con = DriverManager.getConnection("jdbc:odbc:HEPV","system","system");
Statement st = con.createStatement();
rs1 = st.executeQuery("select * from super_graph
where src = '"+ src + "'
and dests = '"+ dest +"'");

if(rs1.next())
{
SPW = rs1.getFloat("wts");
```

```
String hops = rs1.getString("hops");
int fragidd = rs1.getInt("fragid");

System.out.print(src + " -> " + hops);

frame.highlight(src); // colour buttons
frame.highlight(hops);

int fragids = 0,flag = 1;
String hopd = null;

rs = st.executeQuery("select fragid from frag where src  = " + src + "
and dest = " + hops + " and wt = (select min(wt) from frag
where src = " + src + " and dest = " + hops +")");

if(rs.next())
{
fragids = rs.getInt("fragid");
bordern(border, fragids); // border[] is used for src
}
rs.close();

if(fragids == fragidd && fragids != 0) // src and dest are in same fragment
{
while(!hops.equals(dest))
{
frag.set(hops,dest,fragids);
hops = frag.getHop();
System.out.print(" -> " + hops);
frame.highlight(hops);
}
}
else // traverse more than one fragment
{
int fragid = 0,indxs = 0;
float min = 999.0f,wt;
String hop = null;
```

```
for(int i = 0;i < border.length;i++)
{
if(src.equals(border[i]) == false)
{
frag.set(hops,border[i],fragids);
wt = frag.getWt();
if(min > wt)
{
min = wt;
indxs = i;
hop = frag.getHop(); // hops -> hop -> border[i]
}
}
}
if(hop.equals(null) == false)
{
System.out.print(" -> " + hop); // hops -> hop -> border[indxs]
frame.highlight(hop);
while(hop.equals(border[indxs]) == false)
{
frag.set(hop,border[indxs],fragids);
hop = frag.getHop();
System.out.print(" -> " + hop);
frame.highlight(hop);
}
}
else
{
System.out.print(" -> " + border[indxs]);
frame.highlight(border[indxs]);
}
String[] borderd = new String[10]; // border -> borderd
bordern(borderd, fragidd); // borderd is used for dest
min = 999.0f;
int indxd = 0;
for(int i = 0;i < borderd.length;i++)
{
if(border[indxs].equals(src) == false)
```

```
{
if(border[indxs].equals(borderd[i]))
// if we get same border node in fragids and fragidd
{
indxd = i;
break;
}
else
{
rs = st.executeQuery("select fragid from super_graph
where src = " + border[indxs] + " and dests = " + borderd[i]);
if(rs.next())
fragid = rs.getInt("fragid");
frag.set(border[indxs],borderd[i],fragid);
wt = frag.getWt();
if(min > wt)
{
min = wt;
hop = frag.getHop();
indxd = i;
}
rs.close();
}
}
}
if(border[indxs].equals(borderd[indxd]) == false)
// 2 different border nodes in different fragment
{
while(!hop.equals(borderd[indxd]))
{
frag.set(hop,borderd[indxd],fragid);
hop = frag.getHop();
System.out.print(" -> " + hop);
frame.highlight(hop);
}
}
hopd = borderd[indxd];
while(!hopd.equals(dest))
```

```
{
frag.set(hopd,dest,fragidd);
hopd = frag.getHop();
System.out.print(" -> " + hopd);
frame.highlight(hopd);
}
}
}
else
SPW = 999.0f;
st.close();
con.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
else if(flags == 1 && flagd == 0) // Case 2: The src is a border node
//, but the dest is a local node (in fragment)
{
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = null;
ResultSet rs = null;
ResultSet rs1 = null;
ResultSet rs2 = null;
con = DriverManager.getConnection("jdbc:odbc:HEPV","system","system");
Statement st = con.createStatement();
rs = st.executeQuery("select * from super_graph
where src = '" + src + "'");
if(rs.next())
{
int fragids = 0,fragidd = 0;
String hop = null;

for(int j = 1;j <= p;j++)
```

```
{
rs1 = st.executeQuery("select * from frag
where fragid = " + j + " and dest = " + dest + " and bor = 0");
if(rs1.next())
{
fragidd = j;
bordern(border, fragidd);

float swt = 0.00f,fwt = 0.00f;

for(int k = 0; border[k]!= null; k++)
{
superg.set(src,border[k],0);
frag.set(border[k],dest,fragidd);

swt = superg.getWt();
fwt = frag.getWt();

if((swt + fwt) < SPW)
{
hop = border[k];
rs2 = st.executeQuery("select fragid from super_graph
where src  = '" + src + "' and dests = '" + border[k] + "'");
if(rs2.next())
{
fragids = rs2.getInt("fragid");
}
rs2.close();
SPW = swt + fwt;
}
}
System.out.print(src);
frame.highlight(src);
while(!src.equals(hop))
{
frag.set(src,hop,fragids);
src = frag.getHop();
System.out.print(" -> " + src);
```

```
frame.highlight(src);
}
while(!hop.equals(dest))
{
frag.set(hop,dest,fragidd);
hop = frag.getHop();
System.out.print(" -> " + hop);
frame.highlight(hop);
}
rs1.close();
break;
}
}
}
rs.close();
st.close();
con.close();
} //try
catch(Exception e)
{
System.out.println(e);
}
}

else // Case 3: Both the src and dest nodes are local nodes (in fragment)
{
try
{
int flag = 0;
String hops = null,hopd = null;

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = null;
ResultSet rs = null;
ResultSet rs1 = null;
con = DriverManager.getConnection("jdbc:odbc:HEPV","system","system");
Statement st = con.createStatement();
int fragids = 0,fragidd = 0;
```

```
for(int i = 1; i <= p; i++)
{
rs = st.executeQuery("select * from frag
where fragid = " + i + " and src = '" + src + "'");
if(rs.next())
{
bordern(border, i); // border[] is used for src
fragids = i;
rs.close();
break;
}
rs.close();
}
String[] borderd = new String[10];
for(int j = 1; j <= p; j++)
{
rs = st.executeQuery("select * from frag
where fragid = " + j + " and dest = '" + dest + "'");
if(rs.next())
{
bordern(borderd, j); // borderd is used for dest
fragidd = j;
rs.close();
break;
}
rs.close();
}
if(fragids!=0 && fragidd!=0)
{
float swt,fwts,fwtd;

for(int i = 0; border[i]!= null; i++)
{
for(int j = 0; borderd[j]!=null; j++)
{
frag.set(src,border[i],fragids);
fwts = frag.getWt();
```

```
superg.set(border[i],borderd[j],0);
swt = superg.getWt();
frag.set(borderd[j], dest,fragidd);
fwtd = frag.getWt();
if((swt + fwts + fwtd) < SPW)
{
hops = border[i];
hopd = borderd[j];
SPW = swt + fwts + fwtd;
flag = 1;
}
}
}
}
if(fragids == fragidd && fragids != 0)
{
float fwt;

frag.set(src,dest,fragids);
fwt = frag.getWt();
if(fwt < SPW)
{
hops = frag.getHop();
SPW = fwt;
flag = 2;
}
}
if(flag == 1)
{
int fragid = 0;

System.out.print(src);
frame.highlight(src);

while(!src.equals(hops))
{
frag.set(src,hops,fragids);
src = frag.getHop();
```

```
System.out.print(" -> " + src);
frame.highlight(src);
}

rs = st.executeQuery("select fragid from super_graph
where src = '" + hops + "' and dests = '" + hopd + "'");
if(rs.next())
{
fragid = rs.getInt("fragid");
}

while(!hops.equals(hopd))
{
frag.set(hops,hopd,fragid);
hops = frag.getHop();
System.out.print(" -> " + hops);
frame.highlight(hops);
}

while(!hopd.equals(dest))
{
frag.set(hopd,dest,fragidd);
hopd = frag.getHop();
System.out.print(" -> " + hopd);
frame.highlight(hopd);
}
}
else if(flag == 2)
{
System.out.print(src + " -> " + hops);
frame.highlight(src);
frame.highlight(hops);

while(!hops.equals(dest))
{
frag.set(hops,dest,fragids);
hops = frag.getHop();
System.out.print(" -> " + hops);
```

```
frame.highlight(hops);
}
}

rs.close();
st.close();
con.close();
} //try
catch(Exception e)
{
System.out.println(e);
}
}
}
else
System.out.println("Source and Destination are same. Thus, shortest distance = 0.0");

return SPW;
}
private int flatUpdate(String src, String dest,float linkwt)
{
int fragid = 0;

try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = null;
ResultSet rs = null;
con = DriverManager.getConnection("jdbc:odbc:HEPV","system","system");
Statement st = con.createStatement();
st.executeUpdate("Update flat_graph set wt = " + linkwt + "
where src = '" + src + "' and dest = '" + dest + "'");
rs = st.executeQuery("select fragid from flat_graph
where src = '" + src + "' and dest = '" + dest + "'");
if(rs.next())
{
fragid = rs.getInt("fragid");
}
```

```
rs.close();

st.close();
con.close();
}
catch(Exception e)
{
System.out.println(e);
}
return fragid;
}

public void HEPVUpdate()
{
int fragid[] = new int[p];
String ch, borderc[] = new String[10], bor[] = new String[10];

for(int i = 0; i < p; i++)
fragid[i] = 0;

do
{
Scanner sc = new Scanner(System.in);

System.out.println("Enter src: \t");
String src = sc.nextLine();

System.out.println("Enter dest: \t");
String dest = sc.nextLine();

System.out.println("Enter new linkwt:\t");
float linkwt = Float.parseFloat(sc.nextLine());

int id = flatUpdate(src,dest,linkwt);
fragid[--id] = 1;

System.out.println("Do you want change any links?");
ch = sc.nextLine();
```

```
}while(ch == "y");


for(int i = 0; i < p; i++)
{
if(fragid[i] == 1)
encode(i+1);
}


for(int i = 0; border[i]!= null; i++)
{
border[i] = null;
borderc[i] = null;
}

for(int i = 0; i < p; i++)
{
if(fragid[i] == 1)
{
bordern(border,i+1);

for(int j = 0; border[j] != null; j++)
{
for(int k = 0; border[k] != null; k++)
{
if(j != k)
{
Supergraph superg = new Supergraph();
superg.set(border[j], border[k], i+1);
superg.update(null,999.00f);
}
}
}
}
}
```

```
for(int k = 1; k <= p; k++)
{
for(int i = 0; border[i]!=null; i++)
{
border[i] = null;
}
bordern(border,k);

for(int l = 0; l < p; l++)
{
if(fragid[l] == 1)
{
for(int i = 0; borderc[i] != null; i++)
{
borderc[i] = null;
}
bordern(borderc,l+1);

int c = 0;

for(int u =0; border[u] != null;u++)
// create bor[] for Ni, Nj that belong to both //border[] and borderc[]
{
for(int v = 0; borderc[v] != null;v++)
{
if(border[u] == borderc[v])
{
int i;
for(i = 0; bor[i] != null; i++) // no duplicate entries
{
if(bor[i] == border[u])
break;
}
if(bor[i] == null)
bor[c++] = border[u];
}
}
}
```

```
for(int i = 0; bor[i] != null; i++)
{
for(int j = 0; bor[j] != null; j++)
{
if(bor[i] != bor[j])
{
Supergraph superg = new Supergraph();
Fragment frag = new Fragment();

superg.set(bor[i], bor[j], l+1);
frag.set(bor[i], bor[j], l+1);

float fwt = frag.getWt();
float swt = superg.getWt();

if(fwt < 999.00f && swt > fwt)
{
superg.update(frag.getHop(),fwt);
}
}
} // for j
} // for i
}
} // for l
} // for k

encode(0); // encode supergraph
}
}
```

# Chapter 6

## PROJECT TESTING

### 6.1 MANUAL TESTING

| Test Case ID | Test Case Name | Test Case Description | Steps | Expected Result | Actual Result | Test Status(P/F) | Test Priority |
|---|---|---|---|---|---|---|---|
| Login01 | Login | To verify username and password | Enter invalid username and valid password | An error message ”Enter valid username | An error massage Displayed | P | high |
| Login02 | Login1 | To verify username and password | Enter valid username and valid password | Next Form shoule display | Next form displayed | P | high |
| Encode01 | Encode | To check whether fragment is encoded | Click on Encode button | A message ”successfully Encoded” display Fragment gets encoded | Successfully Encoded | P | high |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Encode02 | Encode1 | To check whether super-graph is created | Click on Encode button | A message ”successfully Encoded” display. super-graph is created | Successfully Encoded. Super-graph created | P | high |
| Update01 | Update | To check whether database upadated | click on update button | A message ”Successfully Updated” should display. Database updated | A message not displayed. Database not updated. | F | medium |
| PathDisplay01 | PathDisplay | To verify proper path display | Select source and destination | The path between source and destination get hidglighted | Path is highlighted | P | high |
| Distance01 | Distance | To verify shortest distance display | select source and destination | The shortest distance displayed | The shortest distance not displayed | F | high |

# Chapter 7

## CONCLUSION

Implemented hierarchical graph model supports efficient optimal path retrieval. By extending the encoded path view to the hierarchical graph model, we achieve an excellent compromise between the complete computations of paths on-demand versus precomputing all paths. Path retrieval is still significantly faster than A*. In addition, the memory requirement of HEPV is also much less compared to FEPV.

**The contributions of our project can be summarized as follows:**

1. Implemented hierarchical graph model (HEPV) exploits path materialization strategies.
2. These algorithms are used to create and maintain HEPV.
3. Implemented a shortest path retrieval algorithm which can be shown to be optimal.

# Chapter 8

## REFERENCES

[1] Ning Jing, Yun-Wu Huang, Elke Rundensteiner ”Hierarchical Optimization of Optimal Path Finding for Transportation Applications,”
[2] R. Agrawal, S. Dar and H. V. Jagadish, Direct Transitive Closure Algorithms: Design and Performance Evaluation, ACM TODS, Vol. 15, No. 3, Sep. 1990, pp. 427 458.
[3] R. Agrawal and H. V. Jagadish, Hybrid Transitive Closure Algorithms, Proc. of the 16th VLDB Conf., 1990, pp. 326 334.
[4] T. Cormen, C. Leiserson, and R. L. Rivest, Introduction to Algorithms, The MIT Press, 1993.
[5] M. A.W. Hustma, F. Cacace, and S. Ceri, Parallel Hierarchical Evaluation of Transitive Closure Queries, Proc. of the 1st Int. Conf. on Parallel and Distributed Information Systems, 1990, pp. 130 137.