Institution för datavetenskap

Department of Computer and Information Science

Final Thesis

Using Case-Based Reasoning

For

Intelligent Time of Arrival Estimation

By

Irfan Nazir

LIU-IDA/LITH-EX-A—12/006—SE

2012-02-03

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköping university
Department of Computer and Information Science

Final Thesis

Using Case-Based Reasoning For Intelligent Time of Arrival Estimation

By

Irfan Nazir

LIU-IDA/LITH-EX-A—12/006—SE

2012-02-03

Supervisor

Prof. René Meier

School of Computer Science and Statistics
Trinity College Dublin, Dublin 2, Ireland

Examiner

Prof. Petru Eles

Dept. of Computer and Information Science

Linköpings University, Linköping, Sweden

Dedication

To my loving parents, brother and sisters for their endless prayers and support which helps me to get success in life.

Abstract

Traffic congestion and over saturation is a common worldwide problem. This problem is more severe in urban areas due to rapid increase in number of vehicles. City traffic often results in traffic blockage at peak office hours. The same situation can also be observed on freeways, especially at entrance/exit areas of a city. In most cases the drivers are totally unaware about road traffic situations in advance.

Intelligent transportation systems offer various applications to improve road traffic. In this manner the timely and reliable traffic information can be greatly beneficial for travelers. Real time travel time information can be adopted by vehicles as an excellent tool which can result to reduce the impact of continuous traffic increase on urban roads.

Real time travel time estimation is a complex task as it involves various factors. This travel time estimation is even more difficult for urban road networks. In our dissertation, we have investigated an estimated time of arrival approach in order to inform the drivers in advance about travel time. We have introduced Estimated Time of Arrival (ETA) in a clever and intelligent fashion by developing a Case-Based Reasoning engine. We believe that by adopting this simple and intelligent approach, it can greatly help improving traffic congestion overall. We also believe that this strategy will also help in reducing the emission of environment unfriendly gases, thus helping mankind.

Acknowledgement

I would like to express my in-depth gratitude to my supervisor Prof. Rene Meier for giving me such a nice opportunity to work under his kind supervision in the Trinity College Dublin, Ireland. In fact it is his ideas, guidance and prompt support throughout my dissertation work which helped me in understanding the scope and finally in achieving the desired goal.

I am also thankful to my examiner Prof. Petru Eles at Linkoping University, Sweden.

And lastly, thanks to all of my friends at Linkoping University and at Trinity College Dublin who always tried to keep me happy time to time with their nice company and attitudes.

Irfan Nazir

Table of Contents

1. Introduction *1*

1.1. Motivation 1

1.2. Problem Statement 2

1.3. Objective 3

1.4. Methodology 3

1.5. Solution 3

1.6. Thesis Organization 4

2. Background and Related Work *6*

2.1. Introduction 6

2.2. Intelligent Transport Systems 6

2.3. Intelligent Traffic Management 8

2.4. Estimated Time of Arrival 9

2.5. Case-Based Reasoning 19

2.5.1. Overview 19

2.5.2. Earlier work and Historical Background 20

2.5.3. Case-Based Reasoning using Cases 21

2.5.4. Case Representation in Case-Based Reasoning 21

2.5.4.1. Traditional Approaches to Case Representation 22

2.5.4.1.1. Features-Value Representation 22

2.5.4.1.2. Object Oriented Representation 22

2.5.4.1.3. Textual Representation 22

2.5.4.2. Advanced approaches to case representation 22

2.5.4.2.1. Hierarchical Case Representation 23

2.5.4.2.2. Generalized Case Representation 23

2.5.4.2.3. Case Representation for Specific Applications 23

2.5.5. Case-Based Reasoning Cycle 23

2.5.5.1. Retrieve 24

2.5.5.2. Reuse 25

2.5.5.3. Revise 25

2.5.5.4. Retain 25

3. A Case-Based Approach to Time of Arrival Estimation ***28***

3.1. Design Principle 28

3.2. Case Library 29

3.2.1. Route Cases 30

3.2.2. Segment Cases 32

4. CBR Engine ***38***

4.1. CBR Routing 38

4.2. CBR Segmentation 39

4.3. Use Cases 39

4.3.1. Use Case 1 39

4.3.2. Use Case 2 41

4.3.3. Use Case 3 44

4.3.4. Use Case 4 45

4.3.5. Use Case 5 47

4.3.6. Use Case 6 48

4.3.7. Use Case 7 50

5. Implementation ***53***

5.1. System Choice 53

5.2. Implementation Description 54

6. System Evaluation and Results ***58***

6.1. Evaluation Description 58

6.2. Performance Metrics 60

6.2.1. Latency 60

6.2.2. Cases Similarity 62

7. Conclusion and Future Work ***68***

7.1. Conclusion 68

7.2. Future Work 69

8. Appendices ***75***

8.1. Appendix A: Data Base Organization of CBR Engine 75

8.2. Appendix B: Source Code of CBR Engine 76

8.2.1. ETA.java 76

8.2.2. SegmentDescription.java _____ 81

8.2.3. SegmentSolution.java_____ 83

8.2.4. Thresholdvalue.java _____ 84

8.2.5. ClockInterval.java _____ 85

8.2.6. Databaseconfig.xml _____ 86

8.2.7. Hibernate.cfg.xml _____ 86

8.2.8. SegmentDescription.hbm.xml _____ 87

8.2.9. SegmentSolution.hbm.xml_____ 88

List of Figures

FIGURE 1-1: JOURNEY DISTRIBUTION USING MODE OF TRAVEL [20] 2
FIGURE 1-2: JOURNEY DISTRIBUTION INDICATING JOURNEY PURPOSE [20] 2
FIGURE 2-1: 1ST AND 2ND MOST COMMON ROUTES BETWEEN INTELLISHARE STATIONS [14] 10
FIGURE 2-2: FREQUENCY HISTOGRAM OF ROUTES BETWEEN STATION 1 AND STATION 2 OF INTELLISHARE SYSTEM [14] 11
FIGURE 2-3: DATA COLLECTION SCHEME [8] 12
FIGURE 2-4: PREDICTION AND UPDATE OF ARRIVAL TIMES AT STOPS [8] 12
FIGURE 2-5:STUDY SEGMENTS ALONG WEST MADISON STREET [21] 14
FIGURE 2-6:THE ARCHITECTURE OF PREDICTION ENGINE [10] 16
FIGURE 2-7:TIME OF ARRIVAL ESTIMATION CALCULATION [10] 17
FIGURE 2-8: CASE-BASED REASONING CYCLE [2] 24
FIGURE 2-9: CASE-BASED REASONING CYCLE, NEW SOLUTION BY IAN WATSON [24] 26
FIGURE 3-1: DEPARTURE AND DESTINATION LOCATIONS, ROUTES AND ROUTE SEGMENTS 31
FIGURE 4-1: USE CASE 1. VEHICLE WANTS TO GET AN ESTIMATED TIME OF ARRIVAL 41
FIGURE 4-2: USE CASE 2. ADDING A NEW ROUTE TO CBR WHEN NO DESTINATION INFORMATION IS SPECIFIED 42
FIGURE 4-3: USE CASE 2. ADDING A NEW ROUTE TO CBR WHEN DESTINATION INFORMATION SPECIFIED 43
FIGURE 4-4: USE CASE 3. EMPTY CBR 45
FIGURE 4-5: USE CASE 4. PARTIAL CASE MATCHING OF ROUTE 46
FIGURE 4-6: USE CASE 5. PARTIAL ROUTE SEGMENT'S CASE MATCHING 48
FIGURE 4-7: USE CASE 6. SPLIT CBR 49
FIGURE 4-8: USE CASE 7. UPDATE CBR 51
FIGURE 6-1: CASE BASE LATENCY EVALUATION OF ROUTE SEGMENTS S1-S7 61
FIGURE 6-2: CASE BASE LATENCY EVALUATION OF ROUTE SEGMENTS S8-S13 61
FIGURE 6-3: CASE BASE LATENCY EVALUATION OF ROUTE SEGMENTS S14-S18 62
FIGURE 6-4:CASE SIMILARITY EVALUATION BY ASSIGNING 25% OF CASES AS QUERY 63
FIGURE 6-5: CASE SIMILARITY EVALUATION BY ASSIGNING 50% OF CASES AS QUERY 64
FIGURE 6-6: CASE SIMILARITY EVALUATION BY ASSIGNING 50% OF CASES AS QUERY AND REPEATED FOR TWO TIMES 65

FIGURE 6-7: CASE SIMILARITY EVALUATION BY ASSIGNING 75% OF CASES AS QUERY _____ 65

FIGURE 6-8: CASE SIMILARITY EVALUATION OF EACH CASE IN THE CASE BASE WITH REMAINING ALL ONES _____ 66

FIGURE 8-1: DATA BASE ORGANISATION OF CBR ENGINE TO DETERMINE ETA _____ 75

List of Tables

TABLE 2-1: HISTORICAL DATA ORGANIZATION [8] 13
TABLE 2-2: INPUTS REQUIRED FOR PREDICTION OF TRAVEL TIME [8] 13
TABLE 2-3: SAMPLE SIZE CALCULATION [21] 15
TABLE 2-4: SCHEDULE ASSOCIATED WITH ROUTE SEGMENTS AND TRAVEL NODES [10] 17
TABLE 3-1: ROUTE CASE BASE 30
TABLE 3-2: JOURNEY FROM LOCATION L1 TO LOCATION L6 31
TABLE 3-3: SEGMENT CASE STRUCTURE 32
TABLE 3-4: LOCATION NAMES WITH THEIR GPS COORDINATES 34
TABLE 3-5: ROUTE SEGMENTS WITH THEIR GPS COORDINATES AND DISTANCE 35
TABLE 6-1: SEGMENT CASE BASE FOR ROUTE SEGMENT S5 59
TABLE 6-2: NUMBER OF CASE AND TIME REQUIRED FOR CASE BASES OF ROUTE SEGMENTS S1-S7 60
TABLE 6-3: NUMBER OF CASES AND TIME REQUIRED FOR CASE BASES OF ROUTE SEGMENTS S8-S13 61
TABLE 6-4: NUMBER OF CASES AND TIME REQUIRED FOR CASE BASES OF ROUTE SEGMENTS S14-S18 62

Nomenclature

ITS	Intelligent Transport system
ITM	Intelligent Traffic Management
ETA	Estimated Time of Arrival
CBR	Case-Based Reasoning
4 R's	Retrieve, Reuse, Revise, Retain
AVL	Advance Vehicle Location
GPS	Global Positioning System
GIS	Geographic Information System
Algo	Algorithm
ATIS	Advance Traveller Information System

Chapter 1

1. Introduction

1.1. Motivation

With the increase of world population and rapid migration towards main cities, we can observe tremendous amount of road traffic on urban roads. As a result of this, increased traffic volume is affecting our life styles in many respects. We can observe traffic congestion often. An immense part of this traffic volume consists of private vehicles. Due to more traffic on road, the journey time also increases.

According to the national travel survey 2009 by the central statistics office Ireland [20], journeys in urban areas are observed at higher frequency than in rural areas. The average journey travelled in rural areas was nearly double in distance as compared to urban areas but it makes no or negligible effect in average travel time. Most of the journeys were travelled by private vehicles giving a figure of about 73% of the total travel. The journey by bus and rail service is adding only about 5% in total.

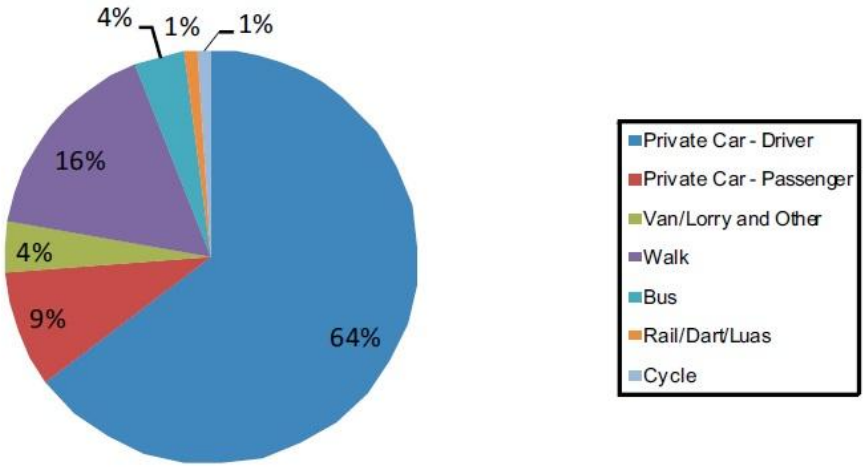

Figure 1-1: Journey Distribution using Mode of Travel [20]

It has also been observed that most of the journeys purpose is related to work, shopping and to visit family/friend making up to 65% of total journeys.

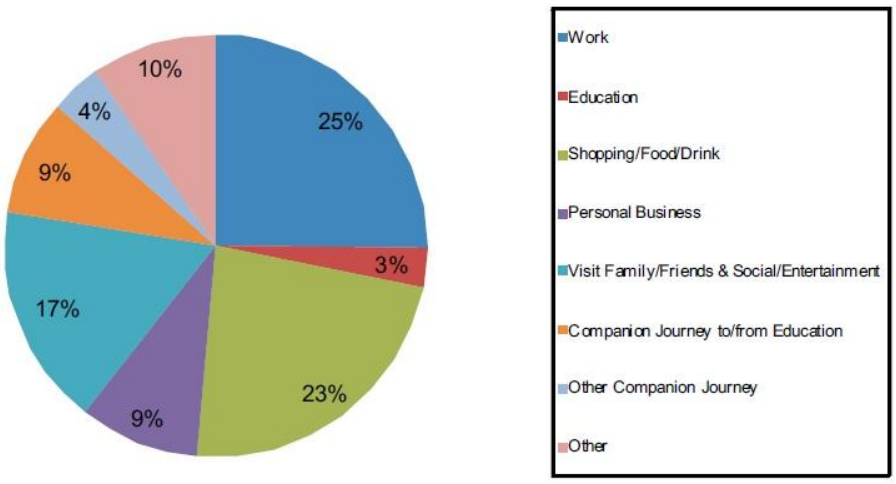

Figure 1-2: Journey Distribution indicating Journey purpose [20]

The increase in traffic volume is not only affecting humans directly, it is also leaving a long term effect. More traffic volume means more environment unfriendly gases are emitted, which finally results in raising atmospheric pollution.

One of the best ways to reduce the traffic volume can be the use of public transport. This is not only helpful in reduction of traffic congestion but also the support eco friendly environment.

1.2. Problem Statement

Today, Intelligent Transport Systems (ITS) are playing an enormous role in urban traffic management. The Advanced Traveler Information System (ATIS) is an essential

component of ITS. Various Real-time traffic applications exist and are helpful for travelers by giving a range of quality services. The Real-time travel time information system is one among such applications. Real-time travel time estimation is not an easy task to develop. It requires in-depth knowledge and up-to-date technology.

Most travel time applications available so far use various technologies to determine an Estimated Time of Arrival (ETA). Some use distance, speed, location information and other parameters but no application proves to be 100% accurate and none of them fulfils every kind of traveler. Similarly, some applications require more processing and resources so it is difficult to get in time ETA results.

1.3. Objective

The goal of our dissertation is to model the ETA approach in an intelligent fashion so that the application may work with minimum resources and processing but on the other hand gives output in an a quick and reliable manner.

1.4. Methodology

We started by a state of the art review and requirement analysis. Various research papers were studied in order to get the helpful information regarding our project scope. Major focus on the case based reasoning methodology and the use of this approach to determine the estimated time of arrival. In this way we designed the model of the ETA approach and named it CBR Engine. The last phase was to evaluate the implementation and produce the thesis report.

1.5. Solution

Our design approach is based on the case-based reasoning technique. We have designed the CBR Engine which further consists of two sub modules: the CBR Routing and the CBR Segmentation.

The CBR Routing module first initiates and determines appropriate route information the vehicle most likely will travel. Then the CBR Segmentation fetches information about route segments in the route and retrieves most appropriate route segmentation information fulfilling that particular time situation. Later the CBR Segmentation module reuses this information to produce the total time of arrival for the journey.

1.6. Thesis Organization

The dissertation is organizes as follows:

- **Chapter 1. Introduction**

This chapter provides a general introduction and motivation in understanding the project, project goal, methodology and the solution approach.

- **Chapter 2. Background and Related Work**

This chapter covers all theoretical background, current state of the art in the related areas of our dissertation like Intelligent transport System, Estimated Time of Arrival, Intelligent traffic management and Case-Based Reasoning.

- **Chapter 3. A Case-Based Reasoning Approach for Time of Arrival Estimation**

This chapter describes the approach we have used in this project. It puts lights on the design principles of case-based reasoning in detail.

- **Chapter 4. CBR Engine**

In this chapter we have discussed in detail about the reasoning engine i.e. CBR Engine and both of its submodules.

- **Chapter 5. Implementation**

This chapter illustrates the implementation of the reasoning engine and presents the framework used.

- **Chapter 6. Results and Evaluation**

This chapter evaluates the CBR engine.

- **Chapter 7. Conclusion and Future Work**

This chapter summarizes the dissertation work and discusses some possible future research.

Chapter 2

2. Background and Related Work

2.1. Introduction

This chapter focuses on the current state of the art in those areas which have direct impact on this dissertation. These areas include ITS (Intelligent Transport Systems), ITM (Intelligent Traffic Management), ETA (Estimated Time of Arrival) and CBR (Case-Based Reasoning).

2.2. Intelligent Transport Systems

The core idea behind Intelligent Transport Systems (ITS) is an imaginary world where vehicles can move freely and without any hazard, vehicles can navigate to their destination in quickest possible time by using shortest route, vehicles can use up-to-date traffic information and reports in order to identify their route information, can easily identify their parking place and, besides these, vehicles should also be environment friendly by emitting less carbon dioxide and by using less fuel. Another possible addition to ITS is automatic driving.

There are different means of transportation available. Private vehicles are the most common way which we are using as transportation medium. Hybrid and electric vehicles are an addition which are quite environment friendly. In some parts of the world, cycling as a mean of transport is gaining popularity being eco friendly and a cheap way to travel

short distances. Similarly, public transport systems are another way that gives benefits in reducing traffic congestion and has a pollution friendly nature. There is continuously ongoing research in the area of transportation systems so that it may be more and more beneficial.

Today, many different ITS applications exist. Many countries are adopting these ITS applications so the community may get maximum benefits from them. One common example of ITS, which can be seen in many cities, is the Advance Vehicle Location (AVL) system installed on public transport. This system gives benefits to both the management and passengers waiting for buses in the sense that it provides current bus information and next bus arrival time on bus stops. For real time traffic monitoring and information, a real time AVL system is required [21]. This AVL system contains a range of different communication technologies and location aware devices. The onboard computer system records data from the bus and updates the central server regularly at some certain time interval by a communication channel [21].

ITS provides many benefits. One of the main motivation of ITS is the improvement in road safety. It is fact that a great number of people die every year in road side accidents. The figures of injured people in such circumstances are even many times more. By using ITS, it can greatly help in reducing these figures.

One of the most important applications of ITS, which is widely used today, is the navigation system. Vehicles have installed GPS devices which help drivers to determine the route information with the collaboration of satellite navigation systems and maps. Similarly, many safety applications are also in use which help in avoiding collision and accidents along roadside traffic.

These are few ITS applications that are in use in many countries in some ways:

- Satellite navigation system
- Real time bus information system
- Traffic signal control system
- Traffic diversion in case of an emergency or traffic congestion
- Vehicle tracking system
- Automatic vehicle number plate recognition
- Speed monitoring by cameras
- Parking identification and information system

2.3. Intelligent Traffic Management

This section illustrates different aspects of intelligent traffic management. ITM increases the efficiency of traffic management in many different ways. It greatly helps vehicles to move and pass through the road network in an easy manner and, as a result, it reduces the unnecessary traffic congestion. This allows travelers to save their time, especially in more congested and crowded cities. ITM also gives benefits of environment friendliness as vehicles tend to emit less environment unfriendly gases [10].

Today, in ITS systems communication technologies are used and are playing an important role. The old way of communication between vehicles, especially between bus service and their office, was to call by driver's phone. In recent years, tremendous enhancement is in progress. Now, vehicles can update their information by any way using technology oriented wireless communication.

To get the ITS benefits, it needs to be managed in an efficient manner. Traffic light signals at roads play an important role in smooth traffic flow. Managing of these traffic light signals at road intersections greatly helps to reduce traffic congestion if properly managed and provides safety against traffic accidents as well. Today, in many cities intelligent traffic lights signals have been installed. Sometimes, situations occur that vehicles are waiting along one side of the road due to red light signal, where as there is no traffic on the other side. Such situations can be avoided by implementing sensors at these locations. These sensors detect vehicles and depend on cameras, induction loops or laser oriented technology and, thus, when implemented in an efficient way, reduce the long queues along roadsides. The intelligent traffic management system intends to make available a solution that may be helpful for the vehicles waiting along each side of junction, helps emergency vehicles on roads to pass through traffic and intersections, helps to assign priorities to certain vehicles like public transport etc. Another area of intelligent traffic management at intersection is to inform the junction about the number of vehicles approaching to it from each side at some certain interval of time.

To attain the above goals, the priority and arrival time of each vehicle must be communicated to the intersection within time limits. For this purpose, vehicles have to be equipped with certain GPS devices so that these can get information about their GPS coordinates. In addition, vehicles would also need to process their travel history to analyze their estimated time of arrival to such intersections by observing their travel pattern in an intelligent manner. Information about the travel history may be maintained locally or at some distributed points. Vehicles also need certain transmitters by which

GPS coordinates can be transmitted to intersection controllers and communication infrastructure.

The intersection controller plays a key role in the development of such systems. It manages all traffic lights in order to get them synchronized. Intersection controllers get information regarding alerts from the approaching vehicles, get synchronized with all approaching vehicles and adjust the signal lights according to this information.

2.4. Estimated Time of Arrival

In this section we focus on the state of the art in Estimated Time of Arrival techniques in transportation systems. Moreover, this part and the next section play a major role to achieve the goal of our project.

Today, advanced traveler information systems play an important role in our transport mechanisms. They help in relieving congestion on roads and helping an increased level of service [21]. Real time traffic information systems are installed along roadsides which continuously help travelers with respect to real time traffic monitoring, warnings, road accidents or in any emergency.

Estimated time of travel is a popular and hot area in Advanced Vehicle Location (AVL) systems and an important component of ITS, especially for public transport services. Many researchers have investigated techniques to get accurate travel time. Some researchers used speed or distance or both between origin and destination and then used these parameters to get an estimated time of travel [22] [27]. Karbassi and Barth's [14] research is very important in this aspect as they developed a UCR IntelliShare care sharing system. This research is carried out by the joint collaboration of University of California and Honda Motor industry with the focus to develop methods that can be helpful to travelers with respect to get accurate, in time and precise travel time information. They use the Global Positioning System (GPS) technology with 30 seconds time span to get the coordinates information of the vehicle. Further, they used the point-to-curve map matching methodology in order to find the closest roadway link on the map. Figure 2-1 shows the two most frequent routes. Figure 2-2 shows a frequency histogram of those routes that have been travelled by all drivers in the last 3 years between the above described stations.

After map matching, the authors present the algorithm used for real-time route prediction. This algorithm uses information like the origin and estimated destination station information and trip start time of the vehicle. The authors collected routes data for three years and maintained these data in the form of frequency histograms for all station pairs

taking part in their research. For efficient results, they created these histograms by dividing a day into three intervals; the AM travel is from 7AM to 11AM, the noon travel is from11AM to 2PM and, lastly, the PM travel is from 2PM to 6PM. As the vehicle checks out for the journey, its real-time route prediction algorithm starts. Based on the precomputed probabilities of the routes, the algorithm selects that route which has the highest probability for the period of the day. With the upcoming GPS data from real time receiver, the algorithm recalculates the most probable route by using the map matching algorithm and hierarchical data structure. In this way new routes are always calculated whenever there is a change like unexpected turn in the real time pre calculated route.

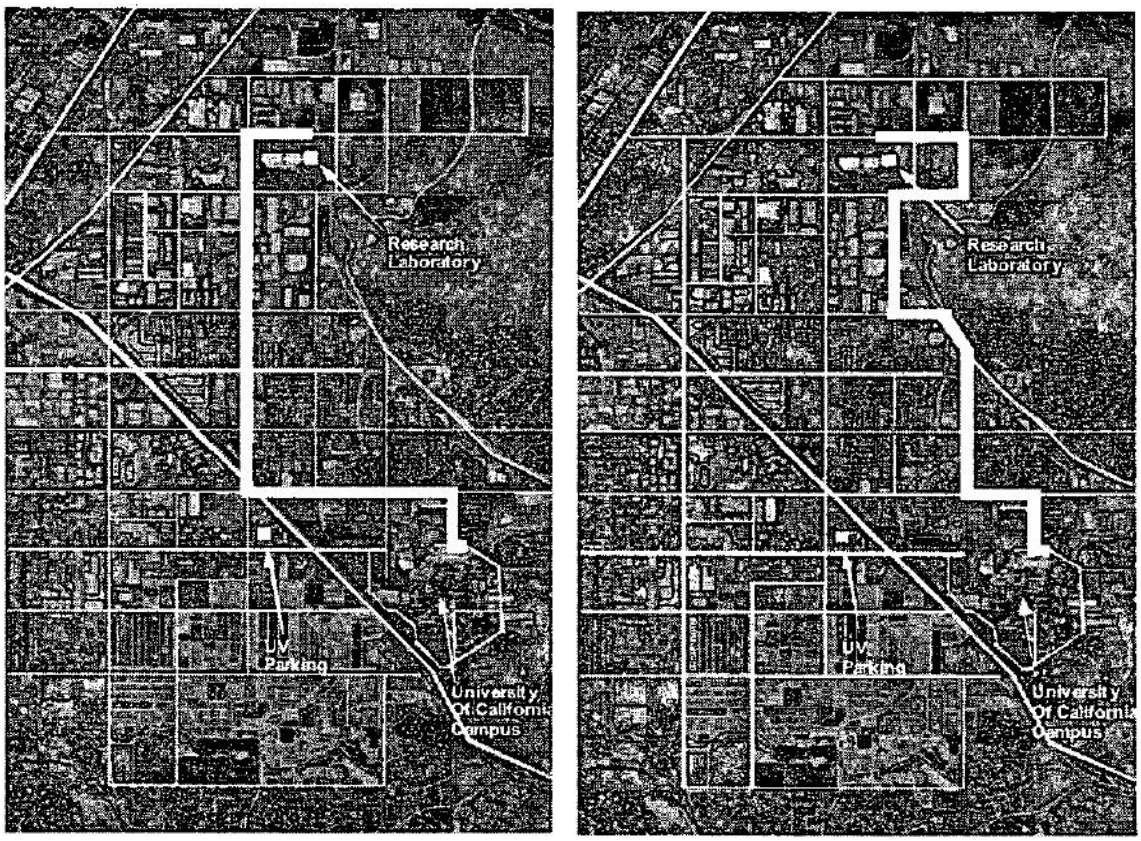

Figure 2-1: 1st and 2nd most common routes between IntelliShare stations [14]

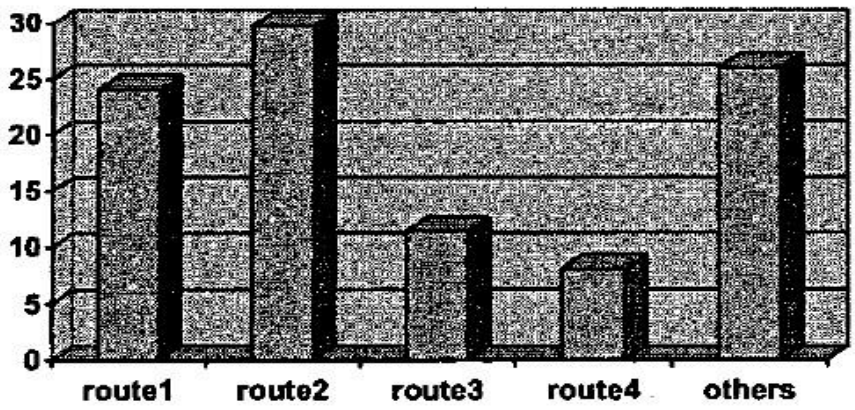

Figure 2-2: Frequency histogram of routes between station 1 and station 2 of IntelliShare system [14]

After route prediction is over, the time of arrival is calculated by using historical travel time data for this current route. This historical travel time for each route is stored in the database for different periods of the day as described previously.

In this paper, the authors introduced the concept of map matching with the real time vehicle's GPS coordinates and then estimate the travel time based on the previous route history consisting of 3 years. However, the authors did not describe how to deal in case of an emergency like traffic accident on the way or traffic congestion which requires more travel time.

Chung and Shalaby [8] also worked on the time of arrival estimation. They developed a model to determine time of arrival for school buses which uses real time global positioning data. Their main objective is to develop such a system that can predict arrival time on individual routes in all weather circumstances and road congestion.

Chung's proposed model depends on historical data. This historical data is in the form of arrival and departure times for all stops in the way to school trip. Figure 2-3 shows the data collection scheme they used. Besides historical data, Chung also collects current vehicle's trip data in the form of arrival and departure time. They keep current data in the database and transmit it to the central server. For the identification of bus stops, they adopt GIS based algorithms to match bus stops names against their GPS coordinates.

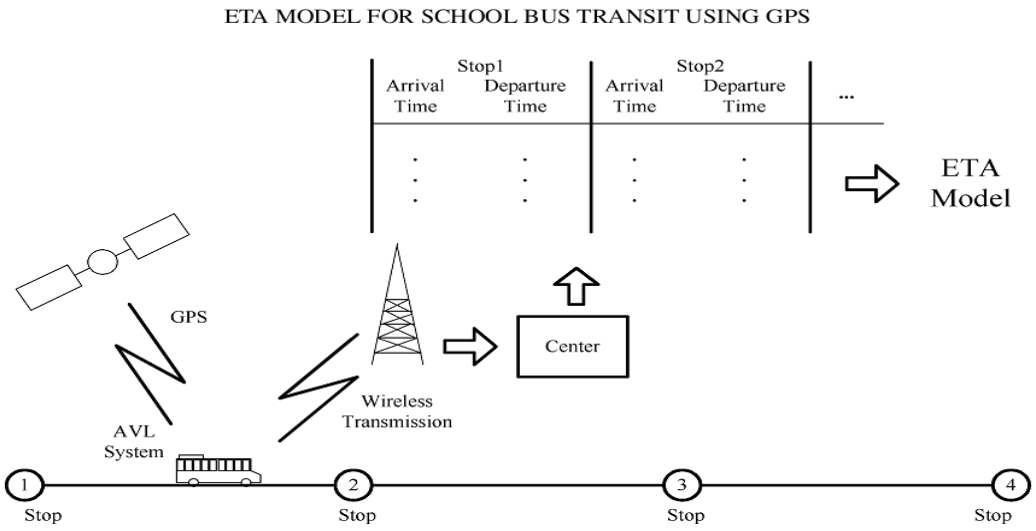

Figure 2-3: Data Collection Scheme [8]

The travel time prediction model is illustrated in Figure 2-4. It determines the arrival time at stop i, i+1, i+2 and so on. When a vehicle stops at a stop, the next bus stop's arrival time is computed by adding travel time to next stop and actual arrival time at current stop. Thus, travel time to next stop is calculated by the difference of arrival times between these two stops. Similarly for stops i+2, i+3 etc, the arrival time is calculated by adding the expected arrival time at previous stop and travel time to target stop.

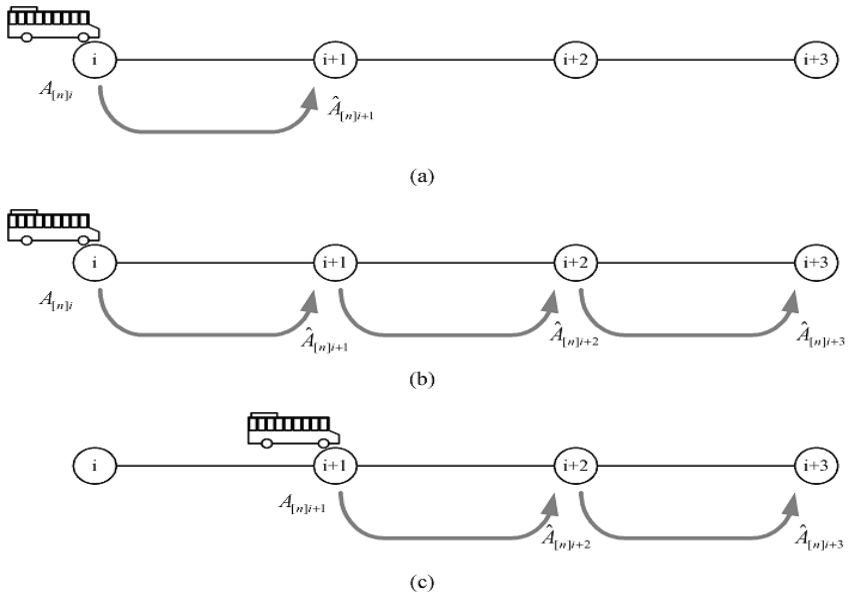

Figure 2-4: Prediction and Update of Arrival Times at Stops [8]

For travel time prediction, Chung and Shalaby used the current operational condition and the bus travel time observed in previous days, and merged both these approaches. For operational condition, they have incorporated current weather condition and bus schedule,

capturing whether the bus reached at right time or it is late for its intended arrival on a bus stop. The above approaches strongly inspired over project. So, as a first step, we have implemented an ETA model only on the basis of historical data set. As a next step, we wish to incorporate current operational conditions, among which the most important are the weather condition and traffic signal in our ETA model, as possible future work.

Table 2-1: Historical Data Organization [8]

From Stop i to Stop i+1

Day	Weather	Delay	Travel Time	Normalized Travel Time
⋮	⋮	⋮	⋮	⋮
n-4	$W_{[n-4]}$	$D_{[n-4],i}$	$T_{[n-4],i,i+1}$	$NT_{[n-4],i,i+1}$
n-3	$W_{[n-3]}$	$D_{[n-3],i}$	$T_{[n-3],i,i+1}$	$NT_{[n-3],i,i+1}$
n-2	$W_{[n-2]}$	$D_{[n-2],i}$	$T_{[n-2],i,i+1}$	$NT_{[n-2],i,i+1}$
n-1	$W_{[n-1]}$	$D_{[n-1],i}$	$T_{[n-1],i,i+1}$	$NT_{[n-1],i,i+1}$
n	$W_{[n]}$	$D_{[n],i}$	$T_{[n],i,i+1}$	$NT_{[n],i,i+1}$

Further, Chung normalized travel time for each record of historical data set; the data format is described in Table 2-1. Finally, to predict the time of arrival, the authors used weather and delay attributes as input and used these inputs to calculate the effect of current operational conditions. Similarly, normalized time of arrival is used to calculate the base time of travel. Table 2-2 describes inputs and travel time prediction.

Table 2-2: Inputs Required for Prediction of Travel Time [8]

Day	Weather	Delay	Travel Time	Normalized Travel Time
⋮	⋮	⋮	⋮	⋮
n-4	$W_{[n-4]}$	$D_{[n-4],i}$	$T_{[n-4],i,i+1}$	$NT_{[n-4],i,i+1}$
n-3	$W_{[n-3]}$	$D_{[n-3],i}$	$T_{[n-3],i,i+1}$	$NT_{[n-3],i,i+1}$
n-2	$W_{[n-2]}$	$D_{[n-2],i}$	$T_{[n-2],i,i+1}$	$NT_{[n-2],i,i+1}$
n-1	$W_{[n-1]}$	$D_{[n-1],i}$	$T_{[n-1],i,i+1}$	$NT_{[n-1],i,i+1}$
n	$W_{[n]}$	$D_{[n],i}$	$T_{[n],i,i+1}$	$NT_{[n],i,i+1}$
n+1	$W_{[n+1]}$	$D_{[n+1],i}$	$\hat{T}_{[n+1],i,i+1}$	

$f_{[HS],i,i+1}$

$f_{[OC],i,i+1}$

Wenjing Pu, Jie Lin and Liang Long [21] also performed research to estimate travel time in urban street areas. For this purpose, they used transit bus service to examine the sensitivity and relationships between buses and cars. The researchers tried to focus on bus data and travel history so that the information may be useful for other vehicles, especially the cars, in order to determine real-time travel time and speed estimation.

In the framework designed by Pu, the study segment is divided into smaller links. Then, they used historical data of both the bus and car speed relationships as both play a key role. For the estimation of real-time speed and travel time of a car, they used the mean bus speed denoted by b of the newly incoming bus. This mean bus speed b is compared with the confidence interval of bus historic mean speed b^0. No updates of b^0 is required in case the mean bus speed b lies within the confidence interval, otherwise a Bayesian update is needed. In an event if there is no any record of bus speed while determining estimation, the historic bus speed b^0 is used. Finally, latest bus average speed is applied to the historic data to estimate the car mean speed and mean travel time for that segment link, and all these link travel times are added to determine total travel time for the segment.

For this research, the authors used east bound Madison and west bound Madison streets of Chicago, Illinois, USA. Each of these streets consists of about 2.6 miles having several bus stops and signalized intersections. The street layout is shown in figure 2-5. The distance from one bus stop to the next is also shown along the travelling line of direction.

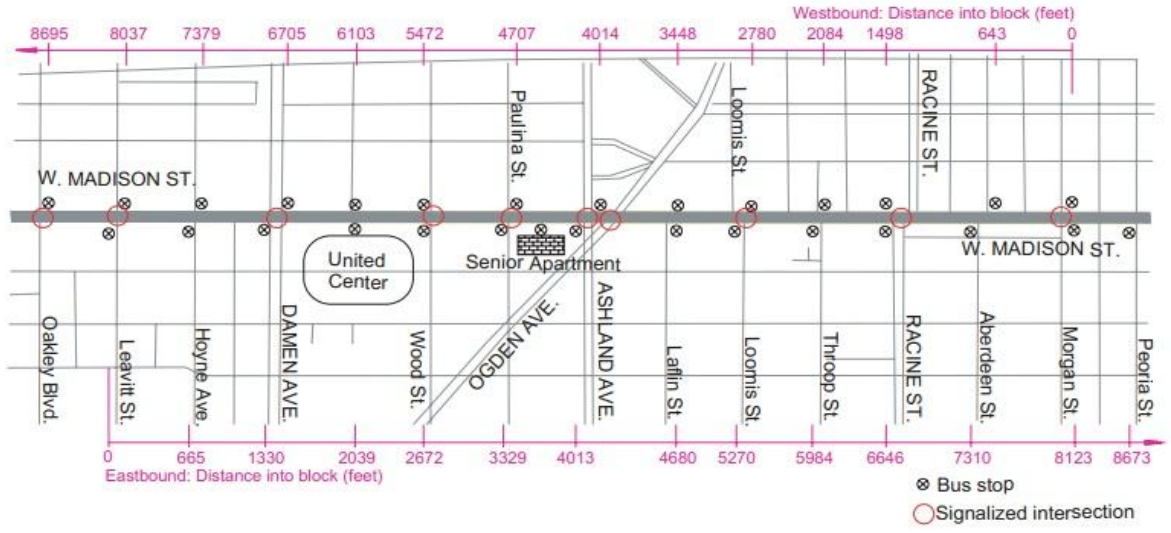

Figure 2-5:Study Segments Along West Madison Street [21]

Researchers used real-time AVL bus data which was already collected by the Chicago transport authority. To measure the traffic condition, a GPS enabled passenger car is used.

They observed and collected data for continuous 9 days in the morning and in the afternoon sessions. It was observed from dataset that at morning time both street segments showed no sign of congestion whereas in the afternoon session, traffic congestion was observed at westbound segments. For a period of 9 consecutive days, total travel time is measured along street segments at a 15 minute interval. Later, mean and standard deviation is calculated for these observations, the sample size is shown in following table 2-3.

Table 2-3: Sample Size Calculation [21]

	Eastbound			Westbound		
	Mean (s)	SD (s)	Required Sample Size (m)	Mean (s)	SD (s)	Required Sample Size (m)
10:30–10:45 a.m.	263	25	1.4	250	23	1.3
10:45–11:00 a.m.	272	29	1.8	260	26	1.5
11:00–11:15 a.m.	255	13	0.4	256	26	1.6
11:30–11:45 a.m.	254	27	1.8	270	14	0.4
5:30–5:45 p.m.	264	20	0.8	260	31	2.2
5:45–6:00 p.m.	267	23	1.1	284	55	5.8
6:00–6:15 p.m.	254	22	1.1	251	26	1.6
6:15–6:30 p.m.	256	28	1.8	272	32	2.1

Researchers observed that travel time for the street segments remains almost steady for week days in the same 15 minute time interval slot. So, due to this little variation, they assumed that only one observation is sufficient for record keeping and to use for further time estimation calculation.

The bus data for the same consecutive 9 days was also extracted from the original data set. Then, this bus data is used to build a historic relationship between general categories of vehicles and bus. Later, in order to determine the car‘s mean speed, the end result of the bus speed updating are observed and applied on bus-car speed correlation. Car’s mean travel time is calculated by segment length and car’s mean speed. Finally, this car’s mean travel time for all street segments are summed. The results indicate that, in most cases, the estimated car travel time remains quite accurate with respect to the actual observed time.

The research which is most relevant and helpful to our project is by Derek Fagan and Rene Meier [10]. In fact it is their approach which is quite influential in all respects to our project dissertation. The goal of their research is to present an ETA technique which includes certain important parameters in order to determine an accurate time of arrival. Fagan and Meier incorporated driver’s travel behavior and traffic flow patterns as they

believe that by inclusion of these parameters, vehicle's time of arrival estimation can significantly be improved.

The authors designed a system and named it as prediction engin, which needs to be installed on a vehicle as part of some location aware GPS device. The prediction engin consists of several components; these are described in figure 2-6.

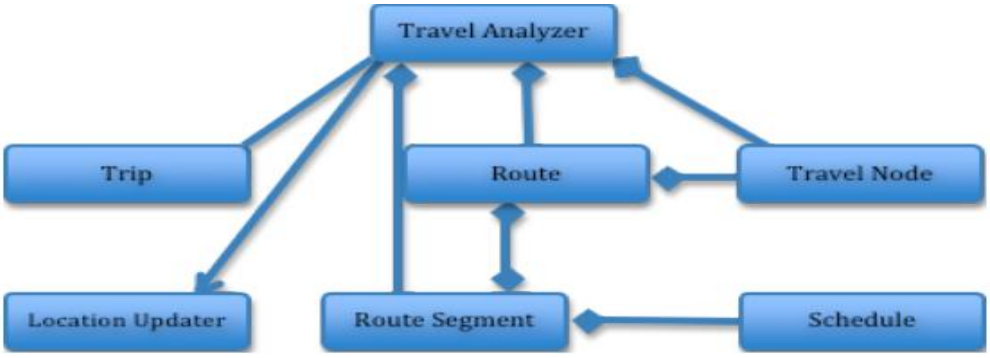

Figure 2-6:The Architecture of Prediction Engine [10]

The main component of the prediction engine is the travel analyzer; it uses all other components to determine travel time. Another major component is the route which may consist of several segments called route segments and travel nodes. The route segments are represented by continuous GPS coordinates whereas the travel nodes are those points where a vehicle can stop or where route segments are connected. The authors believe that both the route and route segment represent the driver travel behavior pattern. Another major component of the prediction engine is the schedule; the schedule is associated with each route segment and travel node.

For each route segment, a schedule is associated. This schedule represents information about travel time and time required to travel a route segment. The authors also argued that such schedules are also required for travel nodes. It is because during a journey, a vehicle may have to spend some time at any junction or stop. Table 2.4 represents a schedule pattern.

Table 2-4: Schedule associated with route segments and travel nodes [10]

Day Traveled	Time Traveled	Time Taken [min]
Sunday	8:25	16
Sunday	14:30	17
Monday	8:25	16.5
Monday	14:30	17
Tuesday	8:20	17.5
Tuesday	14:25	15.5
Wednesday	8:30	16.5
Wednesday	12:20	18.5
Thursday	8:15	18
Thursday	14:00	17.5
Friday	10:00	19
Friday	16:35	13.5

The trip component saves the unique routes data whenever a vehicle travels a new journey. Moreover, this trip component retrieves information of route segments when a vehicle travels a journey that has already been recorded. The authors used the bounding box algorithm to represent and retrieve information from route segments.

The authors also discuss how new route segments are added and how a route segment is divided into two or more segments. We have used these ideas of addition and division a route segment in our case based reasoning approach for this project. As an addition or division of a route segment happens, the schedule associated with it also needs to be updated.

Finally, the authors demonstrated the time of arrival estimation mechanism of prediction engine by stating that during a journey, the first responsibility of the prediction engine is to determine the route segments at which the vehicle is travelling. After this, it analyzes all possible routes the vehicle can travel. The time weights of the current route segment and all preceding ones are added to estimate the possible time of arrival at a destination. This mechanism of time weights addition is described in figure 2-7.

Figure 2-7:Time of Arrival Estimation Calculation [10]

This paper focused on the short term estimated time of arrival prediction in a quick manner. We have incorporated these ideas in our project and tried to enhance ETA by introducing a CBR engine which focuses on both the short term as well as the long term ETA for a journey.

2.5. Case-Based Reasoning

2.5.1. Overview

Case-Based Reasoning is a relatively simple and recent problem solving technique; it is purely based on past experiences and lies under the umbrella of Artificial Intelligence. In recent years, case-based reasoning is gaining more attention in many different commercial applications.

A case-based reasoner solves new problems by adapting solutions that were used to solve old problems [23].

In general, case-based reasoning is a methodology which is used to model the human reasoning and human thinking to produce intelligent computer systems.

The way and methodology of case-based reasoning is straightforward. Here a reasoner tries to remember and recall a previous situation which is similar or close enough to the current problem. And then this previous solution is used to solve the new problem. Case-based reasoning can be used in different ways [15].

1. Case-based reasoning can use old solutions to explain new situations.
2. Case-based reasoning can be used to get a predicted solution by using previous cases and their solutions.
3. Case-based reasoning can use old cases to critique new solutions.
4. Case-based reasoning can use different old reasonings in order to interpret a new situation.
5. Case-based reasoning can also be used to create a reasonable solution to a new problem.

In our daily life we use case-based reasoning techniques. We can notice that in medical practice, the doctor uses previous history of a patient to diagnosis any disease, in kitchen people use old cooking recipes to cook any new dish, the lawyers use case-based reasoning to interpret and relate different scenarios to make easy judgment for any judiciary case. Similarly, we use our common sense reasoning while selection of any food, restaurant, hotels, hostels, airlines etc. This all leads to case-based reasoning principles.

According to Kolodner [15], the quality of the solution derived by using case-based reasoning primarily depends on the following:

1. The experience. More experience means better understanding, so an experienced reasoner will always have edge on a less experienced reasoner. Normally, these experiences cover both successful as well as non successful events and results. If, on one side, successful events and results help deriving the solution part, the failed ones alarm and help to understanding the potential for failure.

2. Ability to understand new situations while deriving this from those of old cases. Here, the ability to find appropriate cases is the key to get success. Then, one another step is involved which leads to compare the old solutions to give the new solution. Normally, this interpretation is used whenever there is less understanding of the problem.

3. Solution's adeptness at adaptation. In order to meet the requirements, the old solutions are used in such a way that they give or derive a solution to the new problem in an efficient manner. Experts have defined many ways to adapt a solution.

4. Solution's adeptness at evaluation. The evaluation of a solution is also equally important; it helps to determine the quality of the solution. It helps greatly to minimize the mistake ratio.

2.5.2. Earlier work and Historical Background

The work on case-based reasoning started in the 80's. Probably, the first system that can be treated as case-based reasoning system was known as CYRUS developed at Yale University by Kolodner [17]. It was a questioning answering system based on the principles and work of Roger Schank on dynamic memory. The case based model described in CYRUS later becomes a model for many CBR applications.

In contrast to the USA, work on CBR started very late, in early 90's, in Europe. Initially, CBR conferences were held separately by region like ICCBR, ECCBR, EWCBR and many more. Currently, from 2010, both ICCBR and ECCBR have merged into a single most popular conference [5].

In the Handbook of Software Engineering and Knowledge Engineering, Althoff [6] tried to figure out domain areas where CBR applications can be produced. These domain areas include designing, engineering, process, quality, financial, medical science, planning, forecasting, shortly, in almost every aspect.

2.5.3. Case-Based Reasoning using Cases

A case represents a problem solution. Broadly speaking, case-based reasoning can be used by either problem solving or in an interpretive manner. The problem solving method is used mostly by those scenarios involved in AI based applications. This methodology leads to accurate or near perfect solutions. Here, old solutions guide to derive and to determine new solution for any new problem. In problem solving situations, the major focus is to use the old cases and use them efficiently to get solutions. The problem solving methodology is mostly used in designing problems, planning problems, for diagnosis and for explanation scenarios [15].

The interpretive method differs from problem solving methods as here the new problem is evaluated in the context of old problems. When there is no availability of any clear method to derive a new problem solution, then interpretive methods are used. While using the interpretive reasoning, we mainly focus on cases for justification, criticism etc. Interpretive case-based reasoning is also very useful and can be used in many areas and ways like for classification of any situation, evaluation, argumentation, interpretation, planning, justification, projection and decision of any plan. A very good example which we see in our daily life is in judiciary where both the judges as well as lawyers use interpretive reasoning by giving arguments etc. [15]

2.5.4. Case Representation in Case-Based Reasoning

According to Kolodner [16] a case is a "contextualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner"

So this experience can be structured in many ways. The simplest form by which a case can be structured is to divide t he experience into problem and solution part. The problem description part describes the state of the world, what is the problem which needs to be solved. The solution part defines the solution to the problem.

Besides the above simplest form of a case, it can be organized in some complex form i.e. an additional part named as "outcome" can also be incorporated which represents the resulting state after the case usage. Still some more complex form of the structure of a case is also possible depending on the requirement and intended use. For example, Kolodner [16] proposes a more comprehensive case representation which consists of these factors:

1. Situation and its goal

2. Solution, also sometimes the means of deriving it
3. Result of carrying it out
4. Result explanations
5. Lessons learned from the experience [3]

As a case represents a particular problem solution, researchers have proposed many ways to represent the experience. This case representation is very important in CBR as it is closely related to the case retrieval i.e. how a similarity of a case is measured. In a precise way a case representation can be classified into two main categories: traditional and advanced approaches.

2.5.4.1. Traditional Approaches to Case Representation

In traditional case representation approaches, cases are represented at a single level of abstraction. This approach can be further classifies into three categories: feature value pair, object oriented and textual representation [3]

2.5.4.1.1. Features-Value Representation

The most basic form to represent a case can be defined as a pair of attribute-value. Each case has some attributes, and each of these attribute has assigned a value. Although this type of case representation contains some shortcomings, it is probably the most easy to implement. It requires less effort to store and do manipulation on such a case feature list [3] [7].

2.5.4.1.2. Object Oriented Representation

This representation approach uses an object oriented paradigm. Cases representation is performed as a collection of objects and these objects are described by attribute-value structures. Object oriented case representation is normally used in complex situations i.e. when cases belong to different structures [3].

2.5.4.1.3. Textual Representation

In textual case representation the experiences are captured in the form of text. Textual CBR systems use this textual knowledge in order to solve and draw conclusion [26].

2.5.4.2. Advanced approaches to case representation

In advance case representation, cases are represented at different levels of abstractions. So one case can present details at multiple levels. Following are some approaches which fall under this category.

2.5.4.2.1. Hierarchical Case Representation

In a hierarchical case representation, a large and complex case is decomposed into small subcases [7] [25]. So, CBR action will have to be performed in every individual sub case and, at the end, the solutions of all subcases are combined to get the global solution.

2.5.4.2.2. Generalized Case Representation

Generalized case representation differs from the rest of case representations as this does not cover a single situation or experience. Instead, it covers and analyzes a group of similar problems which are closely related to each other. This is some sort of additional feature that a case can handle. An ordinary or regular case representation can be switched into a generalized case representation by introducing some additional variables [3].

2.5.4.2.3. Case Representation for Specific Applications

Besides regular case representation techniques, researcher sometimes uses case representation as per the demand of the tasks [3].

2.5.5. Case-Based Reasoning Cycle

Case-based reasoning is different from other Artificial Intelligence techniques in many ways. CBR uses specific knowledge from previous experiences and utilizes this for solving the problem instead of using general knowledge of the problem. Similarly, CBR is an incremental approach as it retains its experience every time it finds or calculates a solution for any new problem.

As CBR is an approach to solve new problems based on the past problem solution setup; these past problems and their solutions are named cases. So, a case is a problem situation along with its interpretation and solution part. The cases are learned, stored in case base and can be reused any time to solve similar problem scenarios in the future. This case base is a repository structure and helps to solve every new problem by adopting and manipulating some clever and intelligent approaches of known similar problems.

In general sense and at the highest level, a CBR cycle consists of the following four processes which further are illustrated in the Figure 2-8:

1. Retrieve
2. Reuse
3. Revise
4. Retain

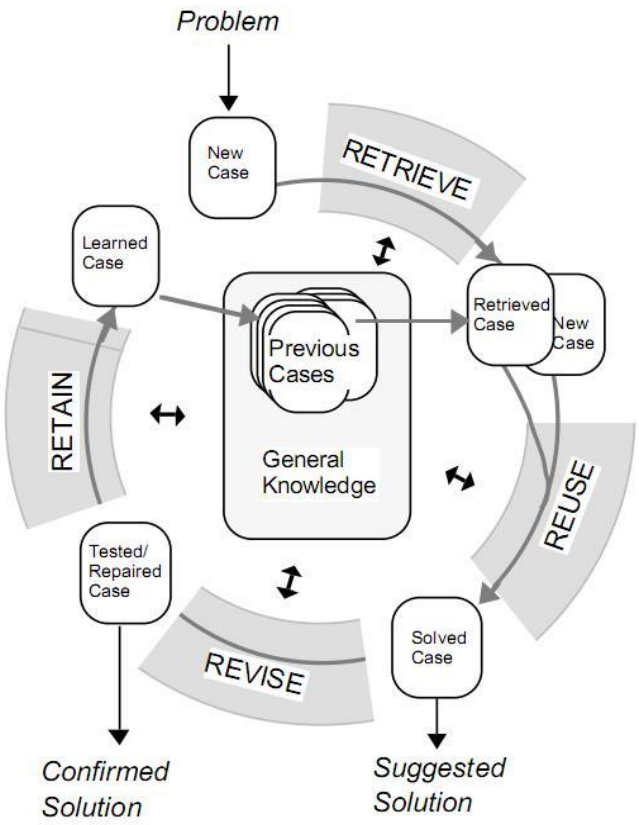

Figure 2-8: Case-Based Reasoning Cycle [2]

2.5.5.1. Retrieve

This is known as the first and most important part in the CBR cycle. It is that part where most of the research has been carried out. The purpose of the retrieval phase is to select those cases from the case base which have nearly or exactly the same solution for the current problem. This phase works on the basic principle that "similar problems have similar solutions". This phase starts with the description of the problem statement and its job is to find the best matching case from its case base [1]. Further, several subtasks are involved in this phase. These are summarized as to index the cases in the case library or case base, identify the relevant features as a case may consist of several features, searching relevant cases from the case base which may result in many cases and, finally, choosing an appropriate or best resulted case from the previous result [6].

In the retrieval phase, the main concept is to compare the incoming problem with every case in the case base and then select the most similar case to the incoming one. So, similarity is one of the most important phenomenons here. In most of the applications, the similarity test is performed by both ways i.e. by local similarity and global similarity. Moreover, the similarity assessment can be performed by assigning weights to problem attributes, by exact match, by considering the goal etc.

2.5.5.2. Reuse

The second phase of the CBR cycle proposes a solution for the incoming problem from the retrieved solutions. Reuse of a solution can be as simple as to use the exact solution of some previous experience or more difficult if there is no exact solution and it is needed to adapt the solution in some efficient way. As a case represents and covers a situation, in simple classification tasks the exact solution of the retrieved case is used. This is done by using the most appropriate case solution. In complex tasks and where a significant difference between the new and stored cases exists, we perform some adaptation of saved solutions in some better way, to solve the new problem [1].

The new problem can be solved in two different ways: either by using a past case solution (it is an easy method) or by using the method of constructing the solution. The first type requires domain knowledge and transformational rules which helps to derive an efficient solution. For the second type we observe how the related problems are solved. This strategy gets information about the methods to solve the related problems and then these methods are used in some new context to derive the solution for the new problem [1].

For some applications it is necessary to define a strategy in order to adopt a solution for the new problem as it is unlikely and difficult to store all cases. Common applications that fall in this category are related to designing, configuration and planning [18].

2.5.5.3. Revise

The third phase of the CBR cycle further can be categorized into two tasks: first, evaluate the solution and second, repair the fault. So it is some kind of opportunity of learning from the failure cases. In the first task, the solution which is adapted in the previous step is evaluated i.e. by reuse in some real environment like evaluation by teacher, domain expert or by performing in real world. Evaluation time may vary due to different application domains and complexity. In the second task, we detect errors in the current derived solution and then repair these. These errors can be repaired by some repair module either by the system or by the user, depending on the application [1].

As the revision phase also deals with faulty results, so to make things better, the revision phase further utilizes explanations from failure results and modifies these results. Finally, it also makes sure that in future such a failure does not happen [1].

2.5.5.4. Retain

The last part of the CBR cycle involves retaining the experience of the new problem in the database of CBR. It depends from application to application whether every new problem and its solution should be retained or not. Further, the retain phase involves

many subtasks in order to efficiently store this experience of the current problem for future use. These involves selecting the appropriate information to store, integrating the case with the existing case base and memory structure, updating the knowledge base and making a suitable indexing [1].

During problem solving, it may happen that the system cannot find a solution or fails in determining a solution. In such a case, the reason for failure needs to be identified and captured in the case base. This helps in further avoiding such failures.

After retaining a new case, this is immediately available in the case base and ready to solve new similar incoming problems.

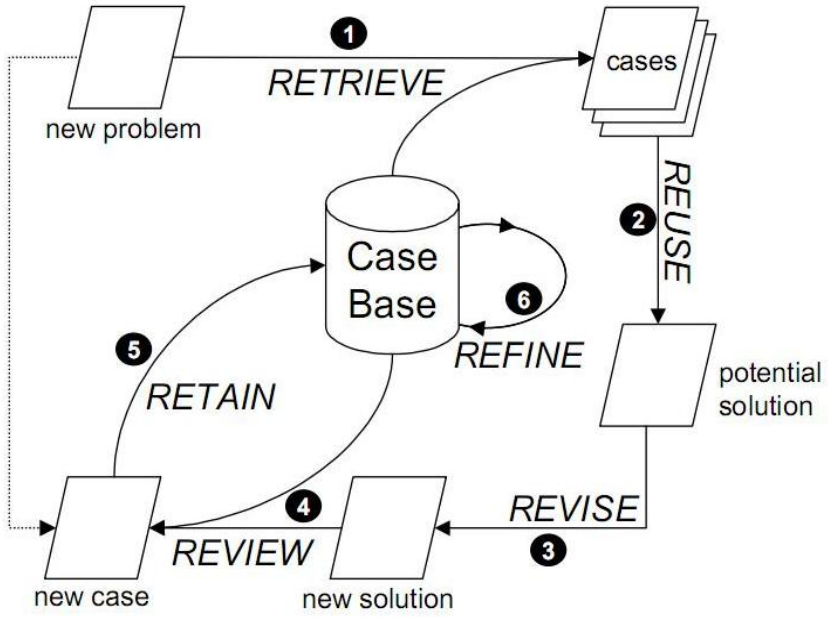

Figure 2-9: Case-Based Reasoning Cycle, New Solution by Ian Watson [24]

The CBR phases are also described in new CBR cycle diagram by Ian Watson which comprises of six stages (Figure 2-9). New suggested stages here are review and refine. In the review phase, the new problem solution is analyzed. This is done by comparing the new case with all those cases that are already in the database. If it is found that the new case is useful it is retained, otherwise it is discarded. In the refine phase, the case base index and case feature weights are refined as much as possible [24].

Chapter 3

3. A Case-Based Approach to Time of Arrival Estimation

This chapter describes the approach which we have adopted in our project for estimating the time of arrival. A case-based reasoning approach is used. The main focus of this chapter is the use of CBR for our project of estimation time of arrival.

Most of the systems described in literature focus on the end to end route prediction. They give information related to the final destination of a vehicle's current trip. Similarly, some systems aim to determine the short term route information. These systems try to figure out the next few segments or roads on some certain route. Both systems have their advantages and shortcomings. We tried to give attention to both these issues. On one hand, we are concentrating on long term route prediction based on the previous travel information stored in the case base, whereas, on the other hand, we estimate the time of travel for individual segments in that current route, as well, in parallel manner.

3.1. Design Principle

First we start by giving a brief introduction and introduced important terminology used in this chapter. A vehicle travels on some intended route and this route consists of route segments. A route starts at an origin and ends at a particular destination. Accordingly, we have used departure and destination for a route's starting and ending location. These departure and destination information are stored in the case base in the form of GPS latitude and longitude coordinates. Then, we have used the term route segment, which is a

subset of route. A route may have one or many route segments. To express a route segment, we have used the idea from Fagan and Meier [10]. Hence, we represented each route segment by the highest and lowest longitude and latitude coordinates information. This gives us the boundary of a particular route segment. While travelling on a route, the vehicle's current GPS coordinates are taken at each certain interval of time. These coordinates are checked against the route segments which are stored in the case base in order to find which route segment the vehicle is following.

This strategy to represent a route segment has several advantages with respect to our approach to determine the time of arrival. It confines route segments from growing rapidly which is a key factor in case based reasoning. Further, it also allows to minimize computations to determine a route segment.

For this project we assume that a GPS tracking java enabled device should be installed on every vehicle using this system. We are not going in detail about the device specification as it is beyond the scope of this project. We argue that this GPS device will start its working as the vehicle starts the travel and will continuously update the vehicle's location information.

As the vehicle will start its journey for the very first time, we assume that its database will be empty. With every ongoing journey, location information about departure and destination stations, routes and route segments information will be measured, calculated, and finally incorporated in the case base. When the vehicle will travel on this same route in future, it will retrieve information from the case base to conclude about time of travel for the journey.

3.2. Case Library

In most of the CBR applications the size of the case base grows with the increase in time while the structure of the case base should remain uniform. We have given important consideration to the size of the case base. By keeping in mind this complexity, we have divided the case structure into two types. So the case library of this project consists of two types of cases:

1. Route cases
2. Segment cases

Route cases are used to predict a vehicle's route and thus route cases are used at start. Route cases give their output to the segment cases in the form of a list structure of

segments that are part of the route. Segment cases determine the estimated time of travel for individual segments.

3.2.1. Route Cases

Route cases describe the situation when we wish to determine the route segments along a route before or during a vehicle's journey. Each Route case comprises a problem and its solution. When a vehicle's engine starts, its GPS device will be activated. The GPS coordinates about this journey's starting location will be taken as Route_Start_Latitude and Route_Start_Longitude. Further we assume here that the traveler will give information regarding his/her intended destination. This destination information will be evaluated (as we discuss in the next chapter) and translated into Route_End_Latitude and Route_End_Longitude coordinates. This whole information is given a name and added as another attribute in the Route case. We have also incorporated a Route_Count attribute in the Route case about which we believe, will help us to identify that route which has been travelled maximum times between a particular source and destination location. And, finally, to uniquely identify a case, we have added a Route_ID attribute.

The solution part of the Route case consists of a Route_Segments_List which is in the form of a linked list data structure containing all route segments taking part in a particular Route case. The Route case structure is described in table 3-1:

Table 3-1: Route case base

Route_ID
Route_Name
Route_Start_Latitude
Route_Start_Longitude
Route_End_Latitude
Route_End_Longitude
Route_Count
Route_Segments_List

The table fully describes all attributes of a Route case. Multiple routes may exist between one departure and destination points. This is described in figure 3-1:

Figure 3-1: Departure and destination locations, Routes and route segments

In the figure, location informations (either departure or destination) are represented by red marks and names are described in black text. The blue lines represent route segments while their names are described in dark red text.

If we take an example of a journey from L1 to L6, there are many possibilities to reach at destination L6 from location L1. These possible routes with their route segments are described in table 3-2:

Table 3-2: Journey from Location L1 to Location L6

Route	Route Segments
R1	S1, S2, S3, S4, S5
R2	S1, S6, S7, S8, S5
R3	S1, S2, S9, S8, S5
R4	S1, S6, S10, S11, S12
R5	S1, S6, S7, S13, S12

It is the solution part of the Route case for any selected route which gives intimation about route segments in details. Further whenever a route has been travelled, its Route_Count attribute will be incremented. This will help in determining how many times this route has already been travelled if the vehicle will travel the same journey in future.

3.2.2. Segment Cases

Similar to the Route case, the Segment case also comprises of a problem and its solution part. Every Segment case is uniquely identified by a Segment_ID attribute. We are using the Segment_Highest_Latitude, and Segment_Highest_Longitude, Segment_Lowest_Latitude and Segment_Lowest_Longitude attributes to identify the boundary of a route segment. Every route segment's distance is also incorporated as part of the attributes in the segment case structure. Further, two important attributes Segment_Day and Segment_Time_Travelled represents the day and time of travel when the segment has been travelled in the past. Another attribute, Segment_Count, indicates how many times this route segment has been travelled.

Table 3-3: Segment case structure

Segment_ID
Segment_Highest_Latitude
Segment_Highest_Longitude
Segment_Lowest_Latitude
Segment_Lowest_Longitude
Segment_Distance
Segment_Day
Segment_Time_Travelled
Segment_Count
Segment_Estimated_Time_of_Travel

The solution part of a Segment case consists of the Segment_Estimated_Time_of_Travel attribute, which indicates the time in the form of minutes and seconds a vehicle needs to travel a route segment. The segment case structure is illustrated in detail in table 3-3:

We are using Segment_Distance in order to measure the distance of the route segment. This is the distance between two GPS locations. To determine the distance between any two locations, we are using the following formula [4].

ACOS(COS(RADIANS(90-Lat1)) *COS(RADIANS(90-Lat2)) +SIN(RADIANS(90-Lat1)) *SIN(RADIANS(90-Lat2)) *COS(RADIANS(Long1-Long2))) *6371

For example, in figure 3.1, a route segment S1 starts from location L1 and ends at location L2. The location coordinates of L1 are following:

Latitude: 53.341992

Longitude: -6.250578

And the location coordinates of location L2 are:

Latitude: 53.344125

Longitude: -6.249297

By using these values in the above equation, we get the distance of route segment S1 0.251964 km. Moreover, the location coordinates and distance of route segments in figure 3.1 are calculated and described in table 3.4 and table 3.5.

This segment's distance value of the Segment_Distance attribute will be helpful for our CBR approach in many ways. Two reasons are following:

1. Segment_Distance will help to split route segments and adjust their estimated time of travel accordingly.
2. When a vehicle is travelling a journey which has not already been incorporated in the case base, this feature may help to define the boundary of a route segment in order to add it in the case base.

Table 3-4: Location names with their GPS coordinates

Location ID	Location Name	Latitude (Degree, Minute, Second)	Longitude (Degree, Minute, Second)	Latitude (Decimal)	Longitude (Decimal)
1	L1	53 20 31.17 N	6 15 02.08 W	53.341992	-6.250578
2	L2	53 20 38.85 N	6 14 57.47 W	53.344125	-6.249297
3	L3	53 20 41.01 N	6 15 08.27 W	53.344725	-6.252297
4	L4	53 20 42.94 N	6 15 18.46 W	53.345261	-6.255128
5	L5	53 20 45.25 N	6 15 18.37 W	53.345903	-6.255103
6	L6	53 20 50.91 N	6 15 18.09 W	53.347475	-6.255025
7	L7	53 20 44.33 N	6 15 25.09 W	53.345647	-6.256969
8	L8	53 20 40.09 N	6 15 33.77 W	53.344469	-6.259381
9	L9	53 20 35.86 N	6 15 33.39 W	53.343294	-6.259275
10	L10	53 20 43.66 N	6 14 54.53 W	53.345461	-6.248481
11	L11	53 20 48.57 N	6 14 53.22 W	53.346825	-6.248117
12	L12	53 20 50.85 N	6 15 06.60 W	53.347458	-6.251833
13	L13	53 20 44.88 N	6 15 07.73 W	53.3458	-6.252124
14	L14	53 20 35.54 N	6 14 40.49 W	53.343217	-6.244581
15	L15	53 20 49.01 N	6 15 32.26 W	53.346947	-6.258961
16	L16	53 20 50.33 N	6 15 26.62 W	53.347314	-6.257394
17	L17	53 20 59.58 N	6 15 08.45 W	53.349883	-6.252347
18	L18	53 20 59.21 N	6 15 36.83 W	53.349781	-6.260231
19	L19	53 21 04.51 N	6 14 58.85 W	53.351253	-6.249681
20	L20	53 20 21.61 N	6 15 40.16 W	53.339336	-6.261156

Table 3-5: Route segments with their GPS coordinates and distance

Segment ID	Segment Name	Segment Start Latitude	Segment Start Longitude	Segment End Latitude	Segment End Longitude	Segment Distance
1	S1 (L1 - L2)	53.341992	-6.250578	53.344125	-6.249297	0.251964
2	S2 (L2 - L3)	53.344125	-6.249297	53.344725	-6.252297	0.210029
3	S3 (L3 - L4)	53.344725	-6.252297	53.345261	-6.255128	0.197154
4	S4 (L4 - L5)	53.345261	-6.255128	53.345903	-6.255103	0.071406
5	S5 (L5 - L6)	53.345903	-6.255103	53.347475	-6.255025	0.174875
6	S6 (L2 - L10)	53.344125	-6.249297	53.345461	-6.248481	0.158124
7	S7 (L10 - L13)	53.345461	-6.248481	53.3458	-6.252124	0.244749
8	S8 (L13 - L5)	53.3458	-6.252124	53.345903	-6.255103	0.198082
9	S9 (L3 - L13)	53.344725	-6.252297	53.3458	-6.252124	0.120085
10	S10 (L10 - L11)	53.345461	-6.248481	53.346825	-6.248117	0.153583
11	S11 (L11 - L12)	53.346825	-6.248117	53.347458	-6.251833	0.256512
12	S12 (L12 - L6)	53.347458	-6.251833	53.347475	-6.255025	0.21189
13	S13 (L13 - L12)	53.3458	-6.252124	53.347458	-6.251833	0.18537
14	S14 (L4 - L7)	53.345261	-6.255128	53.345647	-6.256969	0.129528
15	S15 (L7 - L8)	53.345647	-6.256969	53.344469	-6.259381	0.206869
16	S16 (L8 - L9)	53.344469	-6.259381	53.343294	-6.259275	0.130843
17	S17 (L14 - L2)	53.343217	-6.244581	53.344125	-6.249297	0.328949
18	S18 (L16 - L7)	53.347314	-6.257394	53.345647	-6.256969	0.187497
19	S19 (L15 - L7)	53.346947	-6.258961	53.345647	-6.256969	0.19591
20	S20 (L8 - L15)	53.344469	-6.259381	53.346947	-6.258961	0.276948
21	S21 (L1 - L9)	53.341992	-6.250578	53.343294	-6.259275	0.595238
22	S22 (L9 - L20)	53.343294	-6.259275	53.339336	-6.261156	0.457483
23	S23 (L15 - 18)	53.346947	-6.258961	53.349781	-6.260231	0.326207

When we observe traffic on an urban road network in our daily life, we come to know that its intensity is different at different intervals of day time. Normally, there is more traffic at morning. Similarly, at evening time, we can observe some rush hours again. Road traffic almost remains quiet at night hours.

To make time estimation more intelligent, we gave attention to the route segment case structure. We have incorporated the above mentioned phenomenon in the Segment_Day and Segment_Time_Travelled attributes. When a vehicle travels a certain journey, its estimated travel time will be calculated on the basis of that week day and that clock time.

By considering the volume of cases in the case base, a major challenge to the performance of CBR applications, we have defined certain levels against each Segment case attribute as:

Segment_ID, Segment_Highest_Latitude, Segment_Highest_Longitude, Segment_Lowest_Latitude, Segment_Lowest_Longitude, Segment_Distance and Segment_Count all these attributes have single possible value representation.

Segment_Day: It has been observed that there exists a change in traffic volume during different week days. However, we can imagine no big difference at weekends i.e. on Saturday and Sunday. Similarly traffic scenario almost remains constant during mid week days i.e. Tuesday to Thursday whereas we observe more congestion on Monday and Friday. By keeping these factors we considered 4 levels for the Segment_Day attribute. These are:

1. Level 1: Monday
2. Level 2: Tuesday-Thursday
3. Level 3: Friday
4. Level 4: Saturday-Sunday

Segment_Time_Travelled: We observe a great decrease in traffic volume at night time whereas traffic volume is continuously changing at different hours during day interval. So we considered a smooth flow of traffic pattern from 9 pm to 7 am. For the rest, we considered a 30 minutes interval. By this grouping, 29 traffic pattern intervals are defined (since we considered 1 interval for 30 minutes from 7 am to 9 pm, so it makes 28 intervals. One more interval is added for night traffic pattern) for a day.

So the total number of possible cases for each route segment which cover the traffic pattern in all respect is 116 (29 time interval x 4 day level). Hence, for each route segment, 116 possible cases needed to be stored in the case base in order to get more efficient and intelligent time estimation.

Chapter 4

4. CBR Engine

In this chapter we describe the design of the reasoning engine which is a major part of this project. Throughout this project we are concentrating on vehicle's routes and route segments. Thus, we divided the main CBR engine into two submodules:

1. CBR Routing
2. CBR Segmentation

Both modules have their own specific features and functionalities.

The proposed reasoning engine i.e. CBR Engine is to use reasoning techniques i.e. all 4 R's (Retrieve, Revise, Reuse and Retain) and all these R's perform their functions in precise manner.

4.1. CBR Routing

CBR Routing deals with the Route case base. It is responsible to take input from both the GPS device installed on the vehicle and from the vehicle's driver. Based on the information provided to CBR Routing, it retrieves the most suitable route to the intended destination. This route will also contain a list of route segments. Further detail of CBR Routing is described in use cases.

4.2. CBR Segmentation

CBR Segmentation deals with individual route segments. CBR Segmentation takes its input from CBR Routing in the form of route segments list. CBR Segmentation is responsible to take action on these route segments one by one. It retrieves these segments from the segment case base, takes estimated time information and display output. Besides appropriate segment retrieval, it also performs other reasoning functions on route segments.

4.3. Use Cases

By keeping in mind the scope of our project, we have exemplified the functionality of CBR Engine with both its submodules in the form of use case diagrams. These possible use cases are described in detail here:

- 1. Calculate Estimated Time of Arrival.
- 2. Add a New Route to the CBR.
- 3. Empty CBR.
- 4. Partial Route's Case Matching.
- 5. Partial Route Segment's Case Matching.
- 6. Splitting a Route Segment.
- 7. Updating a CBR after Route Segment's division.
- 8. Updating a CBR after updating existing data.

4.3.1. Use Case 1

Calculate Estimated Time of Arrival

Description:

Driver and Vehicle wants to get an Estimated Time of Arrival.

Before the start of a journey, the driver wishes to know about the total estimated time of journey. We assume that the GPS device is synchronized with the vehicle's engine, so as the vehicle's engine turns ON, the GPS device will also turn ON and starts receiving current GPS coordinates. This will be one input in the form of departure location coordinates to the CBR engine's routing part.

In this use case, we assume that the second input to the CBR engine in the form of destination address will be given manually by the vehicle's driver. This destination address may be in some string format. The CBR Routing module first converts this

destination address to the destination coordinates which are already saved against each visited destination.

Once the input information is completed, the CBR Routing starts the search from the Route Case Base to find the best route to the destination. As several routes to a destination may exist, the CBR Routing selects the most visited route i.e. that route whose Route_Count value will be maximum. In this way we argue that the system follows the driver's travel behavior pattern.

After selecting the route case, the CBR Routing takes the route segments list from this case which is basically the solution part of this route case. This route segments list is passed to the CBR Segmentation module. The CBR Segmentation takes route segments IDs one by one from this list information and searches the most appropriate route segment's case from its case base which fully qualifies according to current day and clock time. After selecting the top matching route segment case, the CBR Segmentation takes its solution part which is the estimated travel time of this route segment. This process is repeated for all route segments list delivered by CBR Routing. Finally, the CBR Segmentation module adds up these estimated time of travel used for every route segment and displays the total time of arrival for the whole journey in advance. Figure 4.1 gives a pictorial representation of the above procedure.

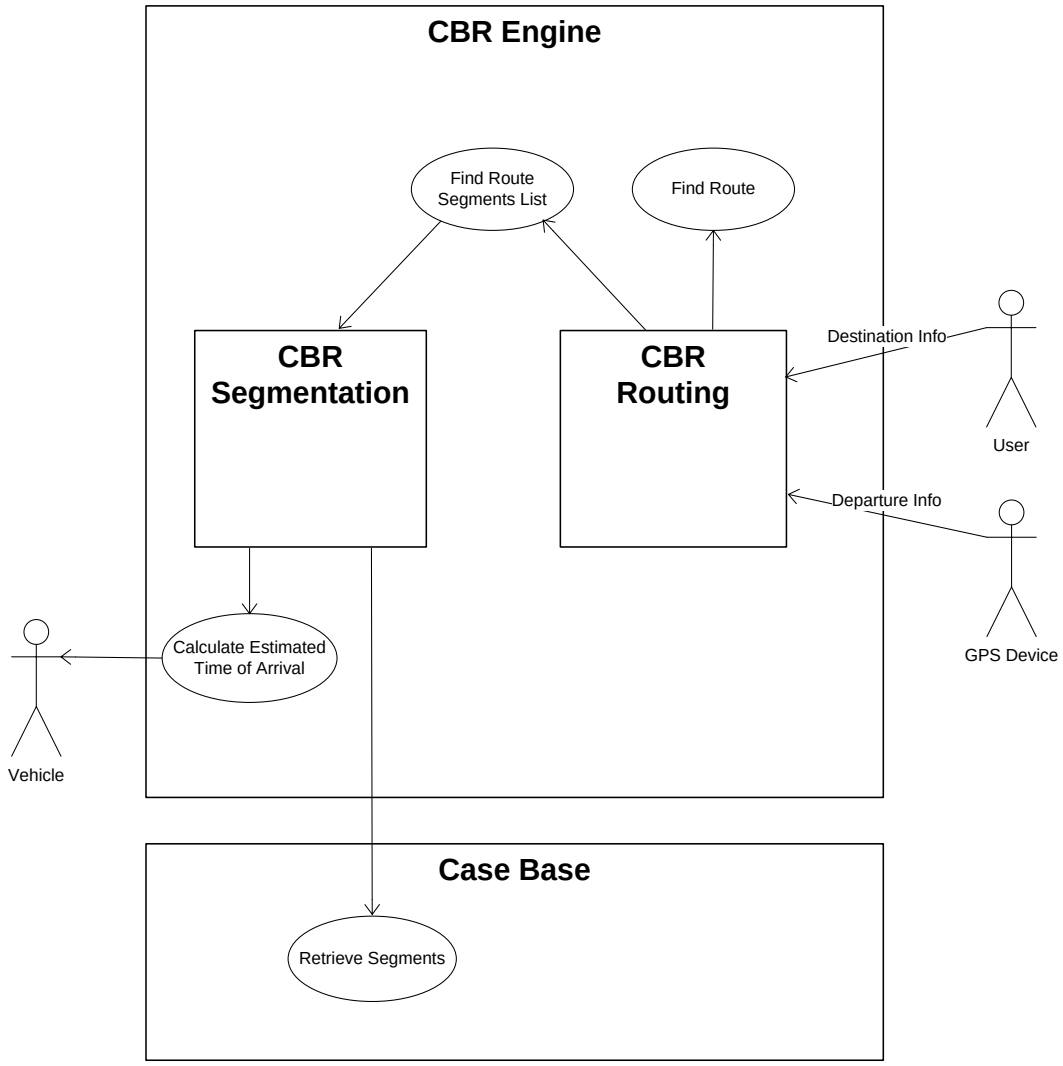

Figure 4-1: Use Case 1. Vehicle wants to get an estimated time of arrival

4.3.2. Use Case 2

Add a new Route/Route Segment to the CBR

To add a new route/route segment into the CBR case base, we examined two possibilities. These are described here under Case 1 and Case 2.

Case 1:

Description:

When a user or vehicle's driver doesn't enter any destination information, it is assumed that this journey is going to take place the very first time. The destination address has not been already saved in the database. This journey should be saved as a new route and then needs to be added into the CBR Route.

The use case diagram in figure 4.2 illustrates the above case. As the journey starts, the vehicle's GPS device receives departure coordinates information. Moreover, the GPS device will be programmed to receive current location coordinates and update these to the CBR Segmentation module after every 10 seconds interval. CBR Segmentation will check these coordinates against its route segment case base to determine whether some already travelled route segment exists or it is a new one.

Here we assume that no previous history of route segment in the case base exists. Now the CBR Segmentation will keep an eye on the vehicle's current location coordinates and continuously calculate the distance of route segment. When it reaches the defined maximum limit of segment length, it will save this as a new route segment in the segment case base. This process continues till the end of the journey. When the vehicle stops, these coordinates are saved as destination and a name is given to identify it. All new route segment's ID are passed to CBR Routing which will create a new route for this journey and add these route segments list in it.

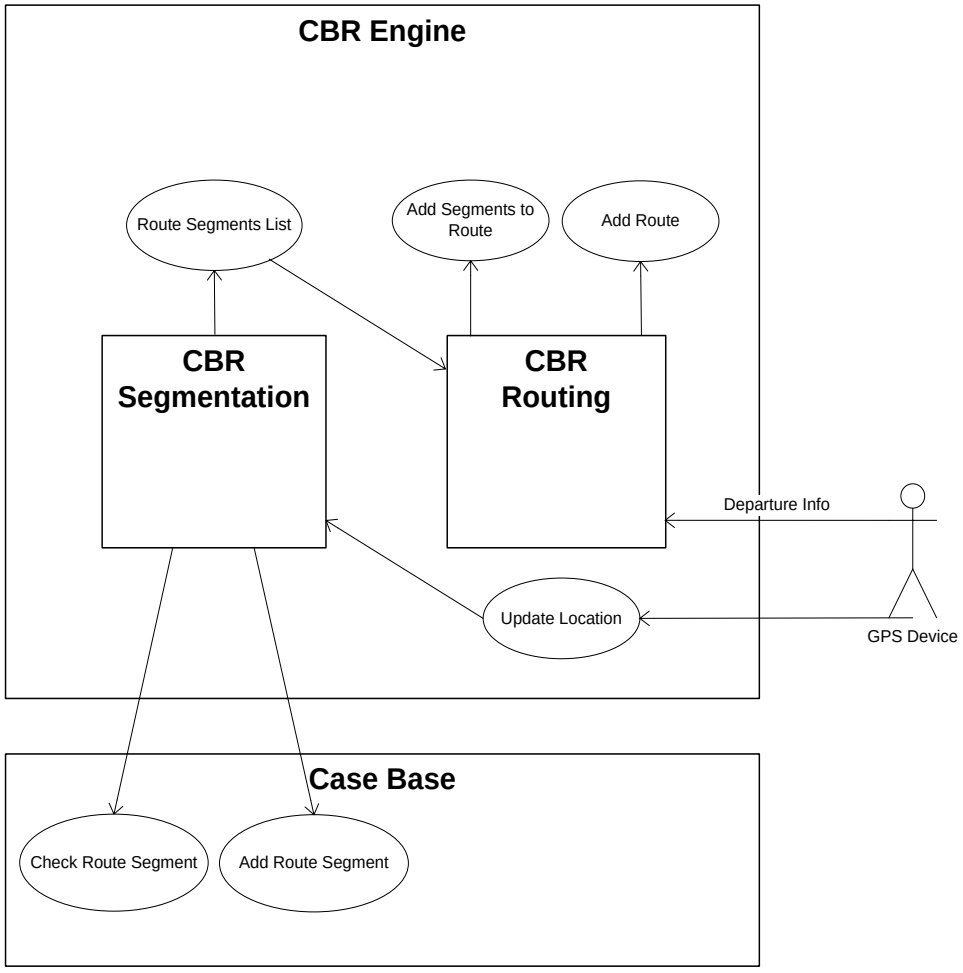

Figure 4-2: Use Case 2. Adding a new route to CBR when no destination information is specified

Case 2:

Description:

In this case we imagine that a vehicle's driver enters a destination information. The driver can do this by either selecting a predefined location or by writing a destination location name. The GPS device receives the vehicle's departure location coordinates. The CBR Engine's Routing module will search an estimated route from the route case base. Again, we assume here that the destination information does not exist in the CBR Engine. This indicates that the route for this intended journey does not exist in the CBR Routing module of the CBR Engine. Hence, after the journey finishes, the route should be added to the Route case base.

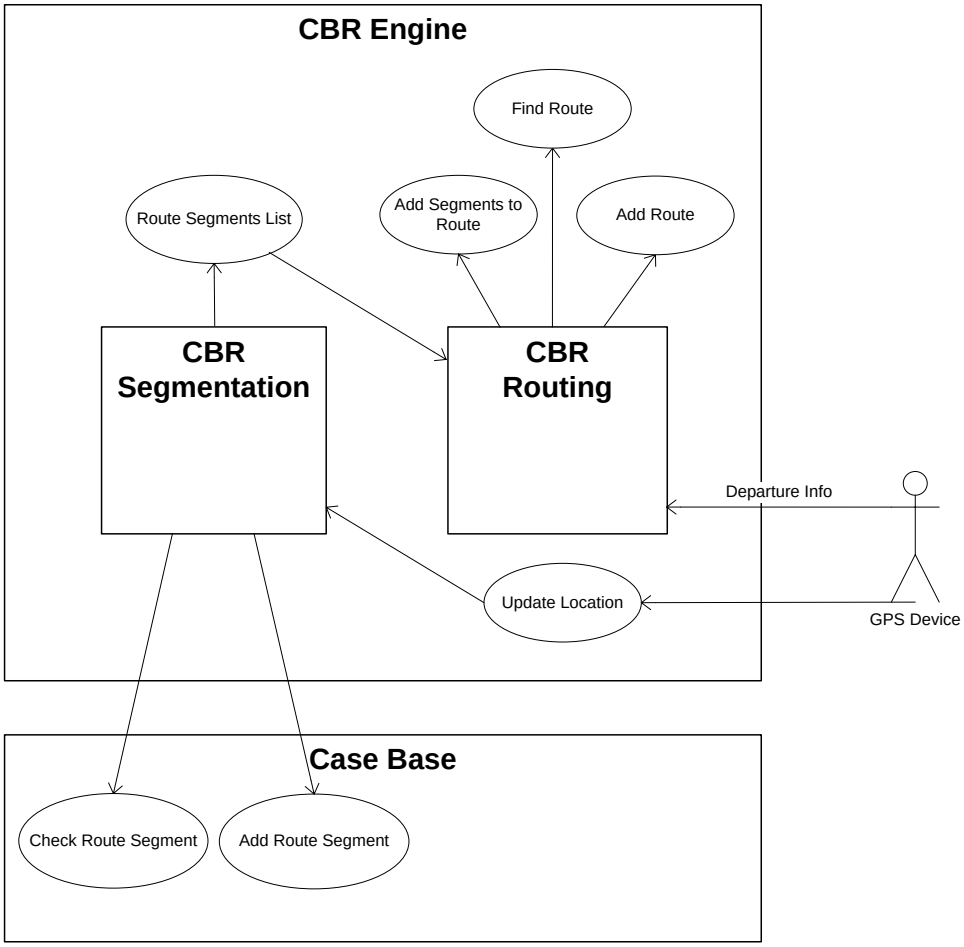

Figure 4-3: Use Case 2. Adding a new route to CBR when destination information specified

Similar to the previous case, the GPS device keeps the vehicle's current location coordinates updated after certain intervals of time to the CBR segmentation. CBR segmentation confirms whether the current route segment already exists in the segment case base or not. If it already exists, then the CBR segmentation module keeps in its

record. If the current route segment does not exist, then the CBR Segmentation adds this segment as a new one in the similar way as we have discussed in the previous case. This process of route segments identification and addition continues till the journey ends, at which the final destination's given name (as described by the driver at the start of journey) is saved according to the current location coordinates. Moreover, the route segments ID list is passed to the CBR Routing module so that it may add this new route in the route case base.

4.3.3. Use Case 3

Empty CBR

Description:

At the very first time, after integrating the CBR engine with the GPS device on a vehicle, the system is empty. Our proposed CBR engine works for every kind of vehicle, according to its own behavior. The CBR engine starts learning routes and route segments depending on the vehicle's nature, driver's behavior and traffic flow conditions. The intention behind this is to estimate travel time in an accurate fashion.

As neither any route case nor route segment case does exist in their case bases at the very first travel, any journey performed at this time will be directly added in the form of their respective cases in the CBR Engine. Information about destination and departure locations, route, and route segments and about their attributes is incorporated in real-time manner. The flow of events is similar as discussed in the use case 2. Figure 4.4 describe this use case.

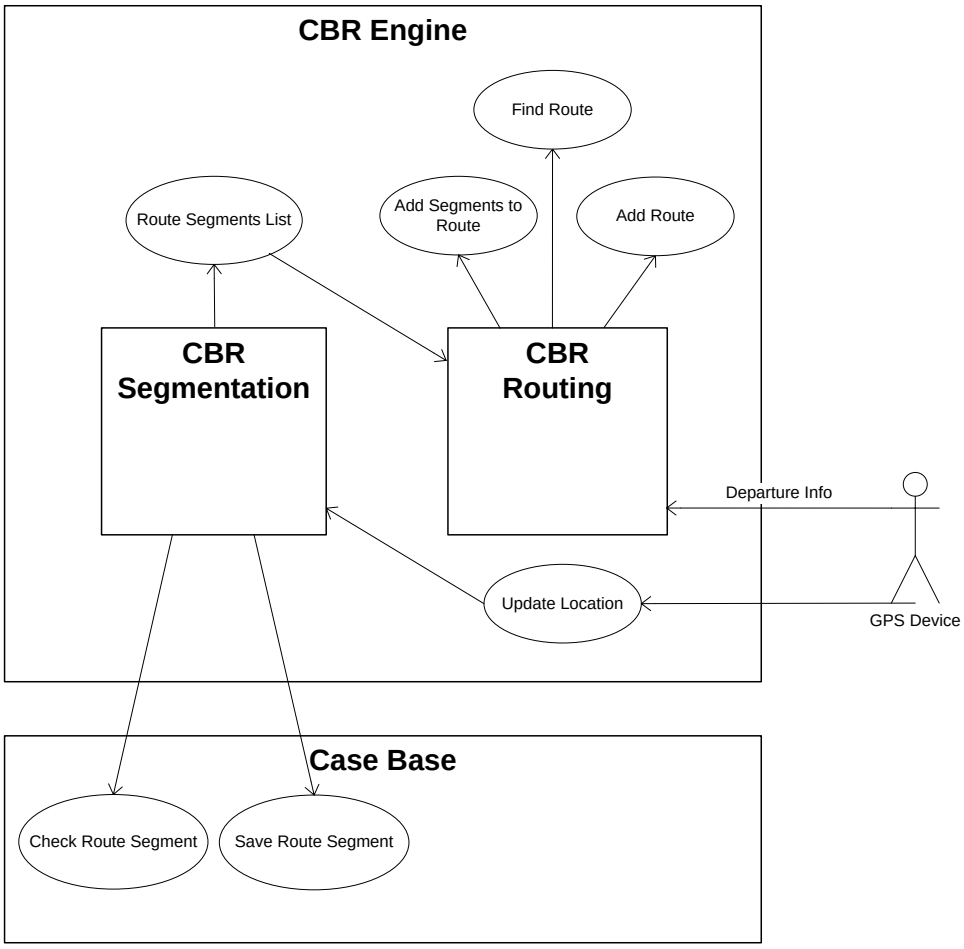

Figure 4-4: Use Case 3. Empty CBR

4.3.4. Use Case 4

Partial Route's Case Matching

Description:

In this use case, our intension is to find the nearest route to the specified destination address in case the CBR Engine does not find an exact route. After taking input from both the vehicle's driver and from the GPS device in the form of destination and departure location coordinates, the CBR Engine's Routing module searches the route case from its case base. When it does not find an exact route, it searches the given destination's GPS coordinates with all routes in its case base. The intention to do this is to find that route from the case base whose destination coordinates is nearest to the current journey. One important thing about this use case is that the user should give information about the destination in the form of GPS coordinates.

After finding the nearest route to the destination, the CBR Routing module passes its list of route segments to the CBR Segmentation. CBR Segmentation works in parallel on two

tasks. On one hand it searches the best case against each route segment, while on the other hand it calculates the distance between the specified destination location and the retrieved route's destination. This is done by comparing location coordinates. Although this distance measurement is in straight line form, it is enough helpful in determining an estimated time for the remaining distance.

The estimated time for this remaining distance is calculated in the following way: First the CBR Segmentation module gets the total estimated time and distance for all individual route segments one by one of the nearest retrieved route to the specified destination. Then it calculates an average time for certain distance. CBR Segmentation maps the average distance time ratio to the remaining distance and calculates the approximate travel time for the remaining distance.

Finally, CBR segmentation adds both estimated times i.e. the partial predicted route and the estimated remaining route and displays as an output information.

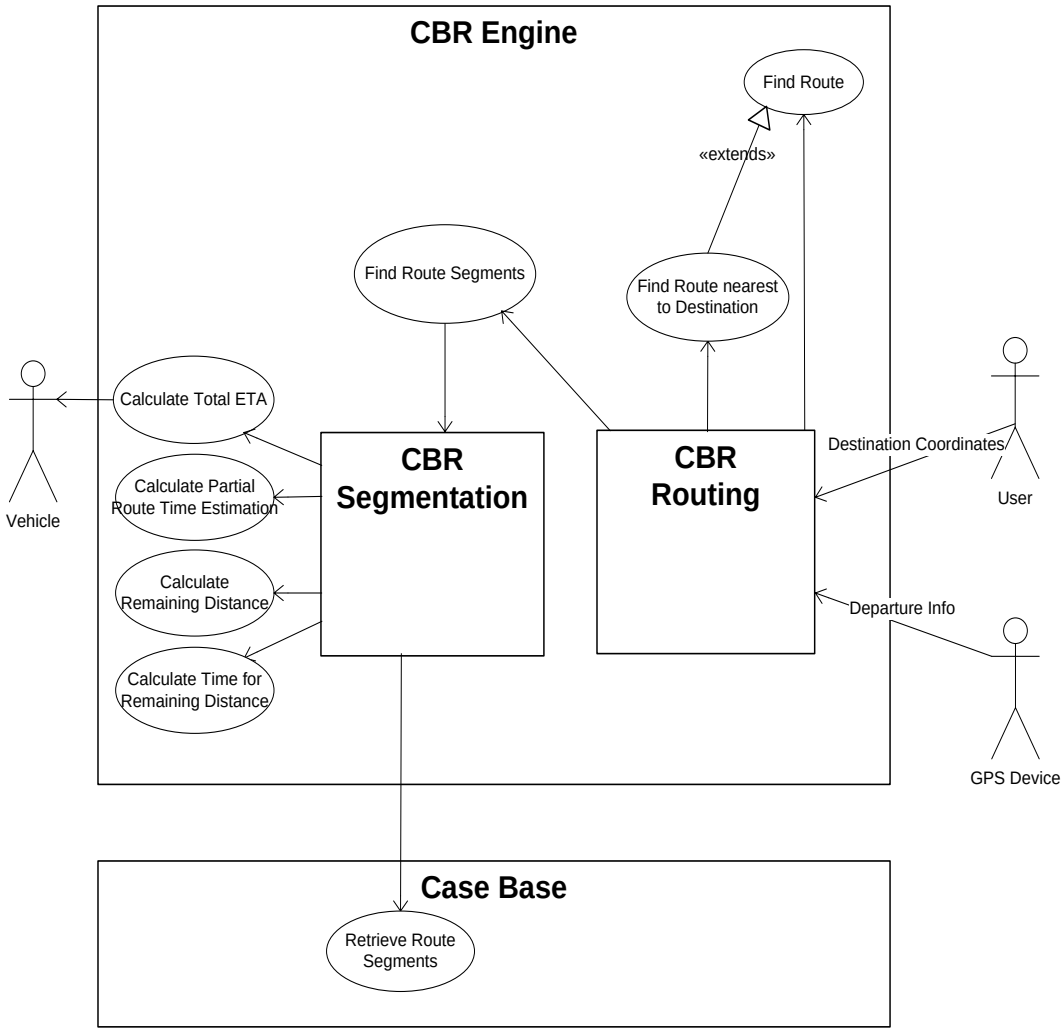

Figure 4-5: Use Case 4. Partial case matching of route

4.3.5. Use Case 5

Partial Route Segment's Case Matching

Description:

Often it happens in our CBR Engine that the CBR Segmentation module does not find an exact case against each route segment. Consider that a vehicle travels on a route R5 (as explained in figure 3.1 and table 3.2) for the first time on Wednesday at 12:35 pm. This route consists of route segments S1, S6, S7, S13 and S12. Being the first time travelling, the travel time spent on these route segments along their distance is saved in the segment case base and route information is saved in the route case base. Now, suppose the vehicle travels the same route R5 for the second time on Thursday at 2:45 pm. The CBR Engine's Routing module retrieves this route information from the route case base. The CBR Segmentation module gets the list of route segments but does not find an exact route segment's case match. This is because the current journey is travelling on a different day and at different clock timing. Without an exact route segment's case match, the CBR Segmentation still can estimate an accurate time of travel.

In such a scenario, CBR Segmentation adopts an existing solution of similar cases. Those cases which are matching at maximum level to the current situation are retrieved and used. Like in this case, we give more weight to the current journey's day and time of travelling. The CBR Segmentation retrieves those cases that have been travelled on the same day group (4 levels are described at page 30) and at nearest clock time. Based on this information, the route cases travelled on Wednesday at 12:35 pm will be retrieved and their solution (which is time of travel) will be adopted and displayed as solution. This use case is described as figure 4.6.

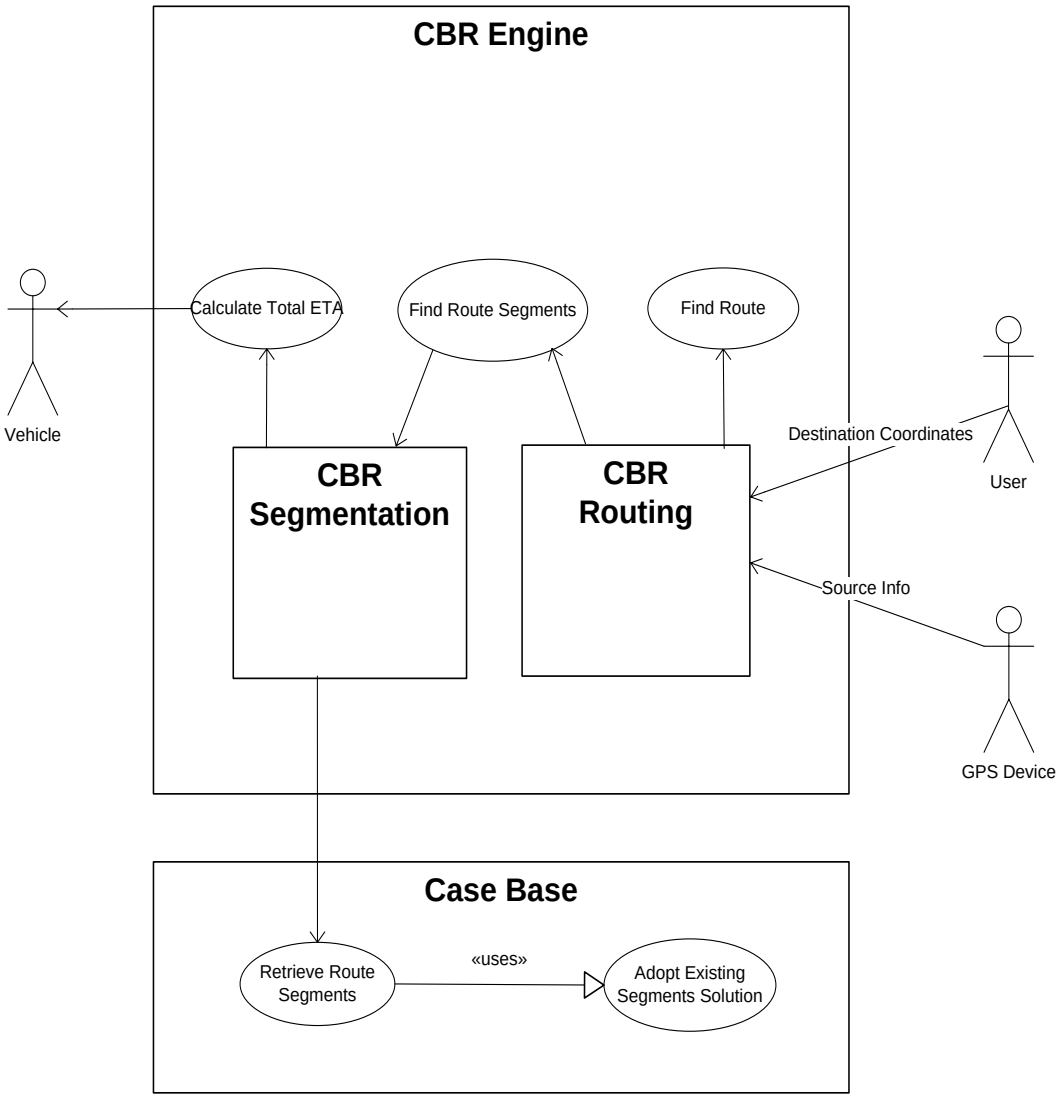

Figure 4-6: Use Case 5. Partial route segment's case matching

4.3.6. Use Case 6

Splitting a Route Segment

Description:

This use case describes the situation when a vehicle travels a journey, the CBR Engine's Routing module found a route case and CBR Segmentation retrieved route segment cases according to the current situation. But on the way to its destination, the vehicle takes some unintended turn. So the vehicle has deviated from its intended route instead of travelling a whole route segment. In such scenarios where a vehicle leaves a route segment in its midway, the route segment needs to be split accordingly. The use case diagram in figure 4-7 illustrates this phenomenon.

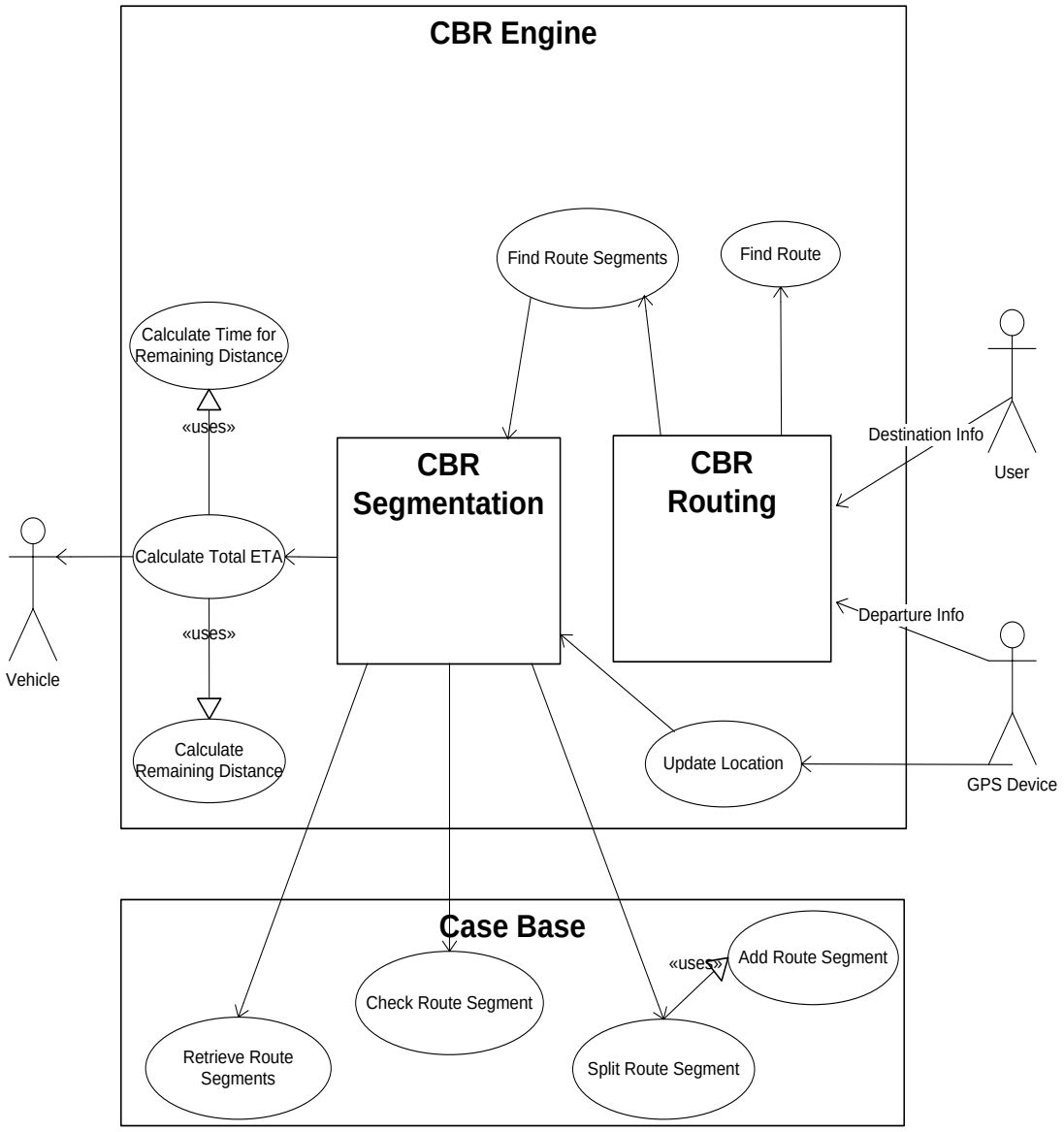

Figure 4-7: Use Case 6. Split CBR

During a journey, the vehicle's GPS device keeps the CBR Segmentation updated about the vehicle's current location coordinates after every 10 seconds interval. CBR Segmentation uses these coordinates to verify the retrieved route segment case. As each route segment is represented by a bounding box (highest and lowest latitude and longitude), whenever a vehicle leaves the boundary of this route segment's representation, CBR Segmentation marks this location coordinates. This marked location cuts the existing route segment into two segments. This incident is described by the following example. Consider a vehicle is travelling route segment S7 as described on figure 3.1. While travelling, the vehicle takes a right turn in mid way. So, at the exact point where the vehicle leaves the segment S7, CBR Segmentation module splits segment S7 into two route segments S7a and S7b. Route segment S7a will be represented from the

starting point of S7 till the vehicle's turning point whereas the route segment S7b will start from this location till the end point of the original segment S7.

In this way CBR Segmentation splits an existing route segment S7 into two route segments S7a and S7b. The original route segment's distance also divides between both new route segments as per of actual readings. Moreover, due to this division, both new route segments need to be included in the route segment's case base.

4.3.7. Use Case 7

Updating a CBR after Route Segment's Division

Description:

In the previous use case we illustrated how a route segment may be divided into two segments if the vehicle leaves it before travelling completely. This use case describes how the CBR engine updates the case base after route segment's division. Both the route case base as well as the segment case base needs to be updated.

CBR Segmentation takes care of the route segment's case base. When a route segment is divided into two segments, the CBR Segmentation searches all cases of this route segment from the segment case base. It divides each case into two cases with the same attributes as the new route segments. As an example, consider route segment S7. Before its division, this route segment consists of 50 cases. So after its division into S7a and S7b (as explained in the previous use case), both these route segments should also consists of 50 cases each. For all these cases, the latitude, longitude and distance attributes are updated according to S7a and S7b. To update the estimated time of travel of these cases, a clever calculation is needed. The estimated time of arrival needs to be divided between new segments according to their distance. So, first of all, the estimated time of arrival to distance ratio of all old cases is calculated. The time of arrival of new route segments is adjusted by multiplying the result calculated in the previous step with the new route segment's distance. This way, an estimated time of arrival is distributed evenly and in precise manner. Other route segment's attributes like Segment_Day, Segment_Time_Travelled and Segment_Count remain the same as the old cases.

Similarly, CBR Routing is responsible to update the route case base. It receives new segments ID's from CBR Segmentation. It searches all routes case base to find those route's cases which need to be updated with these new segment's ID's. This use case is described in figure 4.8.

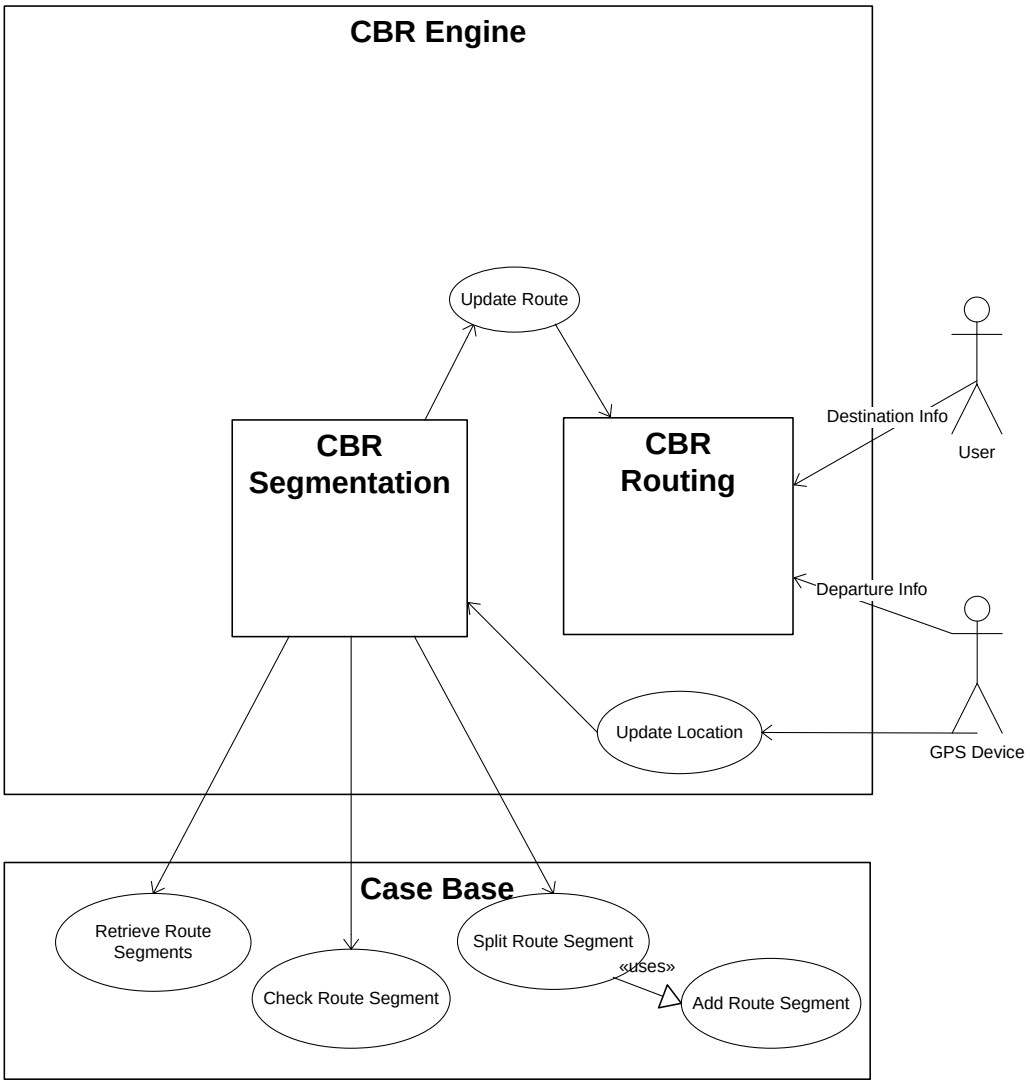

Figure 4-8: Use Case 7. Update CBR

Chapter 5

5. Implementation

The main motivation of our project includes the designing and implementation of the CBR engine for the accurate time of arrival estimation. In this chapter we concentrate on the implementation of one module of the CBR engine. We decided to realize the CBR Segmentation module and implemented the CBR cycle on the route segment.

5.1. System Choice

We have preferred Java as the implementation language to build the CBR module. The most basic reason for this choice is the availability of libraries which may be helpful in fulfilling the requirements of this dissertation.

We have used the jCOLIBRI framework [12] [11]. It is an object oriented and open-source reference rich platform to develop standard CBR applications [13]. It is quite helpful for researchers to build rapid prototypes for their CBR systems.

To develop our CBR engine module, we have used eclipse as integrated development environment [9] in order to use java. Eclipse supports the jCOLIBRI framework and loads its necessary libraries to build CBR application in an easy manner.

Further we have used MySQL as database of our project. It is an open source database and used extensively by most researchers and developers [19]. We have used it to store case bases and to persistently retain new cases as per the application requirement.

5.2. Implementation Description

One key concern at the early stage of this development process was how to represent the cases. In this regard, we used MySQL database and named the newly created database as "etatest". A single table named "segmentcasebase" is created in order to fulfill the requirements of case representation. This table covers both the description as well as the solution parts. Appendix A represents the database design organization of CBR Engine.

We have implemented the CBR segmentation module as a set of classes presented in appendix B.

ETA.java

This is the main implemented class which uses various libraries of jCOLIBRI. ETA.java uses standard CBR Application interface. This class is used to implement reasoning technique. Major functionalities introduced here are the establishment of the similarity and retrieval of the most similar cases according to the query description.

First of all the preCycle () method is used to load the case base; it loads all route segment cases from database to memory. Next the cycle () method executes the whole CBR cycle i.e. it performs all 4 R's. For similarity measures of a given query and case base, we use global as well as local similarity functions. The global similarity method uses the nearest neighbor approach in order to calculate similarity of compound attributes whereas the local similarity methods are used to calculate similarity for simple case attributes.

In our implementation, we selected the top most similar case in retrieval phase and used it for further CBR phases. Finally, we have implemented the postCycle () to finish or terminate the connection.

SegmentDescription.java

This class is used in order to design and represent the route segment cases within the application. Each route segment case is described by many description attributes and a single solution attribute. In the SegmentDescription.java class, we have defined only the description part of route segment. To do so, a java bean is implemented. This java bean describes all description attributes of the route segment. Finally, getter and setter methods are added for each attribute defined in the previous step.

We have defined enumerated data type to represent SegmentDay in this class. For all other attributes, normal data types are used.

SegmentSolution.java

To represent the solution part of a route segment case, we have developed SegmentSolution.java class. This class is exactly the same as that of SegmentDescription.java class. Solution bean is implemented and then getter and setter methods are added for SegmentETA attribute.

Thresholdvalue.java

We have defined location attributes of route segment's case by Double data type. While getting a location's coordinates value by a GPS based device, values may slightly vary each time. To incorporate this varying factor, Thresholdvalue.java class is constructed. The compare functions in this class compare both values i.e. actual location coordinate reading by a GPS device and the case reading that exists in the case base. This function returns 1 if the difference between two double values is less than the defined threshold value, otherwise it returns 0.

ClockInterval.jav

In our design approach of a case, one case can represent one clock hour time interval. As 24 hours exist in a day, to incorporate this factor we have developed ClockInterval.java class. This class calculates the similarity between two clock hour times inside the interval of 24. It does so by using this similarity function and returns the value;

Similarity (caseTime, queryTime) = 1 - (|caseTime – queryTime| / interval)

Database Configuration

We used databaseconfig.xml in order to configure the database connector. This contains information of other classes and files used.

Hibernate Configuration

To access database management system, hibernate.cfg.xml configuration file is used.

Mapping Configuration

Mapping files are used to map java bean with the data base tables. We are using two mapping configuration files. Both files define segmentcasebase table used as database table and then configure its columns.

1. **SegmentDescription.hbm.xml**

In this file, mapping configuration is done for the bean attributes introduced in SegmentDescription.java class against each related column of the database table.

2. **SegmentSolution.hbm.xml**

In this file, mapping configuration is done for the bean attributes introduced in SegmenSolution.java class against each related column of the database table.

Chapter 6

6. System Evaluation and Results

After implementation of CBR segmentation, we have performed its evaluation. System evaluation is necessary in order to demonstrate the advantages and limitations of the case-based reasoning approach and to analyze the performance. We have considered many aspects in this regard which might influence the performance of the application.

6.1. Evaluation Description

In order to evaluate CBR segmentation, we need a large database containing cases for route segments. So we have assumed route segments as shown in figure 3-1 and described in table 3-5. Further we have assumed estimated time of arrival for these route segments: one example of such a database for route segment s5 used for evaluation is shown in table 6-1.

At first instant, we used different sizes of case base and analyzed the retrieval efficiency. We have assumed that each route segment can contain maximum 168 numbers of cases. This is calculated by assuming 24 cases (1 case for each clock hour) for each day of the week, hence becoming 168 in total.

Exact hit of a case retrieval depends on the number of cases a case base contains. This exact retrieval ratio can easily be calculated by the following simple calculation.

Case match = total number of cases in a case base/total number of possible cases*100. This indicates that by using the case base in table 6-1, we can obtain a precise case retrieval accuracy of approx 12.5% only.

This result demonstrates the behavior of case-based reasoning that when a case base contains more cases, it gives more hits against a query. In short, if a single route case base contains its maximum 168 cases, query will be responded by 100% hit retrieval result. These results favor the assumption of case-based reasoning that the system gives more accurate retrieve results when we continue to extend our case base.

Table 6-1: Segment case base for route segment s5

Segment Id	Segment StartLat	Segment StartLon	Segment EndLat	Segment EndLon	Segment Distance	Segment Day	Segment Time	Segment Count	Segment ETA
1	53.345903	-6.255103	53.347475	-6.255025	0.174875	Monday	8	1	00:00:23
2	53.345903	-6.255103	53.347475	-6.255025	0.174875	Monday	9	1	00:00:26
3	53.345903	-6.255103	53.347475	-6.255025	0.174875	Monday	17	1	00:00:24
4	53.345903	-6.255103	53.347475	-6.255025	0.174875	Tuesday	7	1	00:00:18
5	53.345903	-6.255103	53.347475	-6.255025	0.174875	Tuesday	10	1	00:00:22
6	53.345903	-6.255103	53.347475	-6.255025	0.174875	Tuesday	13	1	00:00:23
7	53.345903	-6.255103	53.347475	-6.255025	0.174875	Wednesday	5	1	00:00:16
8	53.345903	-6.255103	53.347475	-6.255025	0.174875	Wednesday	9	1	00:00:26
9	53.345903	-6.255103	53.347475	-6.255025	0.174875	Wednesday	18	1	00:00:24
10	53.345903	-6.255103	53.347475	-6.255025	0.174875	Thursday	8	1	00:00:20
11	53.345903	-6.255103	53.347475	-6.255025	0.174875	Thursday	10	1	00:00:20
12	53.345903	-6.255103	53.347475	-6.255025	0.174875	Thursday	15	1	00:00:20
13	53.345903	-6.255103	53.347475	-6.255025	0.174875	Friday	9	1	00:00:23
14	53.345903	-6.255103	53.347475	-6.255025	0.174875	Friday	11	1	00:00:24
15	53.345903	-6.255103	53.347475	-6.255025	0.174875	Friday	13	1	00:00:25
16	53.345903	-6.255103	53.347475	-6.255025	0.174875	Saturday	5	1	00:00:15
17	53.345903	-6.255103	53.347475	-6.255025	0.174875	Saturday	8	1	00:00:18
18	53.345903	-6.255103	53.347475	-6.255025	0.174875	Saturday	18	1	00:00:18
19	53.345903	-6.255103	53.347475	-6.255025	0.174875	Sunday	7	1	00:00:16
20	53.345903	-6.255103	53.347475	-6.255025	0.174875	Sunday	10	1	00:00:19
21	53.345903	-6.255103	53.347475	-6.255025	0.174875	Sunday	13	1	00:00:21

Besides this benefit of large case base, we can observe some lismitations as well due to an increase in case base size. With continuous increase in number of cases in the case base, the database size carries on to grow. Eventually it requires more processing and time to retrieve the most appropriate case from case base. We imagine implementing such a system to some location aware device on a vehicle. Such location aware devices always have limited memory and computation power, so a balance between the case base sizes is required.

6.2. Performance Metrics

We have chosen the following two performance metrics to evaluate our system.

1. Latency
2. Cases similarity

6.2.1. Latency

The performance of CBR application can easily be determined by observing the latency required to retrieve a recommendation against a given query. To study latency results in detail, we have constructed case bases for all 23 route segments. All these case bases have variable number of cases. It has been observed that as the case base size grows, the time required for retrieval recommendation also increases.

The following results have been derived by observing two groups of case bases. The first group consists of route segments s1 to s7: the number of cases and the time required to retrieve a result for these case bases is shown in table 6-2, whereas the second group consists of route segments s8 to s13: the description of their number of cases and time required is described in table 6-3. Graph diagrams in figure 6-1 and 6-2 respectively show the latency for retrieval. Broad lines in these graphs show the actual time required while the narrow line depicts an idea of the tendency of increase in latency.

Table 6-2: Number of case and time required for case bases of route segments s1-s7

Segment	Number of Cases	Time Taken
s1	168	1353
s2	126	1324
s3	84	1302
s4	42	1250
s5	21	1224
s6	14	1200
s7	7	1200
s1Single	1	1197

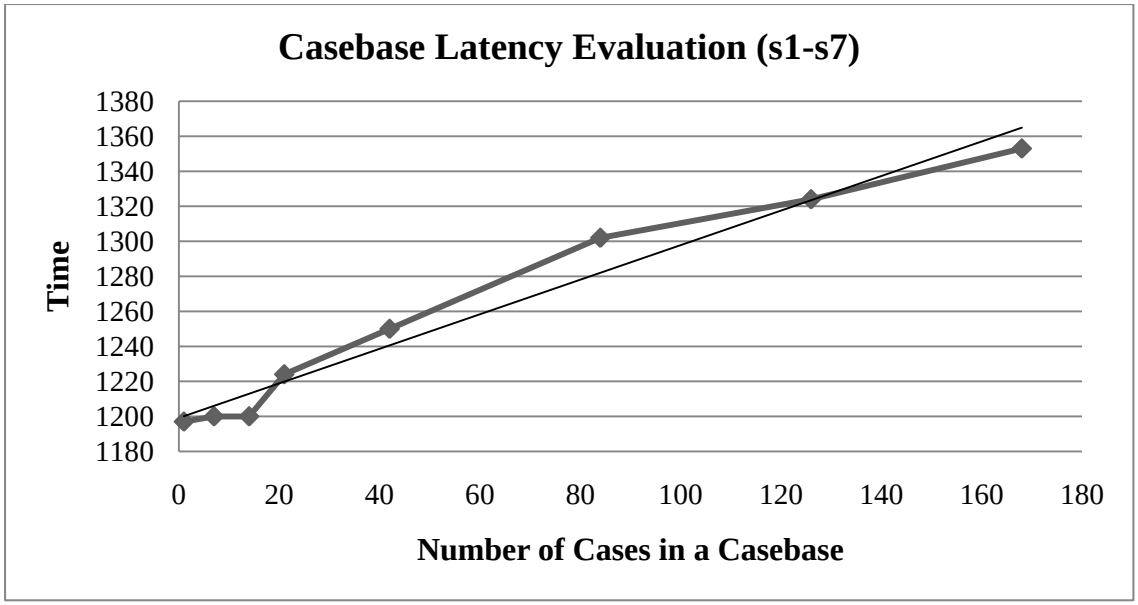

Figure 6-1: Case base latency evaluation of route segments s1-s7

Table 6-3: Number of cases and time required for case bases of route segments s8-s13

Segment	Number of Cases	Time Taken
s8	144	1356
s9	120	1321
s10	96	1312
s11	72	1299
s12	48	1262
s13	24	1230

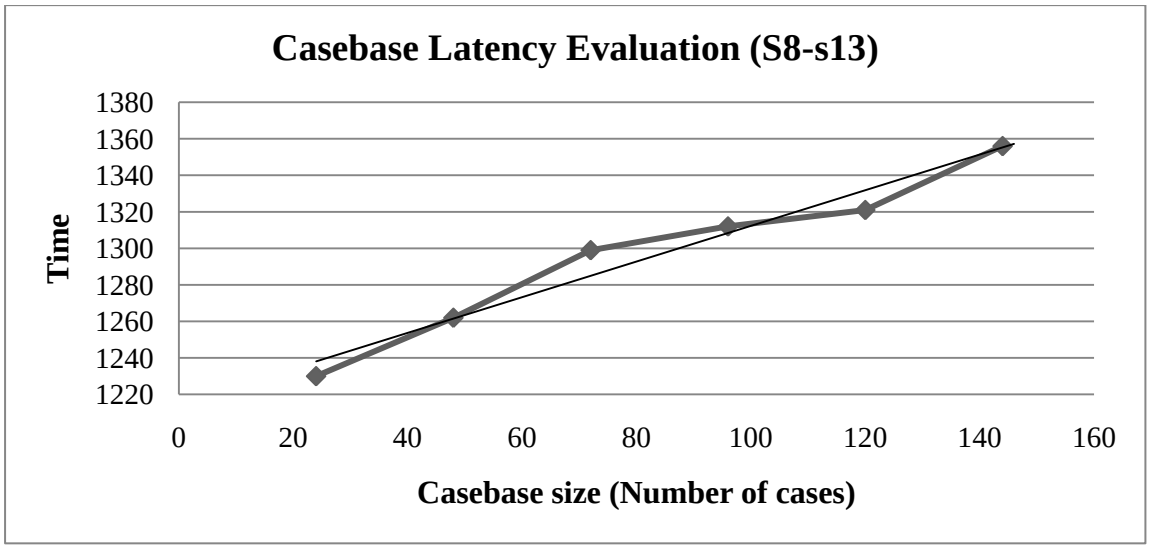

Figure 6-2: Case base latency evaluation of route segments s8-s13

We have also observed that case bases having similar number of cases require almost the same amount of time to retrieve a case. Table 6-4 consists of route segments s13 to s18, each case base having 168 numbers of cases. The bar diagram in figure 6-3 shows the same retrieval behavior for all segments.

Table 6-4: Number of cases and time required for case bases of route segments s14-s18

Segment	Number of Cases	Time Taken
s14	168	1363
s15	168	1371
s16	168	1368
s17	168	1356
s18	168	1360

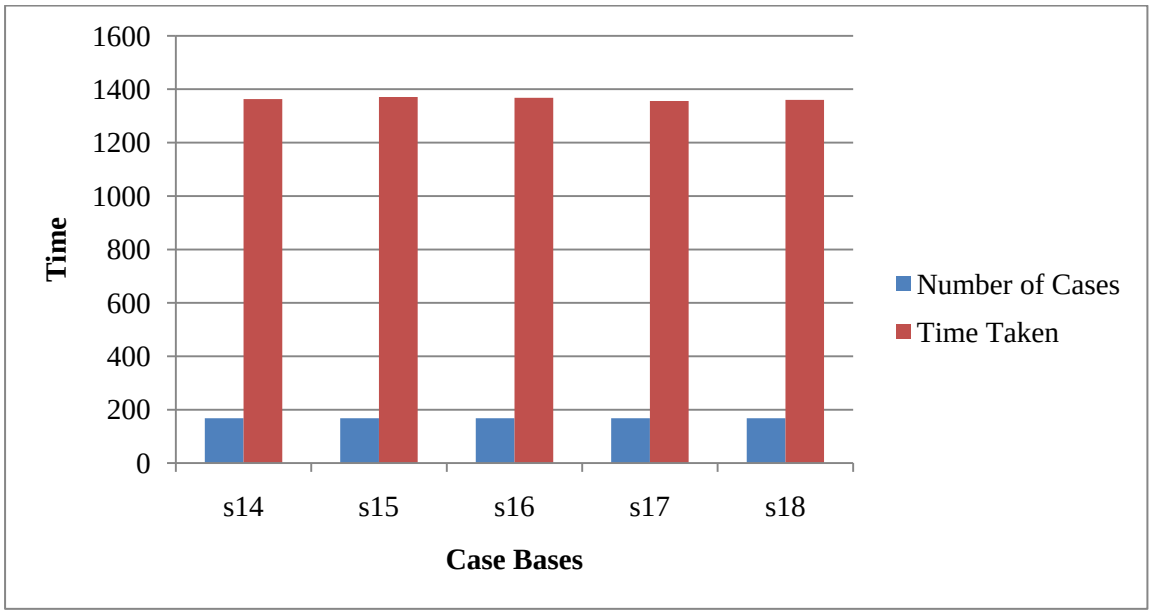

Figure 6-3: Case base latency evaluation of route segments s14-s18

6.2.2. Cases Similarity

For the second performance metric, first we have constructed a case base having 268 cases by selecting random cases belonging to all 23 route segments. Then, we performed cases similarity tests within the boundary of this case base. This cases similarity evaluation demonstrates how cases are related to each other.

We have used two different methodologies for this type of evaluation. In the first methodology, we have split the case base into two portions: a query and a normal case

base. In this type of evaluation, we have taken results by breaking the case base in different variations.

First the 25% of this case base is assigned as query part. All cases in the query part are compared with the normal case base part and the similarity result is analyzed. The graph diagram in figure 6-4 demonstrates the similarity results.

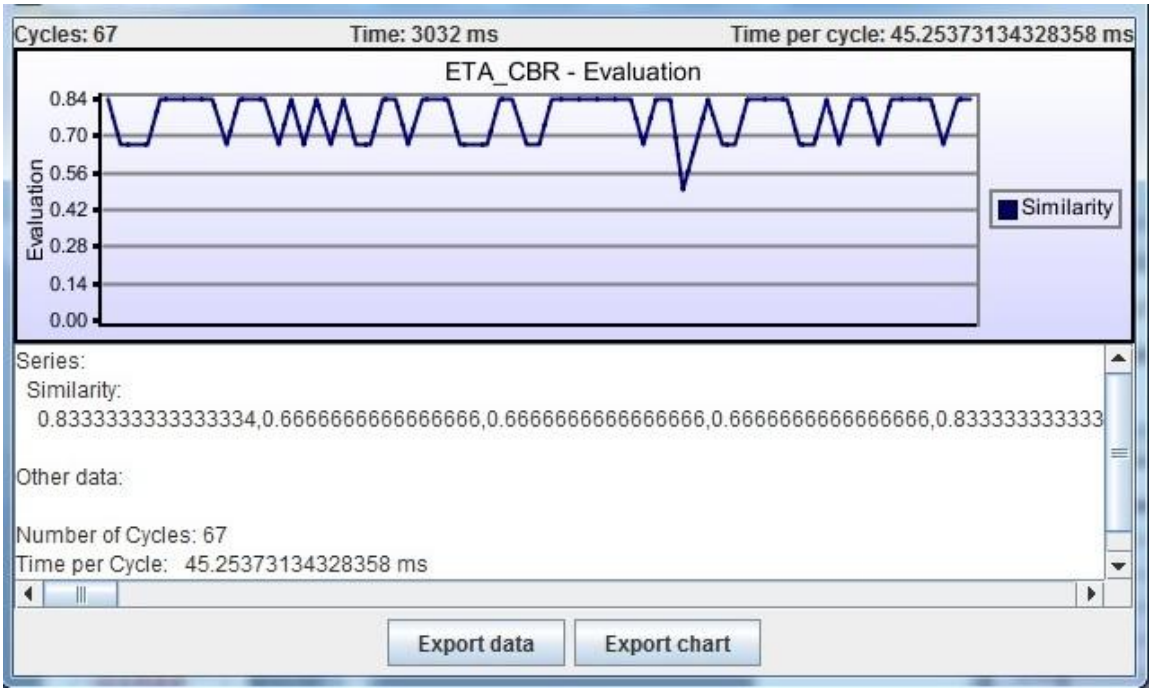

Figure 6-4:Case similarity evaluation by assigning 25% of cases as query

We have compared 6 attributes of a case in evaluation. Among these 6 attributes, 4 belong to origin and destination's latitude and longitude information. The remaining 2 attributes are Segment_Day and Segment_Time_Travelled: all these have been discussed in detail in chapter 3. When a case is compared with other cases of the same route segment, 4 attributes belonging to coordinates information are the same. This gives about 67% similarity result and can be clearly observed in the figure. When a query case's 5th attribute also matches with the case base, we can observe 83% similarity result. As in evaluation, an actual case of the case base is treated as query, and we do not have two similar cases in the case base, so we are not observing 100% similarity in the figure.

Besides similarity measures, the figure also depicts latency measures. As the retrieval cycle executes 67 times, total and average time taken for a cycle can also be observed on the diagram.

In figure 6-5 the same case similarity test is performed again, but here we have assigned 50% cases as query cases.

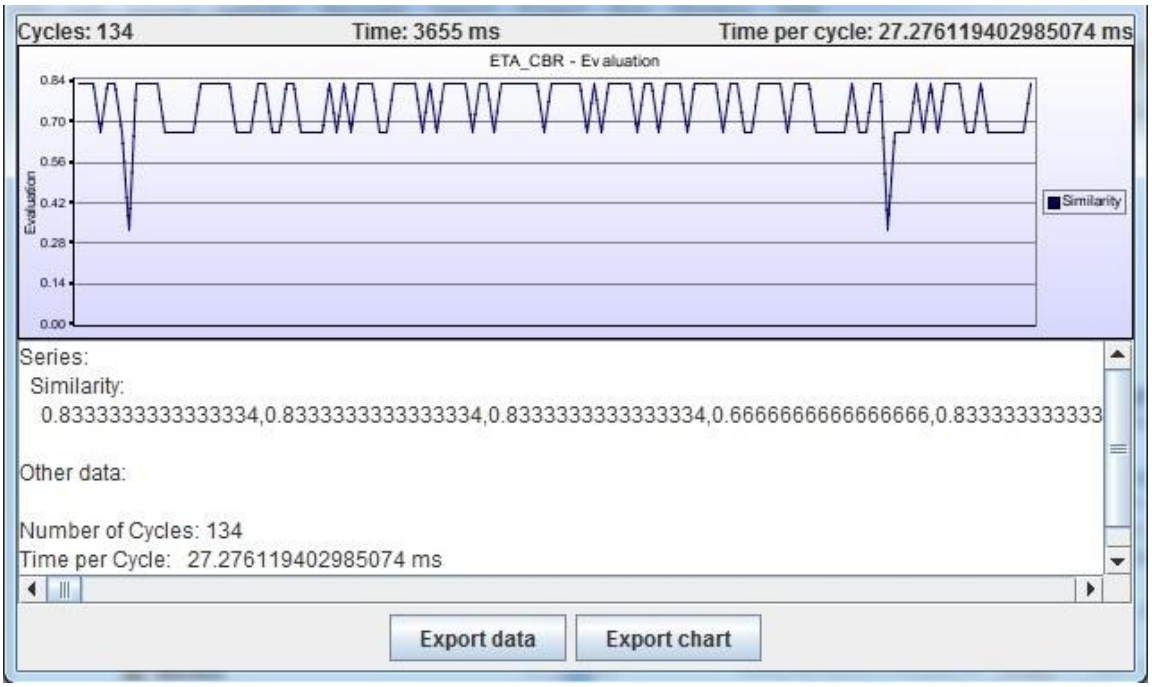

Figure 6-5: Case similarity evaluation by assigning 50% of cases as query

In figure 6-6, the same test as above is performed again, but here it is repeated twice. In both graphs, we notice some low similarity results. This is due to mismatching in location coordinates. We also observe that the total time increased due to increase in the number of cycles caused by repetition.

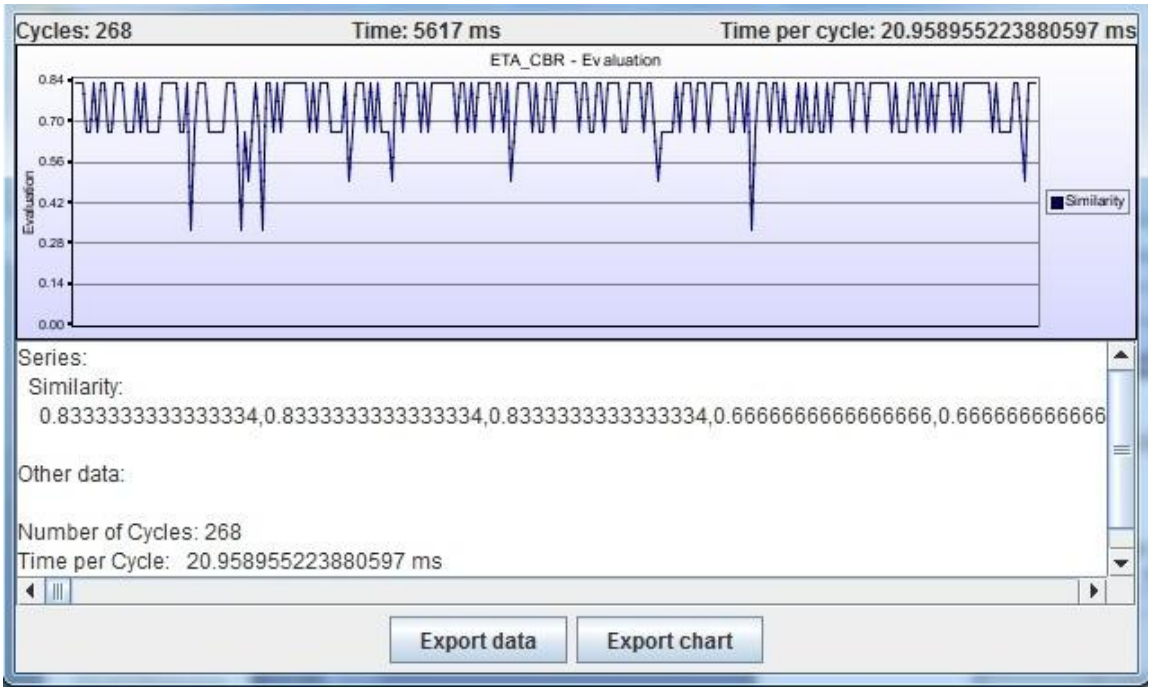

Figure 6-6: Case similarity evaluation by assigning 50% of cases as query and repeated for two times

Here in figure 6-7, the case similarity test is performed by assigning 75% of case base as query part.

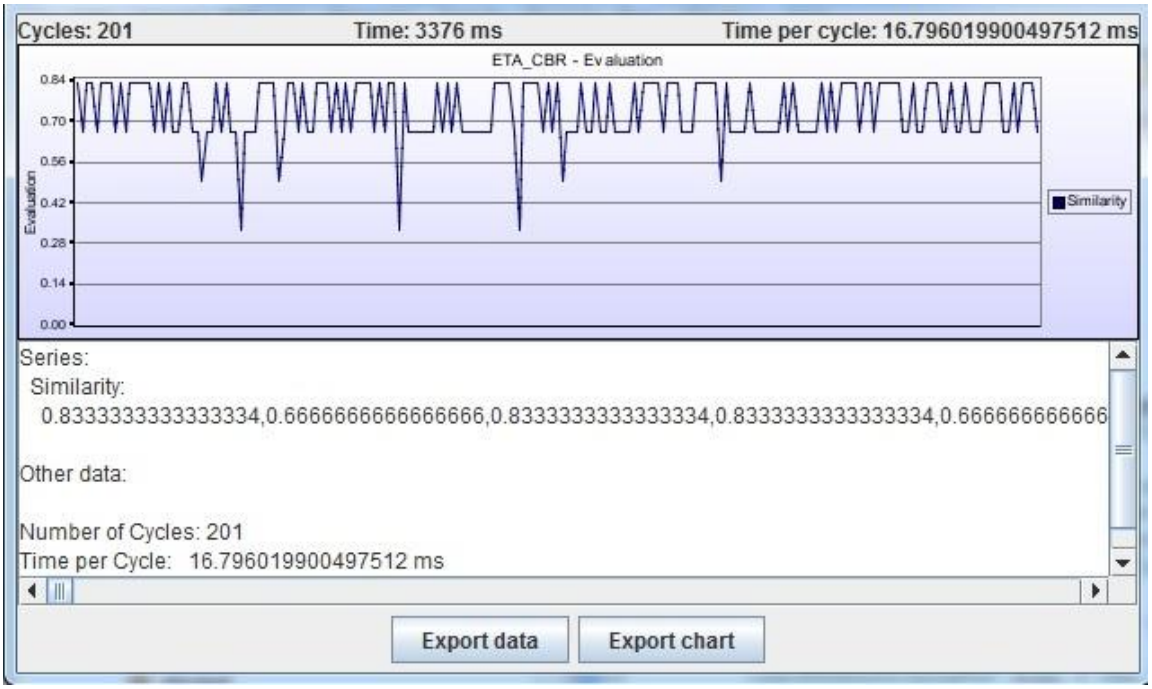

Figure 6-7: Case similarity evaluation by assigning 75% of cases as query

All of the above diagrams from 6-4 to 6-7 demonstrate results by splitting a case base into a query and a normal case base part. In the following graph diagram, instead of dividing a

case base, we have performed a case similarity test for each case in the case base with all the remaining cases. Hence, all cases are used as query one by one against the whole case base for similarity measure.

The case base we have used in evaluation consists of cases belongs to all 23 route segments, and it has more than one case for each route segment. That is why we can observe in the graph diagram that similarity results vary from 67% to 83%.

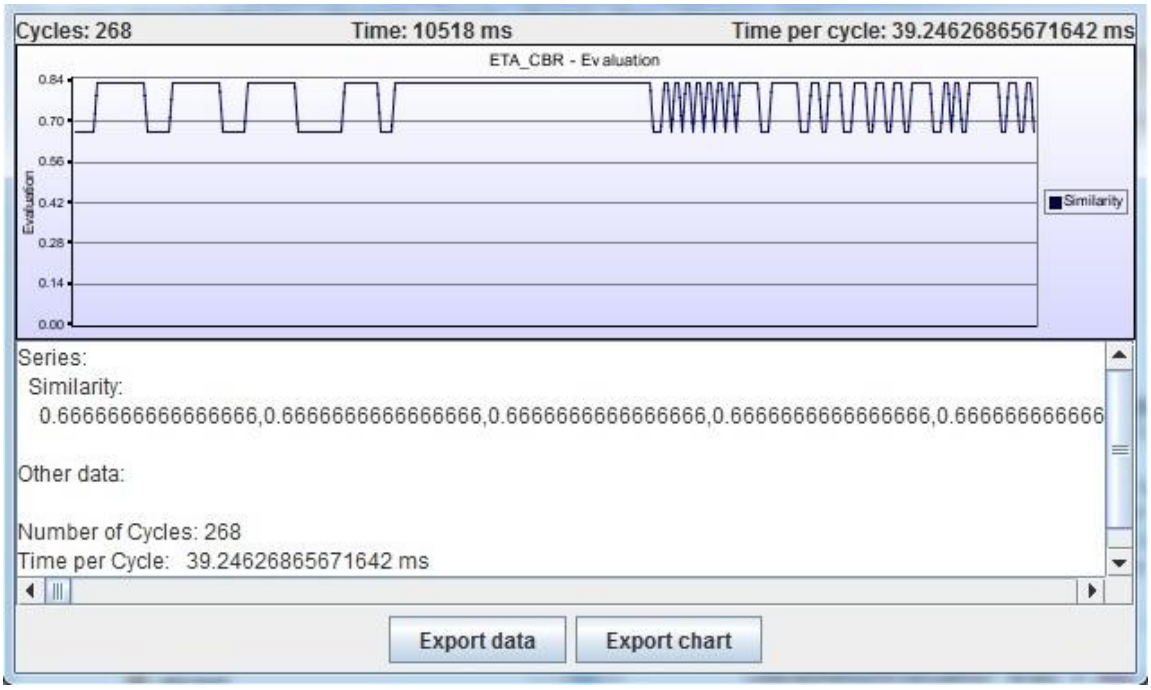

Figure 6-8: Case similarity evaluation of each case in the case base with remaining all ones

Chapter 7

7. Conclusion and Future Work

7.1. Conclusion

The use of case-based reasoning methodology in estimation time of arrival is a rather easy technique when we implement it for a category of vehicles. We believe that being a pretty simple model, this approach turns out good results even at times with serious congestion on road networks.

Furthermore, our approach shows results in advance about congestion on different routes at any particular time. This can be very helpful for drivers to choose their route plan before travelling.

As CBR is a memory based methodology, our emphasis remains to store minimum number of cases. This is indeed necessary for a quick response against a journey query. Moreover, the approach of CBR engine we have modeled and presented, uses real-time statistics for better performance.

7.2. Future Work

The CBR engine's model description needs to be checked and verified in an actual scenario in order to measure its impact and feasibility. Moreover, we have implemented only a part of the CBR module. The whole module needs to be implemented to better evaluate it. We have incorporated the bounding box strategy in order to represent a route segment. Although this strategy helps in using minimum memory and processing resources but it seems to be not an efficient way of route segment representation.

Besides the above issues, we believe that weather also plays an important role in traffic flow and congestion. By adding the weather factor in the CBR engine, it can improve the overall accuracy.

References

[1] Aamodt, A. *Case-Based Reasoning - An Introduction.*

[2] Aamodt, A., & Plaza, E. (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *Artificial Intelligence Communications, IOS Press,* , 39-59, Vol. 7: 1.

[3] Bergmann, R., Kolodner, J., & Plaza, E. (2005). Representation in case-based reasoning. *The Knowledge Engineering Review* , 209-213, Volume 20 Issue 3.

[4] Calculate distance between 2 latitude, longitude (lat/lon) points (GPS positions). *http://bluemm.blogspot.com/2007/01/excel-formula-to-calculate-distance.html [Accessed: 2011-08-16 16:09].*

[5] *Case-Based Reasoning Wiki : http://cbrwiki.fdi.ucm.es/mediawiki/index.php/Meetings, [Accessed: 2011-10-29 12:45].*

[6] Chang, S. (2002). Handbook of Software Engineering and Knowledge Engineering, Volume 1. World Scientific Pub Co Inc.

[7] Chen, G., Chen, J., Zhao, Z., & Ruan, X. Y. (2005). An object-oriented hierarchical case representation of automotive panels in a computer-aided process planning system. *The International Journal of Advanced Manufacturing Technology* , 1323-1330, Volume: 26, Issue: 11-12.

[8] Chung, E.-H., & Shalaby, A. (2007). Expected Time of Arrival Model for School Bus Transit Using Real-Time Global Positioning System-Based Automatic Vehicle Location Data. *Journal of Intelligent Transportation Systems* , 157-167.

References

[9] Eclipse IDE. *http://www.eclipse.org/*
[Accessed: 2011-08-25 11:25].

[10] Fagan, D., & Meier, R. (2011). Intelligent Time of Arrival Estimation. *Integrated and Sustainable Transportation System (FISTS), IEEE Forum*, (pp. 60-66).

[11] GAIA - Group of Artificial Intelligence Applications. *http://gaia.fdi.ucm.es/projects/jcolibri/*
[Accessed: 2011-08-22 10:15].

[12] GAIA - jCOLIBRI. *http://gaia.fdi.ucm.es/research/colibri/jcolibri*
[Accessed: 2011-12-01 20:35].

[13] GAIA - People, Users and Applications. *http://gaia.fdi.ucm.es/research/colibri/people-users-apps*
[Accessed: 2011-12-02 12:52].

[14] Karbassi, A., & Barth, M. (2003). Vehicle route prediction and time of arrival estimation techniques for improved transportation system management. *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE* , (pp. 511-516).

[15] Kolodner, J. (1992). *An Introduction to Case-Based Reasoning.* Morgan-Kaufmann Publishers Inc.

[16] Kolodner, J. (1993). *Case-Based Reasoning.* Morgan Kaufmann.

[17] Kolodner, J. L. (1983). Maintaining Organization in a Dynamic Long-Term Memory. *Cognitive Science* , 243-280 Volume 7 Issue 4 .

[18] Mantaras, R. L., McSherry, D., Bridge, D., Leake, D., & Smyth, B. (2005). Retrieval, reuse, revision and retention in Case-Based Reasoning. *The Knowledge Engineering Review* , 215-240, Volume 20 Issue 3.

[19] MySQL Community Server. *http://www.mysql.com/*
[Accessed: 2011-08-29 10:46].

[20] *National Travel Survey.* Ireland, C. S. (2009). http://www.cso.ie/releasespublications/documents/transport/2009/nattravel09.pdf [Accessed: 2011-10-28 12:43].

[21] Pu, W., Lin, J., & Long, L. (2009). Real-Time Estimation of Urban Street Segment Travel Time Using Buses as Speed Probes. *Transportation Research Board of the National Academies, Washington DC* , 81-89.

[22] Rashid, O., Coulton, P., Edwards, R., Fisher, A., & Thompson, R. (2005). Mobile information systems providing estimated time of arrival for public transport users. *Vehicular Technology Conference, IEEE*, (pp. 2565-2569 Vol.5).

[23] Riesbeck, C. K., & Schank, R. C. (1989). *Inside Case-Based Reasoning.*

[24] Watson, I. (May 21-23 2001). Case-Based Reasoning and Knowledge Management: a Perfect Match? *In, Proc. FLAIRS-2001. 14th Int. FLAIRS Conference* (pp. 118-123). Key West Florida, Menlo Park CA, US: AAAI Press.

[25] Watson, I., & Perera, S. (1998). A Hierarchical Case Representation using Context Guided Retrieval. *Knowledge Based Systems* , 285-292.

[26] Weber, R. O., Ashley, K. D., & Bruninghaus, S. (2006). Textual Case-Based Reasoning. *The Knowledge Engineering Review* , 255-260, Vol 20:30.

[27] Weigang, L., Koendjbiharie, M. W., Juca, R. C., Yamashita, Y., & Maciver, A. (2002). Algorithms for estimating bus arrival times using GPS data. *Proceedings IEEE 5th International Conference on Intelligent Transportation Systems* , (pp. 868-873).

Chapter 8

8. Appendices

8.1. Appendix A: Data Base Organization of CBR Engine

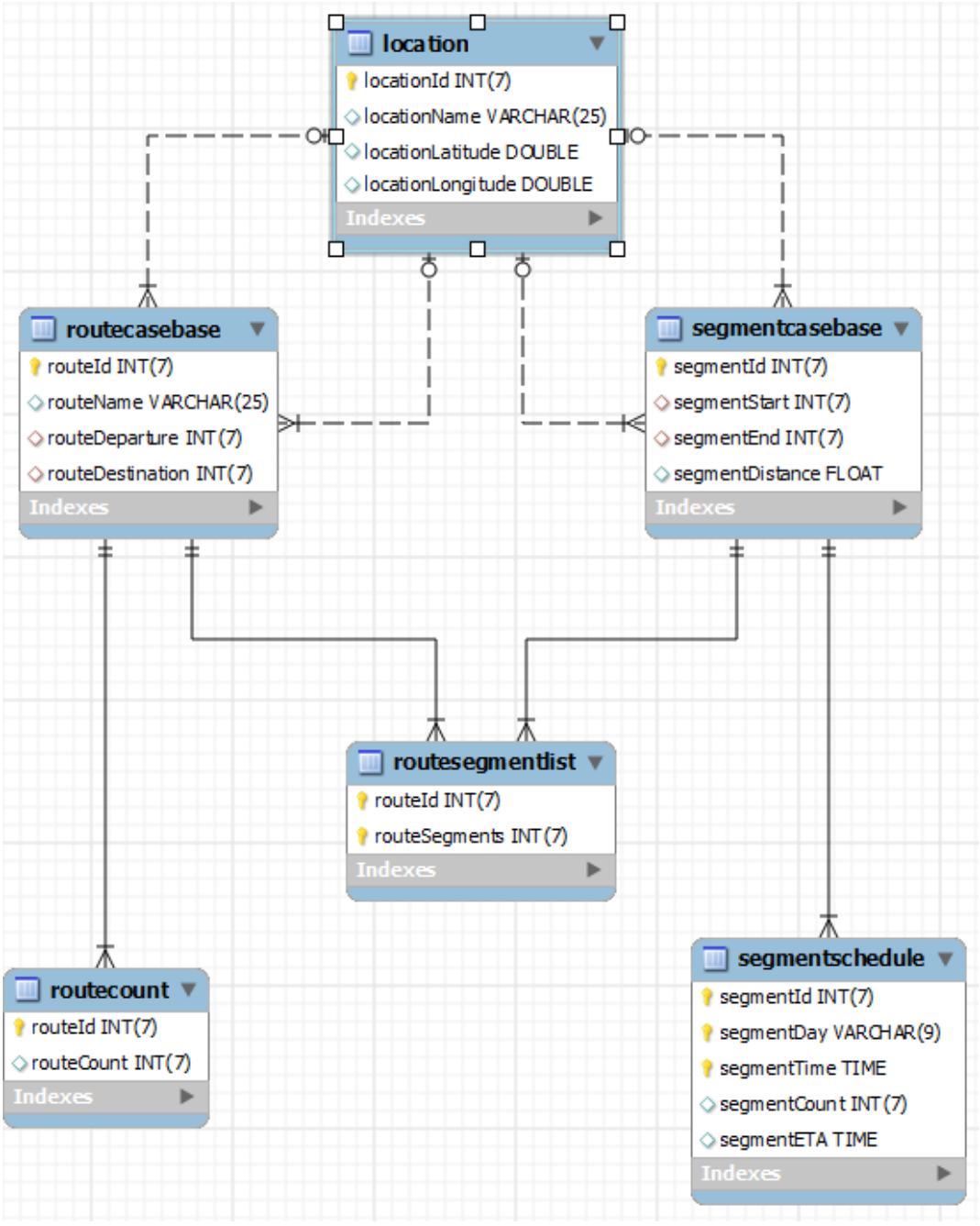

Figure 8-1: Data Base Organisation of CBR Engine to determine ETA

8.2. Appendix B: Source Code of CBR Engine

8.2.1. ETA.java

```
/*
 * ETA.java
 * implemented by using jCOLIBRI framework
 * @author Irfan Nazir
 * Dated: 2011-11-15
 * Version 1.0
 */
package eta.cbr;

import java.text.SimpleDateFormat;
import java.util.Collection;
import java.util.Date;
import java.util.HashMap;

import eta.cbr.data.SegmentDescription;
import eta.cbr.data.SegmentDescription.Day;
import jcolibri.casebase.LinealCaseBase;
import jcolibri.cbraplications.StandardCBRApplication;
import jcolibri.cbrcore.Attribute;
import jcolibri.cbrcore.CBRCase;
import jcolibri.cbrcore.CBRCaseBase;
import jcolibri.cbrcore.CBRQuery;
import jcolibri.cbrcore.Connector;
import jcolibri.connector.DataBaseConnector;
import jcolibri.exception.ExecutionException;
import jcolibri.method.retrieve.RetrievalResult;
import jcolibri.method.retrieve.NNretrieval.NNConfig;
import jcolibri.method.retrieve.NNretrieval.NNScoringMethod;
import jcolibri.method.retrieve.NNretrieval.similarity.global.Average;
import jcolibri.method.retrieve.NNretrieval.similarity.local.ClockInterval;
import jcolibri.method.retrieve.NNretrieval.similarity.local.EnumCyclicDistance;
import jcolibri.method.retrieve.NNretrieval.similarity.local.Thresholdvalue;
import jcolibri.method.retrieve.selection.SelectCases;
import jcolibri.method.reuse.CombineQueryAndCasesMethod;

public class ETA implements StandardCBRApplication
{

    //  Connector object by using jcolibri.cbrcore.Connector
    Connector etaConnector;

    // CaseBase Object by using jcolibri.cbrcore.CBRCaseBase
    CBRCaseBase etaCaseBase;

    // Object for global similarity measure
```

```
    NNConfig etaConfig;

    @Override
    public void configure() throws ExecutionException
    {
        try{
            // Database connector creation
            etaConnector = new DataBaseConnector();

            // Database connector initializing with the configuration file

    etaConnector.initFromXMLfile(jcolibri.util.FileIO.findFile("eta/cbr/data/databaseconfig.xml")
);

            // Lineal case base creation
            etaCaseBase = new LinealCaseBase();
        } catch (Exception e){
            throw new ExecutionException(e);
        }
    }

    @Override
    // Precycle phase to read and organize cases into the case base.
    public CBRCaseBase preCycle() throws ExecutionException
    {
        // load cases from etaConnector to etaCaseBase
        etaCaseBase.init(etaConnector);

        // Display all cases
        java.util.Collection<CBRCase> allCases = etaCaseBase.getCases();
        System.out.println();
        System.out.println("Displaying all cases:");
        for(CBRCase singleCase : allCases)
            System.out.println(singleCase);
        return etaCaseBase;
    }

    @Override
    // CBR cycle execution takes place by giving input in the form of a query
    public void cycle(CBRQuery query) throws ExecutionException
    {
        //  KNN Configuration
        etaConfig = new NNConfig();

        // Average() global similarity function is used for the description of the eta case
        etaConfig.setDescriptionSimFunction(new Average());

        // SegmentStartLat --> Thresholdvalue()
        // original 53.341992. Acceptable 53.341892 to 53.342092 i.e +0.000100 to -0.000100
```

```
        etaConfig.addMapping(new Attribute("SegmentStartLat", SegmentDescription.class), new
Thresholdvalue(0.0001));

        // SegmentStartLon --> Thresholdvalue()
        etaConfig.addMapping(new Attribute("SegmentStartLon", SegmentDescription.class), new
Thresholdvalue(0.0001));

        // SegmentEndLat --> Thresholdvalue()
        etaConfig.addMapping(new Attribute("SegmentEndLat", SegmentDescription.class), new
Thresholdvalue(0.0001));

        // SegmentEndLon --> Thresholdvalue()
        etaConfig.addMapping(new Attribute("SegmentEndLon", SegmentDescription.class), new
Thresholdvalue(0.0001));

        // SegmentDay --> EnumCyclicDistance()
        Attribute s_day = new Attribute("SegmentDay", SegmentDescription.class);
        etaConfig.addMapping(s_day, new EnumCyclicDistance());
        etaConfig.setWeight(s_day, 0.5);

        // SegmentTime --> Interval(24). Considering 24 hour clock timings from 00:00 to 23:00
        Attribute s_time = new Attribute("SegmentTime", SegmentDescription.class);
        etaConfig.addMapping(s_time, new ClockInterval(24));
        etaConfig.setWeight(s_time, 0.75);

        // Display given query
        System.out.println();
        System.out.println("The Query Description is as: ");
        //System.out.println(query);
        System.out.println(query.getDescription());
        System.out.println();

        // Execute nearest neighbor algorithm
        Collection<RetrievalResult> evalNN =
NNScoringMethod.evaluateSimilarity(etaCaseBase.getCases(), query, etaConfig);

        evalNN = SelectCases.selectTopKRR(evalNN, 5);

        System.out.println("Following are top Retrieved cases: ");
        for(RetrievalResult retrievedCases: evalNN)
            System.out.println(retrievedCases);

        // Select top priority cases
        Collection<CBRCase> selectedTopCase = SelectCases.selectTopK(evalNN, 1);

        System.out.println();
        System.out.println("Following is the selected case from all Retrieved cases: ");
        for(CBRCase priorityCase: selectedTopCase)
            System.out.println(priorityCase);
```

```
        // Reuse Process
        //Combining the query description with cases solutions and obtaining a list of new cases
        Collection<CBRCase> newReuseCases =
CombineQueryAndCasesMethod.combine(query, selectedTopCase);
        System.out.println();
        System.out.println("Following are combined cases after Reuse process: ");
        for(jcolibri.cbrcore.CBRCase reuseCase: newReuseCases)
            System.out.println(reuseCase);

        // Revise Process
        // Best case is selected to store
        CBRCase topCase = newReuseCases.iterator().next();

        // Defining new ID's
        HashMap<Attribute, Object> newIDs = new HashMap<Attribute,Object>();
        newIDs.put(new Attribute("segmentDescriptionId",SegmentDescription.class), 101);
        jcolibri.method.revise.DefineNewIdsMethod.defineNewIdsMethod(topCase, newIDs);

        System.out.println();
        System.out.println("Following are cases with new ID's: ");
        System.out.println(topCase);

        // Retain Process

        // In order to store cases permanently, needs to uncomment next line of code
        //jcolibri.method.retain.StoreCasesMethod.storeCase(etaCaseBase, topCase);

        System.out.println("Cycle finished.");
    }

    @Override
    public void postCycle() throws ExecutionException
    {
        // TODO Auto-generated method stub
        this.etaCaseBase.close();
    }

    // main function
    public static void main(String[] args)
    {
        // New application is creating
        //StandardCBRApplication estimation = new ETA();

        //Execution Time required for retrieval
        ETA estimation = new ETA();
        try
        {
            // Application configuration
            estimation.configure();
```

```
            // Starting pre-cycle phase to load cases.
            estimation.preCycle();

            //Query object is creating
            CBRQuery etaQuery = new CBRQuery();

            // Segment Description object is creating
            SegmentDescription sd = new SegmentDescription();
            sd.setSegmentStartLat(53.341992);
            sd.setSegmentStartLon(-6.250578);
            sd.setSegmentEndLat(53.344125);
            sd.setSegmentEndLon(-6.249297);
            //String ss = "Friday";

            Day etaDay;
            etaDay = Day.Friday;

            //Enum ee = (Enum) ss;

            //sd.setSegmentDay(SegmentDescription.Day.Thursday);
            sd.setSegmentDay(etaDay);

            Date date = new Date();
            SimpleDateFormat time_in_hour;

            //time_in_hour = new SimpleDateFormat("hh");
            time_in_hour = new SimpleDateFormat("kk");

            sd.setSegmentTime(time_in_hour.format(date));
            System.out.println(time_in_hour.format(date));
            //sd.setSegmentTime("15");

            etaQuery.setDescription(sd);

            // Starting cycle phase
            estimation.cycle(etaQuery);

            System.out.println(time_in_hour.format(date));

            // Starting post-cycle phase
            estimation.postCycle();

        }catch (Exception e)
        {
           org.apache.commons.logging.LogFactory.getLog(ETA.class).error(e);
        }
    }
}
```

8.2.2. SegmentDescription.java

```
/*
 * SegmentDescription.java
 * implemented by using jCOLIBRI framework
 * @author Irfan Nazir
 * Dated: 2011-11-15
 * Version 1.0
 */
package eta.cbr.data;

import jcolibri.cbrcore.Attribute;

public class SegmentDescription implements jcolibri.cbrcore.CaseComponent {

public enum Day {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};

    Integer segmentDescriptionId;
    Double SegmentStartLat;
    Double SegmentStartLon;
    Double SegmentEndLat;
    Double SegmentEndLon;
    Float SegmentDistance;
    //String SegmentDay;
    Day SegmentDay;
    String SegmentTime;
    //Time time;
    Integer SegmentCount;
    //String SegmentETA;

    public String toString() {
        return
"("+segmentDescriptionId+";"+SegmentStartLat+";"+SegmentStartLon+";"+SegmentEndLat+";"
+SegmentEndLon+";"+SegmentDistance+";"+SegmentDay+";"+SegmentTime+";"+SegmentCou
nt+")";
    }

    public Integer getSegmentDescriptionId() {
        return segmentDescriptionId;
    }

    public void setSegmentDescriptionId(Integer segmentDescriptionId) {
        this.segmentDescriptionId = segmentDescriptionId;
    }

    public Double getSegmentStartLat() {
        return SegmentStartLat;
    }

    public void setSegmentStartLat(Double segmentStartLat) {
```

```
        SegmentStartLat = segmentStartLat;
    }

    public Double getSegmentStartLon() {
        return SegmentStartLon;
    }

    public void setSegmentStartLon(Double segmentStartLon) {
        SegmentStartLon = segmentStartLon;
    }

    public Double getSegmentEndLat() {
        return SegmentEndLat;
    }

    public void setSegmentEndLat(Double segmentEndLat) {
        SegmentEndLat = segmentEndLat;
    }

    public Double getSegmentEndLon() {
        return SegmentEndLon;
    }

    public void setSegmentEndLon(Double segmentEndLon) {
        SegmentEndLon = segmentEndLon;
    }

    public Float getSegmentDistance() {
        return SegmentDistance;
    }

    public void setSegmentDistance(Float segmentDistance) {
        SegmentDistance = segmentDistance;
    }

    public Day getSegmentDay() {
        return SegmentDay;
    }

    public void setSegmentDay(Day segmentDay) {
        SegmentDay = segmentDay;
    }

    public String getSegmentTime() {
        return SegmentTime;
    }

    public void setSegmentTime(String segmentTime) {
        SegmentTime = segmentTime;
    }
```

```
    public Integer getSegmentCount() {
        return SegmentCount;
    }

    public void setSegmentCount(Integer segmentCount) {
        SegmentCount = segmentCount;
    }

    @Override
    public Attribute getIdAttribute() {
        // TODO Auto-generated method stub
        return new Attribute("segmentDescriptionId", this.getClass());
    }
}
```

8.2.3. SegmentSolution.java

```
/*
 * SegmentSolution.java
 * implemented by using jCOLIBRI framework
 * @author Irfan Nazir
 * Dated: 2011-11-15
 * Version 1.0
 */
package eta.cbr.data;

import jcolibri.cbrcore.Attribute;

public class SegmentSolution implements jcolibri.cbrcore.CaseComponent {

    Integer segmentSolutionId;
    String SegmentETA;

    public String toString () {
        return "("+segmentSolutionId+";"+SegmentETA+")";
    }

    public Integer getSegmentSolutionId() {
        return segmentSolutionId;
    }

    public void setSegmentSolutionId(Integer segmentSolutionId) {
        this.segmentSolutionId = segmentSolutionId;
    }

    public String getSegmentETA() {
        return SegmentETA;
    }
```

```
    public void setSegmentETA(String segmentETA) {
        SegmentETA = segmentETA;
    }

    @Override
    public Attribute getIdAttribute() {
        // TODO Auto-generated method stub
        return new Attribute ("segmentSolutionId", this.getClass());
    }
}
```

8.2.4. Thresholdvalue.java

```
/*
 * Thresholdvalue.java
 * implemented by using jCOLIBRI framework
 * @author Irfan Nazir
 * Dated: 2011-11-16
 * Version 1.0
 */
package jcolibri.method.retrieve.NNretrieval.similarity.local;

import jcolibri.exception.NoApplicableSimilarityFunctionException;
import jcolibri.method.retrieve.NNretrieval.similarity.LocalSimilarityFunction;

/* This class is applicable to comparison of Double values for GPS coordinates */

public class Thresholdvalue implements LocalSimilarityFunction {

    double thresholdvalue;

    public Thresholdvalue (double Threshold) {
        thresholdvalue = Threshold;
    }

    /*
     * This function returns 1 if the difference between two double values is less than the defined
threshold value,
     * otherwise it returns 0.
     */

    public int compare (double x, double y) {
        if (Math.abs(x - y) > thresholdvalue)
            return 0;
        else
            return 1;
    }

    @Override
    public double compute(Object caseObject, Object queryObject)
```

```
            throws NoApplicableSimilarityFunctionException {
        // TODO Auto-generated method stub
        if ((caseObject == null) || (queryObject == null))
            return 0;
        if (!(caseObject instanceof java.lang.Double))
            throw new jcolibri.exception.NoApplicableSimilarityFunctionException(this.getClass(),
caseObject.getClass());
        if (!(queryObject instanceof java.lang.Double))
            throw new jcolibri.exception.NoApplicableSimilarityFunctionException(this.getClass(),
queryObject.getClass());

        Double d1 = (Double) caseObject;
        Double d2 = (Double) queryObject;
        return (double) compare(d1.doubleValue(), d2.doubleValue());
    }

    @Override
    public boolean isApplicable(Object caseObject, Object queryObject) {
        // TODO Auto-generated method stub
        if ((caseObject == null) && (queryObject == null))
            return true;
        else if (caseObject == null)
            return queryObject instanceof Double;
        else if (queryObject == null)
            return caseObject instanceof Double;
        else
            return (caseObject instanceof Double) && (queryObject instanceof Double);
    }
}
```

8.2.5. ClockInterval.java

```
/*
 * ClockInterval.java
 * implemented by using jCOLIBRI framework
 * @author Irfan Nazir
 * Dated: 2011-11-16
 * Version 1.0
 */
package jcolibri.method.retrieve.NNretrieval.similarity.local;

import jcolibri.exception.NoApplicableSimilarityFunctionException;
import jcolibri.method.retrieve.NNretrieval.similarity.LocalSimilarityFunction;

/*
 * Calculates similarity between two clock hour times inside interval 24.
 * use similarity function similarity(x, y) = 1 - (|x-y|/interval)
 */

public class ClockInterval implements LocalSimilarityFunction{
```

```
    double clockInterval;

    public ClockInterval(double interval)
    {
        clockInterval = interval;
    }

    @Override
    public double compute(Object caseObject, Object queryObject)
            throws NoApplicableSimilarityFunctionException {
        // TODO Auto-generated method stub
        if ((caseObject == null) || (queryObject == null))
            return 0;

        int caseOb = Integer.parseInt((String) caseObject);
        int queryOb = Integer.parseInt((String) queryObject);

        //Number caseOb = (Number) caseObject;
        //Number queryOb = (Number) queryObject;

        //double x = caseOb.doubleValue();
        //double y = queryOb.doubleValue();
        //return 1 - ((double) Math.abs(x - y) / clockInterval);
        return 1 - ((int) Math.abs(caseOb - queryOb) / clockInterval);
    }

    @Override
    public boolean isApplicable(Object caseObject, Object queryObject) {
        // TODO Auto-generated method stub
        return true;
    }
}
```

8.2.6. Databaseconfig.xml

```
<DataBaseConfiguration>
    <HibernateConfigFile>eta/cbr/data/hibernate.cfg.xml</HibernateConfigFile>
    <DescriptionMappingFile>eta/cbr/data/SegmentDescription.hbm.xml</DescriptionMappingFi
    le>
    <DescriptionClassName>eta.cbr.data.SegmentDescription</DescriptionClassName>
    <SolutionMappingFile>eta/cbr/data/SegmentSolution.hbm.xml</SolutionMappingFile>
    <SolutionClassName>eta.cbr.data.SegmentSolution</SolutionClassName>
</DataBaseConfiguration>
```

8.2.7. Hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>

   <session-factory>

      <!-- Database connection settings -->
      <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="connection.url">jdbc:mysql://localhost:3306/etacbrs1</property>
      <property name="connection.username">root</property>
        <property name="connection.password">admin123</property>

      <!-- JDBC connection pool (use the built-in) -->
      <property name="connection.pool_size">1</property>

      <!-- SQL dialect -->
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

      <!-- Enable Hibernate's automatic session context management -->
      <property name="current_session_context_class">thread</property>

      <!-- Disable the second-level cache  -->
      <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>

      <!-- Echo all executed SQL to stdout -->
      <property name="show_sql">true</property>

   </session-factory>
</hibernate-configuration>
```

8.2.8. SegmentDescription.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping default-lazy="false">
<class name="eta.cbr.data.SegmentDescription" table="segmentcasebase">

    <id name="segmentDescriptionId" column="segmentId">
      <generator class="native"/>
    </id>
    <property name="SegmentStartLat" column="segmentStartLat"/>
    <property name="SegmentStartLon" column="segmentStartLon"/>
    <property name="SegmentEndLat" column="segmentEndLat"/>
    <property name="SegmentEndLon" column="segmentEndLon"/>
    <property name="SegmentDistance" column="segmentDistance"/>
    <property name="SegmentDay" column="segmentDay">
        <type name="jcolibri.connector.databaseutils.EnumUserType">
            <param name="enumClassName">eta.cbr.data.SegmentDescription$Day</param>
    </type>
    </property>
    <property name="SegmentTime" column="segmentTime"/>
```

```
    <property name="SegmentCount" column="segmentCount"/>

</class>
</hibernate-mapping>
```

8.2.9. SegmentSolution.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping default-lazy="false">
<class name="eta.cbr.data.SegmentSolution" table="segmentcasebase">

    <id name="segmentSolutionId" column="segmentId">
    </id>
        <property name="SegmentETA" column="segmentETA"/>
</class>
</hibernate-mapping>
```