DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

REAL-TIME INTELLIGENT JOURNEY PLANNER

by

Feriştah DALKILIÇ

June, 2015

İZMİR

REAL-TIME INTELLIGENT JOURNEY PLANNER

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering

by
Ferіştah DALKILIÇ

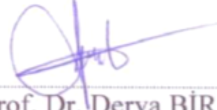
June, 2015
İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"REAL-TIME INTELLIGENT JOURNEY PLANNER"** completed by **FERİŞTAH DALKILIÇ** under supervision of **PROF. DR. ALP KUT** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Alp KUT

Supervisor



Asst. Prof. Dr. Derya BİRANT

Thesis Committee Member

Asst. Prof. Dr. Reyat YILMAZ

Thesis Committee Member

Prof. Dr. Ahmet Faik KAŞLI

Examining Committee Member

Assoc. Prof. Dr. Aybars UĞUR

Examining Committee Member

Prof. Dr. Ayşe OKUR
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to express my utmost gratitude and sincere thanks to my advisor, Prof. Dr. Alp KUT, for his strong support, valuable insights, patience, and encouragement during this study. It was a great privilege for me to work with him.

I extend my thanks to the members of my committee, Assistant Professor Dr. Derya BİRANT and Assistant Professor Dr. Reyat YILMAZ for their contribution to this study and sharing their ideas during development and writing of this thesis.

I would also thank to Sercan KÖKHAN, for sharing her knowledge with me and for her support during the development of the applications.

In addition, I would like to acknowledge the equipment support from the Dokuz Eylul University Scientific Research Projects (Bilimsel Araştırma Projeleri, BAP) Coordination Unit under project number 201190 for this thesis.

I would like to express my special gratitude to my husband Gökhan DALKILIÇ for his support and patience during preparation of this thesis. Finally, I thank my whole family. All of your support and love throughout my life have brought me to this achievement. My little daughter Ela, your presence gives me the strength to work harder.

Feriştah DALKILIÇ

REAL-TIME INTELLIGENT JOURNEY PLANNER

ABSTRACT

Planning a journey by integrating information from diverse sources can be very complicated. A user friendly and informative journey planner system can simplify the journey plan by assisting people in making better use of public transportation.

In scope of this thesis, a service-oriented and inter-model Intelligent Journey Planner System has been developed to assist travelers for planning their journey. Izmir has been selected as the pilot city to operate the system primarily. A WCF web service acting as a path finding engine and a windows service application collecting contemporary transportation data from related sources and transforming it into the GTFS format has been presented. In addition to these services, applications for Mobile Web and Desktop Web portals, Android, iPhone, and Windows Phone platforms have been implemented to provide wide range of usage at anytime and anywhere. Weather forecast, traffic-road condition, and approximate taxi fare services have been intended to be shared with the users. Informing the passengers about important points, social and cultural activities located on travel routes is an innovation performed.

This thesis also introduces the Gradual Path Finding Algorithm that produces alternative journeys according to the user's choice. Modified versions of Dijkstra's algorithm have been used in several stages of the algorithm to reduce search space and run time. In spite of the fact that visited edge counts are increasing over the upper stages of the algorithm, the reduction on search space has been observed as varying percentages from 1.32 to 77.39 in consecutive stages of the algorithm.

Keywords: Google transit feed specification, intelligent transportation systems, journey planning, shortest path problem

GERÇEK ZAMANLI AKILLI SEYAHAT PLANLAMA SİSTEMİ

ÖZ

Farklı kaynaklardan bilgi toplayarak seyahat planlamak çok karmaşık bir işlem olabilmektedir. Kullanıcı dostu bir ara yüze sahip bilgilendirici bir seyahat planlama sistemi, kişilere toplu taşımayı en iyi şekilde kullanabilmeleri konusunda yardımcı olarak seyahat planlama işini basitleştirebilir.

Bu tez kapsamında, toplu taşımayı kullanacak yolculara seyahatlerini planlamaları konusunda yardımcı olmak üzere, servis tabanlı ve çok modlu bir Akıllı Seyahat Planlama Sistemi geliştirilmiştir. Sistem ilk olarak pilot şehir olarak seçilen Izmir için kullanıma sunulmuştur. Yol bulma motoru olarak görev gören bir WCF web servisi ve ilgili kaynaklardan güncel ulaşım verilerini toplayarak bu verileri GTFS formatına dönüştüren bir Windows servis uygulaması geliştirilmiştir. Bu servislere ek olarak, sistemin her zaman ve her yerden kullanılabilmesini sağlamak amacıyla Mobil Web, Web, Android, iPhone, ve Windows Phone platformlarında çalışacak uygulamalar geliştirilmiştir. Hava durumu, trafik ve yol durumu, ortalama taksi tutarı gibi bilgilerin kullanıcılar ile paylaşılması hedeflenmiştir. Yolcuları, seyahat rotası üzerinde yer alan önemli nokta, sosyal ve kültürel aktiviteler hakkında bilgilendirmek bir yeniliktir.

Bu tezde ayrıca, kullanıcının seçimleri doğrultusunda seyahat alternatifleri üreten Kademeli Güzergâh Hesaplama Algoritması tanıtılmıştır. Bu algoritmanın farklı kademelerinde, arama uzayını daraltmak ve çalışma süresini kısaltmak amacıyla Dijkstra algoritması değiştirilerek kullanılmıştır. Kademe ilerledikçe ağ üzerinde gezilen düğüm sayısının artmasına rağmen, arama uzayı yüzde 1,32'den 77,39'a değişen oranlara indirgenmiştir.

Anahtar kelimeler: Google ulaşım besleme tanımlaması, akıllı ulaşım sistemleri, seyahat planlama, en kısa yol problemi

CONTENTS

Page

THESIS EXAMINATION RESULT FORM ii
ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZ v
LIST OF FIGURES ix
LIST OF TABLES xii

CHAPTER ONE – INTRODUCTION 1

1.1 Overview 1
1.2 Izmir Transportation Infrastructure 2
1.3 Motivation 3
1.4 Aim of This Thesis 3
1.5 Thesis Organization 4

CHAPTER TWO – RELATED WORKS 6

2.1 Overview 6
2.2 Graph Theory 6
2.3 Shortest Path Problem 7
2.4 Multi-Modal Routing 9
2.5 Multi-Criteria Routing 10
2.6 Timetable Information 10
2.6.1 Time-Expanded Model 11
2.6.2 Time-Dependent Model 11
2.7 Speed-Up Techniques 12
2.8 Reference Data Models for Public Transport 15
2.8.1 General Transit Feed Specification 17
2.8.2 GTFS-Conversion Application 20

2.8.3 GTFS-realtime 21
2.9 Existing Journey Planner Systems 22

CHAPTER THREE – INTELLIGENT JOURNEY PLANNER SYSTEM 24

3.1 Overview 24
3.2 System Architecture 24
3.3 Database Design 26
3.4 Applications 29
3.4.1 Mobile Applications 30
3.4.2 Web Application 42

CHAPTER FOUR – SERVICES 46

4.1 Overview 46
4.2 Update Service 48
4.2.1 Updating Transportation Information 49
4.2.2 Determination of Neighbor Stops 50
4.2.3 Determination of Accessible Transfer Centers 51
4.2.4 Route Planning Preprocess Between All Transfer Center Pairs 52
4.3 Journey Planner Web Service 53
4.3.1 Service Methods 54
4.3.2 Testing the Web Service 56

CHAPTER FIVE – GRADUAL PATH FINDING ALGORITHM 60

5.1 Overview 60
5.2 Representation of the Transportation Graph 60
5.3 Gradual Path Finding Algorithm 63
5.3.1 Finding Direct Routes 67
5.3.2 Finding Routes Containing One Transfer 67

5.3.3 Finding Routes Containing Two Transfers 69
5.3.4 Finding Routes by Using Transfer Centers 72
5.3.5 Finding Routes Containing N Transfers.. 73
5.3.6 Calculation of the Departure Times 75
5.3.7 Filtering the Alternatives... 76

CHAPTER SIX – EXPERIMENTATIONS 78

6.1 Evaluation Measures 78
6.2 Test Dataset 78
6.3 Search Space 80
6.4 Experimental Results.. 83

CHAPTER SEVEN – CONCLUSION.. 86

7.1 Conclusion... 86
7.2 Future Works 88

REFERENCES 89

LIST OF FIGURES

Page

Figure 2.1 Sample text file from a GTFS feed. 18
Figure 2.2 UML Diagram of the General Transit Feed Specification. 19
Figure 2.3 Generated agency and stop files in GTFS format. 20
Figure 2.4 GTFS validation results for Izmir feed. 20
Figure 3.1 System architecture. 25
Figure 3.2 Three-tier architecture. 26
Figure 3.3 Entity-relationship diagram. 28
Figure 3.3 Entity-relationship diagram (continued) 29
Figure 3.4 Determination of the origin and destination points in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web. 31
Figure 3.5 Determination of the journey parameters in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web. 32
Figure 3.6 Summarized route list in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web. 33
Figure 3.7 Route details in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web. 34
Figure 3.8 Display of the selected route on the map in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web. 35
Figure 3.9 Point of interest lists in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web. 36
Figure 3.10 Event center list in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web. 37
Figure 3.11 Estimated distance and taxi fare in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web. 38
Figure 3.12 Current weather conditions in (a) iOS, (b) Android, (c) Windows Phone. 39
Figure 3.13 Road condition in (a) iOS, (b) Android, (c) Windows Phone. 40
Figure 3.14 Traffic status in (a) iOS, (b) Android, (c) Windows Phone. 41
Figure 3.15 Journey parameter selection page. 42
Figure 3.16 Alternative paths. 43

Figure 3.17 Route details. 43
Figure 3.18 Representation of a path on the map.......... 44
Figure 3.19 Representation of the event centers and POI on the map. 44
Figure 3.20 Nearest places of event list. 45
Figure 3.21 Event list of selected event center. 45
Figure 4.1 IJPS Data Collection and Integration Tool. 48
Figure 4.2 Update Service work flow. 49
Figure 4.3 Distance ranges in determination of neighborhood. 51
Figure 4.4 Transfer centers in Izmir.......... 51
Figure 4.5 Path finding between two transfer centers. 52
Figure 4.6 Development environment of Journey Planner Web Service. 54
Figure 4.7 WCF Test Client window. 56
Figure 4.8 Windows application developed to test Journey Planner Web Service.... 57
Figure 4.9 Display of results gathered from the Journey Planner Web Service. 58
Figure 5.1 Illustration of a transportation graph. 60
Figure 5.2 Izmir transportation graph. 61
Figure 5.3 Classes designed to construct transportation graph. 62
Figure 5.4 Transportation graph with stop and transition objects. 62
Figure 5.5 Pseudo code of Dijkstra's algorithm. 64
Figure 5.6 Stages of Gradual Path Finding Algorithm. 65
Figure 5.7 Classes designed to hold routes. 66
Figure 5.8 Illustration of the routes containing one transfer. 67
Figure 5.9 Pseudo code of the modified Dijkstra's Algorithm for the routes that contain one transfer. 68
Figure 5.10 Illustration of the routes containing two transfers. 70
Figure 5.11 Pseudo code of the modified Dijkstra's Algorithm for the routes that contain two transfers.......... 70
Figure 5.11 Pseudo code of the modified Dijkstra's Algorithm for the routes that contain two transfers (continued). 71
Figure 5.12 Sample path between two stops through one transfer center. 72
Figure 5.13 Sample path between two stops through two transfer centers. 73

Figure 5.14 Pseudo code of the modified Dijkstra's Algorithm for routes containing n transfers. 74
Figure 5.15 Illustration of sample departure time calculation 76
Figure 6.1 Example tuples of the test data. 79
Figure 6.2 First and last stops of the available bus (red), train (green), metro (yellow) and ferry (blue) lines. 79
Figure 6.3 A visualization of the search space for Stage 2. 81
Figure 6.4 A visualization of the search space for Stage 3. 81
Figure 6.5 A visualization of the search space for Stage 4. 82
Figure 6.6 A visualization of the search space for Stage 5. 82
Figure 6.7 Percentages of executed queries for all stages. 84
Figure 6.8 Average runtimes for all stages. 84
Figure 6.9 Average result counts for all stages. 85

LIST OF TABLES

Page

Table 2.1 The shortest path produced by pure Dijkstra's algorithm............................ 9

Table 2.2 Required and optional files of a GTFS feed. 17

Table 4.1 Query types according to the origin and destination parameters. 55

Table 6.1 Visited edge counts for the modified and Pure Dijkstra's algorithms 80

Table 6.2 Average runtime and result counts for stages of the Gradual Path Finding Algorithm. 83

CHAPTER ONE
INTRODUCTION

1.1 Overview

A *journey planner (or trip planner)* is a specialized electronic search engine that finds one or more suggested journeys between an origin and a destination. Origin and destination may be specified as geospatial coordinates, Points of Interest, or stops/stations names. Journey planners have been widely used in the travel industry since the 1970s. Early journey planning engines were typically developed as part of the booking systems. As computing resources became more widely available, journey planning engines were developed to run on personal computers and mobile devices.

A *Road Route Planner* is a journey planner specialized for road network whereas a *Public Transport Journey Planner* is specialized for journeys on Public Transport. *Unimodel Journey Planners* cover a single mode of transport while *Intermodal Journey Planners* cover many transport modes for a combined journey.

We have modeled our system to test for Izmir. Izmir has a population of over 4 million, and each year approximately 200,000 of tourists visit Izmir (Turkish Statistical Institute, 2013). Most travelers in Izmir prefer public transportation because of the increasing traffic congestion and fuel costs. There are four types of public transportation modes available in Izmir, all with different operation schedules. These are bus, train, metro and ferry, and new transportation modes like tram are going to be added in the near future. At present, four different public transportation companies are in operation in Izmir. Route and timetable information of these services are available on their web sites, but there is no information about the connection of other forms of transportation. It is not easy to integrate information from diverse sources to plan a journey. A user friendly and informative journey planner system was needed to assist people in making better use of public transportation. In scope of this thesis, a service-oriented and intermodal Intelligent

Journey Planner System (IJPS) has been developed to assist travelers for planning their journey. IJPS supplies information about urban transportation for domestic and foreign visitors.

Dijkstra's algorithm forms the basis of modern journey planner search algorithms and provides an optimal solution to simple searches. Search engine of the IJPS is designed as a web service and can be used by variable platforms and devices. A five stages path finding algorithm the *Gradual Path Finding Algorithm* has been developed in scope of this thesis. Modified versions of Dijkstra’s algorithm are used in several stages of the algorithm.

IJPS advices alternatives routes, transfer details, departure and arrival times according to user preferred criteria for any origin–destination point. The system provides optimal route choices according to multiple criteria like preferred mode, maximum distance to walk, least changes, shortest travelling time, or lowest fare. IJPS runs on different kinds of platforms to provide wide range of usage at anytime and anywhere. Mobile Web and Desktop Web portals, kiosks, Android, iPhone, and Windows Phone based mobile applications are available in both English and Turkish. Weather forecast, traffic and road condition, approximate taxi fare, activity centers and events on the route are other informative details produced by the system. Despite similar developments in other world cities, Izmir has a complex public transport system, especially in terms of bus transportation. IJPS is a flexible system which can include other public transportation modes and can be adapted to any other city.

1.2 Izmir Transportation Infrastructure

Izmir is Turkey's third largest city and has four existing public transport companies which are listed below:

- ESHOT operates a total of 319 bus lines according to the numbers of June 2014. There are approximately 6700 bus stops in Izmir.
- Izmir Metro Inc. operates the Izmir metro that has approximately 20 km of lines and 17 stations. In the coming years, four different metro lines are expected to be in service.

- IZBAN Inc. operates the rail system consisting of 80 km of lines and 32 stations. Six new stations are going to be added to system in the near future.
- IZDENIZ operates the maritime transport by 8 stations, 22 passenger ships and 3 ferries.

In addition to the existing services, tram is expected to attend public transport in the near future.

1.3 Motivation

The motivation for writing this thesis rests on four pillars:

- There are various alternative sources of transportation and selecting the most appropriate route is getting complicated. A system that assists people in making better use of public transportation is a necessity.
- Route finding problem in a complex multi-model network is one of the most studied areas and still needs to be improved.
- In many countries and cities, journey planning systems are available. But there was no such a service in Izmir.
- In order to create a more social and intellectual community, citizens should be aware of the social activities and events in the city.

1.4 Aim of This Thesis

One of the purposes of this thesis is the making the passengers aware of the travel options available, provide alternative ways that are appropriate for the passengers' choices, and assist them to complete the journey successfully. So that, passengers can minimize the cost of their journeys, and save travel time which is an economic value.

In this study, we wish to reduce dependence on the car and encourage greater use of public transport. Another purpose is to reduce noise and carbon dioxide emissions, and avoid traffic jump and congestion by promoting passengers to public transport.

One of the other intentions in this thesis is to inform the passengers about important points (hospital, pharmacy, public institutions, etc.), car parks and events (concerts, exhibitions, etc.) located on travel routes. Thus, passengers will be promoted of the social and cultural activities carried out in the city. Also, another objective of this study is to increase the use of technology anywhere anytime in the public by especially supporting mobile platforms.

The main contribution of this thesis is to introduce a new route planning algorithm named “Gradual Path Finding Algorithm”. We have provided a web service that uses the route planning algorithm and we have developed web and mobile applications to serve web service to the end-users. We aim to produce results in an acceptable response time for the users.

1.5 Thesis Organization

The thesis consists of seven chapters. In this chapter, we have stated what we are trying to accomplish, what is our motivation and aim of the thesis, and our contribution to the field. The rest of the thesis is organized as follows.

Chapter 2 presents a literature survey on shortest path problems, timetable information and the common approaches that model timetable information, speed-up techniques, reference data models and examples of existing journey planner systems.

Chapter 3 explains system architecture and database design of the IJPS. Developed applications which include the web application, mobile web application and native applications for mobile operating systems are explained by giving some sample screens.

Chapter 4 presents two main services - The Update Service and The Journey Planner Web Service and other auxiliary services have been developed in scope of this thesis.

In chapter 5, we have mentioned about our route planning algorithm named "Gradual Path Finding Algorithm" and its implementation details.

The experimental results and performance overview of IJPS are given in Chapter 6.

Finally, we have presented the conclusions and future directions of the thesis in Chapter 7.

CHAPTER TWO
RELATED WORKS

2.1 Overview

This chapter starts with the graph theory explanation for best understanding of the following sections. In section 2.3, shortest path problems and the proposed solutions to these problems are presented. Multi-model routing and multi-criteria routing are described in the next sections.

Timetable information and the common approaches that model timetable information in public transportation systems are explained and existing speed-up techniques aiming faster query times are described.

Reference data models representing the stops, routes and timetables of the public transportation are introduced in Section 2.8. GTFS and GTFS-realtime specification are described in more detail. Finally, this chapter ends with some examples of existing journey planner systems.

2.2 Graph Theory

A *graph* is generally referred to as *G*, and consists of a set of *nodes (vertices) V* and a set of *edges (arcs)* $E \subseteq V \times V$. The number of nodes is denoted by *n* and the number of edges by *m* respectively.

A graph can be defined as undirected, directed, or mixed. In directed graphs, an edge has the form (u, v), where $u, v \in V$ are ordered distinct nodes whereas in undirected graph, the direction of the edges is not important.

We call two nodes $u, v \in V$ *connected*, if there exists a path from u to v or v to u. If this is true for all pairs of nodes $u, v \in V$, we call the whole graph connected.

A *path* P is a sequence $v_1, e_1, v_2, e_2, \ldots, e_j, v_{(j+1)}$ of nodes and edges such that for

every i ($1 \leq i \leq j$) the edge e_i connects v_i and $v_{(i+1)}$. A path is called as *shortest path* from the source *s* to the target *t*, if the path starts with the node s and ends up with the node t and has the smallest length among all the paths from s to t at time τ.

|P| denotes the number of edges along the path. The *length* of a path P is the sum of its edge weights along the path and is denoted by

$$len(P) = \sum_{i=1}^{j-1} f(v_i + v_{i+1}) \tag{2.1}$$

The *distance* between two nodes u, v ∈ V, written by dist(u, v, τ), for a given departure time τ, is the minimal length of all the paths P from u to v. There might be more than one minimal path from *u* to *v*. A minimal path P between two nodes u and v at time τ is called shortest path from u to v.

2.3 Shortest Path Problem

Computing shortest paths in a graph is used in many real-world applications like route planning, timetable information, or scheduling in transportation networks. Shortest path algorithms differ depending on source and target node. Single-Source Shortest Path Problem finds shortest path from the given source node to all other nodes. All-Pairs Shortest Path Problem determines the shortest paths between all pairs of vertices. The K Shortest Path Problem which is studied in this project finds best K paths in order of increasing cost.

Single-Source Shortest Path Problem finds shortest paths from a source node s to T = V, where V is the set of all nodes in the graph. Dijkstra's algorithm solves the single-source shortest path problem (Dijkstra, 1959). If edge weights are negative, Bellman–Ford algorithm solves the problem (Bellman, 1958). A* search algorithm solves for single pair shortest path using heuristics to try to speed up the search (Hart, Nilsson, & Raphael, 1968). It is an extension of Dijkstra's algorithm.

Many-To-Many Shortest Path Problem is a generalization of the Shortest Path

Problem. Instead of one source node *s* and one target node *t*, we are given a set of source nodes $S \subseteq V$ and a set of target nodes $T \subseteq V$. We then ask for a shortest path $P_{s,t}$ for each pair $(s, t) \in S \times T$. In multi-modal routing, the Earliest Arrival Problem will actually transform to this version of the problem.

All-Pairs Shortest Path Problem finds shortest paths between every pair of nodes where both S and T are the complete node set V of the graph. Floyd–Warshall algorithm solves all pair's shortest paths (Floyd, 1962; Warshall 1962). The algorithm finds the lengths of the shortest paths between all pairs of nodes, but it does not return details of the paths. Johnson's algorithm solves all pairs shortest paths problem between all pairs of nodes in a sparse directed graph (Johnson, 1977). It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist. Computing shortest paths for all pairs of nodes tends to be very expensive both regarding memory consumption and execution time. Hence, this is not a viable approach.

The K-Shortest Path Problem is a generalization of the Shortest Path Problem. The algorithm not only finds the shortest path, but also K – 1 other paths in order of increasing cost. Yen's algorithm is one of the fundamental works dealing with K-Shortest Path Problem (Yen, 1971). It uses the Dijkstra's algorithm or any other shortest path algorithm to find the best path, and then proceeds to find K – 1 deviations of the best path. Hundreds of researches which are offering new solutions exist (Hoffman & Pavley, 1959; Lawler, 1972; Katoh, Ibaraki, & Mine, 1982). Also some comparative studies have been made (Brander & Sinclair, 1995; Hadjiconstantinou & Christofides, 1999; Martins & Pascoal, 2003).

Each of these versions of the Shortest Path Problem can be solved by Dijkstra's algorithm, possibly requiring repeated execution. The algorithm can be augmented to work with time-dependent networks and even further to multi-modal networks. In this study, Earliest Arrival Problem is reduced to the Many-To-Many Shortest Path Problem. Modified versions of Dijkstra's algorithm are applied to multi-model transit graph repeatedly to obtain k-shortest-path in ascending transfer count order.

2.4 Multi-Modal Routing

In multi-model routing, unlike unimodel routing there are more than one transportation modes for traveling thus more than one transportation network. After combining the graphs, the resulting network is a multi-modal network. While planning routes in such a combined network, some constraints as switching the mode of transportation frequently or unacceptable transfer counts must be considered. For instance, Table 2.1 shows the result path produced by pure Dijkstra's algorithm for a query from origin stop '10036-Konak' to destination stop '40120-Tınaztepe Kampüs Son Durak'. The path consists of 23 stops and 12 different lines, thus 11 transfers are required to complete the journey. Producing such an undesired path should be avoided.

Table 2.1 The shortest path produced by pure Dijkstra's algorithm

Step	Stop Id	Stop Name	Line No	Line Name
1	10036	Konak	72	İşçievleri-Konak
2	10023	Bahribaba Alt	7	Sahilevleri-Konak
3	10015	Bahribaba	152	Gaziemir-Konak
4	10241	Kız Yurdu	43	Yapıcıoğlu-Konak
5	12061	Eşref Paşa	23	Uzundere-Konak
6	12063	Yağhaneler	90	Gaziemir- Halkapınar Metro
7	10617	Elka	870	Hıfzıssıhha- Tınaztepe
8	11873	Köprü	870	Hıfzıssıhha- Tınaztepe
9	40001	Nato	576	Tınaztepe- Halkapınar Metro
10	41177	Şirinyer Aktarma	36	Buca-Gümrük
11	40199	Koşu Yolu	36	Buca-Gümrük
12	40201	İstasyon	36	Buca-Gümrük
13	40207	Vali Rahmi Bey	36	Buca-Gümrük
14	40209	Şehitler Parkı	36	Buca-Gümrük
15	40733	Buca Devlet Hastanesi	36	Buca-Gümrük
16	40735	Çevik Bir	36	Buca-Gümrük
17	40737	Buca Sağlık Ocağı	36	Buca-Gümrük
18	40739	Buca Üçkuyular Meydan	604	Sebze Hali-Ayakkabıcılar Sitesi
19	40079	Hasan Ağa Bahçesi	176	Ufuk Mahallesi- Demirciköy
20	40067	Eski Mezarlık	176	Ufuk Mahallesi- Demirciköy
21	40069	Fabrika	176	Ufuk Mahallesi- Demirciköy
22	40071	Begos	671	Narlıdere- Tınaztepe
23	40120	Tınaztepe Kampüs Son Durak		

2.5 Multi-Criteria Routing

When we are planning a journey from s to t, the first goal coming to mind, is minimizing the travel time. Sometimes, the fastest route is not the best route in transportation networks. A passenger may prefer a travel with longer time or longer distance to a travel with less transfers or cheaper cost. Combination of several criteria is even more complicated. Using multiple criteria for optimization in route planning is called multi-criteria search. None of the high-performance approaches developed in the last years can be applied on a multi-criteria routing easily (Muller–Hannemann & Schnee 2007; Pyrga, Schulz, Wagner, & Zaroliagis 2008; Disser, Müller-Hannemann, & Schnee 2008). In this study, a gradual multi-criteria search approach is developed by concentrating to find k-shortest-paths ordered by increasing number of transfers.

2.6 Timetable Information

A timetable consists of data concerning: stops or stations, vehicles (busses, trains, or ferries, etc), transfer stations, departure and arrival times of vehicles at stops stations, and traffic days. More formally, we are given a set of vehicles A, a set of stops B, and a set of elementary connections C whose elements c are 5-tuples of the form $c = (A, S_1, S_2, t_d, t_a)$. An elementary connection c means a vehicle A leaves stop S_1 at time t_d, and the immediately next stop of vehicle A is stop S_2 at time t_a.

Static timetable data describes the state a transportation network is supposed to be in any time t while real-time transit data describes the state a transportation network is currently in. The current real-time state of a transit network can be described with vehicle’s actual GPS positions.

A real-time journey planner produces routes considering vehicle delays, cancellation or route changes because of an accident or construction. Real-time data is rarely available and almost never network-complete. There are a few transportation agencies that provide GPS positions of their vehicles.

There are two common approaches that model timetable information in public transportation systems as shortest-path problems in weighted graphs. These are time-expanded and time-dependent models. Both the time-expanded and the time-dependent models transform static schedule data into a directed graph G = (V, E).

2.6.1 Time-Expanded Model

In the time-expanded approach, every event at a station, e.g., the departure of a train is modeled as a node in the graph. The simplified version of the earliest arrival problem has been studied on time-expanded graphs.

Schulz, Wagner, and Weihe (2000) explicitly use the time-expanded approach to model a simplified version of the earliest arrival problem as a shortest path problem in a static graph, and solve the problem optimally. An extension of the time-expanded approach incorporating train transfers and an extensive experimental study focused on multi-criteria problems is presented by Muller-Hannemann and Weihe (2001). Muller-Hannemann, Schnee and Weihe (2002) focus on more realistic and complex real-world scenarios for timetable information, in particular with respect to space limitations. Bi-criteria problems are presented and an experimental comparison with the time-dependent approach is conducted by Pyrga, Schulz, Wagner and Zaroliagis (2004, 2008). Multi-criteria optimization in the time-expanded graph by a labeling approach is extensively investigated by Muller–Hannemann and Schnee (2007).

2.6.2 Time-Dependent Model

In the time-dependent approach, each $v \in V$ models a station and each $e = (u, v) \in E$; $u, v \in V$ models possible non-stop connections between two nodes. The lengths on the edges are modeled by introducing special cost functions that respect the travel and waiting time and assigned “on-the-fly”.

Orda and Rom (1990, 1991) thoroughly investigated the complexity of time-dependent shortest path problems and gave efficient algorithms for special cases. Nachtigal (1995) used the time-dependent approach with a label correcting method to

calculate the transit function for all starting times with one path search procedure.

Comparing the time-expanded and time-dependent approach by Brodal and Jacob (2004), the time-dependent approach is better than the time-expanded one when the simplified version of the earliest-arrival problem is considered. Also, Pyrga et al. (2004, 2008) have discussed time-expanded and time-dependent models for several kinds of single-criterion and bi-criteria optimization problems on timetable information systems. They have shown that, for the simplified earliest-arrival problem, the time-dependent approach is clearly superior to the time-expanded approach.

2.7 Speed-Up Techniques

Speedup techniques have made enormous progress in the last years. Several speed-up techniques have been developed aiming faster query times for many realistic data sets. These techniques pre-compute and store additional information on shortest paths, which is used in the on-line phase to reduce the running time for solving a shortest-path query. Delling, Sanders, Schultes, and Wagner (2009) have given an overview of these techniques. In general, two approaches exist on how to accelerate s-t queries: goal-directed and hierarchical approaches.

Bidirectional Search, is a natural approach that a second search is started backwards, from the target to the source. The algorithm terminates as soon as some node has been settled from both directions. Experiments of Pohl (1969) showed that search space can be reduced by a factor of 2. Many advanced speed-up techniques use bidirectional search.

A commonly used speedup approach is the *Goal-Directed Search*, also referred to as the A∗ algorithm or the method of potentials, introduced originally by Hart, Nilsson and Raphael (1968). In goal-directed search, the given edge weights are modified to favor edges leading towards the target node. With graphs from timetable information, a speed-up in running time of a factor of roughly 1.5 is reported by Schulz, Wagner, and Weihe (2000).

The *ALT algorithm (A* with Landmarks*) is a combination of A*-search with the improvement of using Landmarks and the Triangle inequality (Goldberg & Harrelson, 2005; Goldberg, & Werneck, 2005). A small number of nodes, called landmarks are selected and distances between each landmark λ and each node v, $d(v, \lambda)$ and $d(\lambda, v)$ are pre-computed. For nodes s and t, the triangle inequality yields for each landmark λ two lower bounds $d(\lambda, t)-d(\lambda, s) \leq d(s, t)$ and $d(s, \lambda)-d(t, \lambda) \leq d(s, t)$. The maximum of these lower bounds is used during an A* search.

The Arc-Flag approach is another goal-directed speed-up technique and a generalization of a partition-based arc labeling approach. The basic idea of the arc-flag method using a simple rectangular geographic partition has been suggested by Lauther (2004). The arc-flag approach divides the graph into regions and gathers information for each arc on whether this arc is on a shortest path in a given region. For each arc, this information is stored in a vector. The vector contains a flag for each region of the graph, indicating whether this arc is on a shortest path in that particular region. Arc-flags are used in the Dijkstra computation to avoid exploring unnecessary paths and achieve an average speed-up of 64 on a typical European road map.

Another speed-up technique, the *Hierarchical Method* requires a preprocessing step at which the input graph G = (V, E) is enriched with additional edges representing shortest paths between certain nodes. The additional edges can be seen as “bridges” or “short-cuts” for Dijkstra’s algorithm. Mainly two methods have been developed to create such a hierarchy, the *Multi-Level Approach* (Schulz, Wagner & Zaroliagis, 2002; Holzer, 2003; Holzer, Schulz & Wagner, 2006; Delling, Holzer, Müller, Schulz & Wagner, 2006) and *Highway Hierarchies* (Sanders & Schultes, 2005, 2006). In Multi-Level Approach, there are three different types of edges being added to the graph: upward edges, going from a node that is not selected at one level to the node selected at that level, downward edges, going from selected to non-selected nodes, and level edges, passing between selected nodes at one level. The weight of such an edge is assigned as the length of a shortest path between the end-nodes. Depending on the given query, only a small fraction of these edges has to be

considered to find a shortest path. Using this technique, speed-up factors of more than 3.5 were observed for road map and public-transport graphs (Holzer, 2003). Timetable information queries could be improved by a factor of 11 (Schulz, Wagner, & Zaroliagis, 2002). These techniques have been combined in many studies (Schulz, Wagner, & Weihe, 2000; Holzer, Schulz, Wagner, & Willhalm, 2006; Bauer et al., 2010).

All developed techniques so far work only either in road or railway networks. There are fewer studies in multi-modal route planning. Mendelzon and Wood (1995) have introduced the *Label Constrained Shortest Paths Problem (LCSPP)* that restricts modal transfers. In this approach, edges are labeled, and the sequence of edge labels must be element of a formal language for any feasible path. Barrett et al. (2008) have conducted an experimental study of this approach, including basic goal-directed techniques.

Access-Node Routing (ANR) is an efficient multi-modal speedup technique that has been proposed by Delling, Pajor, and Wagner (2009). It skips the road network during queries by pre-computing distances from every road node to all its relevant access points of the public transportation network. It has the fastest query times of all the previous multi-modal techniques which are in the order of milliseconds. However, the preprocessing phase predetermines the modal constraints that can be used for queries. Also, it cannot compute short-range queries and requires a separate algorithm to handle them correctly.

Kirchler, Liberti, Pajor, and Calvo (2011) have introduced another approach called *SDALT* that adapts ALT by pre-computing different node potentials depending on the mode of transport. It has fast preprocessing, but both preprocessing space and query times are high, and it also cannot handle arbitrary modal restrictions as query input.

Dibbelt, Pajor, and Wagner (2012) have developed *User-Constrained Contraction Hierarchies* (*UCCH*) that is a multi-modal speedup technique based on Contraction

Hierarchies. In this technique, preprocessing works by heuristically ordering the nodes of the graph by an importance value.

2.8 Reference Data Models for Public Transport

The development of Journey Planning engines has showed a rapid progress by the development of the data standards such as *TRANSMODEL* and *General Transit Feed Specification (GTFS)* for representing the stops, routes and timetables of the network.

TRANSMODEL is the Reference Data Model for Public Transport including the database schema for network description, versioning, vehicle-driver scheduling, personnel disposition, operations monitoring and control, passenger information, fare collection and management (The European Reference Data Model for Public Transport, 2001). It has been fundamental to the development of a number of data models and European Standards including:

- *TransXChange* standard in the UK for bus timetables,
- *The IFOPT (Identification of Fixed Objects in Public Transport)* standard for identifying fixed, transport-related objects,
- *The National Public Tansport Access Nodes (NaPTAN)* database is a UK nationwide system for uniquely identifying all the points of access to public transport in the UK,
- *Service Interface for Real Time Information (SIRI)* is an XML protocol to allow distributed computers to exchange real-time information about public transport services and vehicles.

During the last years, the GTFS has become the most popular format to describe static schedule data of transit networks (General Transit Feed Specification Reference, 2007). GTFS was developed by Google and Portland TriMet transit agency in 2005, and originally known as the Google Transit Feed Specification. Google opened the feed for general use in 2007 and the GTFS format name was changed to the General Transit Feed Specification to represent its use in many different applications outside of Google products in 2010.

Both official and user-generated feeds are available for many transit agencies around the world. According to the data from the GTFS Data Exchange, there are 909 transit agencies providing GTFS Data by the numbers of June, 2015.

GTFS models schedules, provides polylines ("shapes") for single or multiple trips, integrates multiple trips into a single route and holds many other attributes like wheelchair-accessibility, route colors or fare information.

The data models use different terminology when defining the same objects. Differences between the terminologies can be a source of confusion when converting from one format to another. Open Transit group proposes a transport vocabulary that consists of the following ingredient (The Open Transport Vocabulary, 2014).

- stop_point: A stop_point is the location where a vehicle can stop.
- stop_area: A stop_area is a collection of stop_points. Generally there are at least two stop_points per stop_area, one per direction of a line.
- link: This object links two stop_points together (named origin and destination). It is the walkable part of a journey.
- journey_pattern: A journey pattern is an ordered list of stop_points. Two vehicles that serve exactly the same stop_points in exactly the same order belong to the same journey_pattern.
- journey: A journey is a single run of a vehicle along a journey_pattern.
- route: A route is a collection of journey_patterns that match the same commercial direction.
- line: A line is a collection of routes.
- stop_time: A stop_time represents the time when a vehicle is planned to arrive and to leave a stop_point.
- vehicle: A vehicle is an object which can take one or more people from one place to another.
- agency: An agency maintains one or several modes for certain areas.
- mode: A mode is a type of transport.

2.8.1 General Transit Feed Specification

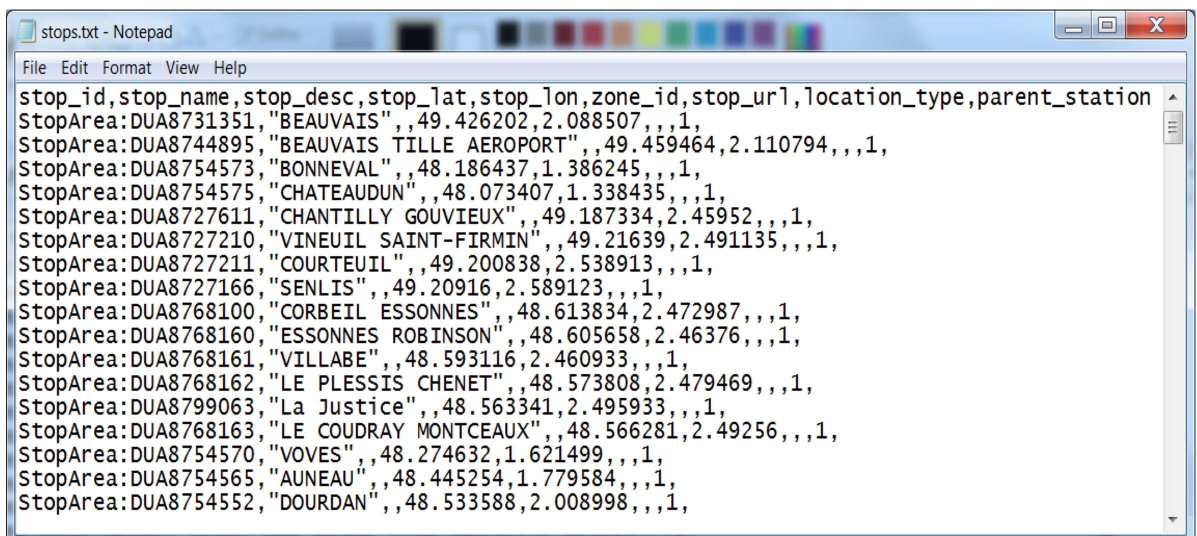
A GTFS feed consists of six required and seven optional CSV-files. Table 2.2 shows a complete list of all files cited from the GTFS reference (General Transit Feed Specification Reference, 2007). All files in a GTFS are saved as comma-delimited text. The first line of each file contains field names. Each subsection of the Field Definitions section corresponds to one of the files in a transit feed and lists the field names used in that file.

Table 2.2 Required and optional files of a GTFS feed.

File Name	Required	Defines
agency.txt	√	One or more transit agencies that provide the data in this feed.
stops.txt	√	Individual locations where vehicles pick up or drop off passengers.
routes.txt	√	Transit routes. A route is a group of trips that are displayed to riders as a single service.
trips.txt	√	Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
stop_times.txt	√	The exact station sequence for each trip. Each station has an arrival and a departure time.
calendar.txt	√	Holds weekly service times referenced by trips.txt.
calendar_dates.txt		Exceptions for the service IDs defined in the calendar.txt file.
fare_attributes.txt		Fare information for a transit organization's routes.
fare_rules.txt		Rules for applying fare information for a transit organization's routes.
shapes.txt		Representation of geographical polylines that describe the exact route a vehicle takes.
frequencies.txt		Headway (time between trips) for routes with variable frequency of service.
transfers.txt		Rules for making connections at transfer points between routes.
feed_info.txt		Additional information about the feed itself, including publisher, version, and expiration information.

In a GTFS feed all field names are case-sensitive. Field values cannot contain tabs, carriage returns or new lines. Field values that contain quotation marks or commas are enclosed within quotation marks. In addition, each quotation mark in the field value is preceded with a quotation mark. Field values cannot contain HTML tags, comments or escape sequences. Each line ends with a CRLF or LF line break character.

A GTFS feed viewed in a file explorer is shown in Figure 2.1 with the text contents of a stops.txt file supplied by the Transilien transport agency operating in France.




```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url,location_type,parent_station
StopArea:DUA8731351,"BEAUVAIS",,49.426202,2.088507,,,1,
StopArea:DUA8744895,"BEAUVAIS TILLE AEROPORT",,49.459464,2.110794,,,1,
StopArea:DUA8754573,"BONNEVAL",,48.186437,1.386245,,,1,
StopArea:DUA8754575,"CHATEAUDUN",,48.073407,1.338435,,,1,
StopArea:DUA8727611,"CHANTILLY GOUVIEUX",,49.187334,2.45952,,,1,
StopArea:DUA8727210,"VINEUIL SAINT-FIRMIN",,49.21639,2.491135,,,1,
StopArea:DUA8727211,"COURTEUIL",,49.200838,2.538913,,,1,
StopArea:DUA8727166,"SENLIS",,49.20916,2.589123,,,1,
StopArea:DUA8768100,"CORBEIL ESSONNES",,48.613834,2.472987,,,1,
StopArea:DUA8768160,"ESSONNES ROBINSON",,48.605658,2.46376,,,1,
StopArea:DUA8768161,"VILLABE",,48.593116,2.460933,,,1,
StopArea:DUA8768162,"LE PLESSIS CHENET",,48.573808,2.479469,,,1,
StopArea:DUA8799063,"La Justice",,48.563341,2.495933,,,1,
StopArea:DUA8768163,"LE COUDRAY MONTCEAUX",,48.566281,2.49256,,,1,
StopArea:DUA8754570,"VOVES",,48.274632,1.621499,,,1,
StopArea:DUA8754565,"AUNEAU",,48.445254,1.779584,,,1,
StopArea:DUA8754552,"DOURDAN",,48.533588,2.008998,,,1,
```

Figure 2.1 Sample text file from a GTFS feed.

Figure 2.2 gives a complete UML diagram describing the relationship between files and their attributes of the General Transit Feed Specification (UML Diagram of the General Transit Feed Specification, 2007).

After an agency creates a GTFS feed and shares it with the public, it is able to be accessed by many different types of applications such as trip planning, timetable creation, data visualization, planning analysis, that are based on GTFS data. An overview of these different types of applications is provided by Antrim and Barbeau (2013).

Standing on a common data format makes an application able to work in all transit systems for which open transit data has been released and the common data is available to any developer to use. In scope of this thesis, an application has been developed to transform the Izmir transportation data into GTFS format and to load any GTFS feed into our database. This application will be explained in the next section by giving a sample GTFS feed which is generated for Izmir. Also, validation results will be mentioned to demonstrate accuracy of the generated feeds.

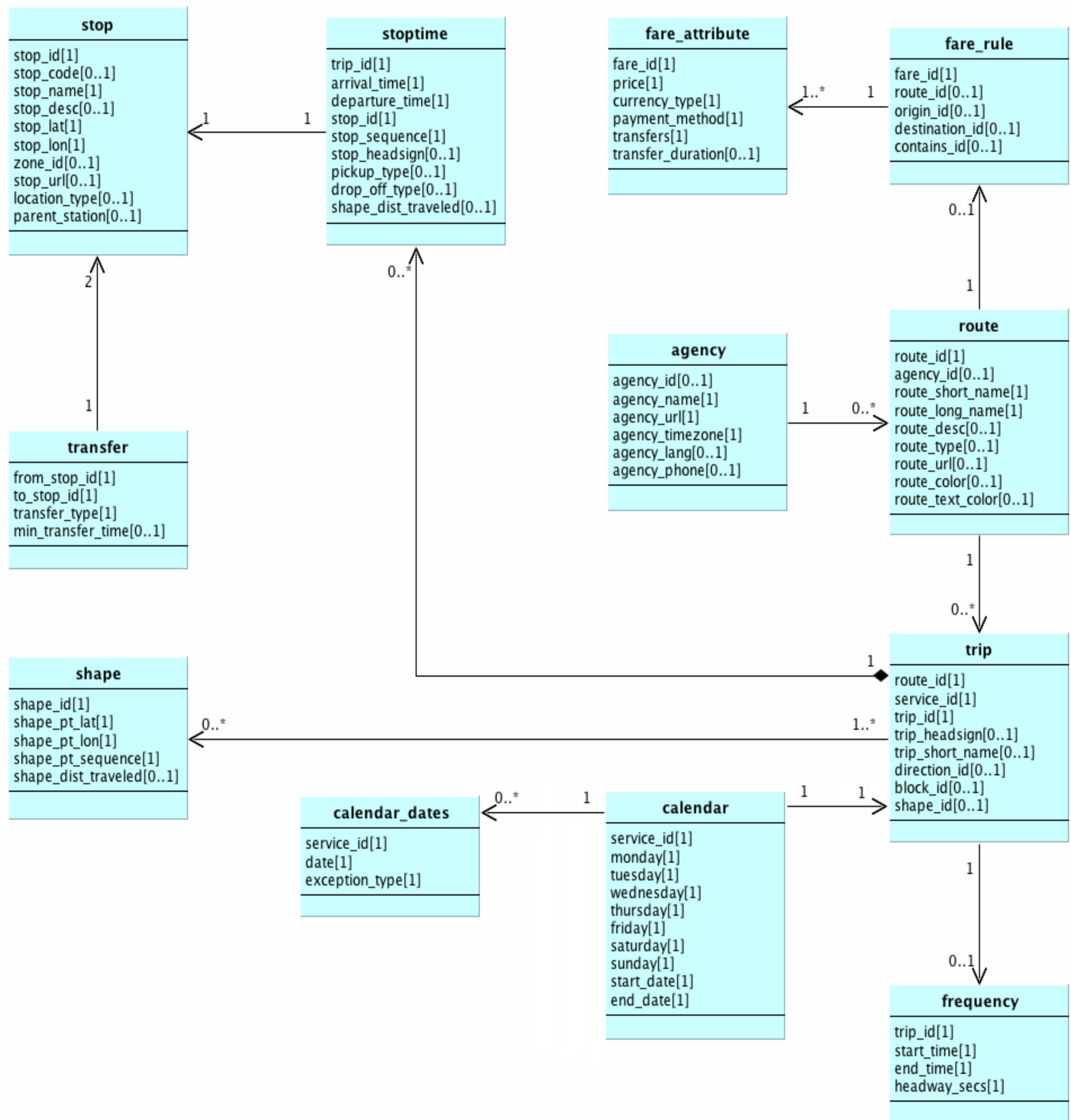


Figure 2.2 UML Diagram of the General Transit Feed Specification.

2.8.2 GTFS-Conversion Application

In scope of this thesis, an application has been developed to transform the Izmir transportation data into GTFS format and to load any GTFS feed into our database. Six required feeds; agency, stops, routes, trips, stop_times, and calendar have been generated for Izmir. Figure 2.3 shows the agency.txt which has been generated by our application.

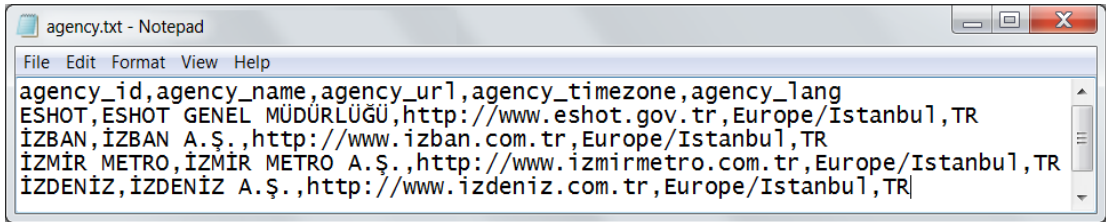


Figure 2.3 Generated agency and stop files in GTFS format.

There is an open source tool available for testing feeds in the GTFS format to verify whether the feed data files match the specifications. We used the *feedvalidator* tool to ensure that our conversion task completed successfully. GTFS validation results for Izmir feeds are given in Figure 2.4.

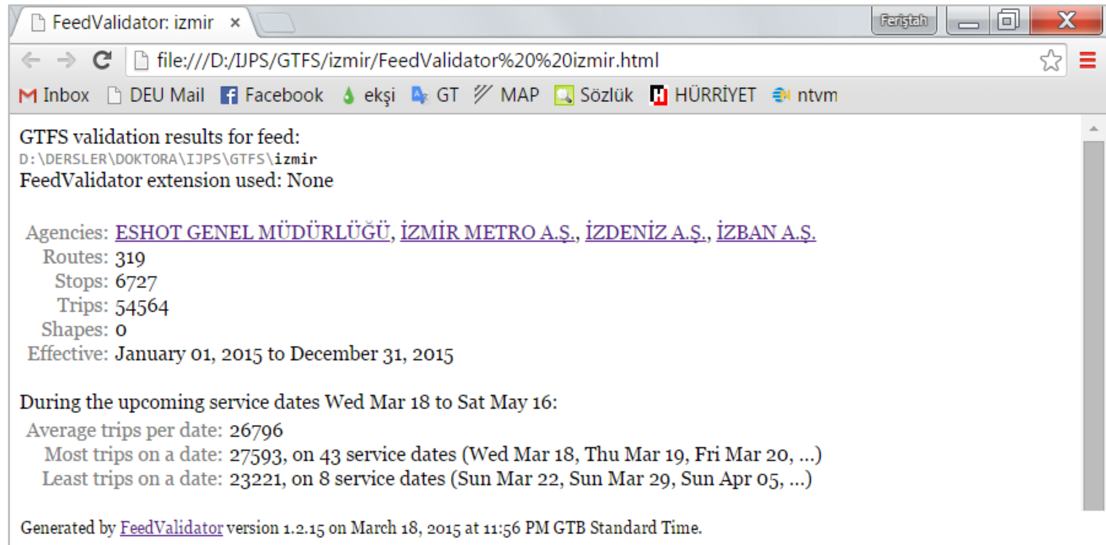


Figure 2.4 GTFS validation results for Izmir feed.

There is no validation error, but some warnings exist. These regular warnings can be exemplified as followings:

- Stop Too Far From Parent Station: Konak (ID 10037) is too far from its parent station Konak (Bahribaba) (ID TRANSFER_CENTER_12) : 127.97 meters.
- Stops Too Close: The stops "Şaşal Köprü" (ID 11105) and "Şaşal Köprü" (ID 11110) are 0.00m apart and probably represent the same location.
- Too Fast Travel: High speed travel detected in trip T13-204-1100-1: Serbest Bölge 2 to Otogar. 12507 meters in 420 seconds (107 km/h).

2.8.3 GTFS-realtime

Since August 2011, the GTFS specification is being extended by a real-time transit data feed (GTFS-realtime Reference, 2011). GTFS-realtime currently provides support for three types of information:

- Trip updates: delays, cancellations, changed routes,
- Service alerts: stop moved, unforeseen events affecting a station, route or the entire network,
- Vehicle positions: information about the vehicles including location and congestion level.

Updates of each type are provided in a separate feed. Feeds are served via HTTP and updated frequently. Trip updates represent fluctuations in the timetable. These updates would give a predicted arrival or departure for stops along the route. Trip updates can also provide more complex scenarios where trips are canceled, added to the schedule or even re-routed.

Service alerts represent higher level problems with a particular entity and are generally in the form of a textual description of the disruption. They could represent problems with stations, lines and the whole network. A service alert usually consists of some text which describes the problem and URLs for more information.

Vehicle position represents a few basic pieces of information about a particular vehicle on the network. Most important are the latitude and longitude the vehicle is at and the other fields bearing, odometer and speed are optional.

2.9 Existing Journey Planner Systems

Many uni-model journey planner systems are available all over the world today. The main focus of this study, multi-model journey planners, can be exemplified by the following systems.

OpenTripPlanner (OTP) is an open source platform for multi-modal and multi-agency journey planning. It follows a client-server model, providing several map-based web interfaces as well as a REST API for use by third-party applications (Open Trip Planner, 2009). OTP relies on open data standards including GTFS for transit and OpenStreetMap for street networks. OTP deployments now exist around the world and OTP is also the routing engine behind several popular smartphone applications.

Google Transit is a public transportation planning tool that combines the latest agency data on Google Maps (Google Transit, 2011). Live Transit Updates is a service providing real-time transit updates to users of Google Maps and Google Maps for mobile. These updates include live departure and arrival times to transit stations, as well as service alerts.

Bristol City Council is a good example of a multi-modal information service, which supports modes like walk, cycle, bus, rail and drive. User can right click on the interactive map to select start and end points, and use alert system to find out if journey could be delayed. *JourneyOn* is being served by Brighton & Hove City Council. This planner compares rail, buses, driving, walking and cycling. *Traveline* is another service that provides journey planning information about public transport services as buses, coaches, trains, ferries, trams, metro and underground throughout England, Wales and Scotland. *Transport Direct* was built upon Traveline for Great Britain and stands out as an example of integrated multi-modal information provision at national level. This service allows users to save their favorite journeys and travel preferences. The service also provides for mobile Internet access, WAP and SMS. *TfL Journey Planner* is another important multi-modal information service for London. This service includes walk, bus and tube journey stages. Journey planning,

travel news, timetables and color maps are available via mobile platforms with real-time travel alerts.

The Public Transport Enquiry System (*PTES*) for Hong Kong is a multi-model journey planner, providing bilingual (English and Chinese) information in the form of interactive maps and text, as well as real-time derivation of optimal travelling routes for users in terms of multiple criteria. *trafiken.nu* compares public transport, car, bike, walk and combinations for each individual journey search in Stockholm. OV9292 has provided travel information on public transport in Netherlands for almost 20 years. *ResRobot* supplies information for all types of transport covering the whole Sweden. *Journey.fi* for planning trips across Finland and *reittiopas.fi* for the Finnish capital and surrounding areas are the journey planners using IPJ by Logica.

Metropolitan Atlanta Rapid Transit Authority (MARTA) presents a journey planner called Five Points for their train and bus services. Deutsche Bahn is the German national railway company using the *DB Bahn* - journey planner of Hacon. *Public Transport Victoria* was established in April 2012 with the aim of improving public transport in Victoria. *Transport Info* provides public transport information to plan an efficient and successful journey in New South Wales.

At the beginning of this study, there was no available journey planner system for Izmir. Two different systems have been released by the current year. The first one *Trafi* is servicing in five countries including Turkey, Lithuania, Latvia, Estonia and Brazil. Istanbul, Ankara, Izmir and Bursa are the Turkish cities which are served by Trafi. The other system *Buradan Oraya* is in use for Istanbul, Ankara, and Izmir. These two systems have Android, iOS, and Web applications. As seen in the examples, journey planner systems are actively in use in many countries of the world. New applications are becoming available day by day.

CHAPTER THREE
INTELLIGENT JOURNEY PLANNER SYSTEM

3.1 Overview

This chapter explains system architecture of the developed system: Intelligent Journey Planner System (IJPS). Database design will also be handled in this section. Finally, developed applications including the web application, mobile web application and native applications for mobile operating systems will be explained by giving some sample screens.

3.2 System Architecture

IJPS stands on a service oriented architecture. The Journey Planner Web Service runs on a web server acts as a routing engine for clients. This web service can be thought as the core of the whole system. There are also some auxiliary services to provide necessary on-the-fly data as shown in Figure 3.1.

Another system component Database Server can serve as Oracle server or MSSQL server. Daily actual data are collected and stored in this server by Update Service. Web, kiosk and mobile web applications are published on another Web Server.

Web Client was developed to be compatible with all common web browsers. Kiosk clients are designed to locate at strategic location for passengers like transfer centers. Their interfaces are the modified version of the web client interfaces to facilitate the use of touch.

Mobile clients are operating native applications for Android, iPhone, and Windows Mobile platforms. These applications are designed suitable with both smart phones and tablets. Mobile web client application was formed especially for other mobile platforms like Blackberry, Symbian etc. which don’t have native applications.

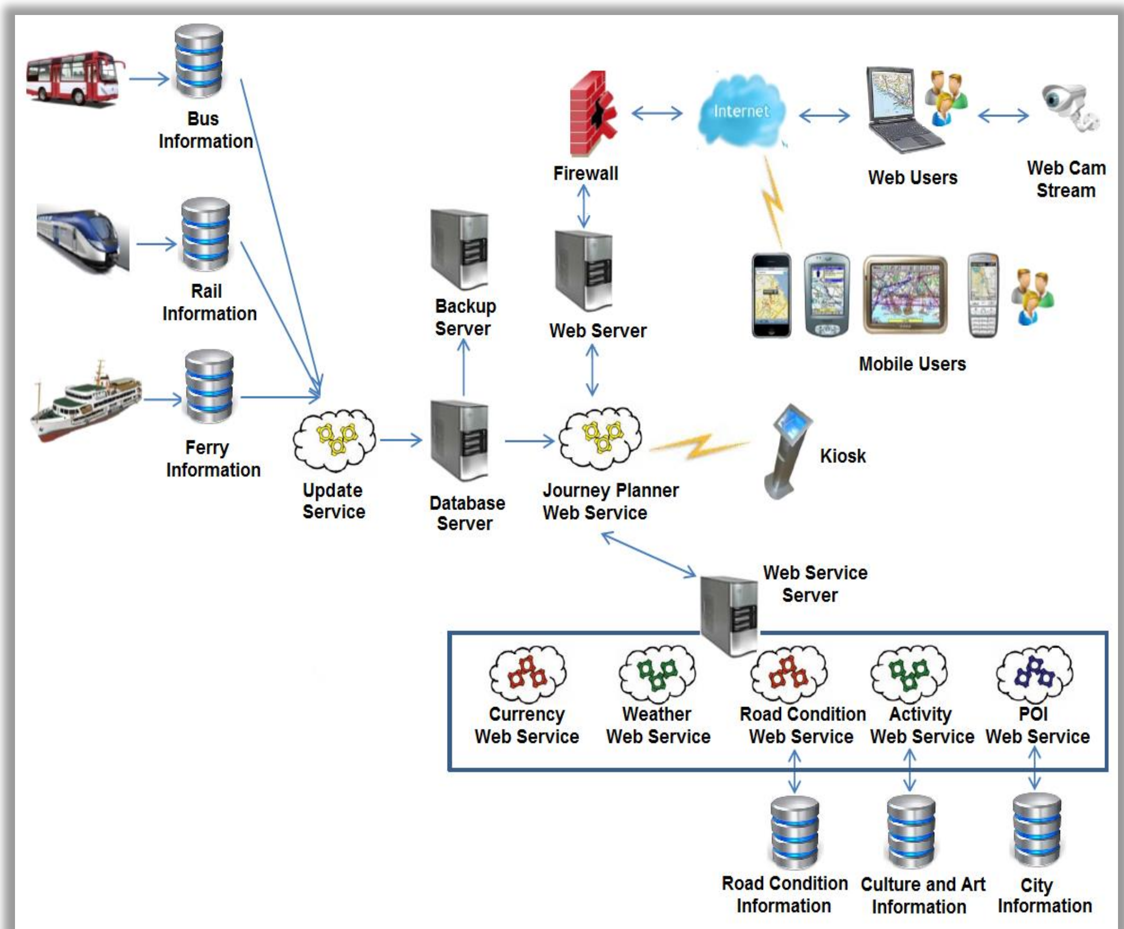


Figure 3.1 System architecture.

The system has been developed based on the three-tier architecture in which presentation, application processing, and data management functions are physically separated as illustrated in Figure 3.2. The three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology.

Presentation tier includes the front-end applications such as web application and mobile applications. In Application tier, business logic is applied by the web services by performing detailed processing. Data tier includes the database servers and the database management system software that manages and provides access to the data.

Windows Communication Foundation protocol provides communication between application and presentation layer while SQL and Oracle protocols and components

allow for communications with data tier and the applications. Separate tiers run on separate physical servers as database server, web service server and web server.

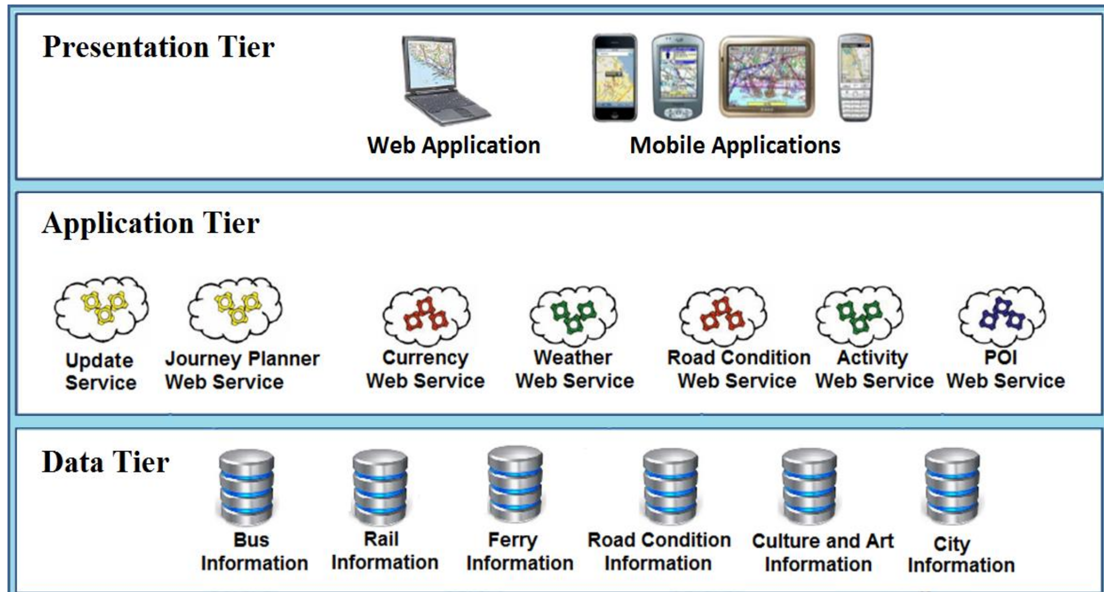


Figure 3.2 Three-tier architecture.

3.3 Database Design

SQL Server Management System is used as the default database management system. Data tables related with the transportation information have been designed in accordance with GTFS format explained in Chapter 2.8.1. When creating the data tables, a data table name convention has been followed. Project description is located at the beginning of the table names (Example: IJPS_ for Intelligent Journey Planner System). The second descriptor is the 3-character extension indicating the content-type of the table. These extensions and the tables they logically grouped are listed below.

- *ADM (Administration):* Administrators, AdminTransactions, Pages, Permissions, Roles
- *GEN (General):* Counties, EducationalStatus, Languages
- *LOG:* Exceptions
- *POI (Point of Interest):* POI, POICategories, POISubCategories, POITranslation, StopPOI

- *PTH (Path):* TransferCenterPaths, TransferCenterPathDetails
- *STT (Status):* WeatherStatus, WeatherStatusIcons
- *TRN (Transportation):* AccessibleTransferCenters, Calendar, CalendarDayTypes, DepartureTimes, Routes, Shapes, SpecialFareRoutes, Stops, StopTimeDistance, Tariffs, TaxiTariffs, TransferCenters, TransferCenterStops, Trips
- *TST*: Test
- *USR (User):* Journeys, Queries, Users, UserTransactions

As mentioned before, there are four types of public transportation modes and each has several lines. All types of lines are kept in "IJPS_TRN_Routes" table with their id, name and vehicle type. Stops are stored in "IJPS_TRN_Stops" table according to their id, name, longitude, latitude, and vehicle type. Distances between sequential stops are also available in "IJPS_TRN_ StopTimeDistance" table. Almost all lines are round-trip. Sequential list of stops for each line are kept in another data table "IJPS_TRN_ Trips" according to their directions.

Transportation system in Izmir is actually insufficient in terms of rail and underground. Maritime transport is also restricted because of the distributed population on geographical area. Therefore, transportation by bus holds an important place in Izmir. Almost all the rail, underground and ferry stations are formed as transfer centers which contain first stops of several bus lines to transfer passengers to their neighborhood. These transfer centers and their stops are stored in "IJPS_TRN_ TransferCenters" and "IJPS_TRN_ TransferCenterStops" data tables, respectively.

Time scheduling is changing at summer/winter time only. Three different timetables are constituted for weekdays, Saturday and Sunday in "IJPS_TRN_ CalendarDayTypes" table. Departure times for each line in two directions organized according to these timetables are kept in the table "IJPS_TRN_ DepartureTimes".

In Izmir, Kentkart, a contactless smart card is used for pricing at all types of transportation. Balance loaded on card decreases according to user type (discounted

for students, and free for senior citizen). The prices for some long-distance lines are doubled. These lines and prices are kept in “IJPS_TRN_ SpecialFareRoutes”.

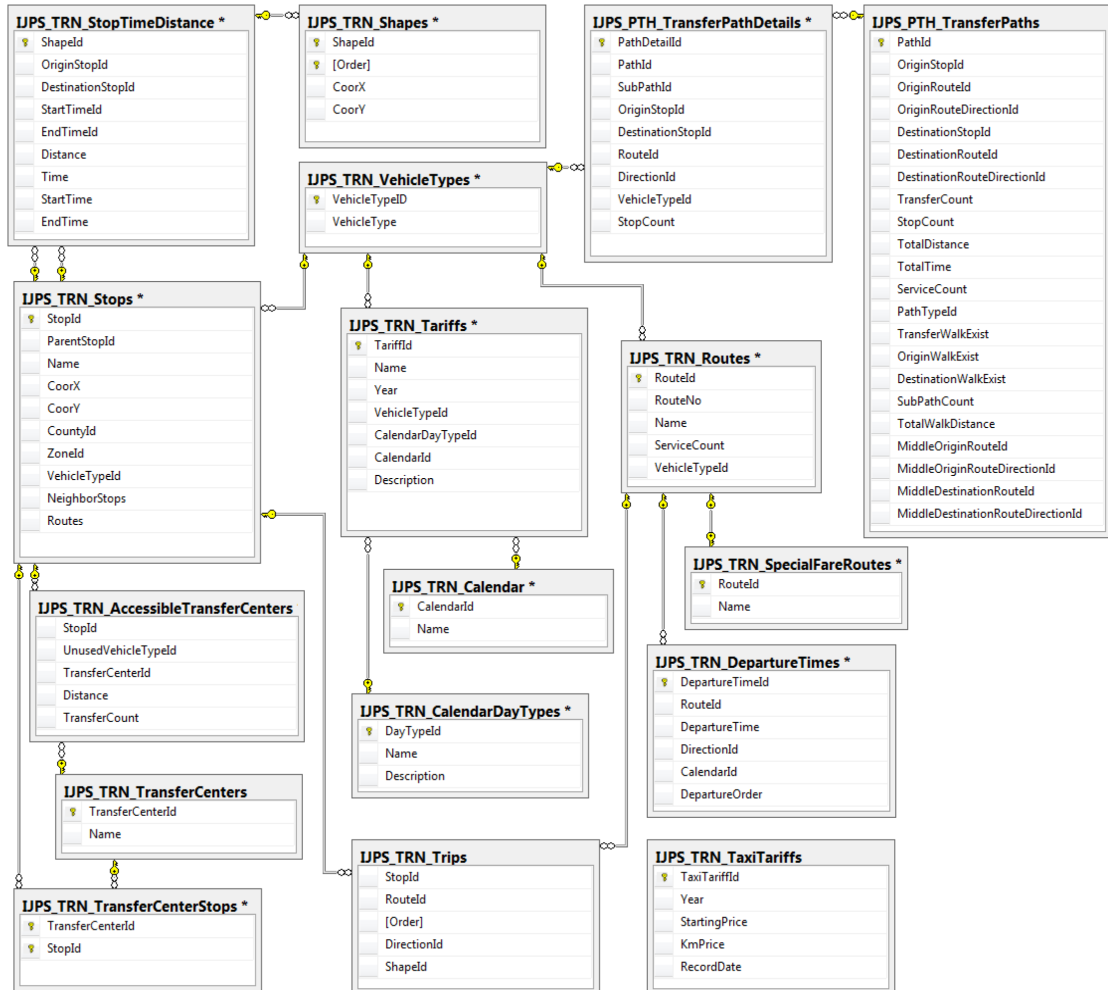


Figure 3.3 Entity-relationship diagram.

Data table “IJPS_POI_ POI” holds named places to select journey origins and destinations. This includes all points of interests (POIs) like popular tourist sightseeing, shopping places, hospitals, schools and all kinds of government offices with their coordinates.

Preprocessed paths between all transfer centers are recorded in “IJPS_PTH_ TransferCenterPaths” and “IJPS_ PTH _ TransferCenterPathDetails” tables in detail. All application specific data tables which are grouped as “ADM”, “USR”, and

“GEN” can be seen in entity-relationship diagrams given in Figure 3.3. 319 lines, 6708 stops, 74,434 departures are recorded currently in the database.

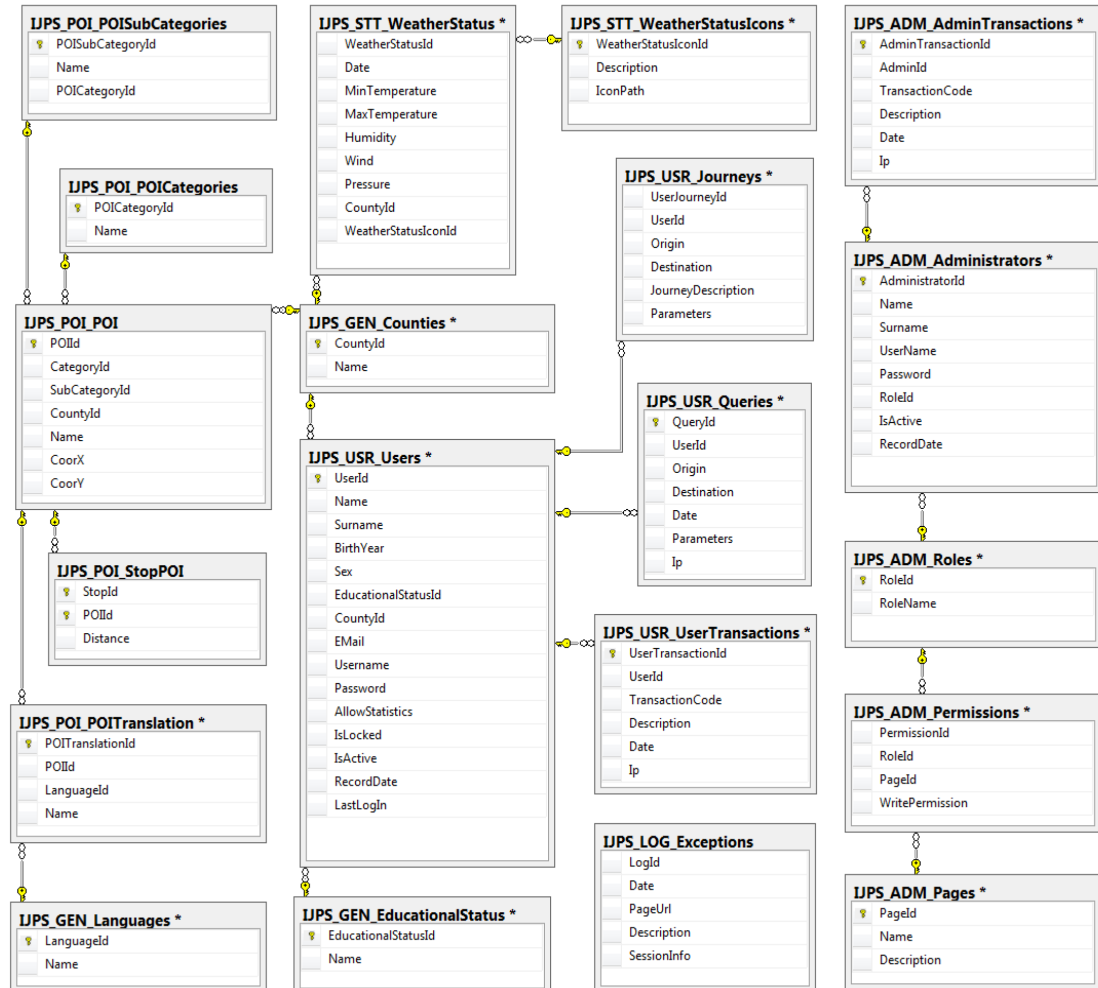


Figure 3.3 Entity-relationship diagram (continued)

3.4 Applications

IJPS consists of Mobile Web and Desktop Web portals, Android, iPhone, and Windows Phone based mobile applications operating in both English and Turkish.

The design of the pages is simple and common UI elements are used to feel users more comfortable. Flow of the usage is not so much different from available journey planner systems to feel user familiar with the system and able to get things done more quickly. Pre-chosen fields reduce the burden on the user. We avoid unnecessary elements and are clear in the language on labels and in messaging.

3.4.1 Mobile Applications

Specific applications have been developed for iOS, Android and Windows Phone mobile operating systems to use system anywhere and at any time effectively. In addition to these mobile applications, a mobile web application is available for other mobile platforms which are not commonly used.

iOS Mobile Application has been developed by using Objective-C programming language and XCode software development environment in MacOSX operating system. WCF method outputs (Soap Service) were read with NSXMLParser. The MKMapView class is used for presenting map data in the application. It provides support for displaying map data, managing user interactions, and hosting custom content.

While developing the Android Mobile Application, the Eclipse platform, Android Software Development Kit and the Java programming language have been used. KSoap2 library which is a Java package that makes establishing the connections easier to WCF method output (Soap Service), has been included into project. The values are taken from the web service properly through this library. Google maps are used to display paths in the Android Mobile Application.

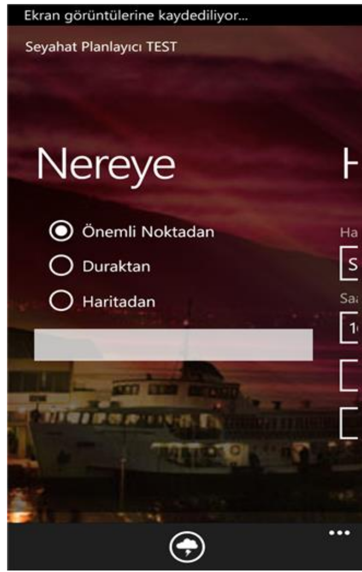
Windows Phone 8 Mobile Application has been developed in Windows 8 operating system by using Windows Phone 8 SDK, Visual Studio 2012 development environment, and C# language. Necessary data is provided by WCF service methods. Windows Phone 8 SDK Map Control provides the Bing maps for route displaying.

Application that runs on mobile Web environments has been developed with the Visual Studio 2010 development environment and C # language. The WCF service methods are used in this application, too.

Properties of the applications will be explained in this section by using sample screenshots. Sub figures A, B, C and D belong to iOS, Android, Windows Phone and Mobile Web applications respectively, in Figure 3.4 to Figure 3.14.

(a)

(b)



(c)

(d)

Figure 3.4 Determination of the origin and destination points in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web.

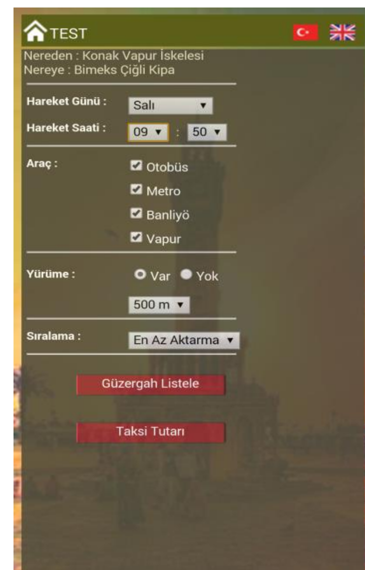
As seen in Figure 3.4, origin and destination points are determined at the beginning of the journey plan. When determining the origin and destination points, three different methods are used. These methods are making a selection from a pool of defined POIs, making selection from stop or station names and marking the location on the map.

Another parameter is the transportation modes that could be used for the user. At present, bus, ferry, subway and train options are available. By default, all options are selected. Elimination of unwanted ones is expected from the user. The next parameter is the desire to walk between transfers and maximum acceptable walking distance. By default, walking is checked as accepted and the maximum walking distance is considered to be 500 m (see Figure 3.5).

(a)

(b)

(c)



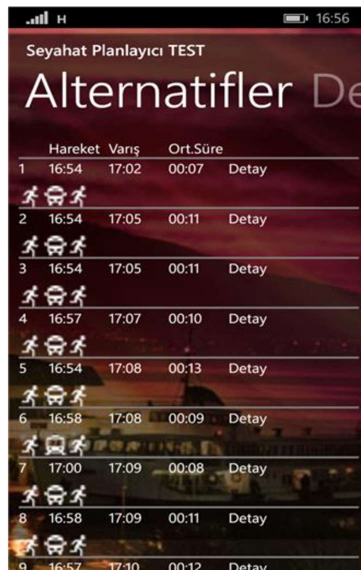
(d)

Figure 3.5 Determination of the journey parameters in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web.

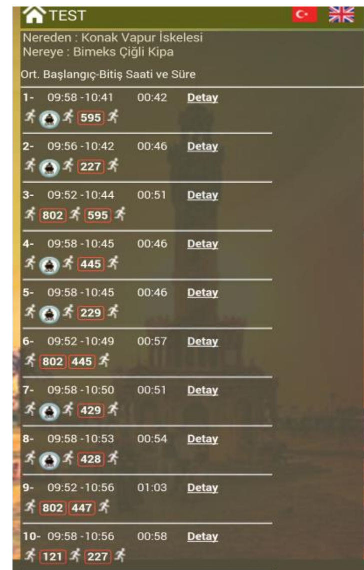
The last parameter is the selection of sorting criteria for the results. Three criteria are available currently. These are least transfer, fastest and cheapest. Least transfer option is selected by default. After taking origin point, destination point and journey parameters from the user, this information are transmitted to the Journey Planner Web Service. The results returned from the service are presented to the user in summary as given in Figure 3.6. Summarized information consists of used lines' icons, departure times and average travel time.

(a)

(b)



(c)



(d)

Figure 3.6 Summarized route list in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web.

Displayed result count can vary depending on the query. For example, when the system produces 5 routes which are direct or has one transfer, the results with two transfers are not required to produce. If Journey Planner Web Service produces more than 10 results, according to the preferred sorting criteria top ten of the results are returned by the Journey Planner Web Service.

(a)

(b)

(c)

(d)

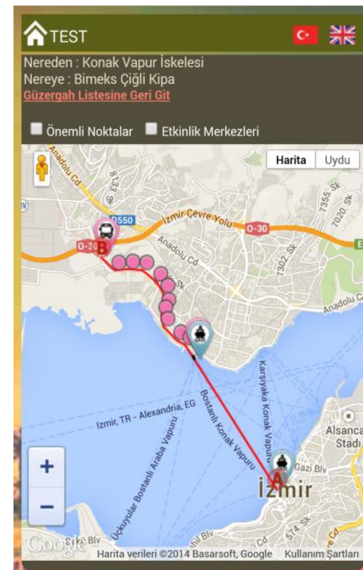
Figure 3.7 Route details in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web.

User can select a route and view the detailed information of the route as exemplified in Figure 3.7. Detailed information includes start and end stops, transportation mode, line number and name, departure times and the average travel time for each sub route. Route details can be sent by e-mail.

(a)

(b)

(c)



(d)

Figure 3.8 Display of the selected route on the map in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web.

Currently selected route can be displayed on the map as screenshots shown in Figure 3.8. Each stop forming the route is marked on the map; transfer stops are also

highlighted. According to user's demand, the event centers and points of interest on the route can be marked on the map. Name tags appear on the map by clicking on stop, event center or important point icons.

(a)

(b)

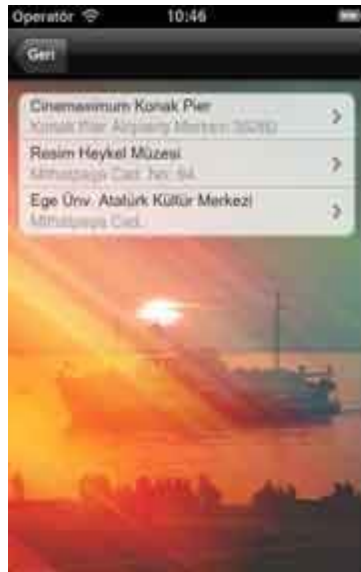
(c)

(d)

Figure 3.9 Point of interest lists in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web.

Important points like marketplace, shopping centers, places of worship, hospitals, parking etc. located on the route are given as a list to the user, as in Figure 3.9. The aim is notifying the user about the places that he/she may need along the route.

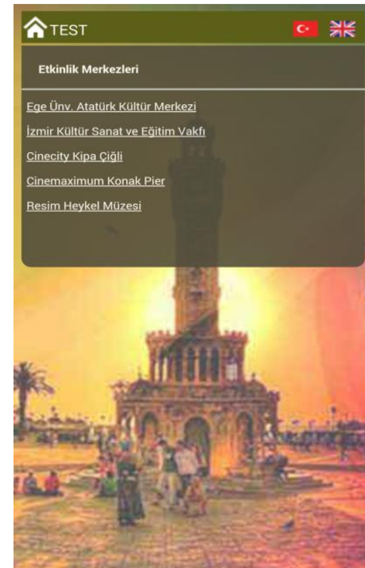
Another list presented to the user includes cultural centers such as museums, and exhibitions, and event centers such as cinema, theater and concert near to the route. As seen in Figure 3.10, when the user selects a center from the list, the events close to the date and time of the journey are displayed. With this function, the people are informed about events that are taking place in the city, and they are intended to greater participation to social and cultural activities.



(a)

(b)

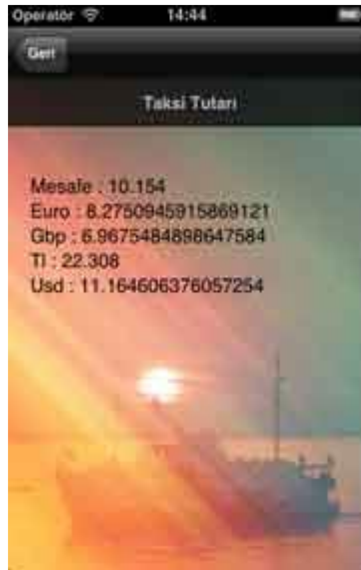
(c)



(d)

Figure 3.10 Event center list in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web.

The distance and taxi fare between selected origin and destination points are presented to the user for guidance purposes only as in Figure 3.11. Estimated taxi fare is calculated by using current exchange rate data and the taxi tariffs, and displayed in four different currencies consisting Tl, Euro, Gbp, and Usd. These prices are only estimated fares and actual fares vary depending on traffic, weather and other unforeseen conditions.



(a)

(b)

(c)

(d)

Figure 3.11 Estimated distance and taxi fare in (a) iOS, (b) Android, (c) Windows Phone, (d) Mobile Web.

Weather condition is an important factor that affects the user's preferences. User can change the preferred modes of transportation or can prefer less transfer according to the weather condition. Therefore, informing the user about weather is helpful when determining the user's query parameters. By selecting a district from the list, weather information about that district can be displayed as shown in Figure 3.12.

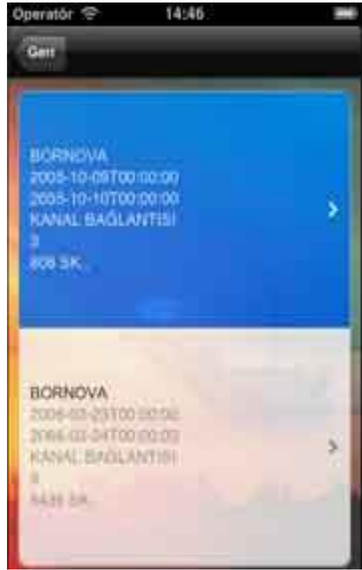
(a)

(b)

(c)

Figure 3.12 Current weather conditions in (a) iOS, (b) Android, (c) Windows Phone.

In addition to all these features, an informative window is available about the road conditions. All the ongoing road construction and infrastructure works can be listed on the basis of the districts. This information is also generated from municipal sources and always up to date.



(a)

(b)

(c)

Figure 3.13 Road condition in (a) iOS, (b) Android, (c) Windows Phone.

Another important feature is information about the traffic status. There is no available service to provide real-time traffic status for Izmir currently. Construction

of Live Camera View system by Municipality of Izmir is in progress on various points of the city to monitor live traffic status. Currently, statistical traffic information provided by the KENTKART Company is used to inform users. Traffic forecasting based on the statistics informs traffic status of each sub path weather heavy or not (see Figure 3.14).

(a)

(b)

(c)

Figure 3.14 Traffic status in (a) iOS, (b) Android, (c) Windows Phone.

3.4.2 Web Application

Web application module of IJPS has been developed in the Visual Studio 2010 development environment and coded with the C# language. WCF service methods are used in this application, too. Web interfaces have been designed in accordance with all common web browsers.

In addition to the functions available in the mobile application, user can create an account and log in to the system from the web application. A registered user is able to save his queries and view them later. User transactions can be used for statistical purposes by the user's permission. Also, users can transmit their opinions and suggestions from the contact page.

The necessary pages for the management operations have also been developed. Management operations include displaying and reporting user queries, monitoring exceptions, and other administrative settings.

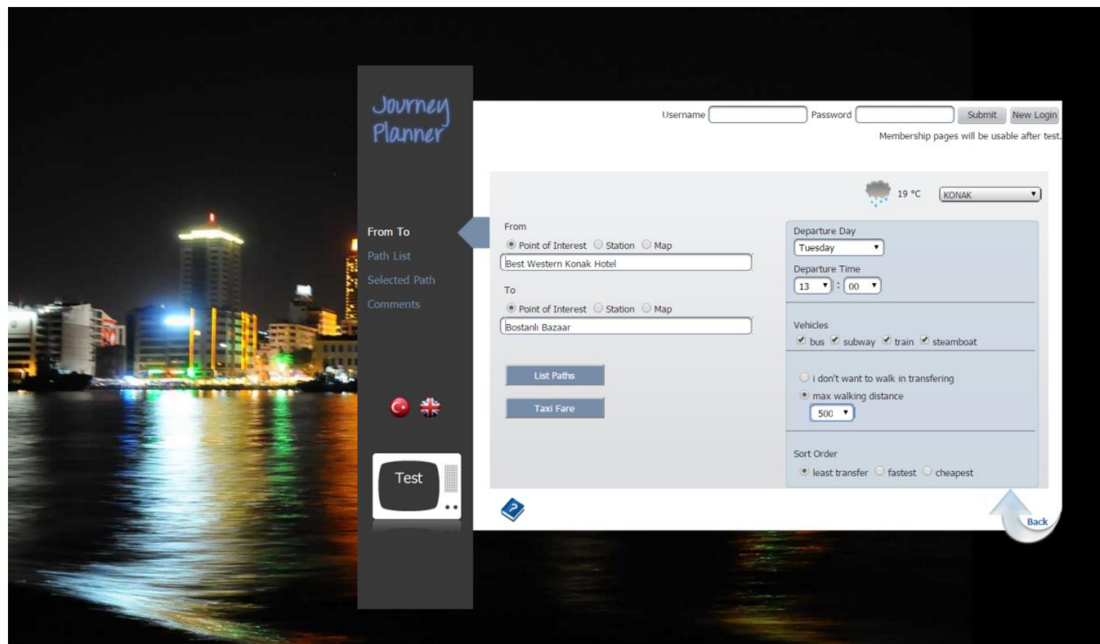


Figure 3.15 Journey parameter selection page.

After determination of the journey parameters (see Figure 3.15), paths produced by the web service according to these parameters are presented as shown in Figure 3.16. User can select a route and view the detailed information as exemplified in Figure 3.17.

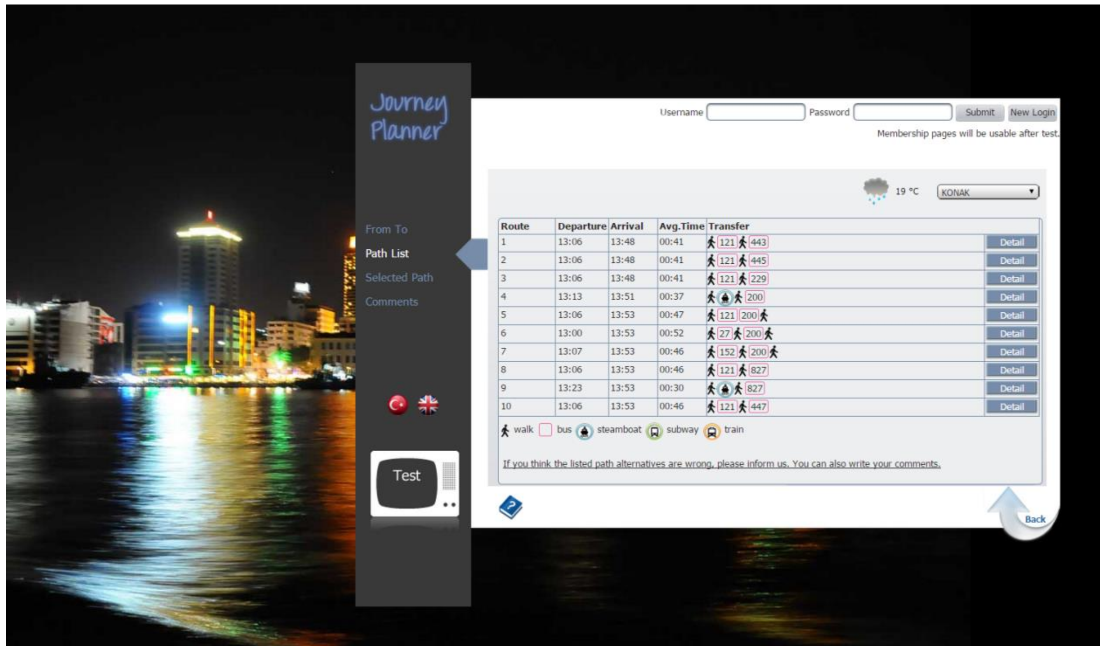


Figure 3.16 Alternative paths.

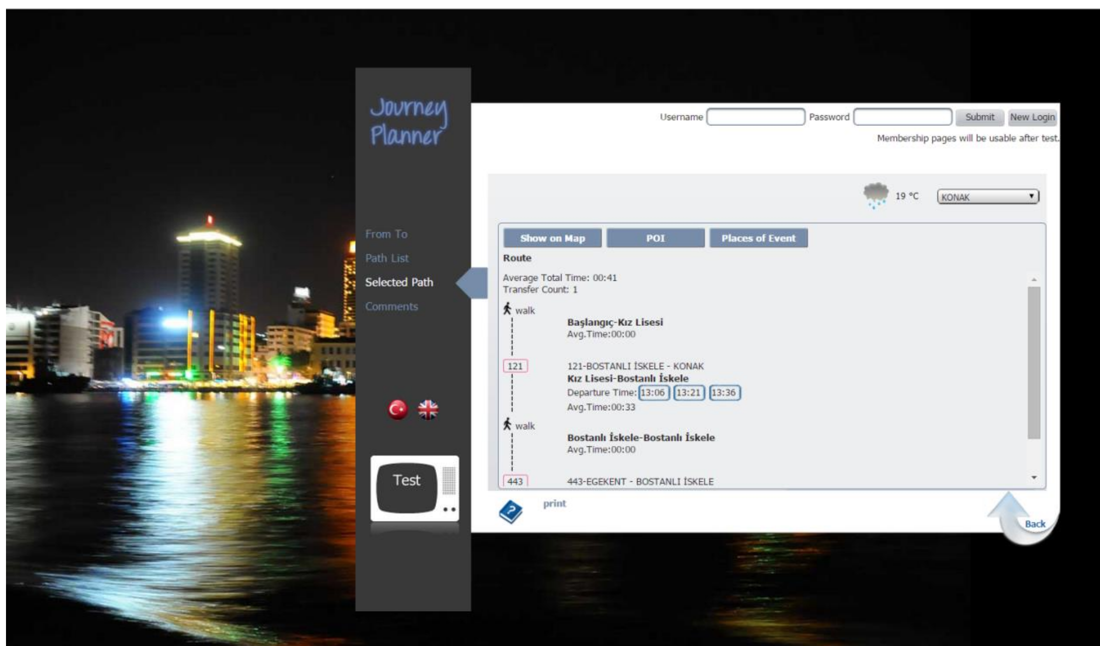


Figure 3.17 Route details.

The selected path can be displayed on the map as screenshots shown in Figure 3.18. Each stop of the path is marked on the map, transfer stops are also highlighted. The event centers and points of interest around the selected path can be marked on the map, too as seen in Figure 3.19.

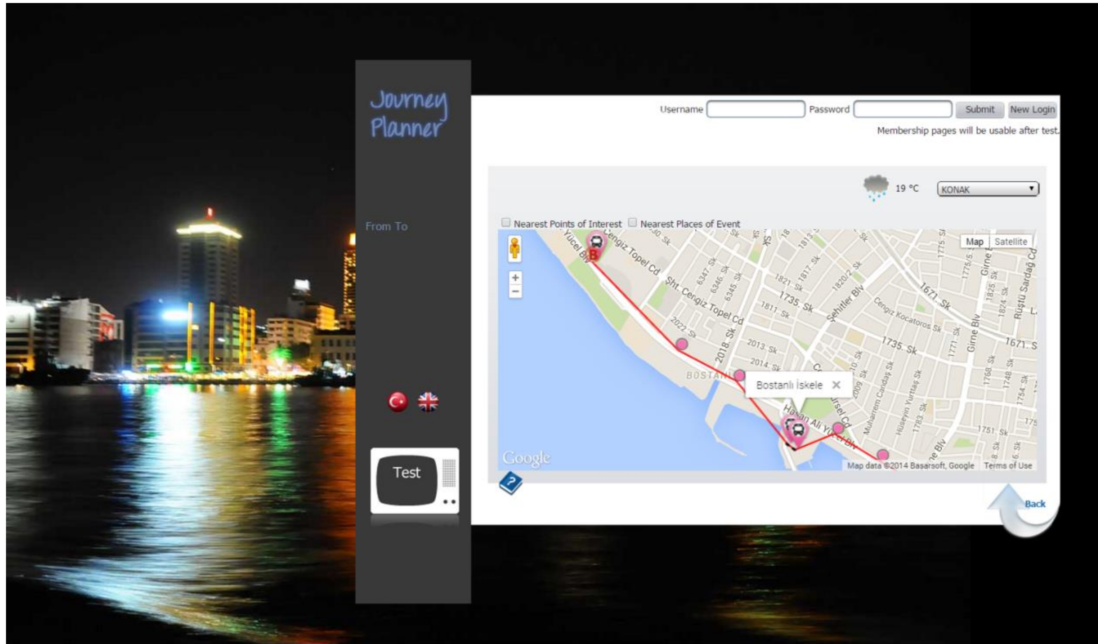


Figure 3.18 Representation of a path on the map.

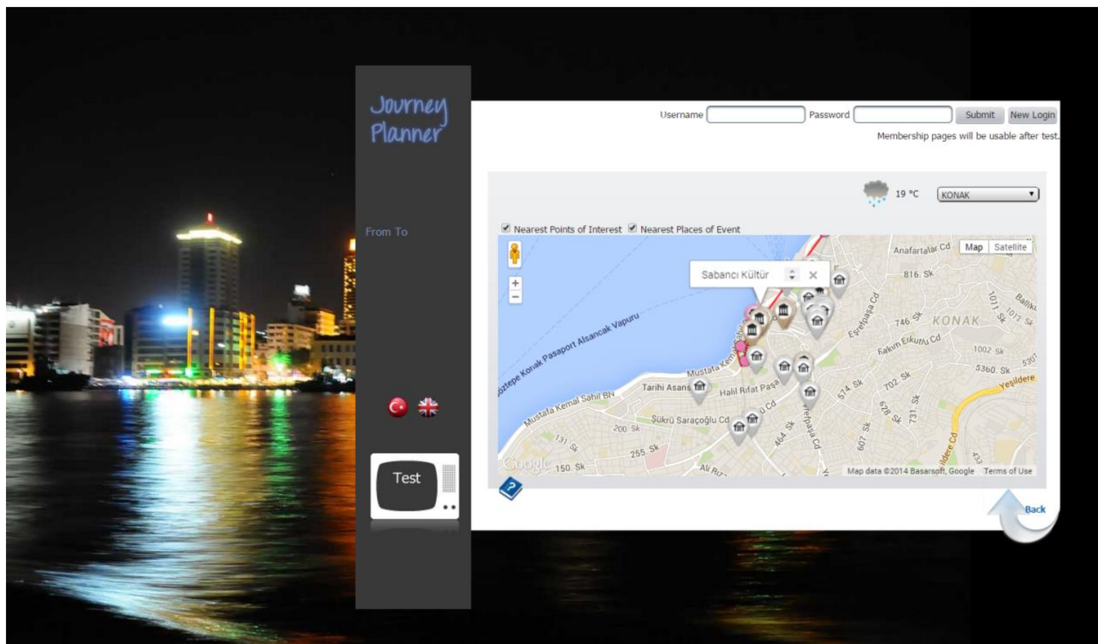


Figure 3.19 Representation of the event centers and POI on the map.

Important points, cultural centers such as museums, and exhibition centers, and event centers such as cinema, theater and auditorium near to the path are listed as seen in Figure 3.20. User can display the events that will take place in a center as in Figure 3.21.

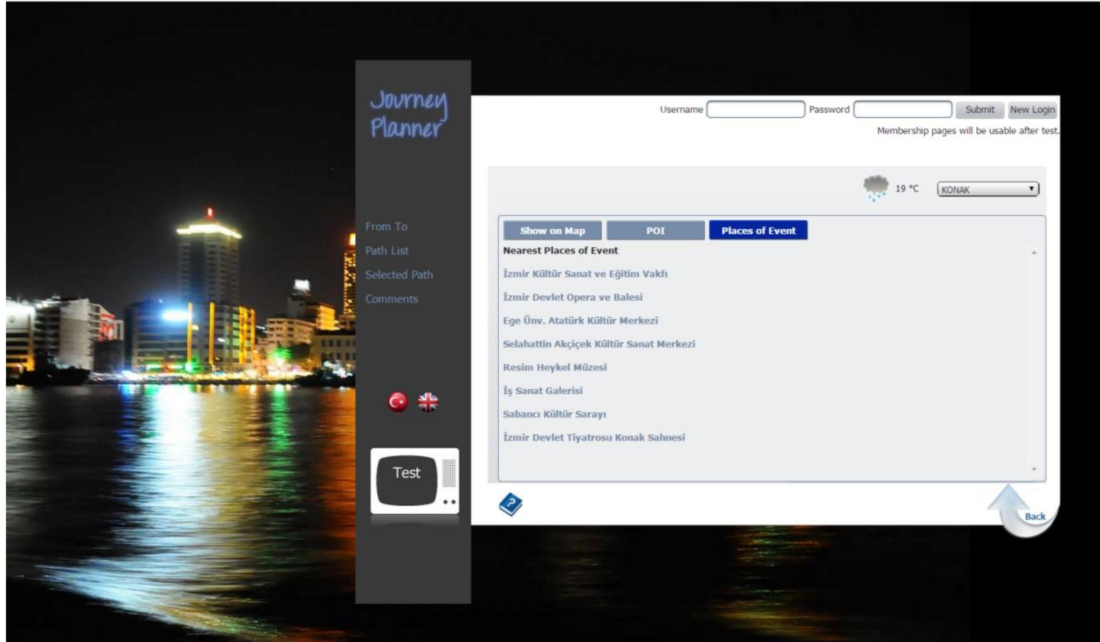


Figure 3.20 Nearest places of event list.

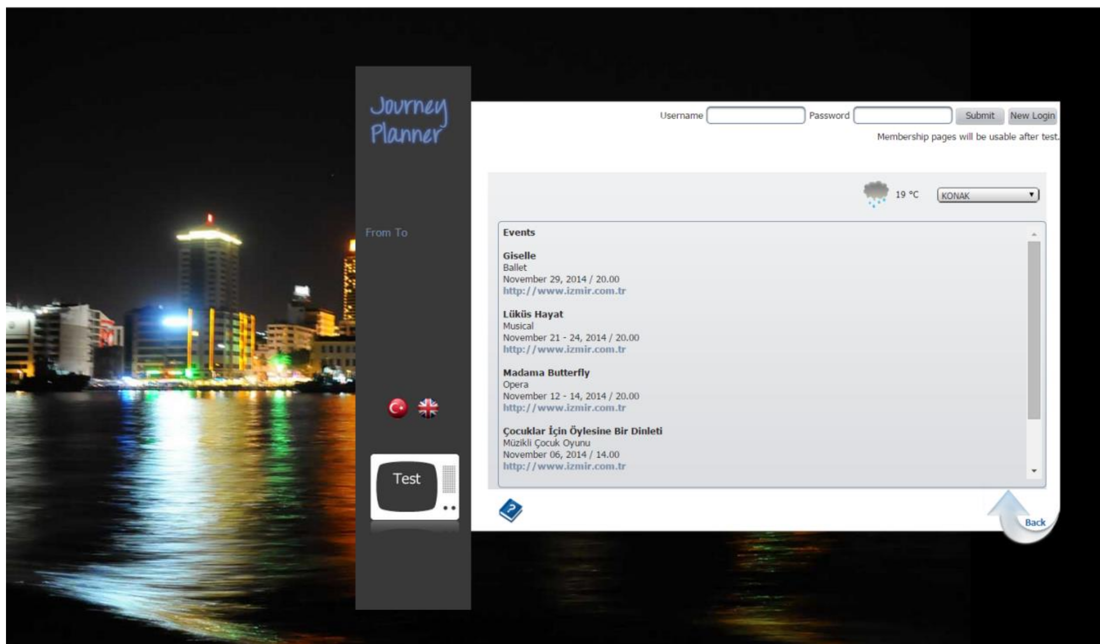


Figure 3.21 Event list of selected event center.

CHAPTER FOUR
SERVICES

4.1 Overview

A distributed application built with Web services is a service-oriented application. ASP.NET Web Services (ASMX) has been available for building Web services since .NET was first released. Then, Microsoft introduced the new service model Windows Communication Foundation (WCF) (What Is Windows Communication Foundation, 2014). WCF provides a number of benefits over ASP.NET Web Services, including:

- Support for sending messages using not only HTTP, but also TCP and other network protocols.
- The ability to switch message protocols with minimal effort.
- Support for hosting services on hosts other than a Web server.
- Built-in support for the latest Web service standards and the ability to easily support new ones.
- Support for security, transactions and reliability.
- Support for sending messages using formats other than SOAP, such as Representational State Transfer (REST).

Therefore, WCF services are very practical approaches looking at the current business trends.

WCF services are developed as interface, operations, and data contracts. The contract specifies the methods clients can call, any arguments the methods take and any values the methods return. The *ServiceContract* attribute identifies the interface as a service contract. To expose a method to clients, the *OperationContract* attribute is used.

Another type of contracts is the *DataContract*. Clients and services exchange data using XML messages. The WCF runtime uses the *Data Contract Serializer* to serialize (convert to XML) and deserialize (convert from XML) data. This serializer has the ability to work with basic .NET types such as strings, integers, DateTime, etc. However, it does not have the built-in ability to work with classes and other complex types. To make a class serializable, a data contract can be created by adding the *DataContract* attribute to the class definition and by adding the *DataMember* attribute to each member of the class which is wanted to be serialized.

A Windows service is a computer program that operates in the background and does not have a user interface. It must conform to the interface rules and protocols of the Service Control Manager; the component responsible for managing Windows services (Services, 2014).

Windows Services can execute even when no user is logged on to the system. They can be configured to start when the operating system is started and run in the background as long as Windows is running. Alternatively, they can be started manually or by an event.

A service can be registered to be started or stopped when a trigger event occurs. This eliminates the need for services to start when the system starts, or for services to poll or actively wait for an event; a service can start when it is needed, instead of starting automatically whether or not there is work to do. These types of services are intended to provide core operating system features such as Web serving, event logging, file serving, printing or error reporting. Not all services are developed by Microsoft. Some applications and drivers install their own services.

In scope of this thesis, two main services and some auxiliary services have been developed. The Update Service has been developed as a windows service application. The Journey Planner Web Service and other auxiliary services have been developed as WCF web services mentioned above. These services will be explained in the following sections.

4.2 Update Service

Update service is a windows service application that runs automatically on the database server every night. This service collects actual data from transportations agencies and integrates them in the local database. After updating data, some preprocessing tasks are taken to fasten on-the-fly processing of user queries. Update service operates in the background and does not have a user interface. A windows form application has been developed to test the update service (see Figure 4.1). This application provides monitoring of process log and current status of the operation.

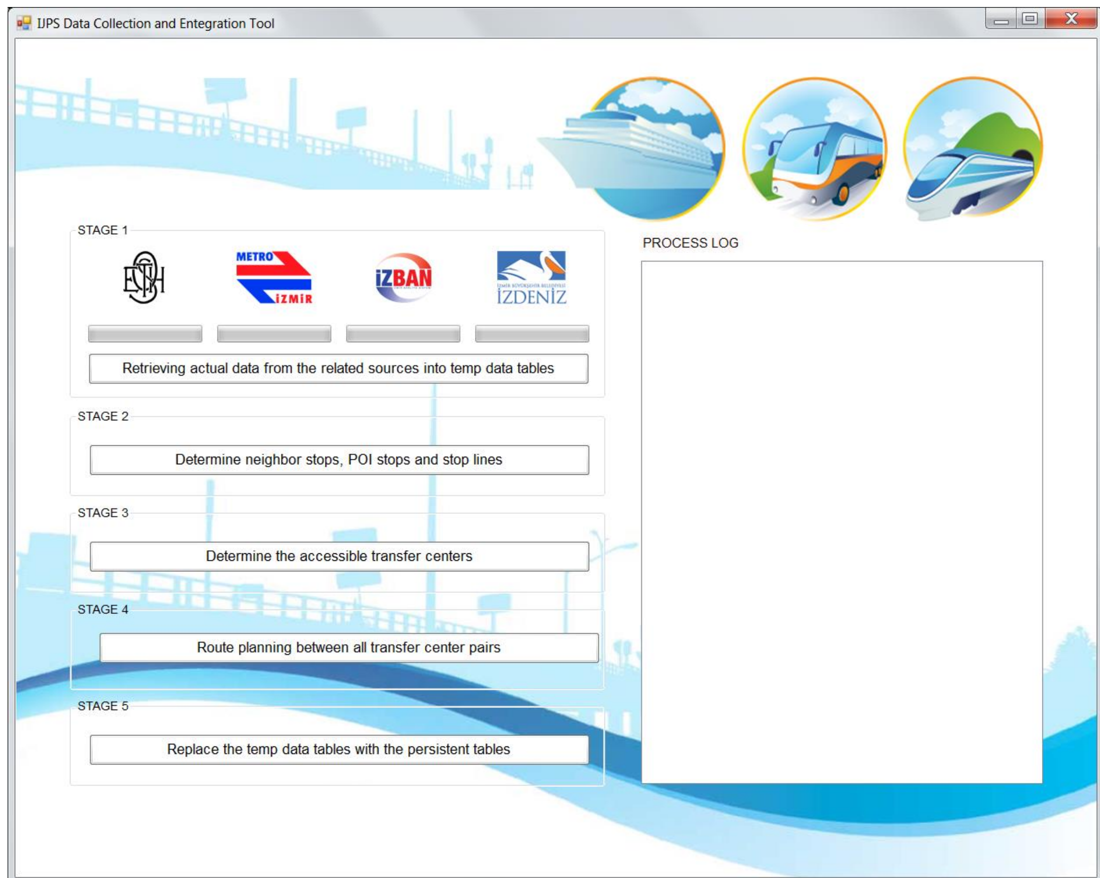


Figure 4.1 IJPS Data Collection and Integration Tool.

Service process starts with the creation of the temporary tables that contain the data to be updated. Current data is retrieved from related sources, verified and stored into the temporary tables. Then, neighbor stops calculation is performed on temporary stops table. After that, transfer centers that can be accessible by up to two transfers are calculated for each stop. Paths between each transfer center pair, which

contain no transfer or one transfer are calculated and stored into the related tables with path details. Finally, temporary data tables are replaced with the persistent ones. General workflow of the Update Service can be seen in Figure 4.2.

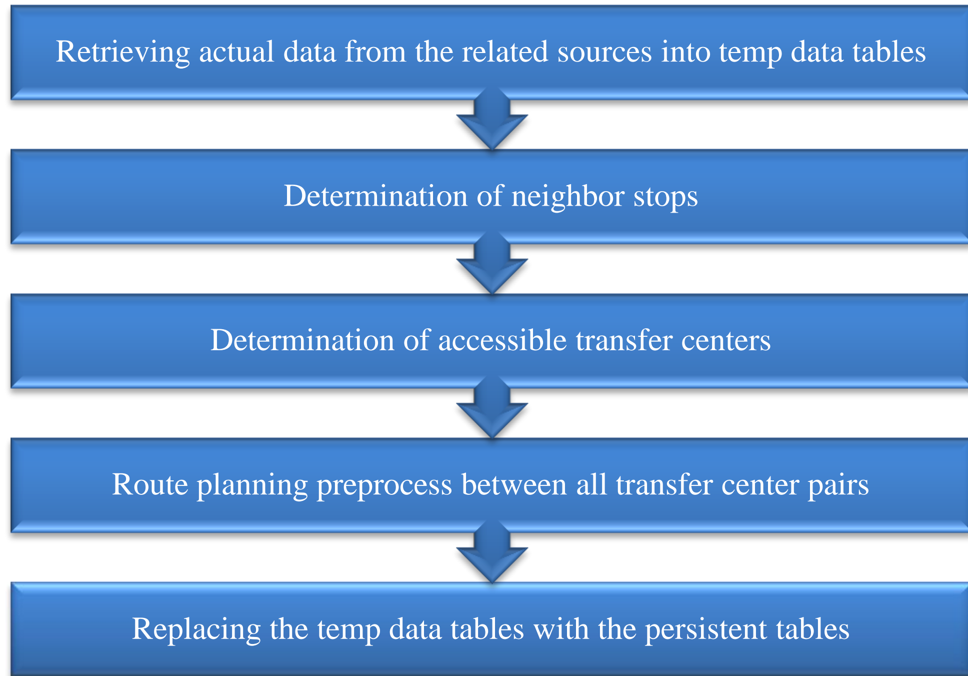


Figure 4.2 Update Service work flow.

The total processing time of local service takes about 50 minutes on the database server. The persistent data tables are not affected by the update operations until the end of process and system keeps running consistently.

4.2.1 Updating Transportation Information

Transportation data is very changeable. New lines can be added, some current lines can be removed or routes can be changed by the time. New stops can be added to the current lines or some of them can be canceled. Even their locations or names can be changed. As the same way, time tables of different types of vehicles change independently. Thus, updating line, route and time table information periodically from related resources is a necessity.

Bus, metro, train and ferry information and time tables are retrieved from databases of ESHOT, METRO, İZBAN and İZDENİZ respectively. Update process

starts at midnight. Firstly, all retrieved data is written into temporary data tables and at the end of the operation, these tables are replaced with the persistent tables. Agencies store transportation data in their own formats. All retrieved data are integrated in a common format and stored in related data tables in accordance with GTFS.

4.2.2 Determination of Neighbor Stops

In transportation graph, stops are connected with edges labeled with their common lines. For example, two sequential stops of a bus line are connected with a directed edge which is labeled with the line id. When planning an intermodal journey, it is a realistic assumption not having a vehicle available everywhere along the journey. To be able to make point to point queries in a transit network, some sort of foot-edges are required, so any stage of the journey can be covered by foot or passengers may walk along the stops while transferring between two different lines. Foot-edges are also providing to link each of the transportation networks (bus, train, metro, ferry etc.) into the resulting multi-modal network G.

Two stops u and v are labeled as neighbor stops by adding foot-edges between them, if the road segment is available to pedestrians and $\mathrm{dist}(u, v)$ is less than maximum walking distance. The m-nearest-neighbors of a stop are computed using the Euclidean distance metric. If two stop$s$ u and v are determined as neighbor stops, then $(u, v) \in E$ and $(v, u) \in E$. Neighborhood determination for all stops is practiced just after the update of transit data.

Train, metro and ferry stations are located further away from other stops when compared to the bus stops. Because of this reason, all of the bus stops in 500 m range of any train, metro or ferry stations are labeled as a neighbor.

Bus stops are close to the other bus stops compared by other stations (ferry, train, and metro). Sometimes 100 m is sufficient for determination of neighbor bus stops. Distance is increased up to 250 m when any neighbor bus stop could not be found in 100 m range, as can be seen in Figure 4.3. A stop is neighbor of a station as long as

the station is neighbor of the stop. If a stop in a transfer center is labeled as a neighbor, all other stops placed in that transfer center are also labeled as neighbors.

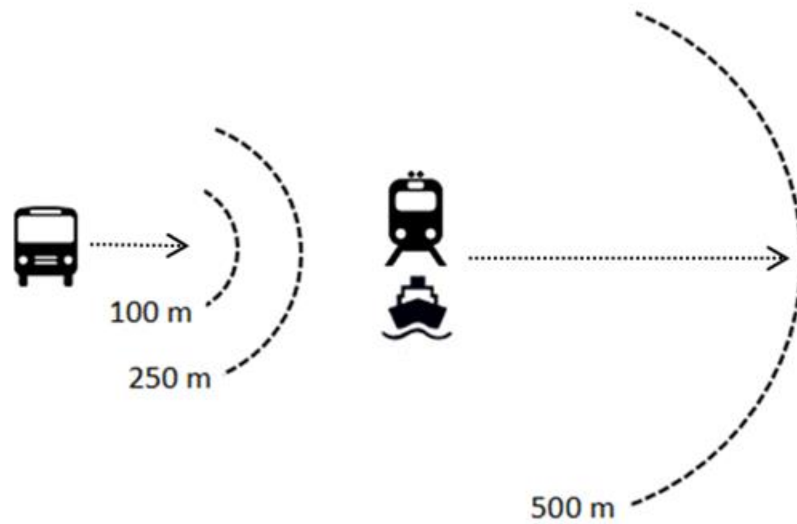


Figure 4.3 Distance ranges in determination of neighborhood.

4.2.3 Determination of Accessible Transfer Centers

The main transportation lines are train, metro and ferry in Izmir. Passengers, who use these lines, can transfer to a bus line to access their neighborhood. Currently, there are 20 transfer centers located mostly around train, metro and ferry stations. Distribution of these centers is shown in Figure 4.4.

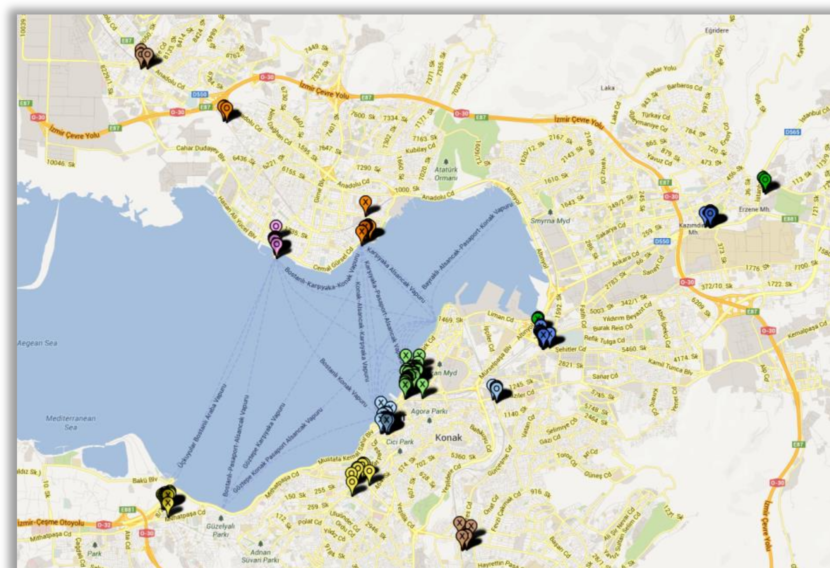


Figure 4.4 Transfer centers in Izmir.

According to the new transportation logic implemented since June 2014 in Izmir, a passenger who travels between two remote points, likely visits one or more of these transfer centers. Our path finding algorithm uses this logic at a stage of route calculation. Thus, determination of accessible transfer centers for each stop, is a preprocess runs after the update of the transportation data. An accessible transfer center for a stop means, at least one path which has up to two transfers, exists from the stop to that transfer center. Under these conditions, there is no stop which does not have an accessible transfer center in Izmir. This logic can be adapted to any transportation network by determining transfer centers automatically considering transportation infrastructure. Also, *parent stations* in GTFS format may cover transfer centers.

4.2.4 Route Planning Preprocess Between All Transfer Center Pairs

When we examine routes between any two transfer centers in Izmir, it can clearly be seen that, at least one route which has zero or one transfer, exists between them. Path finding preprocess, calculates and records the available path between transfer centers, to use them while planning the journey for user queries.

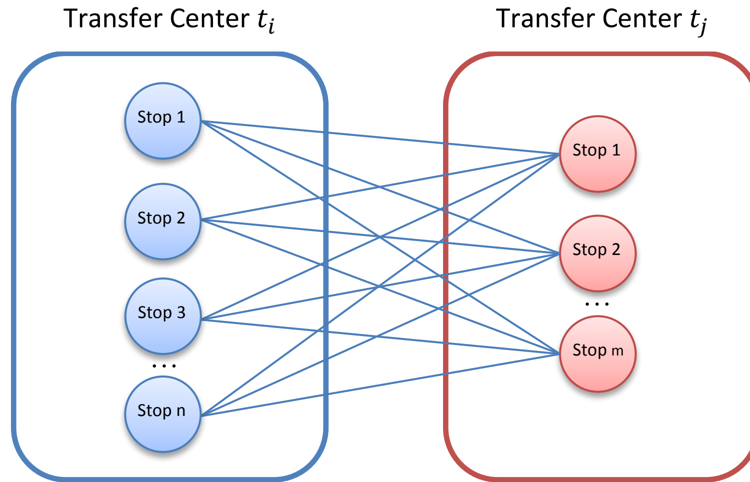


Figure 4.5 Path finding between two transfer centers.

Each transfer center is formed by one or more stops. The maximum stop count in a transfer center is currently 34, in Izmir. If we refer the transportation graph as G that consists of a set of transfer centers T and each transfer center has a set of stops S,

a set of paths $P \subseteq s_n \in t_i \times s_m \in t_j$ for every i and j ($i \neq j$ and $t_i, t_j \in T$) calculated and stored into a database table. $p \in P$ is a path which has no transfer or only one transfer. The illustration of this calculation is shown in Figure 4.5.

Path finding preprocess between transfer centers takes about 30 minutes on the database server. Approximately 25800 paths are stored into the database.

4.3 Journey Planner Web Service

This service is a WCF web service and forms the core of the system. Each type of the clients uses this service for path finding and some other methods. The main method is the route finding method which takes the source and destination points and the user preferences as parameters and returns the calculated routes ordered by preferred criteria. This method is an implementation of the Gradual Path Finding Algorithm which will be explained in Chapter 5 in detail.

The other methods are for retrieving POI and stop names according to the user specified text, finding closed stops according to the given coordinate or named place, calculation of the taxi toll, determination of POIs and social activities around a route etc.

When the service is started, some frequently used data is loaded into the memory statically. Timetables, line-stop list, stop and line information, transfer centers and their stops, preprocessed routes between transfer centers and POIs are stored in static data structures. Loading this data takes place at the start of the web service only once in a day. All requests are answered using this data from memory without database queries.

Journey Planner Web Service has been developed with Visual Studio 2010 development environment and C# language. All the methods which are exposed to clients have been labeled by OperationContract attribute as shown in Figure 4.6. DataContract attribute has been added to the class definitions and DataMember

attribute has been added to each member of the class which is wanted to be serialized.

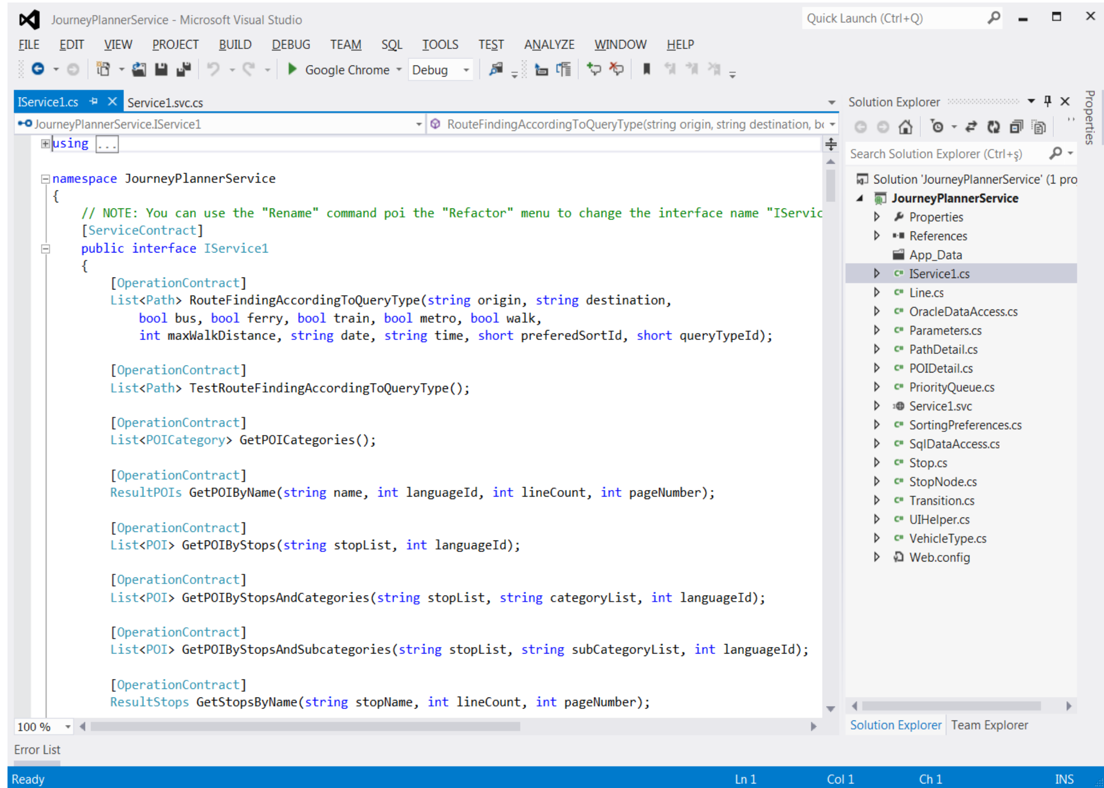


Figure 4.6 Development environment of Journey Planner Web Service.

4.3.1 Service Methods

Several methods have been implemented within Journey Planner Web Service. A few of them are exposed to the clients. The rest of them are the inner methods composed of implementation of business logic. The most important method is the route finding method which takes the origin and destination points and the user preferences as parameters and returns the calculated routes ordered by preferred criteria. This method calculates the routes by using the Gradual Path Finding Algorithm.

The route finding method takes 12 parameters whose values are gathered from the user. These parameters are listed below:

- *Origin and Destination:* These parameters are the string values in one of the three types of location identifier. Origin and Destination values can be a POI id, "|" separated geographical coordinates (Example: 27.0353609432423|38.3929550089826) or "." separated stop ids (Example: 13016 or 10002.10005.10006.10007).
- *Bus, ferry, train, metro, walk:* These parameters specify whether to use the related type of transportation for journey planning.
- *Max Walk Distance:* It gives the maximum distance that a user can walk from the origin to first stop, from last stop to destination or between transfer points. This variable takes a value between 0 and 500 m.
- *Date and Time:* These values specify the desired date and time to begin the journey. Currently, date represents only day of the journey. It takes a numeric value from 1 to 7 corresponding Monday to Sunday.
- *Prefered Sort Id:* It can take three different values from 1 to 3 where 1 for least transfer, 2 for fastest and 3 for cheapest.
- *Query Type Id:* It is a numeric value which is used to identify the types of the origin and destination parameters. Table 4.1 represents the values that this parameter can take according to the type of the origin and destination point.

Table 4.1 Query types according to the origin and destination parameters.

Query Type Id	Origin	Destination
1	POI	POI
2	POI	Stop
3	POI	Location
4	Stop	Stop
5	Stop	POI
6	Stop	Location
7	Location	Location
8	Location	POI
9	Location	Stop

Route finding method has a return type of List<Path>. This list contains up to 10 results.

The other methods are for retrieving POI and stop names according to the user specified text, finding the closed stops according to the given coordinate or named place, calculation of taxi prices, determination of POIs and social activities around a route etc.

4.3.2 Testing the Web Service

The quickest and easiest way to test service methods is to use the WCF Test Client application. The WCF Test Client displays the service and its methods (service operations). To call a service, it must be hosted. When F5 is pressed, the WCF Service Host application starts and hosts the service.

By double clicking a method, the Request section pane that contains the requested parameters and the Response section pane appear as given in Figure 4.7. By clicking Invoke button, the service method is invoked and response message is returned by the service.

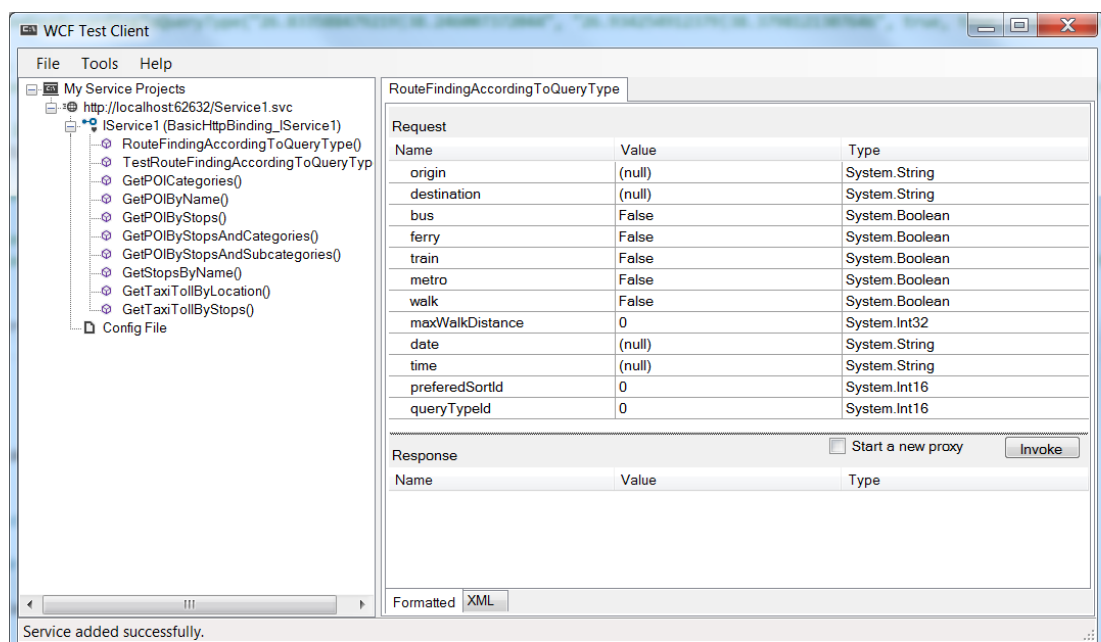


Figure 4.7 WCF Test Client window.

When developing Journey Planner Web Service, testing it with the WCF Test Client was not sufficient and effective. Using a graphical user interface simplifies the test procedure to send parameters. Also a visual user interface simplifies the understanding of the results. For these reasons, a windows form application has been developed to be used in developing process of the service. The application has been developed with Visual Studio 2010 development environment and C# language. This application was used for invocation of the service and representation of the results that were returned by the service. After the development of the web application, the task of testing has inherited to this web application from the windows application.

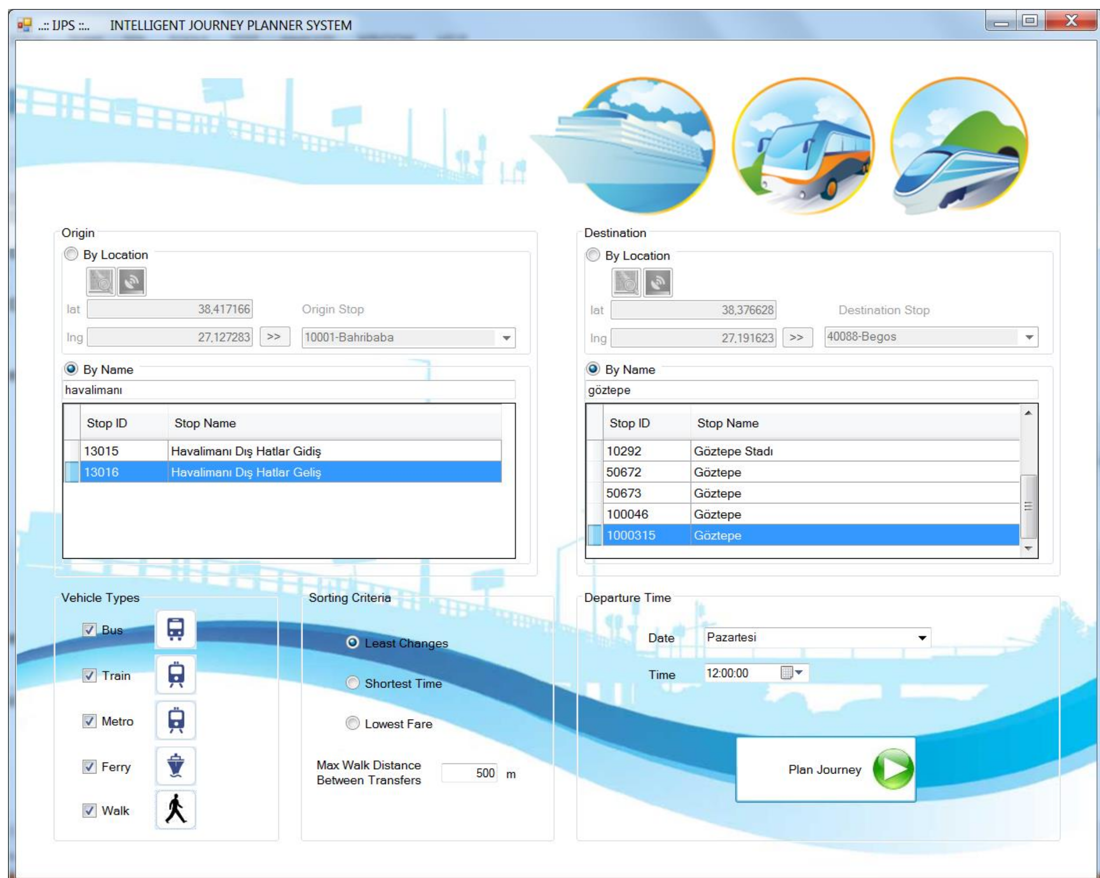


Figure 4.8 Windows application developed to test Journey Planner Web Service.

Figure 4.8 shows the Windows application developed to test Journey Planner Web Service. It has all the necessary interface components for gathering the journey parameters from the user. Origin and destination points can be determined by

indicating the coordinates or typing the name of an important point or a stop. Usage of bus, ferry, train, metro or walk, maximum distance to walk, date and time to start journey, preferred sorting criteria are the other expected information to be filled by the user.

```
Path 1:  202_D - 0_G
Path 2:  0_G - 200023_D - 0_G - 200031_G
Path 3:  0_G - 200023_D - -8_G - 20121_D - -1_G
Path 4:  200_D - 20921_G - -2_G - 200031_G
Path 5:  0_G - 200023_D - -2_G - 24901_G - -2_G - 200031_G
Path 6:  200_D - 20271_G - -17_G - 200031_G
Path 7:  200_D - -2_G - 20351_G - -1_G - 200031_G
Path 8:  0_G - 200023_D - -2_G - 22851_G - 0_G - 200031_G
Path 9:  200_D - 28871_G - 0_G - 200031_G
Path 10: 200_D - 21521_G - 0_G - 200031_G

Path 1
       Sub Path 1
              Id: 0
              Vehicle Id: 1
              Vehicle: Bus
              Line Id: 202
              Line: 202-CUMHURİYET MEYDANI - HAVAALANI
              Line Direction: D
              Origin Stop Id: 13016
              Origin Stop Name: Havalimanı Dış Hatlar Geliş
              Destination Stop Id: 10198
              Destination Stop Name: Vali Konağı
              Stop Count: 16
              Time: 00:33
              Distance: 20,58687km
              Departure Times: 12:00 13:00 14:00
       Sub Path 2
              Id: 1
              Vehicle Id: 0
              Vehicle: -
              Line Id: 0
              Line: Walk
              Line Direction: G
              Origin Stop Id: 10198
              Origin Stop Name: Vali Konağı
              Destination Stop Id: 1000315
              Destination Stop Name: Destination
              Stop Count: 1
              Time: 00:05
              Distance: 0,4627119km
              Departure Times: 12:33

              Total Distance: 21,04959
              Total Time: 00:38
              Total Stop Count: 17
              Transfer Count: 0
              Walk Count: 1
              Total Walk Distance: 0,4627119
              Total Walk Time: 00:05
              Total Cost: 1
              Departure Time: 12:00
              Arrival Time: 12:38
```

Figure 4.9 Display of results gathered from the Journey Planner Web Service.

The test application outputs the results retrieved from the service to a text document. Summary of the results is given at the beginning of the document and the remaining part of the document consists of detailed results. Each path has one or more sub paths. A sub path has several properties as vehicle type, line information, origin and destination stops, stop count, travel time and distance, and departure times. Total distance, time and stop count, transfer count, walk count, total walk distance and time, total cost, departure and arrival times are the other features owned by each path. Figure 4.9 gives a sample display of the results gathered from the Journey Planner Web Service.

CHAPTER FIVE
GRADUAL PATH FINDING ALGORITHM

5.1 Overview

In scope of this thesis, we developed the Gradual Path Finding Algorithm (GPFA) that produces alternative journeys by using the goal-directed speed-up technique. The search is directed toward the target to obtain fewer transfer counts. Modified versions of Dijkstra's algorithm have been used in several stages of the algorithm. Detailed implementation of GPFA will be explained in this chapter. First, our transportation graph representation will be mentioned.

5.2 Representation of the Transportation Graph

In this study, each stop (bus, train, metro, ferry) is represented as a node and a line connecting two consecutive stops in a certain direction is represented as a directed edge to form a transportation graph. This graph is a directed graph as illustrated in Figure 5.1.

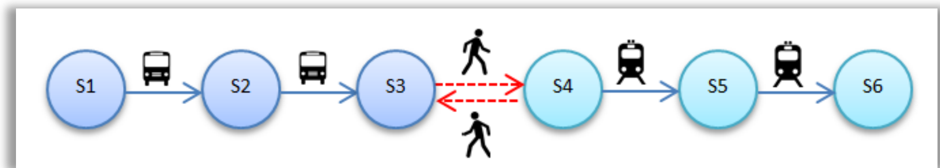


Figure 5.1 Illustration of a transportation graph.

Transportation graph differs from a traditional graph by having multiple edges between any node pair. Each edge corresponds to different vehicles traveling between two stops. Also, if two stops are in walking range, these two stops are connected to each other with edges which are labeled as foot.

Foot edges connect the two nodes in two-ways. These edges are also used to combine different vehicle graphs to obtain a complete transport network. Transferring between different lines occurs by the help of foot edges.

The edge weights of vehicle edges are represented as average travel time to cover the specified road segment. The average travel time is computed by using statistical travel time of the road segment at time t. These statistics have been gathered from KENTKART Company which operates the contactless cards in all modes of public transportation in Izmir. The edge weights of the foot edges are computed by taking the geographical length of the road segment and assuming an average walking speed s of a pedestrian as 6 km/h.

Whole transportation graph of Izmir can be seen in Figure 5.2. Direction of edges could not be specified because of the scale of drawing. As can be seen, node density increases in city center and remote districts are connected to the center by several transportation lines.

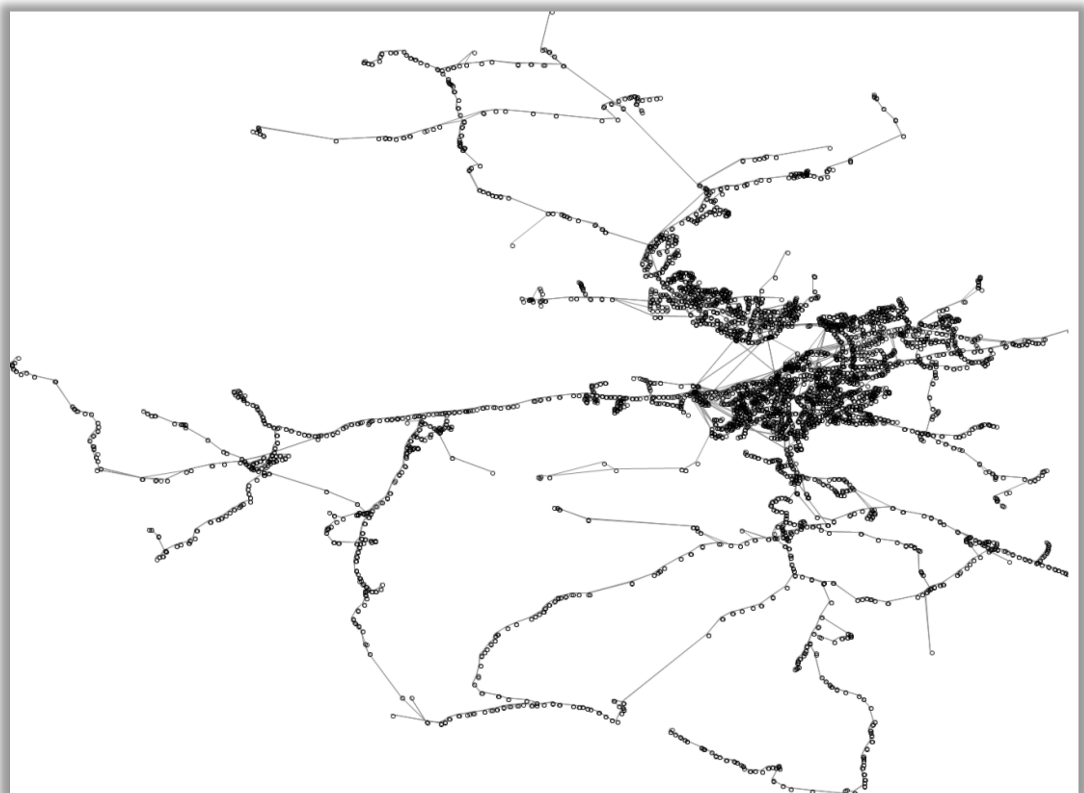


Figure 5.2 Izmir transportation graph.

Transportation graph has been implemented with two main classes: the *Stop* and *Transition* classes which are given in Figure 5.3. Stops are associated with transitions

by *InboundTransitions* and *OutboundTransitions* lists. Transitions are associated with stops by *FromStop* and *ToStop* references.

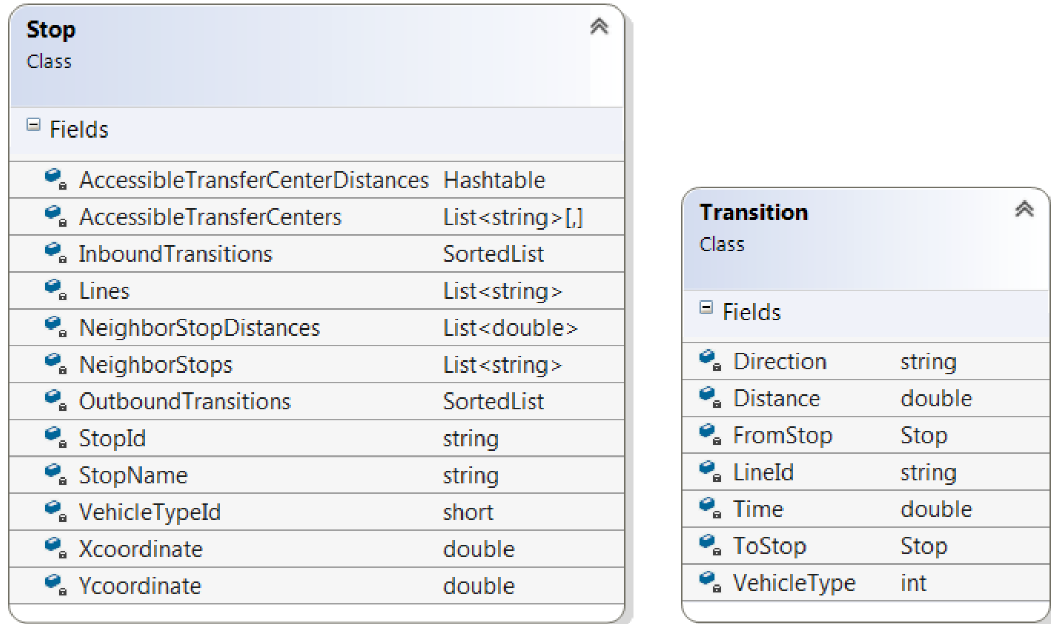


Figure 5.3 Classes designed to construct transportation graph.

Each stop object corresponds to a node and each transition object corresponds to an edge in a transportation graph. Multiple transitions can start and end in a stop as exemplified in Figure 5.4.

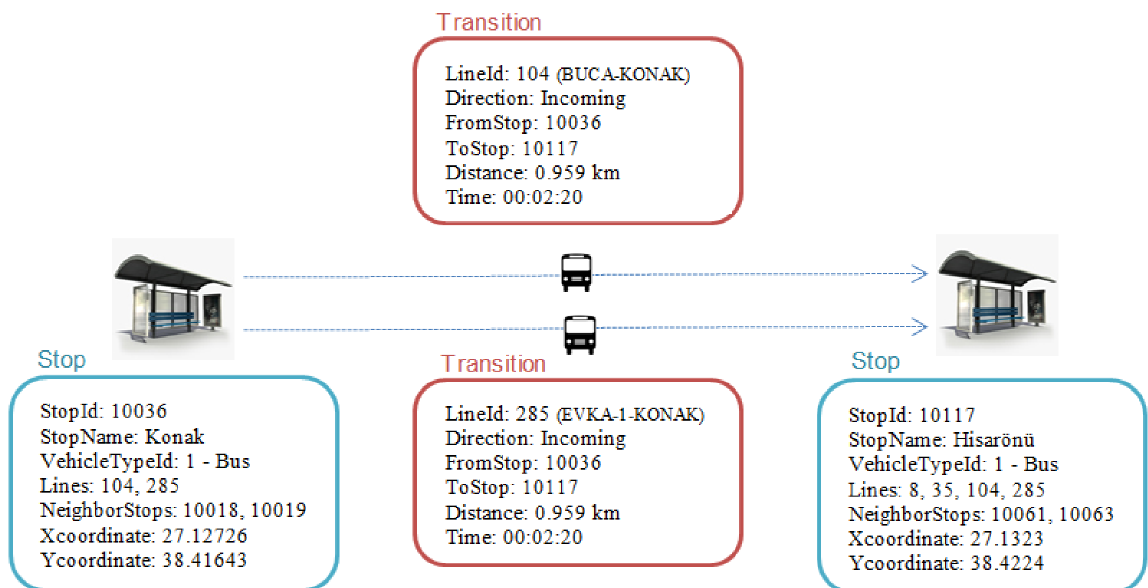


Figure 5.4 Transportation graph with stop and transition objects.

Static transportation data which forms the transportation graph is loaded into the memory once to use during query execution. For each individual query, a new projection graph is constructed by using static graph without any database queries. The nodes of this graph have an id reference of corresponding stop object and extra variables like *minimumDistance* and *previousNode* related with the implementation of Dijksta's Algorithm.

5.3 Gradual Path Finding Algorithm

Finding all possible paths between any two nodes in a graph with large number of nodes is a classic example of problems in the field of computational complexity theory. These type of problems require very large numbers of computations and memory. Real-world implementations of journey planning algorithms involve a tradeoff of computational resource between accuracy and completeness of the answer, and speed of the results. Instead of finding all paths, dealing with just best k path is the preferred technique in general.

In the k shortest paths problem, given a positive integer k>1, we are required to find the k shortest paths from source node to destination node. In this study, we used Dijkstra's algorithm for finding k shortest paths between any pair of nodes in directed transportation graph with non-negative edge weights. Average travel time between two consecutive nodes is used as edge weight. Pseudo code of Dijkstra's algorithm is given in Figure 5.5. The algorithm starts with a priority queue that contains only one node, and inserts new nodes as they are discovered. Process continues with the following steps:

- Step 1. Assign to every node a tentative distance value.
- Step 2. Set the initial node as current. Mark all other nodes unvisited.
- Step 3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
- Step 4. When all of the neighbors of the current node are considered, mark the current node as visited and remove it from the unvisited set.

- Step 5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity then stop. The algorithm has finished.
- Step 6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to Step 3.

```
Data: A weighted graph G = (V, E), s ∈ V and T ⊆ V.
Result: Shortest paths from s to all t ∈ T.

for each vertex v in Graph:
    dist[v] := infinity;
    previous[v] := undefined;
end for

dist[source] := 0;
Q := the set of all nodes in Graph;

while Q is not empty:
    u := vertex in Q with smallest distance in dist[];
    remove u from Q;
    if dist[u] = infinity:
        break;
    end if

    for each neighbor v of u:
        dist_v := dist[u] + dist_between(u, v);
        if dist_v < dist[v]:
            dist[v] := dist_v;
            previous[v] := u ;
            decrease-key v in Q;
        end if
    end for
end while

S := empty sequence
u := target
while previous[u] is defined:
    insert u at the beginning of S
    u := previous[u]
end while ;
```

Figure 5.5 Pseudo code of Dijkstra's algorithm.

According to the transportation experts' opinion, paths with less transfer are almost always preferred by passengers. We developed the Gradual Path Finding Algorithm that produces path primarily with less transfer and then according to other selected criteria. The Gradual Path Finding Algorithm depends on a goal-directed speed-up technique that directs the search toward the target *t* by preferring edges that reach to *t* with fewer transfer counts and by excluding edges that cannot possibly belong to a shortest path to *t*.

Path finding operation is completed in at most five stages. If necessary amount of paths to achieve k-shortest paths could not be found, operation is preceded by the next stage. Then, the stages for departure time calculation of produced paths, filtering and ordering of the results are presented. Stages of Gradual Path Finding Algorithm are given in Figure 5.6. Modified versions of Dijkstra algorithm are used in path determination stages that contain one transfer, two transfers and n transfers (more than two transfers). Detailed implementation of these modified versions will be explained in the next sections.

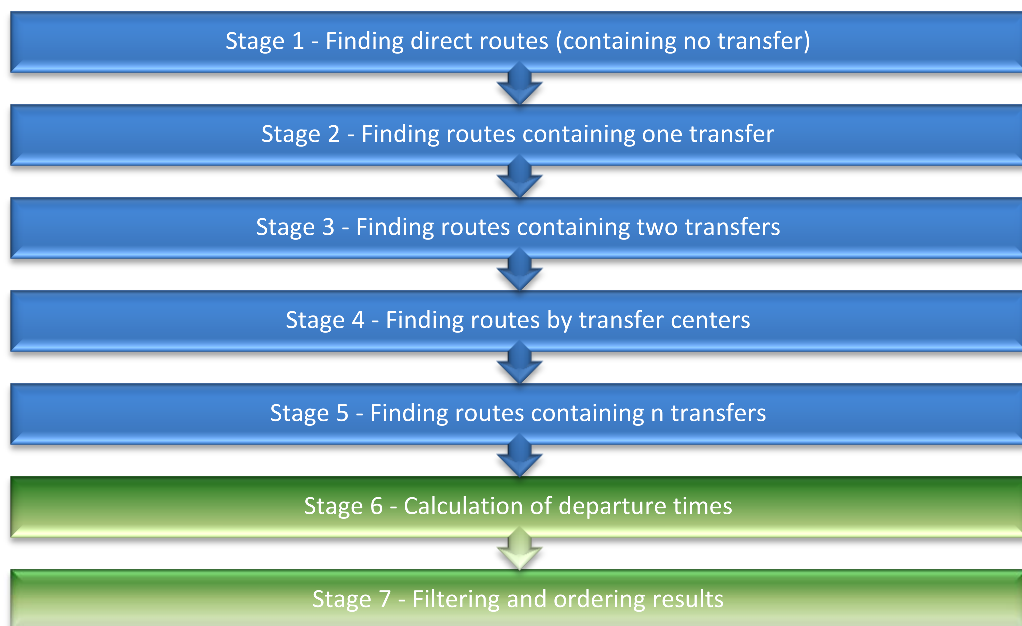


Figure 5.6 Stages of Gradual Path Finding Algorithm.

Before starting path finding process, origin stops S_O and destination stops S_D must be determined. User can determine them by one of the three ways listed below.

- By stop name: An autocomplete list appears when the user types name of origin and destination stops. Selected stop name can correspond to more than one stop located closely. All of the stops with selected name and their neighbor stops become origin stops or destination stops as well.
- By location: When user determines a location by using GPS or map, stops near to that location are calculated by increasing the range until obtaining necessary count of stops. Currently used range values are 150, 250, 400, 500, 750, 1000 meters and these range values are preferred by ESHOT.

- By POI: An autocomplete list appears when user types POI name. We obtain a location by the selection of a POI and origin and destination stops are determined as the selection of stops by location.

After determination of origin and destination stops, we may have multiple origin stops and multiple destination stops. We reduce our problem to single source shortest path problem by running algorithm for each origin stop to all destination stops.

Stop objects in other words nodes in graph have *InboundTransitions* and *OutboundTransitions* lists that have been mentioned before. Origin Lines List is obtained by using outbound transitions of origin stops and Destination Lines List is obtained by using inbound transitions of destination stops. In the next sections we will call origin stop list as S_O, destination stop list as S_D, origin lines list as L_O, and destination lines list as L_D. Obtained routes are kept in *Path* objects. Path objects have properties to store route details as can be seen in Figure 5.7. Each path object has a *SubPaths* list to hold each part of a route.

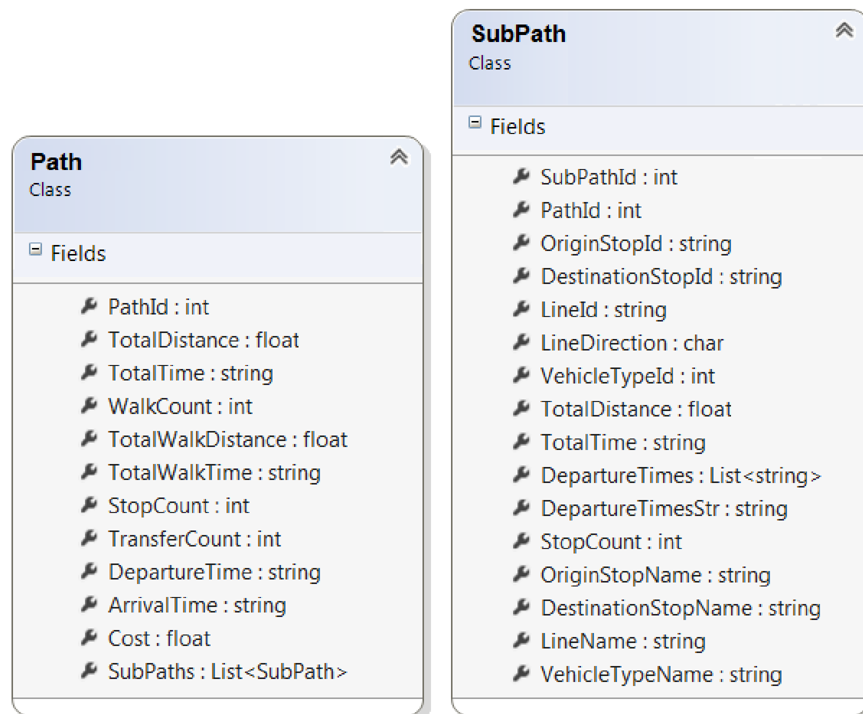


Figure 5.7 Classes designed to hold routes.

5.3.1 *Finding Direct Routes*

Direct paths which have no transfer are calculated by using origin lines list L_O, and destination lines list L_D. For a line *l,* if $l \in L_O$ and $l \in L_D$, then we can say *l* is a line that reaches to the destination from source with no transfer. A *Path* obtained in this stage has only one *SubPath* which contains line *l*.

Each line that forms a direct path is stored in the used-line list L_U to prevent using it in calculation of the paths containing transfer. Direct paths are obtained by using static data without any database queries.

5.3.2 *Finding Routes Containing One Transfer*

In this stage of the algorithm, the aim is to find a path which starts with an origin line and ends up in a destination line. Origin and destination lines must be connected in a transfer stop or a walk must exist between two lines.

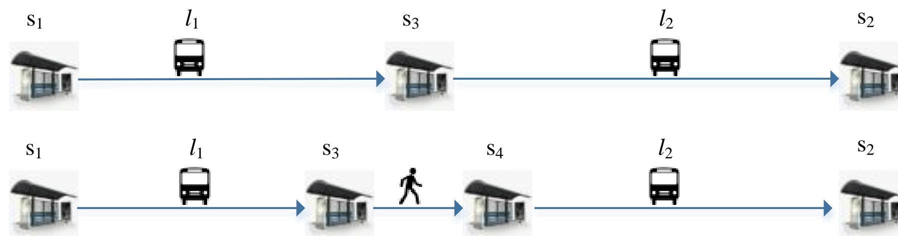


Figure 5.8 Illustration of the routes containing one transfer.

This stage of the algorithm runs once for each origin line. A modified version of the Dijkstra's Algorithm is developed to obtain paths starting with an origin line and ending up in a destination line. The algorithm allows starting with an origin line, transferring to a destination line and arriving to a destination stop with this line as illustrated in Figure 5.8. The rules are:

- $l_1 \in L_O$, $l_2 \in L_D$,
- l_1, $l_2 \notin L_U$,
- $s_1 \in S_O$, $s_2 \in S_D$, and s_3, $s_4 \notin S_O$, S_D.

```
for each node n in Graph:
-n.dist := infinity;
-n.previous := undefined;
end for

source.dist := 0;
Q := Priority queue according to distance;
enqueue source into Q;

while Q is not empty:
-u := node in Q with smallest distance
-remove u from Q;

-for each outbound edge e of node u:
--v := node reachable from u with edge e;
--e.weight := infinity;
--if e.Line not in L_U:
---if u.previous != null:
----prev_e := edge used for reaching to u
----if prev_e.Line in L_D:
-----if e.Line = prev_e.Line:
------e.weight := time of e;
-----end if
----else
-----if e is a line:
------if e reaches a destination stop:
-------e.weight := time of e;
------else if e.Line = prev_e.Line:
-------e.weight:= time of e + transition cost;
------end if
-----else if e is foot-edge:
------e.weight := time of e + walk cost;
-----end if
----end if
---else // u.previous = null:
----if e.Line is l:
-----e.weight := time of e;
----end if
---end if
--end if
-end for

-for each outbound edge e of node u:
--v := node reachable from u with edge e;
--dist_v := u.dist + e.weight;
--if dist_v < v.dist:
---dequeue v from Q with key v.dist;
---v.dist := dist_v;
---v.previous := u;
---enqueue v into Q with key v.dist;
--end if
-end for
end while

S := empty sequence
u := target
while u.previous is defined:
-insert u at the beginning of S
-u := u.previous
end while
```

Figure 5.9 Pseudo code of the modified Dijkstra's Algorithm for the routes that contain one transfer.

In this new algorithm, each node (stop) has a weight vector with the size of its outbound line count. Opposite of a traditional graph, in our approach weight values of edges are not known at the beginning of the process. Weight values are assigned during the run of the algorithm according to the rules. Weights of the edges which do not meet these requirements stay infinity. Minimum cost path that corresponds the rules is obtained as a result and returned as a *Path* object. Pseudo code of the algorithm is given in Figure 5.9.

This algorithm runs once for each start line l. Two cost values *transition cost* and *walk cost* are used in the algorithm. *Transition cost* is used to find paths with fewer stops when *walk cost* is used to find paths with less walk. After finding a path with one transfer, new paths are generated by replacing second line l_2 with its parallel lines. Parallel line means, a line which passes from both l_2's first stop and last stop in the produced path. Then parallel lines are added to the used-line list L_U, too. If necessary count of path could not be found yet, algorithm runs for l_1 again to find an alternative path without using l_2 and its parallel lines. This process continues until producing k paths or a path with one transfer could not be found anymore.

5.3.3 Finding Routes Containing Two Transfers

If necessary count of paths (k paths) could not be found in the first two stages, process continues with finding routes containing two transfers. At the beginning of this stage, *Previous Destination Line List* L_P is constructed with the lines intersect with a destination line at a stop directly or with a walk. The algorithm finds paths which start with an origin line, continue with a previous destination line and end with a destination line. Arbitrary lines can be connected at a transfer stop or a walk could exist between two lines as illustrated in Figure 5.10. The rules are:

- $L_P \cap L_O = \emptyset$ and $L_P \cap L_D = \emptyset$.
- $l_1 \in L_O$, $l_2 \in L_P$, $l_3 \in L_D$,
- $l_1, l_2, l_3 \notin L_U$,
- $s_1 \in S_O$, $s_2 \in S_D$, and $s_3, s_4, s_5, s_6 \notin S_O, S_D$.

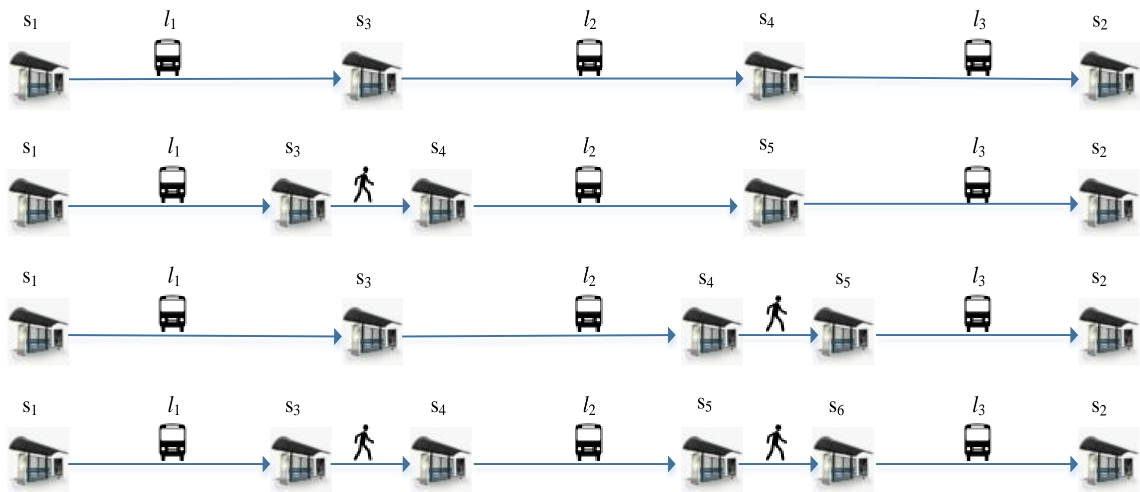


Figure 5.10 Illustration of the routes containing two transfers.

In this version of Dijkstra's Algorithm, each node has a transfer count property in addition to distance property. Transfer count property is used to hold transfer count of the minimum cost path of each node. Pseudo code of algorithm can be seen in Figure 5.11.

After finding a path with two transfers, new paths are generated by replacing second line l_2 with its parallel lines and then by replacing third line l_3 with its parallel lines. Then l_2, l_3 and their parallel lines are added used-line list. If necessary count of path could not be found yet, algorithm runs for l_1 again to find an alternative path without using lines which are in used-line list. This process continues until producing k paths or a path with two transfers could not be found anymore.

```
for each node n in Graph:
-n.dist := infinity;
-n.trCnt := 0; // Transfer Count
-n.previous := undefined;
end for

source.dist := 0;
Q := Priority queue according to distance;
enqueue source into Q;
```

Figure 5.11 Pseudo code of the modified Dijkstra's Algorithm for the routes that contain two transfers.

```
while Q is not empty:
-u := node in Q with smallest distance;
-remove u from Q;
-for each outbound edge e of node u:
--v := node reachable from u with edge e;
--e.weight := infinity;
--if e.Line not in LU:
---if u.previous != null:
----prev_e := edge used for reaching to u;
----if e is a line:
-----if u.trCnt = 2 and e.Line = prev_e.Line:
------e.weight := time of e;
-----else if u.trCnt = 1 and e.Line != prev_e.Line and e.Line in LD:
------e.weight := time of e;
-----else if u.trCnt = 1 and e.Line = prev_e.Line:
------e.weight := time of e;
-----else if u.trCnt = 0 and e.Line != prev_e.Line and e.Line in LP:
------e.weight := time of e;
-----else if u.trCnt = 0 and e.Line = prev_e.Line:
------e.weight := time of e;
-----end if
----else // e is a foot-edge
-----if u.trCnt = 0 and prev_e is a line:
------e.weight := time of e;
-----else if u.trCnt = 1 and prev_e is a line:
------e.weight := time of e;
-----else e.weight:= infinity;
-----end if
----end if
---else // u.previous = null:
----if e.Line is l:
-----e.weight := time of e;
----end if
---end if
--end if
-end for
-for each outbound edge e of node u:
--v := node reachable from u with edge e;
--prev_e := edge used for reaching to u;
--dist_v := u.dist + e.weight;
--if dist_v < v.dist:
---dequeue v from Q with key v.dist;
---v.dist := dist_v;
---v.previous := u;
---v.trCnt := u.trCnt;
---if e.Line != prev_e.Line:
---- v.trCnt := u.trCnt + 1;
---end if
---enqueue v into Q with key dist_v;
--end if
-end for
end while

S := empty sequence;
u := target;
while u.previous is defined:
-insert u at the beginning of S;
-u := u.previous;
end while
```

Figure 5.11 Pseudo code of the modified Dijkstra's Algorithm for the routes that contain two transfers (continued).

5.3.4 Finding Routes by Using Transfer Centers

If necessary count of paths which contain zero, one or two transfers could not be found between source and destination stops, paths with more than two transfers must be searched. Thanks to the traffic infrastructure in Izmir that a person making a trip between two distant points likely will be visiting a transfer center. At this stage, the aim is finding paths that contain more than two transfers and pass from at least one transfer center.

Determination of accessible transfer centers for all stops and path finding preprocess between each transfer center are the tasks accomplished by update service as mentioned before.

Path finding process using transfer centers can be occurring in two cases. In the first case, origin and destination stops have common accessible transfer centers. In this case, paths with zero and one transfer are calculated between origin stop and common transfer center firstly. If there is no path found, then paths with two transfers are calculated. The same process is done between common transfer center and destination stop. It does not be forgotten that a passenger can reach a transfer center with maximum two transfers from any stop. P_1 is the set of the paths found between origin and transfer center and P_2 is the set of the paths found between transfer center and destination as shown in Figure 5.12. Result set P is obtained with cross production of P_1 and P_2, where $p \in P$ and $p \subset P_1 \times P_2$.

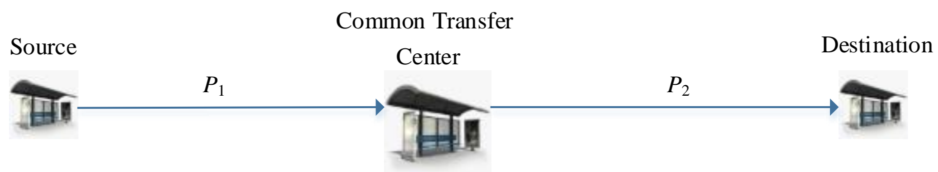


Figure 5.12 Sample path between two stops through one transfer center.

In the second case covers the origin stops and destination stops that do not have any common transfer center. Paths with zero and one transfer are calculated between origin stop and their accessible transfer center T_1, and paths with two transfers are

calculated if necessary. The same is done for transfer center T_2 and destination stops. Paths between two transfer centers are calculated and stored into the database by Update Service. P_1 is the set of the paths found between origin and transfer center T_1, P_2 is the paths between transfer center T_1 and transfer center T_2 stored in the database, and P_3 is the set of the paths found between transfer center T_2 and destination as shown in Figure 5.13. In this case, result set P is obtained by cross production of P_1, P_2 and P_3 where $p \in P$ and $p \subset P_1 \times P_2 \times P_3$.

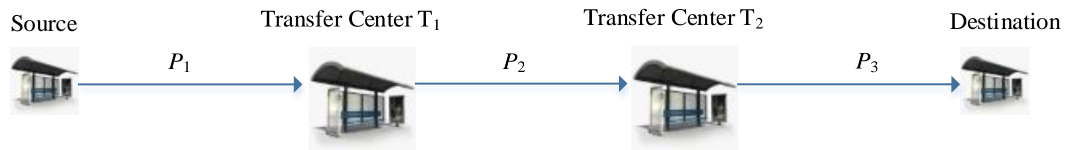


Figure 5.13 Sample path between two stops through two transfer centers.

During cross production, if the destination stop of P_1 is different from the start stop of P_2, a walk is inserted between them. On the other hand, if last line of P_1 is the same as the first line of P_2, then these two parts are combined.

5.3.5 Finding Routes Containing N Transfers

In the first three stages, paths having up to two transfers are obtained. In the previous stage paths through at least one transfer center were obtained and these paths could be containing more than two transfers. If *k* paths could not be produced yet, another modified version of Dijkstra's Algorithm is used to find minimum cost paths. The paths produced in this stage have more than two transfers. This modified algorithm is given in Figure 5.14. Some constants are used to change behavior of the algorithm. *TransferCost* is the constant value added onto the edge weight at each line change. If the transferred line is not in the destination-line list L_D however in the previous-destination-line list L_P, then *coefficient1* is added to the edge weight. If the transferred line is neither in L_D nor in L_P, then *coefficient2* is added onto edge weight. *coefficient2* > *coefficient1* and these values are determined according to the edge weights in graph. *WalkCost* is added onto the weights of the foot-edges, so path consisting less walk also has less cost.

```
for each node n in Graph:
-n.dist := infinity;
-n.previous := undefined;
end for

Q := Priority queue according to distance;
for each source stop s in SO:
-s.dist := 0;
-enqueue s into Q;
end for

while Q is not empty:
-u := node in Q with smallest distance;
-remove u from Q;
-for each outbound edge e of node u:
--v := node reachable from u with edge e;
--e.weight := infinity;
--if e.Line not in LU:
---if u.previous != null:
----prev_e := edge used for reaching to u
----if e is a line:
-----if e.Line != prev_e.Line:
------if e.Line in LD:
-------e.weight := time of e + transferCost;
------else if e.Line in LP:
-------e.weight := time of e + trCost + coeff1;
------else e.weight := time of e + trCost + coeff2;
------end if
-----else // e.Line = prev_e.Line
------e.weight := time of e;
-----end if
----else if e is foot-edge:
-----e.weight := time of e + walk cost;
----end if
---else // u.previous = null:
----e.weight := time of e;
---end if
--end if
-end for

-for each outbound edge e of node u:
--v := node reachable from u with edge e;
--dist_v := u.dist + e.weight;
--if dist_v < v.dist:
 ---dequeue v from Q with key v.dist;
---v.dist := dist_v;
---v.previous := u;
---decrease-key v in Q;
---enqueue v into Q with key dist_v;
--end if
-end for
end while

S := empty sequence;
u := target;
while u.previous is defined:
-insert u at the beginning of S;
-u := u.previous;
end while;
```

Figure 5.14 Pseudo code of the modified Dijkstra's Algorithm for routes containing n transfers.

5.3.6 Calculation of the Departure Times

After we obtained *k* paths at the end of the first five stages, the stage of departure time calculation begins. System offers three departure times for each route according to the journey start time. Calculation of the departure times is illustrated in Figure 5.15. This example route has one transfer and starts with Line A and ends up with Line B. If we suppose journey start time is determined as 10:30 by the user, departure times are calculated as explained in below:

- If the user selected origin is not the first stop of Line A, leaving time of the vehicle from the initial stop is calculated by going back, stop by stop. Average travel time between each arbitrary stop pair is subtracted from the journey start time. Average travel times are obtained from statistics according to the selected time of day.

The time to leave from the initial stop of Line A, is obtained as 10:18:12 to be in origin at 10:30.

- First three departure times after the time to leave initial stop are determined and arrival times to origin are calculated for these three departures.

10:28:00, 10:40:00 and 10:52:00 are determined as departure times from initial stop, and arrival times to origin for these three departures are obtained as 10:39:48, 10:51:48 and 11:04:01, respectively.

- Progress is advanced by starting from the origin and forwarding stop by stop until arriving to the transfer stop. Average travel time between each arbitrary stop pair is added to the origin's first departure time.

A vehicle that is leaving from the origin at 10:39:48, arrives to the transfer stop at 10:46:14.

If the current sub path is not the final sub path, transfer stop is assumed as the origin and arrival time to transfer stop is assumed as journey start time and steps 1, 2 and 3 are repeated until arriving to the destination.

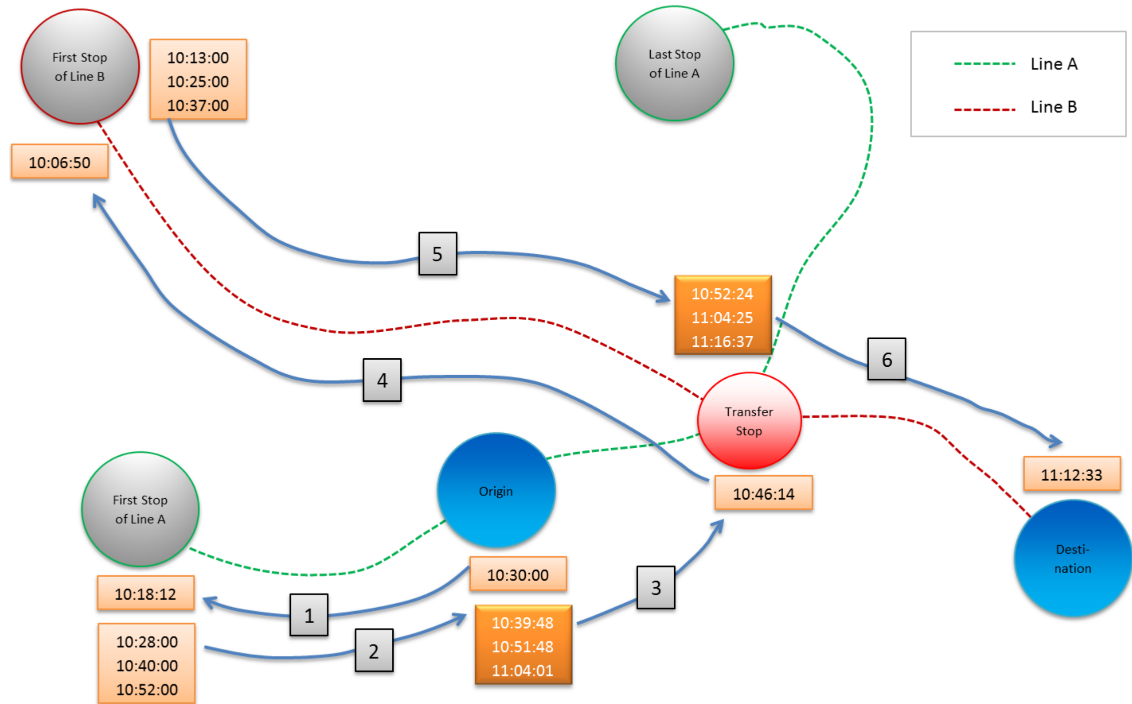


Figure 5.15 Illustration of sample departure time calculation

While back warding to the initial stop of a line and subtracting average time between stops in step 1, it is important to remember that the time may change to the day before. In the same manner, while forwarding stop by stop and adding average time between stops in step 2 and 3, it can be change to next day. In these two situations, corresponding time tables are used.

5.3.7 Filtering the Alternatives

Gradual Path Finding Algorithm produces at least *k* paths. If *k* paths could not be produced yet, algorithm continues with the next stage to produce at least *k* paths. The number of the produced paths can be exceeding *k* by the paths produced at last stage. Because, stages do not end when produced path count reaches to *k*.

The number of the paths returned by the Web Service is changeable and at most equals to 10. For example, if the best path has no transfer or one transfer and the second best path has three or more transfers, then only one result can be returned. In Gradual Path Finding Algorithm, result paths are produced in ascending order by the number of transfers. Then, the paths are ordered by the travel cost or time spent according to the user's preference and best 10 of them are returned.

CHAPTER SIX
EXPERIMENTATIONS

6.1 Evaluation Measures

Success of the algorithm can be evaluated according to three different criteria. Evaluation measures consist of accuracy and completeness of the answer, computational resource usage, and the response time of the system.

Accuracy and completeness of the answer have been verified by the experts from the transportation agencies, firstly. Some improvements have been taken into account according to their feedback. The users of the system are still going to send their feedbacks by using the applications in use. IJPS is a live system which is being improved.

Search space reduction of the path finding algorithm has been analyzed to evaluate computational resource usage. Average response time of the algorithm calculated by using the test dataset which will be explained in the next section.

6.2 Test Dataset

In order to verify the accuracy, reliability and consistency of the IJPS, a test data table has been created. The necessary data has been collected by executing thousands of sample queries. Origin and destination stops, total query execution time, produced result count, time spent and count of the paths that are produced at each stage are the properties recorded for each query.

All properties and some example tuples of the test data can be found in Figure 6.1. From each first stop of all the available bus, train, metro, and ferry lines to each last stop of the lines, a path finding query has been executed as shown in Figure 6.2. A total of 96,107 query results have been recorded in to target the data table.

FERISTAH-PC\SQLEX...dbo.IJPS_TST_Test

TestId	OriginStopIdId	DestinationStopId	Time	TotalPathCount	DirectPathCount	OneTransferPathCount	TwoTransfersPathCount	TransferCenterPathCount	NTransfersPathCount
1	13016	22192	00:00:00.6821581	3	3	0	13	0	0
2	13016	13015	00:00:00.9784421	10	1	0	0	291	0
3	13016	100021	00:00:00.1099912	1	1	5	0	0	0
4	13016	100029	00:00:00.1788916	10	2	15	0	0	0
5	13016	1000232	00:00:00.1188485	4	2	3	0	0	0
6	13016	100024	00:00:00.9825712	1	2	2	40	0	0
7	13016	1000312	00:00:00.3915519	8	0	12	0	0	0
8	13016	1000315	00:00:04.8033591	10	1	3	88	0	0

TimeDirect	TimeOneTransfer	TimeTransferCenter	TimeAktarmaMerkezli	TimeNTransfers	BusExist	IzbanExist	MetroExist	FerryExist	WalkExist	Distance
00:00:00.0066617	00:00:00.0426609	00:00:00.4514018	00:00:00.0000004	00:00:00.0000012	True	True	False	True	True	32,80415726
00:00:00.0000652	00:00:00.0256153	00:00:00.0875075	00:00:00.5399772	00:00:00.0000008	True	True	False	False	True	0,018788295
00:00:00.0001459	00:00:00.0786852	00:00:00.0000004	00:00:00.0000004	00:00:00.0000004	False	True	False	False	True	74,84560394
00:00:00.0002448	00:00:00.0949539	00:00:00.0000004	00:00:00.0000004	00:00:00.0000004	True	True	False	False	True	34,71960831
00:00:00.0000652	00:00:00.0511734	00:00:00.0000004	00:00:00.0000008	00:00:00.0000004	True	True	False	False	True	3,792606592
00:00:00.0002662	00:00:00.0377953	00:00:00.8531882	00:00:00.0000008	00:00:00.0000004	False	True	False	False	True	49,2746048
00:00:00.0000046	00:00:00.3429631	00:00:00.0000004	00:00:00.0000004	00:00:00.0000004	True	True	True	False	True	25,52355957
00:00:00.0000793	00:00:00.1220863	00:00:04.4679580	00:00:00.0000004	00:00:00	True	True	True	False	True	21,04958534

Figure 6.1 Example tuples of the test data.

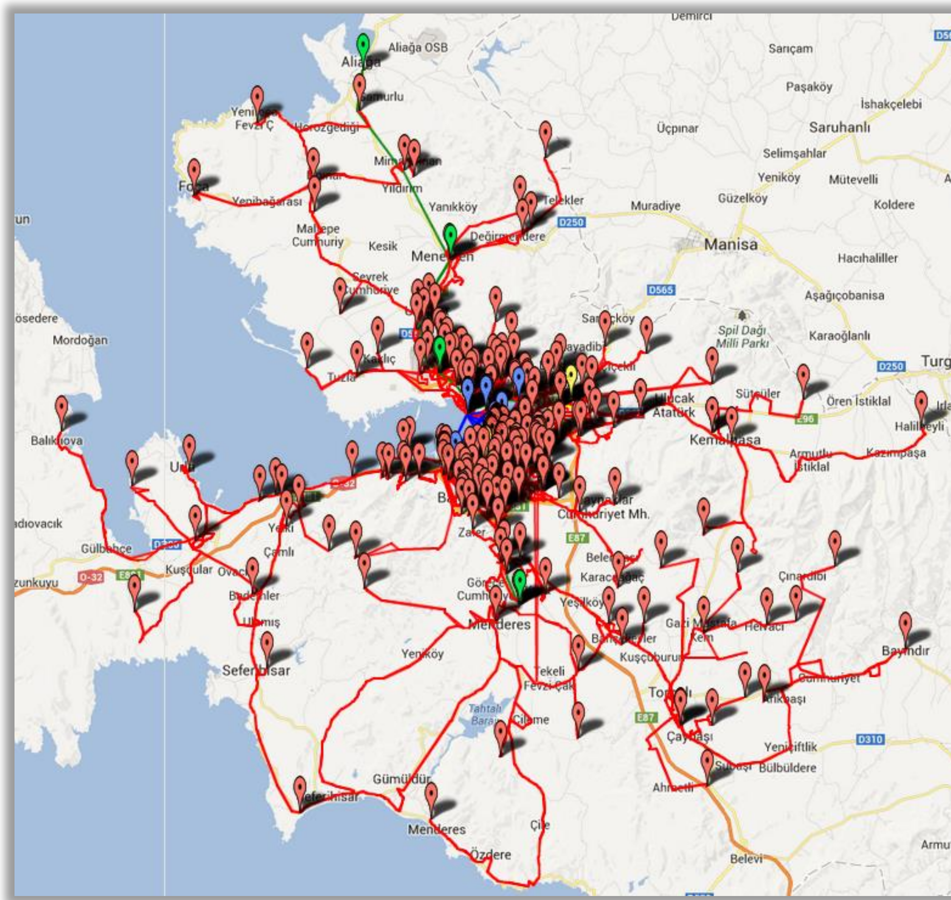


Figure 6.2 First and last stops of the available bus (red), train (green), metro (yellow) and ferry (blue) lines.

6.3 Search Space

In this section, we present an experimental study investigating the reduction of the search space achieved by the modification on the Dijkstra's algorithm. A sample query for O-D pair "31770-Evka 3 Aktarma Merkezi" and "40015-Buca Belediye Sarayı" is executed to evaluate the search spaces of the stages in which Dijkstra's algorithm is used. In Stage 1, memory look-ups are sufficient to find the direct routes. In Stages 2 to 5, Dijkstra's algorithm is used with different modifications.

The implementation details of the algorithms have been given in Section 5.3. While the algorithm is limited to find only the paths containing one transfer in Stage 2, it is limited for the paths containing two transfers in Stage 3. Shortest paths through the nearest transfer centers are searched in Stage 4. In Stage 5, there is no transfer count or transfer center restriction and the algorithm is modified to find the shortest paths with the minimum number of transfers and walk as possible.

Table 6.1 shows the visited edge counts for the modified Dijkstra's algorithms used in Stages 2-5. Visited edge counts are increasing over the upper stages of the algorithm. Increase in the number of visited edges also increases the runtime of the algorithm. Runtimes will be discussed in the next section.

Table 6.1 Visited edge counts for the modified and Pure Dijkstra's algorithms

	Visited Edges	All Edges	Ratio %
Stage 2	364	27,506	1.32
Stage 3	2,169		7.89
Stage 4	5,833		21.21
Stage 5	21,288		77.39
Pure Dijkstra's Alg.	27,464		99.85

A large scale and zoomed visualizations of the search spaces corresponding to the sample query and the optimal paths obtained in Stages 2, 3, 4 and 5 are given in Figure 6.3, 6.4, 6.5 and 6.6, respectively. Black colored lines indicate the traversed edges and the red colored ones show the optimal path obtained.

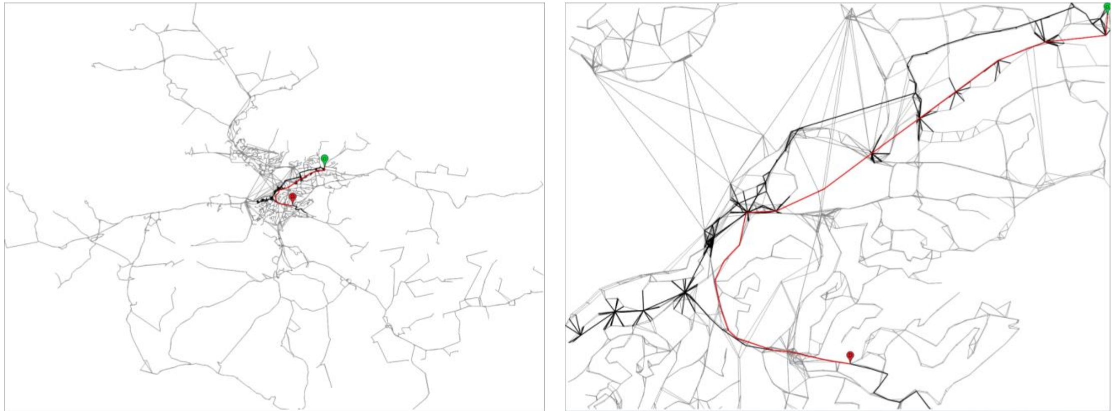


Figure 6.3 A visualization of the search space for Stage 2.

The rules applied in Stage 2 reduce the search space by restricting the traversal of the nodes corresponded to the stops of the origin lines and the destination lines. In stage 3, the search space is reduced to the nodes corresponded to the stops of origin lines, destination lines and the lines connected to them. Reducing this large search space such these extents makes an essential sense on the performance of the algorithm.

Figure 6.4 A visualization of the search space for Stage 3.

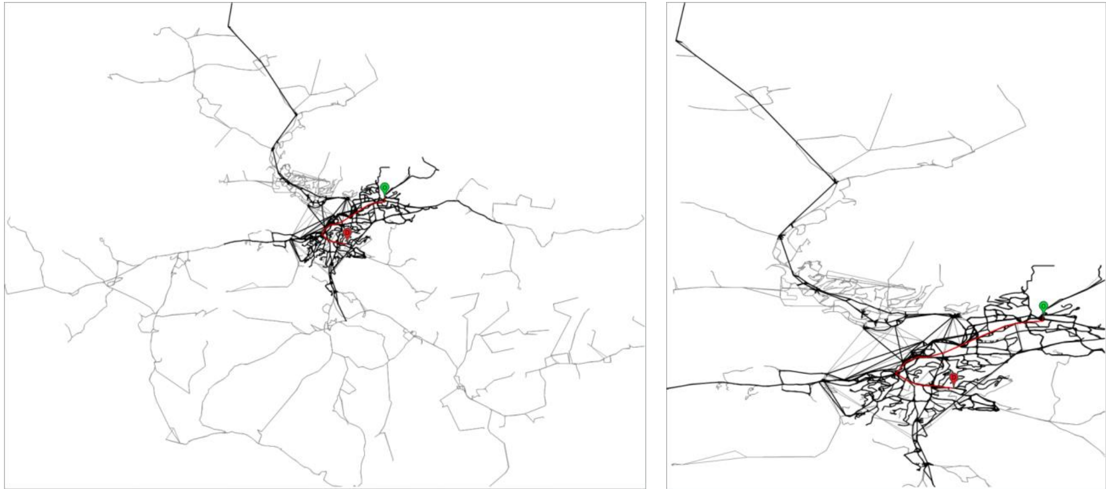


Figure 6.5 A visualization of the search space for Stage 4.

Because the transportation graph is a multi-graph, some u,v node pairs have multiple edges connected to them. Figure 6.6 shows a visualization of the search space for Stage 5. It seems that, the algorithm searches the whole graph, but indeed it only visits 77.39% of all the edges. Visited edges put unvisited parallel edges out of sight.

Figure 6.6 A visualization of the search space for Stage 5.

Goal-directed search is a very useful technique for transportation networks as it simply modifies the edge weights on-the-fly and reduces the search space of the Dijkstra's algorithm, by decreasing the number of the visited nodes and edges. Actually, we are able to reduce the search space to 1.32%, 7.89%, 21.21%, and 77.39% for Stages 2-5, respectively while Pure Dijkstra's Algorithm visits 99.85% of all the edges.

6.4 Experimental Results

When we examine the query results recorded for the test dataset, some inferences can be obtained. Detailed statistical information gathered from the query result is given in Table 6.2. Executed query count, average runtime and average produced result count are calculated for each stage.

Table 6.2 Average runtime and result counts for stages of the Gradual Path Finding Algorithm.

Stage	# of executed queries (Ratio to all queries %)	Avg. runtime(s)	# of queries that generate result (Ratio to executed queries in this stage %)	Avg. result count
	96,107 (100.0%)	0.00005	15,738 (16.4%)	2.24
2	96,107 (100.0%)	0.08743	47,319 (49.2%)	20.46
3	66,500 (69.2%)	0.37313	47,434 (71.3%)	28.92
4	28,905 (30.1%)	0.77419	27,392 (94.8%)	949.27
5	777 (0.8%)	8.47079	91 (11.7%)	2.38

Stage 1 and Stage 2 have been executed for 100% of all the queries independent from the result count. The operation has continued with Stage 3 for 69.2% of the queries because of k-shortest path (k equals to 5 in this case) could not be found in the first two stages. After Stage 3, 30.1% of the queries has been processed in Stage 4 to achieve k-shortest path (k equals to 5 in this case, too). Only 0.8% of the queries that could not produce at least 3 paths have been continued with Stage 5 as can be seen in Figure 6.7. Executed stage count differs for each query. Approximately 30% of all the queries end in the first two stages while approximately 70% of them end in the third stage.

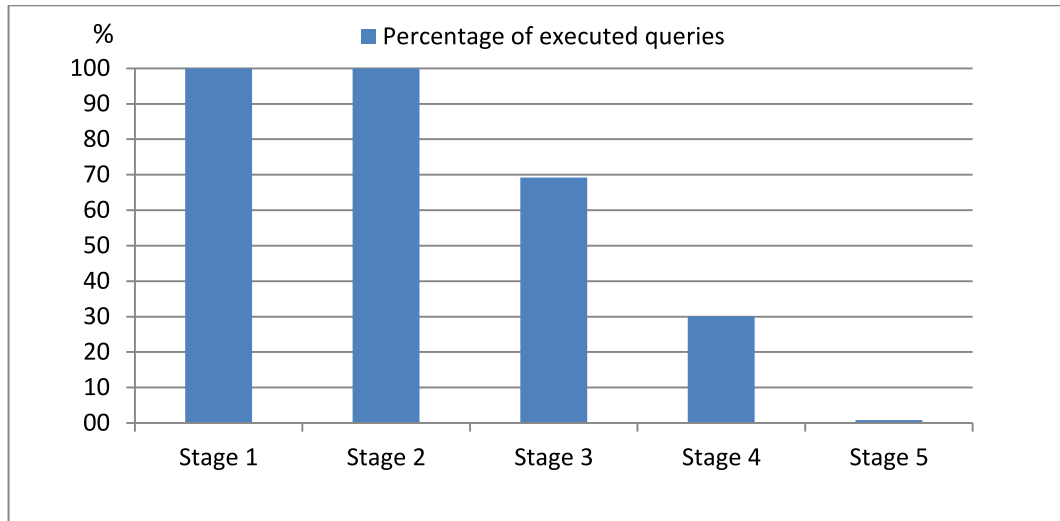


Figure 6.7 Percentages of executed queries for all stages.

In the first four stages average runtimes have been observed under 1 second. But in Stage 5, average runtime has been observed as 8.47 s as shown in Figure 6.8. The fact under that is the similarity of the modified Dijkstra's algorithm used in Stage 5 with the pure Dijkstra's algorithm. It should be noted that all runtimes contain several runs of modified Dijkstra's algorithms. How many times Dijkstra's algorithm run depends on the origin stop count and the origin line count which are explained in Chapter 5.

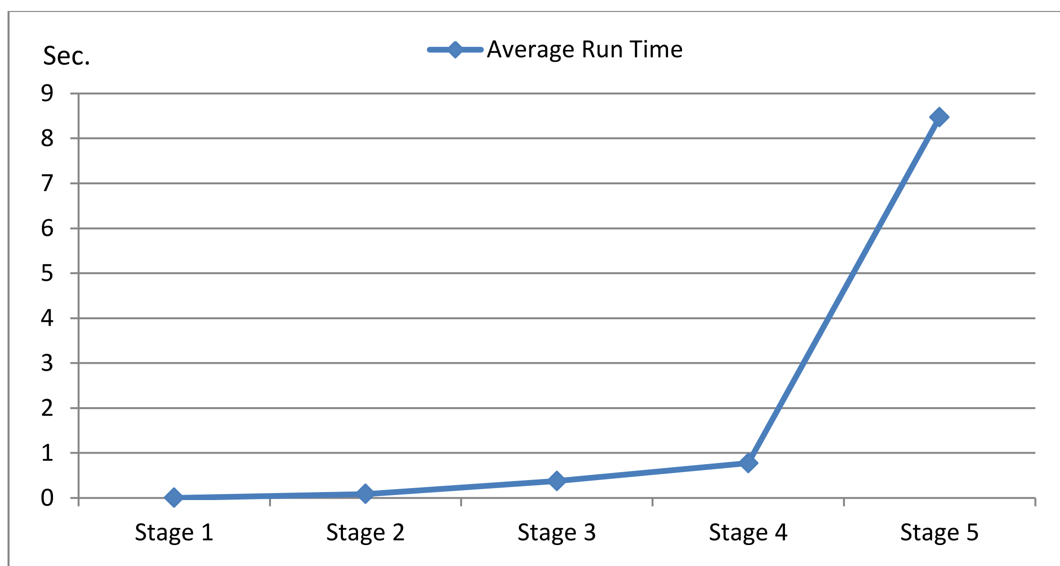


Figure 6.8 Average runtimes for all stages.

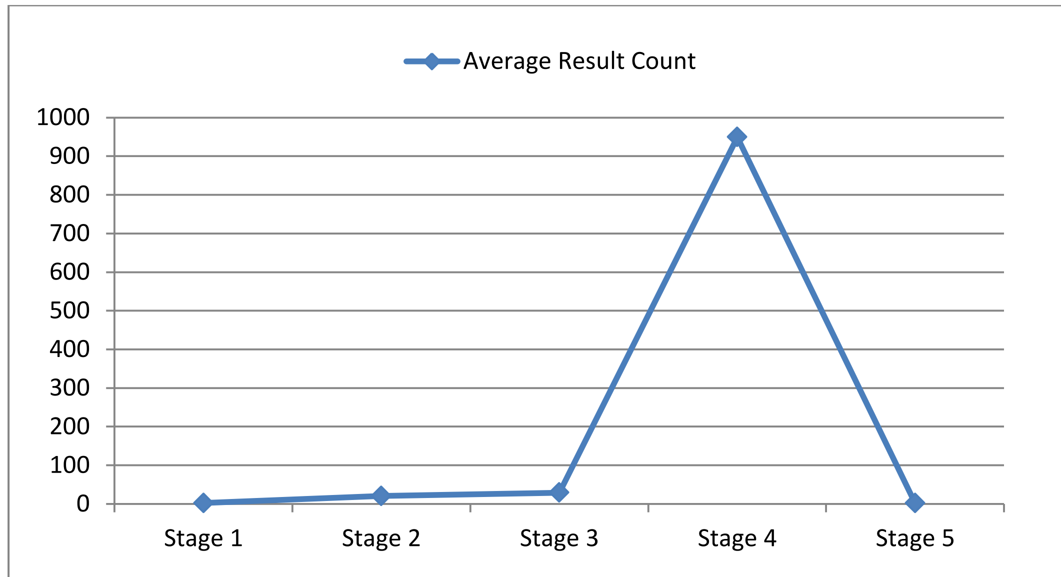


Figure 6.9 Average result counts for all stages.

In stage 1, 2.24 direct paths have been produced by memory look-ups without using Dijkstra's algorithm. In stage 2 and 3 modified Dijkstra's algorithms produced 20.46 and 28.92 result paths in average, respectively. Figure 6.9 shows the average result count graph for all the stages. A pick point attracts the attention as the result count obtained in Stage 4. The average result count obtained in this stage is observed as 949.27, thanks to the cross production of the paths found for each route segments.

When we look at the overall picture, IJPS produces accurate results for all the queries in an acceptable response time for the users of the system, as we target.

CHAPTER SEVEN
CONCLUSION

7.1 Conclusion

In this thesis, we have designed and implemented the service-oriented and inter-model Intelligent Journey Planner System to assist travelers for planning their journey. There are various alternative sources of transportation and it is not easy to integrate information from these diverse sources to plan a journey. In many countries and cities, journey planning systems are in use. But, at the beginning of this study, there was no such a service in Izmir. A system that assists people in making better use of public transportation was a necessity. Now, IJPS supplies information about urban transportation for domestic and foreign visitors in Izmir. So those, passengers can minimize the cost of their journeys and save travel time. In this study, we also aimed to reduce dependence on the car and encourage greater use of public transport. Another purpose was to reduce noise and carbon dioxide emissions, and avoid traffic jump and congestion by promoting passengers to public transport.

IJPS advices alternative routes, transfer details, departure and arrival times according to the user's choice for any determined origin–destination point. The system produces optimal routes according to multiple criteria like preferred mode, maximum distance to walk, least changes, shortest travelling time or lowest fare. The IJPS runs on different kinds of platforms to provide wide range of usage at anytime and anywhere. Specific applications have been developed for iOS, Android and Windows Phone mobile operating systems. In addition to these mobile applications, a mobile web application is available for the remaining mobile platforms which are not commonly used. Web application module of IJPS has been designed in accordance with all common web browsers. IJPS is available in both English and Turkish.

In order to create a more social and intellectual community, citizens should be aware of the social activities and events in the city. To this end, IJPS informs the

users about activity centers and events on the route. User can display selected path, point of interests, activity centers and events on the map. Weather forecast, traffic and road condition, and approximate taxi fare are other informative details produced by the IJPS. Also, another objective of this study is to increase the use of technology anywhere anytime in the public by especially supporting mobile platforms.

In scope of this thesis, two main services and some auxiliary services have been developed. The Update Service has been developed as a windows service application. The Journey Planner Web Service and other auxiliary services have been developed as WCF web services. Update service is a windows service application that runs automatically on the database server every night to collect actual data from transportations agencies. After the update of the transportation data, some preprocessing tasks are executed by the Update service. The Journey Planner Web Service forms the core of the IJPS. Client applications use this service for path finding. The main method is the route finding method which takes the source and destination points and the user preferences as parameters and returns the calculated routes ordered by the preferred criteria.

Route finding problem in a complex multi-model network is one of the most studied areas and still needs to be improved. While planning a route in such a combined network, some constraints as switching the mode of transportation frequently or unacceptable transfer counts must be considered and producing an undesired path should be avoided. Dijkstra's algorithm forms the basis of the modern journey planner search algorithms and provides an optimal solution to simple searches. In this thesis, we propose a new path finding algorithm named the Gradual Path Finding Algorithm. Modified versions of the Dijkstra’s algorithm are used in several stages of the algorithm.

During the last years, the GTFS has become the most popular format to describe static schedule data of transit networks. Both official and user-generated feeds are available for many transit agencies around the world. Relying on a common data format makes an application could work in all transit systems for which open transit

data has been released and a common data is available to any developer to use. Therefore, IJPS stands on the GTFS format. An application has been developed to load any GTFS feed into the IJPS database system and to convert Izmir transport data into the GTFS format. One of the practical results of this study is the GTFS feed for Izmir.

In order to verify the accuracy, reliability and consistency of the IJPS, a test data table has been created. The necessary data has been collected by executing thousands of sample queries. Experimental studies showed that visited edge counts are increasing over the upper stages of the algorithm. Increase in the number of visited edges also increases the runtime of the algorithm. In the first four stages runtimes have been observed under one second. But in Stage 5, average runtime has been observed as 8.47 s. The fact under that is the similarity of the modified Dijkstra's algorithm used in Stage 5 with the pure Dijkstra's algorithm. Edge weights have been assigned on-the-fly and the search space of Dijkstra's algorithm has been reduced by decreasing the number of visited nodes and edges. The reduced search spaces have been observed as 1.32%, 7.89%, 21.21%, and 77.39% for Stages 2-5 respectively while Pure Dijkstra's Algorithm visits 99.85% of all edges.

7.2 Future Works

IJPS is a flexible system which can include any other public transportation mode. In future, new transportation modes that will be operated in Izmir can be integrated into the system. Also, the IJPS can be experimented in any other city. Performance of IJPS can be increased by combination of other multi-model speed-up techniques. Applications are going to be upgraded by the new versions of the mobile platforms.

REFERENCES

Antrim, A., & Barbeau, S. J. (2013). The many uses of GTFS data – Opening the door to transit and multimodal applications. *ITS America's 23rd Annual Meeting & Exposition, Nashville, Tennessee*.

Barrett, C., Bisset, K., Holzer, M., Konjevod, G., Marathe, M. V., & Wagner, D. (2008). Engineering label-constrained shortest-path algorithms. *4th International Conference of Algorithmic Aspects in Information and Management, Shanghai, China*, 23-25.

Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., & Wagner, D. (2010). Combining hierarchical and goal-directed speed-Up techniques for Dijkstra's algorithm. *ACM Journal of Experimental Algorithmics*, *15 (2.5),* 1–31.

Bellman, R. (1958). On a routing problem. *The Quarterly of Applied Mathematics, 16*, 87-90.

Brander, A. W., & Sinclair, M. C. (1995). A comparative study of k-shortest path algorithms. *11th UK Performance Engineering Workshop*, 370-379.

Brodal, G. S., & Jacob, R. (2004). Time-dependent networks as models to achieve fast exact time-table queries. *3rd Workshop on Algorithmic Methods and Models for Optimization of Railways, Electronic Notes in Theoretical Computer Science*, *Elsevier, 92*, 3-15.

Delling, D., Holzer, M., Müller, K., Schulz, F., & Wagner, D. (2006). High-performance multi-level graphs. *9th DIMACS Implementation Challenge*, 52-65.

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2009). Engineering route planning algorithms. *Algorithmics of Large and Complex Networks, Springer, Berlin,* 117–139.

Delling, D., Pajor, T., & Wagner, D. (2009). Accelerating multi-modal route planning by access-nodes. *17th Annual European Symposium on Algorithms (ESA'09), Lecture Notes in Computer Science*, *5757*, 587–598.

Dibbelt, J., Pajor, P., & Wagner, D. (2012). User-constrained multi-modal route planning. *14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, 118–129.

Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1 (1),* 269–271.

Disser, Y., Müller-Hannemann, M., & Schnee, M. (2008). Multi-criteria shortest paths in time-dependent train networks. *7th International Workshop on Experimental Algorithms, Lecture Notes in Computer Science*, *5038*, 347–361.

Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communications of the ACM, 5 (6)*, 345.

General Transit Feed Specification Reference (2007). Retrieved April 8, 2013, from https://developers.google.com/transit/gtfs/reference.

Goldberg, A. V., & Harrelson, C. (2005). Computing the shortest path: A* search meets graph theory. *16th Annual ACM– SIAM Symposium on Discrete Algorithms (SODA'05),* 156–165.

Goldberg, A. V., & Werneck, R. F. (2005). Computing point-to-point shortest paths from external memory. *7th Workshop on Algorithm Engineering and Experiments (ALENEX'05),* 26–40.

Google Transit (2011). Retrieved May 14, 2013, from http://maps.google.com/intl/en/landing/transit/#mdy.

GTFS-realtime Reference (2011). Retrieved April 8, 2013, from https://developers.google.com/transit/gtfs-realtime/reference.

Hadjiconstantinou, E., & Christofides, N. (1999). An efficient implementation of an algorithm for finding K shortest simple paths. *Networks, 34 (2),* 88–101.

Hart, P., Nilsson, N., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4(2)*, 100-107.

Hoffman, W., & Pavley, R. (1959). A method for the solution of the nth best path problem. *Journal of the Association for Computing Machinery (ACM), 6,* 506–514.

Holzer, M. (2003). Hierarchical speed-up techniques for shortest-path algorithms. *Technical report, Departmaent of Informatics, University of Konstanz, Germany*.

Holzer, M., Schulz, F., & Wagner, D. (2006). Engineering multi-level overlay graphs for shortest-path queries. *8th Workshop on Algorithm Engineering and Experiments (ALENEX'06), Philadelphia*, 156–170.

Holzer, M., Schulz, F., Wagner, D., & Willhalm, T. (2006). Combining speed-up techniques for shortest-path computations. *ACM Journal of Experimental Algorithmics, 10 (2.5).*

Johnson, D. B. (1977). Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM, 24 (1),* 1–13.

Katoh, N., Ibaraki, T., & Mine, H. (1982). An efficient algorithm for k shortest simple paths. *Networks, 12*, 411–427.

Kirchler, D., Liberti, L., Pajor, T., & Calvo, R. W. (2011). UniALT for regular language constraint shortest paths on a multi-modal transportation network. *11th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'11), 20*, 64–75.

Lauther, U. (2004). An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. *Geoinformation und Mobilitat - von der Forschung zur praktischen Anwendung, 22*, 219–230.

Lawler, E. L. (1972). A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science, 18*, 401–405.

Martins, E., & Pascoal, M. (2003). A new implementation of Yen's ranking loopless paths algorithm. *Quarterly journal of the Belgian, French and Italian Operations Research Societies, 1 (2),* 121–134.

Mendelzon, A. O., & Wood, P. T. (1995). Finding regular simple paths in graph databases. *SIAM Journal on Computing, 24 (6),* 1235–1258.

Muller–Hannemann, M., & Schnee, M. (2007). Finding all attractive train connections by multi-criteria pareto search. *In Algorithmic Methods for Railway Optimization, Lecture Notes in Computer Science, 4359*, 246–263.

Muller-Hannemann, M., Schnee, M., & Weihe K. (2002). Getting train timetables into the main storage. *2nd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2002), Elsevier Electronic Notes in Theoretical Computer Science, 66.*

Muller-Hannemann, M., & Weihe K. (2001). Pareto shortest paths is often feasible in practice. *5th International Workshop of Algorithm Engineering (WAE 2001), Lecture Notes in Computer Science, 2141,* 185-197.

Nachtigal, K. (1995). Time depending shortest-path problems with applications to railway networks. *European Journal of Operations Research, 83*, 154–166.

Open Trip Planner (2009). Retrieved April 9, 2013, from http://opentripplanner.org/

Orda, A., & Rom, R. (1990). Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM, 37 (3),* 607-625.

Orda, A., & Rom, R. (1991). Minimum weight paths in time-dependent networks. *Networks, 21,* 295-319.

Pohl, I. (1969). Bi-directional and heuristic search in path problems. *Technical Report 104, Stanford Linear Accelerator Center, Stanford, California.*

Pyrga, E., Schulz, F., Wagner, D., & Zaroliagis C. (2008). Efficient models for timetable ınformation in public transportation systems. *ACM Journal of Experimental Algorithmics, 12 (2.4).*

Pyrga, E., Schulz, F., Wagner, D., & Zaroliagis C. (2004). Experimental comparison of shortest path approaches for timetable information. *6th Workshop on Algorithm Engineering and Experiments,* 88–99.

Sanders, P., & Schultes, D. (2005). Highway hierarchies hasten exact shortest path queries. *13th Annual European Symposium (ESA 2005), Lecture Notes in Computer Science, 3669,* 568–579.

Sanders, P., & Schultes, D. (2006). Engineering highway hierarchies. *14th Annual European Symposium (ESA 2006), Lecture Notes in Computer Science, 4168,* 804-816.

Schulz, F., Wagner, D., & Weihe, K. (2000). Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM Journal of Experimental Algorithmics 5, 12.*

Schulz, F., Wagner, D., & Zaroliagis, C. (2002). Using multi-level graphs for timetable information in railway systems. *4th Workshop on Algorithm Engineering and Experiments (ALENEX), Lecture Notes in Computer Science, 2409*, 43–59.

Services (2014). *Microsoft Developer Network*. Microsoft. Retrieved December 15, 2014,from:http://msdn.microsoft.com/enus/library/windows/desktop/ms685141%28v=vs.85%29.aspx

The European Reference Data Model for Public Transport (2001). Retrieved Jun 19, 2013, from http://www.transmodel.org/.

The Open Transport Vocabulary (2014). Retrieved January 7, 2015, from https://github.com/opentransport/vocabulary.

Turkish Statistical Institute (2013). *Turkey's statistical yearbook*. Retrieved January 15, 2015, from http://www.tuik.gov.tr/Kitap.do?metod=KitapDetay&KT_ID=0&KITAP_ID=1.

UML Diagram of the Google Transit Feed Specification (2007). Retrieved April 8, 2013,from:http://www.google.com/help/hc/images/transitpartners_1106431_objecttablelarge_en.gif.

Warshall, S. (1962). A theorem on boolean matrices. *Journal of ACM, 9*, 11-12.

What Is Windows Communication Foundation (2014). *Microsoft Developer Network. Microsoft.* Retrieved December 18, 2014, from http://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx

Yen, J. Y. (1971). Finding the K shortest loopless paths in a network. *Management Science, 17,* 712–716..