A COLLABORATIVE SYSTEM FOR PROVIDING ROUTES BETWEEN LOCATIONS

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
K. Ali Uluğ
June, 2008

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. David Davenport (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Dr. Markus Schaal (Co-Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. H. Murat Karamüftüoğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Savio S. H. Tse

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Dr. Kıvanç Dinçer

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

A COLLABORATIVE SYSTEM FOR PROVIDING ROUTES BETWEEN LOCATIONS

Kerem Ali Uluğ
M.S. in Computer Engineering
Supervisor: Asst. Prof. Dr. David Davenport
Co-Supervisor: Dr. Markus Schaal
June, 2008

Many systems, such as in-car GPS devices and airline company web sites, provide route information between locations. Although such systems are used widely and can provide route information successfully, users of these systems cannot contribute to the data entry process. In these systems, data is entered by the administrators and these systems cannot take advantage of the route expertise of their users.

In this work, we present a collaborative system, which provides routes between locations upon user queries. The data in the system is entered by the users of the system. We present a model which is containing locations, links between locations and relationships between locations (containment, neighborhood and intersection) in order to store the data. For the route finding purpose, we present a customized version of the A* search algorithm. This customized version, named A*CD (A* for Collaborative Data), uses heuristics for estimating the cost remaining to the target location while processing the nodes. A*CD can also provide alternative routes, exclude certain link types in the searches according to user preferences and handle the problems associated with multiple stop transportation lines. As the cost models, we use duration and financial cost.

We also present the intuitive connections concept. Even if a route does not exist between the selected locations, the system can provide a route with missing links. The gap(s) between the disconnected locations are filled by the help of the relationships between locations.

In order to evaluate the performance of the A*CD algorithm, we present automated tests. These tests show that the costs of the routes that are provided by the A*CD algorithm are close to the actual shortest routes. In order to demonstrate the intuitive connections concept, we also present manual test queries.

Keywords: Heuristic search, collaborative systems, A* search

ÖZET

MEKANLAR ARASINDA ROTALAR SUNMAK İÇİN KATILIMCI BİR SİSTEM

Kerem Ali Uluğ
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi : Y. Doç. Dr. David Davenport
Yardımcı Tez Yöneticisi: Dr. Markus Schaal
Haziran, 2008

Otomobiller için GPS cihazları ve havayolu şirketlerinin ağ siteleri gibi bir çok sistem, mekanlar arasında rota bilgileri sunmaktadır. Bu tip sistemler yaygın olarak kullanılmalarına ve rota bilgilerini doğru bir şekilde sunmalarına rağmen, kullanıcıların veri girişine izin vermemektedir. Bu tip sistemlerde tüm veri, sistem yöneticileri tarafından girilmekte ve sistemler kullanıcılarının rota tecrübelerinden faydalanamamaktadır.

Bu çalışmada, kullanıcı sorguları karşılığında mekanlar arasında rotalar sunan katılımcı bir sistem sunulmuştur. Sistemdeki veriler kullanıcılar tarafından girilmektedir. Verilerin saklanması için, mekanların, mekanlar arasındaki bağlantıların ve mekanlar arasındaki ilişkilerin (kapsama, komşuluk, kesişme) tanımlandığı bir model sunulmuştur. Rotaları bulabilmek için, A* arama algoritmasının özelleştirilmiş bir uyarlaması sunulmuştur. A*CD (A* for Collaborative Data) olarak adlandırdığımız bu uyarlama, arama esnasında mekanları işlerken, hedef mekana kalan tahmini bedeli hesaplamak için buluşsal yöntemler kullanmaktadır. Ayrıca alternatif rotalar sunmak, belli bağlantı tiplerini hariç tutmak ve çok sayıda durağa sahip taşım araçları ile ilgili sorunlara çözüm getirmek için A* arama algoritması üzerinden yapılmış değişiklikler sunulmuştur. Bedel modeli olarak seyahat süresi ve seyahat maliyeti (finansal) kullanılmaktadır.

Çalışmamızda, sezgisel bağlantılar kavramı da sunulmuştur. Seçilen mekanlar arasında bir rota bulunamaması durumunda bile, sistem eksik bağlantılara sahip bir rota dönebilmektedir. Eksik bağlantılar, mekanlar arasındaki ilişkiler yardımıyla doldurulabilmektedir.

A*CD algoritmasının performansını değerlendirmek amacıyla otomatik testler sunulmuştur. Bu testler A*CD algoritması ile bulunan rotaların bedellerinin en düşük bedelli rotaya çok yakın olduğunu göstermektedir. Sezgisel bağlantılar kavramını örneklemek için otomatik olmayan testler sunulmuştur.

Anahtar Sözcükler: Buluşsal yöntemlerle arama, katılımcı sistemler, A* arama

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to Dr. Markus Schaal due to his suggestions, and support during this research. I have learned a lot from him.

I am also indebted to Dr. David Davenport and Dr. H. Murat Karamüftüoğlu for their support and comments.

I would like to thank Dr. Savio S. H. Tse and Dr. Kıvanç Dinçer for accepting to read and review this thesis.

CONTENTS

Chapter 1

Introduction

The best way to learn how to get from one location to another is usually to ask a person who knows the route. The route might consist of buses, taxis, sidewalks or any means of transportation. A person with an expertise on the desired route will be able to combine necessary transportation knowledge and offer the best route. On the other hand, a person has expertise on a limited number of routes. This important drawback can be eliminated by combining people's knowledge in a collaborative system. In a collaborative system, all the data is managed by the system's users. Wikipedia and Facebook are two popular examples of collaborative systems.

There are many systems that are providing routes between locations. In-car GPS devices provide routes by using the city road and traffic information. National train network web sites exploit train transportation information. Municipality web sites contain public bus and metro transportation information to provide routes between selected locations. Although such systems are widely used and can provide routes to the users successfully, they cannot take advantage of the route expertise of their users.

Existing systems also have another important disadvantage. They cannot take advantage of the relationships between locations. There might be different types of relationships between locations such as containment (Ankara contains Bilkent University) or neighborhood (Bilkent University and Middle East Technical University are neighbors). These relationships can be used in order to provide useful information to the user even if a route cannot be found. Consider the following scenario;

- User requests a route from Bilkent University (contained in Ankara) to Berlin Airport.
- There are two airports in Istanbul, Atatürk Airport and Sabiha Gokcen Airport.
- There exists a route from Bilkent University to Atatürk Airport.
- There exists a route from Sabiha Gokcen Airport to Berlin Airport.
- There is no route between Bilkent University and Berlin Airport.

In this scenario, the system can conclude that two airports are connected since they are both contained in Istanbul. So, the route from Bilkent University to Atatürk Airport and also the route from Sabiha Gokcen Airport to Berlin Airport can be provided to the user together with the containment information.

1.1 Problem Definition

In this thesis, our main problem is to provide users with routes between locations, using data that has been entered by the users themselves. Gathering the data from the users is the collaborative aspect of the context. This brings a new problem associated with the quality of

the data. There might be missing links between locations although the links actually exist in real life. In the following scenario, a link is missing (Middle East Technical University – Bilkent University link)

- A route from *Kizilay (a district in Ankara)* to *Bilkent University* is requested
- There is a route from *Kizilay* to *Middle East Technical University*
- There is no route from *Kizilay* to *Bilkent University*
- *Bilkent University and Middle East Technical University are neighbors.* This information has been entered by a user.

In this scenario, the system should be able to provide the route from *Kizilay* to *Middle East Technical University and also the neighborhood information.*

Also, missing links might create problems because of the duplicate information in the system. Two users might enter the same location with different names to the system. The following scenario demonstrates a situation in which there is duplicate information; two users, independent from each other, entered two locations to the system, which are both referring to the same real world location. The missing link is between the duplicate locations in this scenario.

- There are two locations named "*Bilkent Engineering Building*" and "*Bilkent University Engineering Building*" in the system. They refer to the same real world location.
- Both of these locations are contained in a location named "*Bilkent University*". Data for these relationships exists in the system.
- User requests a route from a location, *Kizilay (a district in Ankara)* to *Bilkent Engineering Building*.
- There is a route from *Kizilay* to *Bilkent University Engineering Building*.
- There is no route from *Kizilay* to *Bilkent Engineering Building*.

In this scenario the system should be able to provide the information to the user indicating that *Bilkent Engineering Building* can be reached from *Kizilay* by following the *Kizilay* to *Bilkent University Engineering Building* route. Since both buildings are contained in the same location, *Bilkent University*, the system should be able to conclude that they are connected.

Such data quality problems might be handled if additional information, such as relationships between locations, exists in the system. Although there might be missing links, if the relationships have been entered properly, a solution might still be provided. We name this problem as missing links problem; there might be gaps in a route but these gaps can be filled by the help of the relationships between locations.

The provided routes' costs should be as close as possible to the actual lowest cost routes. We don't define a threshold for the ratio of the cost of a provided route to the actual lowest cost route. On the other hand, applying tests for a large number of location pairs and presenting the average cost ratio is in scope of this thesis.

In order to improve the running time of the algorithm, the route search should be guided by taking advantage of the data in the system (geographic coordinates, location relationships, etc.). The time and space complexity is not discussed in this work.

Our problem definition also contains some extensions beyond the main problem. Users should be able to request alternative routes from the system and specify either minimum duration or minimum financial cost. This choice specifies if they are interested in a fast route that will take minimum amount of time to travel or a cheap route in which they will pay less. Additionally, the system should be able to handle the problems associated with transportation lines with multiple stops.

1.2 Thesis Outline

In Chapter 2, we present details of the missing links problem and our solution to it. In Chapter 3, we give detailed descriptions of the extensions of the main problem and present solutions to them. In Chapter 4, first we present the related work for graph search algorithms and then the details of the search algorithm that we are using, A*CD (A* for Collaborative Data). In Chapter 5, we explain our implementation, mainly the modules for data entry. In Chapter 6 we present the results of our tests. We conclude the thesis with Chapter 7.

Chapter 2

Incomplete Information and Virtual Links

In this chapter, first we explain the extended unidirectional graph model for storing the data in the system. This model contains locations, links and location relationships together with their properties. Afterwards, we explain the situations in which we want to find intuitive connections for missing links, despite data problems in the base graph. Finally, we present our solution, virtual links, for the missing links problem.

2.1 A Base for Solution: Extended Unidirectional Graph

We are using a unidirectional graph as a base. A graph G is a pair (V, E). V is a set of nodes (vertices). E is a set of links (edges) between the vertices,

$$E \subseteq \{(u,v) \mid u, v \in V\}$$

Each node in the graph corresponds to a location in our system. A location can be of any type, for example, it can be a country, a city, an office building or even an individual room.

Locations in our system have properties associated with them. Location type, explanation, geographical coordinates are some examples of locations' properties. These properties are used for storing more information about a singe location.

Each edge in the graph corresponds to a unidirectional link between two locations. A link can be of any type, e.g. an international flight or a 5 minute walk. If a reverse link also exists, this is represented by another link in the system. Cycles of any length (number of nodes) are allowed in our system.

Links have several properties associated with them. Guessed duration, guessed cost (money), guessed cost currency and explanation are the link properties. We name these properties as "guessed" in order not to create a confusion with the term "estimated", which will be used in the A*CD search algorithm.

Cost information is mandatory for links. Users should enter this information while entering a link to the system. For the financial cost, we do not request the currency rate from the users. We only take the financial cost and the currency. We maintain a currency rate table for relating all currencies with Turkish Lira so that all financial costs can be converted to Turkish Lira whenever required.

In addition to links, locations, link properties and location properties, location relationships should also be stored. There are three kinds of relationships; containment, neighborhood and intersection.

We name two locations as parent and child if they are related with a containment relationship. We name two locations as neighbor if they are related with a neighborhood relationship. We say two locations intersect if they are related with an intersection relationship.

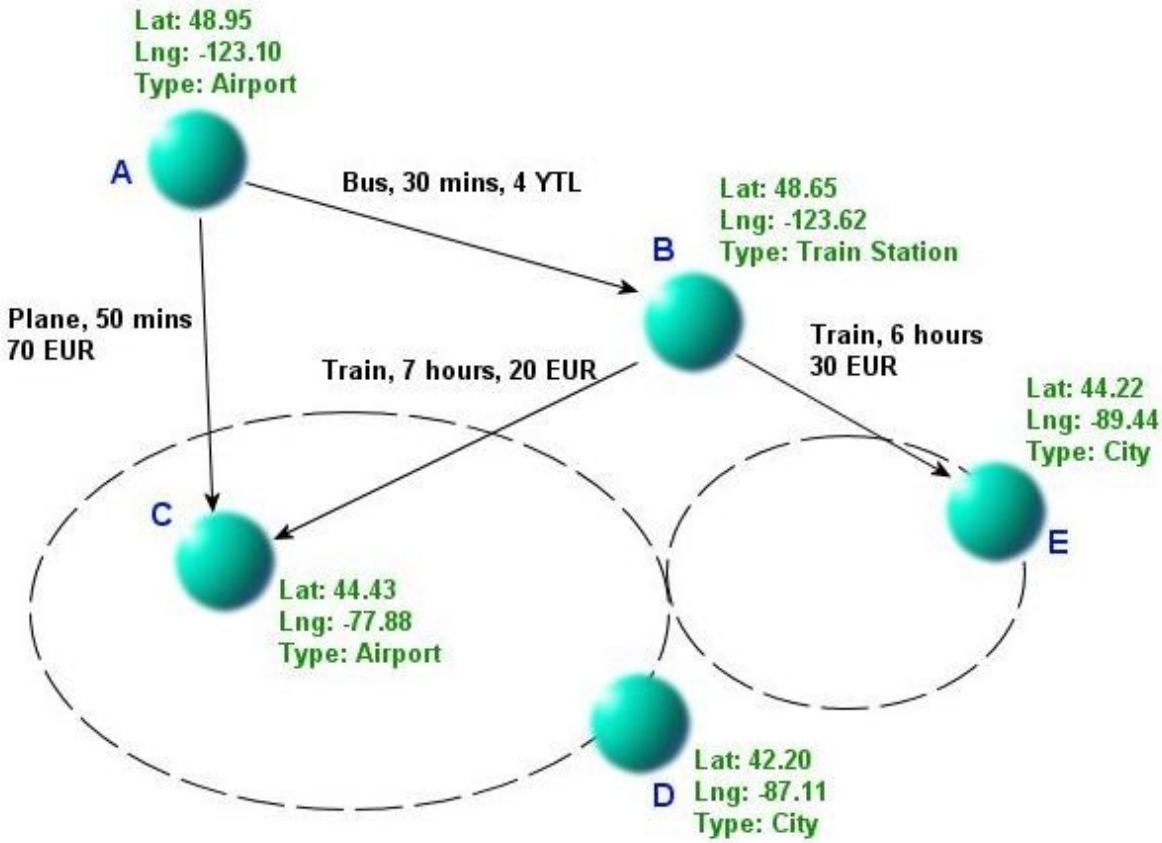

Figure 2.1 – An instance of the base model. Nodes have latitude, longitude and type information. Links have type, guessed duration and guessed financial cost. Node *D* contains node *C*. Nodes *D* and *E* are neighbors.

2.2 Intuitive Connections

Location relationships have an important effect on the solutions provided by the system. They help provide intuitive connections so that even if a complete sequence of user entered links from the source node to the target node cannot be found, a solution can still be provided to the user, in the form of the incomplete route together with the intuitive connection(s).

In order to present the details of the intuitive connections concept, we introduce **basic routes**. A basic route is a sequence of user entered links. A route might consist of user entered links and intuitive connections but a basic route contains only user entered links.

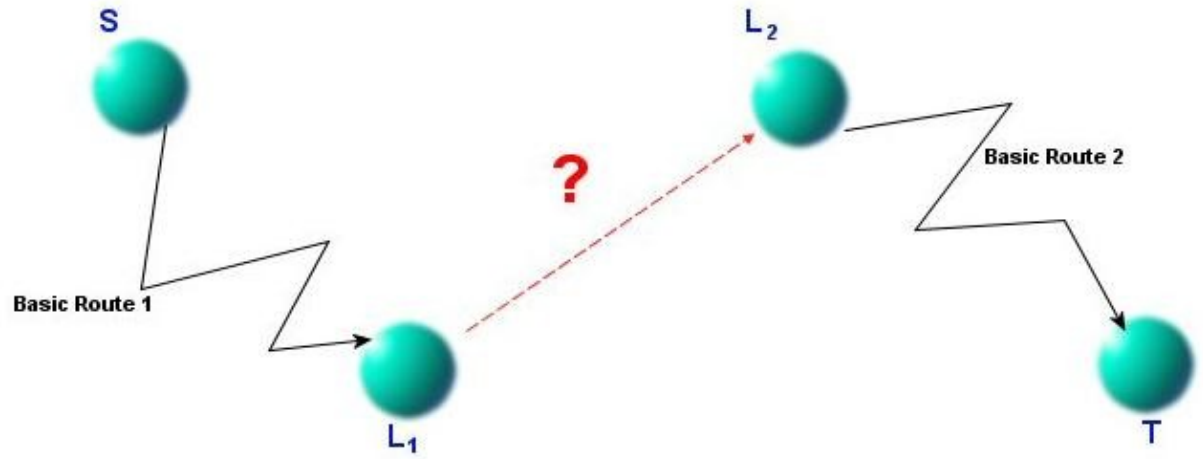

Figure 2.2 – Intuitive connections.

Intuitive connections are used to fill the gaps between locations while finding routes. In order to provide a solution to the user, the system might use more than one intuitive connection; there might be two or more gaps in the route. Also, it is possible that $S = L_1$. This means that the source node is connected to L_2 with an intuitive connection. It is possible that $L_2 = T$. This means that the target node is reached from L_1 with an intuitive connection. Figure 2.2 illustrates intuitive connections. In this figure, there is no basic route from L_1 to L_2.

Figure 2.3 presents a scenario in which three intuitive connections are used. In this scenario, for the first intuitive connection, $S = L_1$. Second intuitive connection connects intermediate nodes and for the third intuitive connection $T = L_2$.

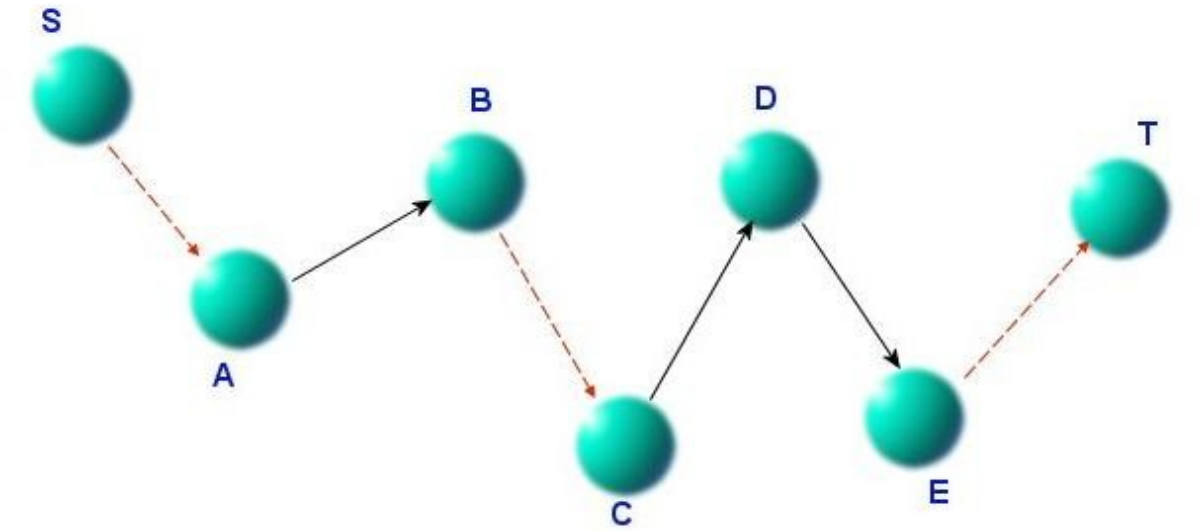

Figure 2.3 – A scenario in which three intuitive connections are used. $S – A$, $B – C$ and $E – T$ routes are missing.

There are five types of intuitive connections.

Intuitive Connection 1 – Common Parent

If there is a location *P* such that

- L_1 is a child of *P*
- L_2 is a child of *P*

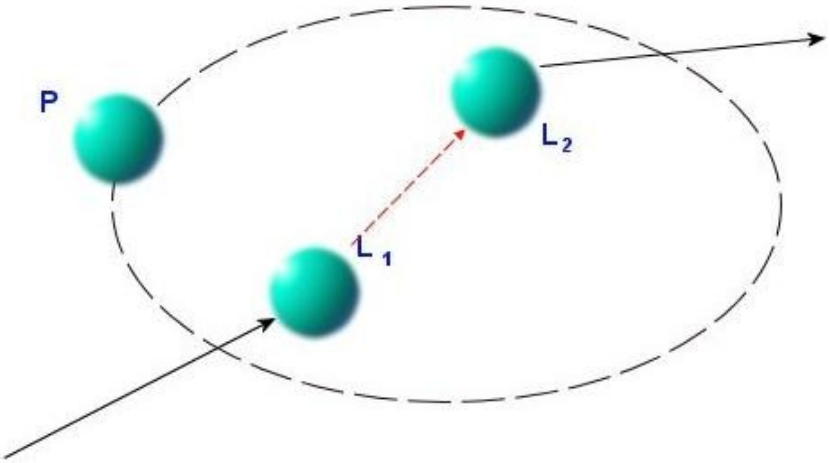

Figure 2.4 – Common parent intuitive connection.

The following scenario shows the usage of this type of intuitive connection..

- A route from *Antalya Intercity Bus Station* to *Istanbul Airport* is requested
- There is a basic route from *Antalya Intercity Bus Station* to *Ankara Intercity Bus Station* (L_1)
- There is a basic route from *Ankara Airport* (L_2) to *Istanbul Airport*
- There is no basic route between *Ankara Intercity Bus Station* and *Ankara Airport*
- *Ankara Intercity Bus Station* and the *Ankara Airport* are contained in *Ankara*

Intuitive Connection 2 – Containment-1

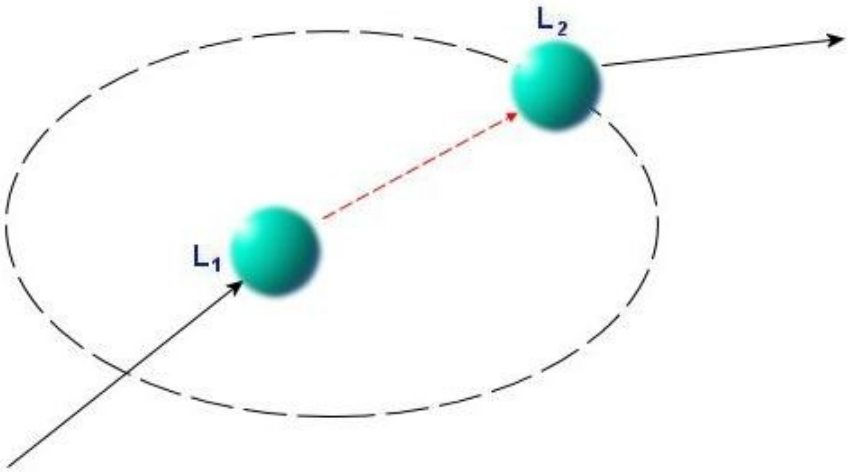

Figure 2.5 – Containment-1 intuitive connection.

If there is a relationship between L_1 and L_2 such that

- L_1 is a child of L_2

The following scenario shows the usage of this type of intuitive connection..

- A route from *Cayyolu (a district in Ankara)* to *Ulus (a district in Ankara)* is requested
- There is a basic route from *Cayyolu* to *Bilkent Bus Stop (L_1)*
- There is a basic route from *Bilkent University (L_2) to Ulus*
- There is no basic route from *Bilkent Bus Stop* to *Bilkent University*
- *Bilkent Bus Stop* is contained in *Bilkent University*

Intuitive Connection 3 – Containment-2

If there is a relationship between L_1 and L_2 such that

- L_1 is a parent of L_2

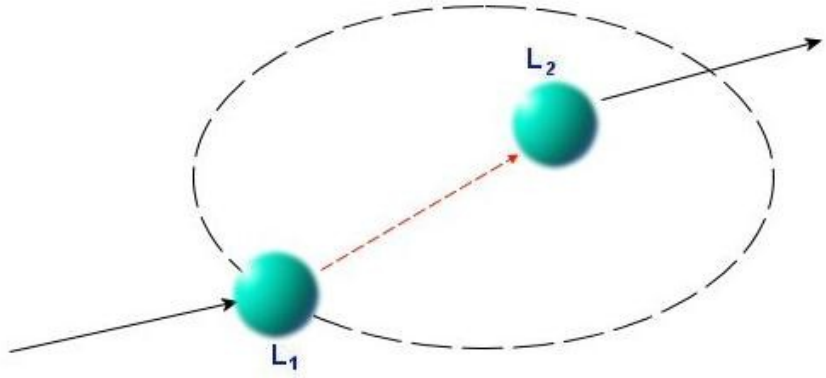

Figure 2.6 – Containment-2 intuitive connection.

The following scenario shows the usage of this type of intuitive connection..

- A route from *Cankaya (a district in Ankara)* to *Bestepe (a district in Ankara)* is requested
- There is a basic route from *Cankaya* to *Kizilay (a district in Ankara – L_1)*
- There is a basic route from *Kizilay Metro Station (L_2) to Bestepe*
- There is no basic route from *Kizilay* to *Kizilay Metro Station*
- *Kizilay Metro Station* is contained in *Kizilay*

Intuitive Connection 4 – Neighborhood

If there is a relationship between L_1 and L_2 such that

- L_1 and L_2 are neighbors

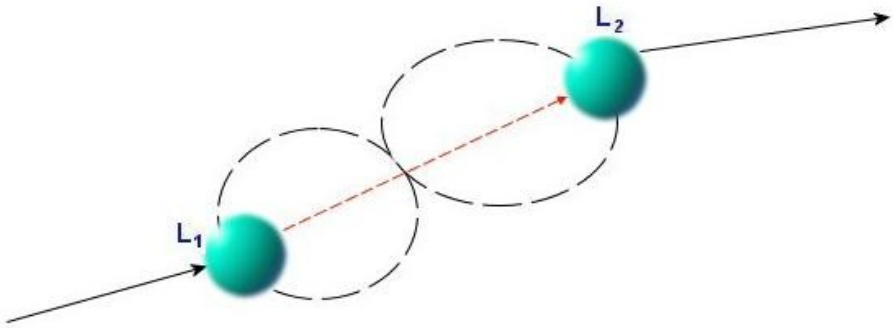

Figure 2.7 – Neighborhood intuitive connection.

The following scenario shows the usage of this type of intuitive connection..

- A route from *Cankaya (a district in Ankara)* to *Bestepe (a district in Ankara)* is requested
- There is a basic route from *Cankaya* to *Kizilay Main Bus Station (L_1)*
- There is a basic route from *Kizilay Metro Station (L_2) to Bestepe*
- There is no basic route from *Kizilay Main Bus Station* to *Kizilay Metro Station*
- *Kizilay Main Bus Station and Kizilay Metro Station are neighbors*

Intuitive Connection 5 – Intersection

If there is a relationship between L_1 and L_2 such that

- L_1 and L_2 intersects

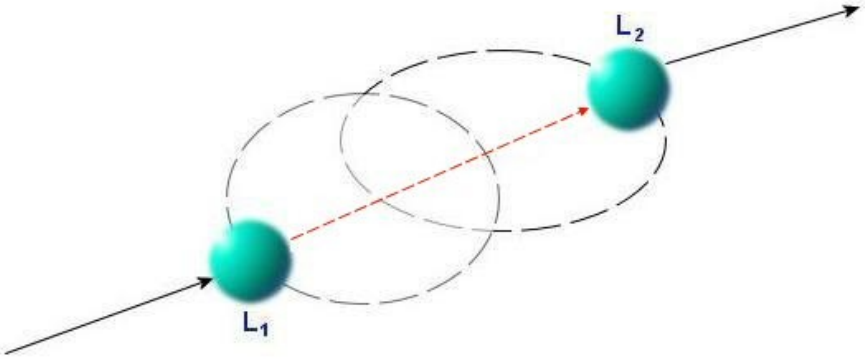

Figure 2.8 – Intersection intuitive connection.

The following scenario shows the usage of this type of intuitive connection..

- A route from *Kemer Town Center (a town center in Antalya)* to *Antalya City Center* is requested
- There is a basic route from *Kemer Town Center* to *Olympos Beach (L_1)* by a boat
- There is a basic route from *Cirali (a town in Antaya – L_2) to Antalya City Center* by a bus
- There is no basic route from *Olympos Beach* to *Cirali*
- *Olympos Beach* and *Cirali* intersects

2.3 Virtual Links

In order to provide the intuitive connections, we came up with the idea of virtual links. A virtual link is a special type of link, which is maintained by the system. The users cannot maintain these links. Virtual links are maintained automatically; as users enter or delete data, the system creates or deletes virtual links accordingly. There are four types of virtual links. These types are described and illustrated in sections 2.3.1 – 2.3.4. Virtual link maintenance is described in section 2.3.5.

2.3.1 Virtual Link Type-1

If location *A* is in location *B* and a link from *A* to *C* exists, virtual link is from *B* to *C*.

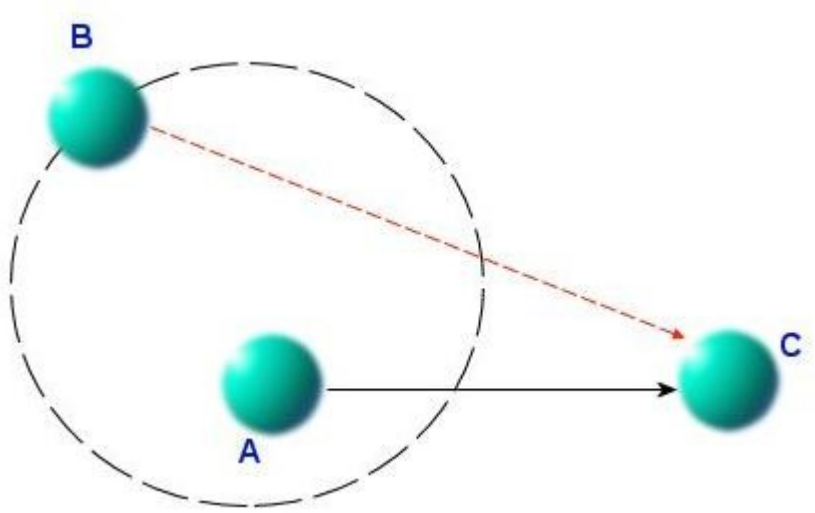

Figure 2.9 – Virtual Link Type-1. A link from *A* to *C* exists in the system. *B* is a parent of *A*. Virtual link is from *B* to *C*.

2.3.2 Virtual Link Type-2

This virtual link is inserted for each containment relationship. If there exists a containment relationship such that; location *A* resides in location *B*, a virtual link from *A* to *B* is added to the system.

This kind of virtual relationship “connects” the locations that are contained in the same parent location. By this, even if the system cannot find a direct route for a query, it can supply a route by using a location that is sharing a common parent.

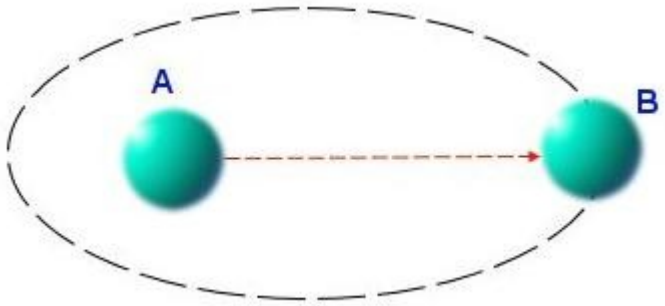

Figure 2.10 – Virtual Link Type-2. *B* is a parent of *A*. Virtual link is from *A* to *B*.

2.3.3 Virtual Link Type-3

This virtual link is inserted for each neighborhood relationship. If there is a neighborhood relationship between two locations *A* and *B*, two virtual links are added; one from *A* to *B* and the other from *B* to *A*.

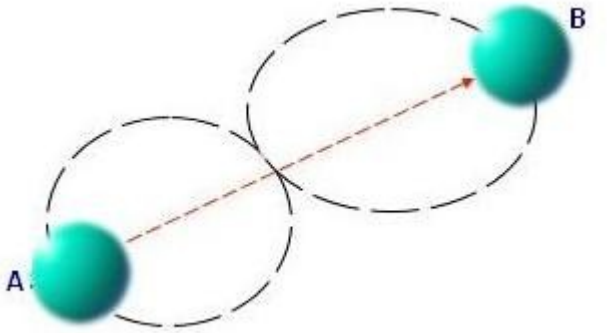

Figure 2.11 – Virtual Link Type-3. *A* and *B* are neighbors.

2.3.4 Virtual Link Type-4

This virtual link is inserted for each intersection relationship. If there is an intersection relationship between two locations *A* and *B*, two virtual links are added; one from *A* to *B* and the other from *B* to *A*.

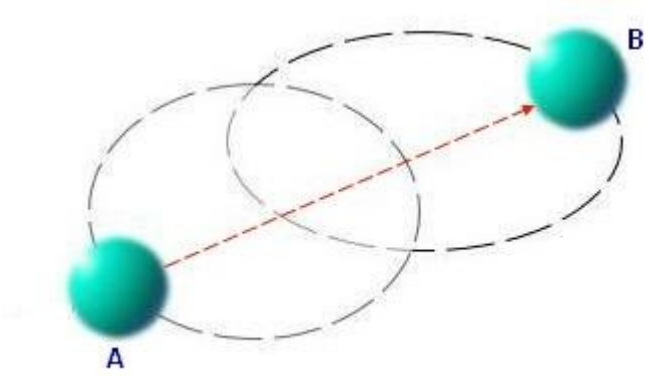

Figure 2.12 – Virtual Link Type-4. *A* and *B* intersects.

2.3.5 Virtual Link Maintenance

The virtual links are all maintained by the system. Users cannot modify, delete or add virtual links to the system. We have come up with two solutions to manage virtual links and decided on the second one.

Batch Virtual Link Maintenance

In this approach, an automated script is executed and the script creates the type 1 and type 2 virtual links automatically. Before creation all the virtual links are deleted from the system.

The advantage of this approach is that, there are no extra operations during data entry of the users. On the other hand, this approach leaves the database unsynchronized between script executions. The virtual links are not created immediately after data entry.

The highly coupled nature of the virtual links makes it hard to implement virtual link maintenance in real time. For the type-1 virtual links, if B is also a parent of C, the virtual link is not created. The reason for eliminating such virtual link candidates is that, with those in the system, the search algorithm gives results that are meaningless to human users.

Deletion and re-creation of around 10000 virtual links takes around 3500 ms, so running the script frequently (in the order of minutes) even if the data increases rapidly, might decrease the disadvantages due to this approach.

Real-Time Virtual Link Maintenance

This approach creates and deletes virtual links when a user-entered data is changed in the system. Virtual links are affected by containment relationships between locations and links. So, any change in these information changes the virtual links in the system. We have categorized trigger changes as follows;

- Add containment relationship (effects type-1 & type-2)
- Delete containment relationship (effects type-1 & type-2)
- Add link (effects type-1)
- Delete link (effects type-1)

- Add neighborhood relationship (effects type-3)
- Delete neighborhood relationship (effects type-3)
- Add intersection relationship (effects type-4)
- Delete intersection relationship (effects type-4)

Adding Containment Relationships

Users can enter containment relationships to the system. We are considering an atomic operation of relating a location with another, with relation type containment. Such a change in the system effects both type-1 and type-2 virtual links. The following algorithm maintains virtual links (type-1) upon entry of containment relationship.

```
AddContainment_ManageType1(child, parent)
    for each Link p originating from child
        if p is not a virtual link and destination(p) is not in parent
            add a type 1 virtual link from parent to destination(p)
        end if
    end for

    RecreateIncomingType1Links(child)
```

Figure 2.13 – Pseudo code for managing type-1 virtual links upon containment addition

First part of the algorithm creates type-1 virtual links that are originating from the parent. For that, all links originating from child are retrieved.

```
RecreateIncomingType1Links(loc)
    for each Link p ending at loc
        if p is a type 1 virtual link
            delete p
        end if
    end for

    for each Link p ending at loc
        if p is not a virtual link
            for each parent of source(p)
                if destination(p) is not in parent
                    add a type 1 virtual link from parent to
                    destination(p)
                end if
            end for
        end if
    end for

    for each Location child in children(loc)
        RecreateIncomingType1Links(child)
    end for
```

Figure 2.14 - Pseudo code for recreating incoming type-1 virtual links

Second part, which is recreating type-1 virtual links, is a recursive algorithm. The rule of the type-1 virtual links is that, as indicated before, if destination is also a child of the parent, the link won't be added. This is why type-1 virtual links that are ending at the new child and also children of the new child should be checked upon entry of a new containment relationship.

```
AddContainment_ManageType2(child, parent)
    Add a virtual link from child to parent
```

Figure 2.15 - Pseudo code for managing type-2 virtual links upon containment addition

Deleting Containment Relationships

Users can delete containment relationships from the system. Such an operation effects both type-1 and type-2 virtual links in the system.

The following algorithm is for maintaining virtual links (type-1) upon deletion of a containment relationship.

```
DeleteContainment_ManageType1(child, parent)
    for each Link p originating from parent
        if p is a type 1 virtual link due to child parent relationship
            delete p
        end if
    end for

    RecreateIncomingType1Links(child)
```

Figure 2.16 – Pseudo code for managing type-1 virtual links upon containment deletion

In the first part of the algorithm, all type-1 virtual links that are created because of the relationship between the child and parent are deleted from the system. Afterwards, incoming type 1 virtual links to child are recreated.

The following algorithm maintains virtual links (type-2) upon deletion of a containment relationship.

```
DeleteContainment_ManageType2(child, parent)
    for each Link p ending at parent
        if p is a type 2 virtual link originating from child
            delete p
        end if
    end for
```

Figure 2.17 – Pseudo code for managing type-2 virtual links upon containment deletion

Adding Links

Users can add a link from any location to any other location. Upon such a data change, type-1 virtual links should be maintained.

The following algorithm maintains virtual links (type-1) upon addition of a link.

```
AddLink_ManageType1(source, destination)
    for each Parent parent of source
        if destination is not a child of parent
            add a type 1 virtual link from parent to destination
        end if
    end for
```

Figure 2.18 – Pseudo code for managing type-1 virtual links upon link addition

Deleting Links

Users can delete any link from the system. Upon such a data change, type 1 virtual links should be maintained.

The following algorithm maintains virtual links (type-1) upon addition of a link.

```
DeleteLink_ManageType1(source, destination)
    for each Link p ending at destination
        if p is a type 1 virtual link that is generated due to source
        destination actual link
            delete p
        end if
    end for
```

Figure 2.19 - Pseudo code for managing type-1 virtual links upon link deletion

Adding Neighborhood Relationships

Users can add neighborhood relationships between locations. Upon adding these data, the system should maintain type-3 virtual links.

```
AddNeighborhood_ManageType3(location1, location2)
    add a type-3 virtual link from location1 to location2
    add a type-3 virtual link from location2 to location1
```

Figure 2.20 – Pseudo code for managing type-3 virtual links upon neighborhood addition

Deleting Neighborhood Relationships

Users can delete neighborhood relationships between locations. Upon deleting these data, the system should maintain type-3 virtual links.

```
DeleteNeighborhood_ManageType3(location1, location2)
    delete the type-3 virtual link from location1 to location2
    delete the type-3 virtual link from location2 to location1
```

Figure 2.21 – Pseudo code for managing type-3 virtual links upon neighborhood deletion

Adding Intersection Relationships

Users can add intersection relationships between locations. Upon adding these data, the system should maintain type-4 virtual links.

```
AddIntersection_ManageType3(location1, location2)
    add a type-4 virtual link from location1 to location2
    add a type-4 virtual link from location2 to location1
```

Figure 2.22 – Pseudo code for managing type-3 virtual links upon intersection addition

Deleting Intersection Relationships

Users can delete intersection relationships between locations. Upon deleting these data, the system should maintain type-4 virtual links.

```
DeleteIntersection_ManageType3(location1, location2)
    delete the type-4 virtual link from location1 to location2
    delete the type-4 virtual link from location2 to location1
```

Figure 2.23 – Pseudo code for managing type-3 virtual links upon intersection deletion

2.4 Accepted Target Set

Accepted target set is used to provide intuitive connection 3 where $L_2 = T$ *and also provide intuitive connection 1 where* $L_2 = T$. This set stores each parent of the target node, which is not a parent of the source node. We do not add parents of both the source and the target node to the accepted target set since virtual link type-2 connects a child to its parent. Consider that they are added to the target set and consider the following scenario.

- The search is from the node *S* to the node *T*
- There exists a node, *P,* which is a parent of both *S* and *T*

In this scenario, *S* – *P* link (virtual link type-2) will be provided as a solution and this information is not useful for the user.

2.5 Why These Virtual Link Types

The main drawback of virtual links is that they slow down the algorithm since they introduce new edges to the graph. More edges means the algorithm will process more nodes and routes.

We had two main goals while designing virtual links;

- They should cover all intuitive connections.
- Their number should be kept at minimum. The reason is to reduce the search space as much as possible. More links mean more processing.

Using these four types, it is possible to cover three intuitive connections that are described in section 2.2. These virtual links cannot cover intuitive connection 1 and intuitive connection 3 completely. The cases where $L_2 = T$ are missed since we do not have virtual links from parent nodes to child nodes. In order to cover these cases, we have introduced the accepted target set. Instead of such an approach, we could have defined another type of virtual link, type-5; A link type that is from the parent location to the child location, i.e. the reverse of the type-2 virtual

link. By such a virtual link, the accepted target set would be unnecessary, but it would increase the number of virtual links. As the number of links increases, the algorithm has to process more links so the running time also increases. A comparison between the used approach and the approach with the type-5 virtual links is given in Table 2.1.

We can say that the cost increase of the approach with type-5 virtual links is more than the cost increase of the currently used approach.

	Used Approach	**Approach With Type-3 VL**
Summary	Only four types of virtual links are used. Accepted target set is also used.	Five types of virtual links are used. Accepted target set is not used.
Place of Extra Processing	Whenever successor nodes of the processed node are retrieved, due to virtual links, some extra nodes are retrieved. Also, after target check, a check is done to see if the node is in the accepted target set.	Whenever successor nodes of the processed node are retrieved, due to virtual links, some extra nodes are retrieved.
Cost of Extra Processing	Extra processing due to more successor nodes is omitted since four types of virtual links also affect the second approach the same way. Checking accepted target set is done in constant time by the help of a hash table. So, if number of processed nodes is P, the cost will change from O(P * 1) to O(P * (1 + c)) where c is a small constant.	Extra processing due to more successor nodes that are connected to the processed node by type-5 virtual links increases the processing time. If number of processed nodes without type-5 virtual links is P and if there are R number of containment relationships, the cost will change from O(P) to O(P + R).

Table 2.1 – Comparison between the applied approach and the approach with type 3 virtual links

Chapter 3

Problem Extensions

In this chapter, we explain the three extensions in our problem definition. First we explain the details of the multiple stop transportation line problem and present a solution. Then, we explain the alternative routes problem and its solution. Finally, we explain the search preferences problem and its solution.

3.1 Multiple Stop Transportation Lines

Until now, all links in the system are considered to be unrelated with each other. But in real life, links between locations can be related.

Consider a bus line in *Istanbul*, that is connecting four bus stops;

Besiktas – Zincirlikuyu – Levent – Maslak

If the algorithm finds a route from source location to target location, in which there are two or more links from the same bus line, it should display it as just one link. So,

Source Location - ... - Besiktas – Zincirlikuyu – Levent - ... - Target Location

is not a desired solution. Instead of this, the algorithm should provide the following route

Source Location - ... - Besiktas – Levent - ... - Target Location

In order to solve this problem we use a new property for the links, MultipleStopLineId. In our approach, all links, which are due to a single transportation line, will share the same MultipleStopLineId value. Other than these links, no other link will have the same value.

For the above example there will be three links in the system.

- *Besiktas – Zincirlikuyu*
- *Zincirlikuyu – Levent*
- *Levent – Maslak*

We solve this problem during the display of the data to the user. During display, if we realize that two consecutive links, *L1* and *L2*, in the solution has the same MultipleStopLineId, we display these two links as one link in which the source is the source of *L1* and the target is the target of *L2*.

3.2 Alternative Routes

In addition to providing a route from the source location to the target location, the system should also provide alternative routes to the users.

There might be alternative routes between the selected pair in different cities or countries, using trains, using planes or using intercity buses. Inside the same city, same pair of locations might be connected by buses, ferries or metro networks.

Although "k-shortest route (path) problem" [9, 12, 16, 17, 21, 23, 24, 28] in the literature is generally studied in order to find the k optimal routes from a source to a target, it contains many similarities with the problem at hand.

In order to provide more than one route, we have to process each node more than once. We use a constant, *k*, in order to define the maximum number of times a node can be processed in the search algorithm. This number defines the minimum number of routes the system can provide if routes exist. It is the minimum number since there might be alternative routes that are not sharing any common node other than the source and the target nodes.

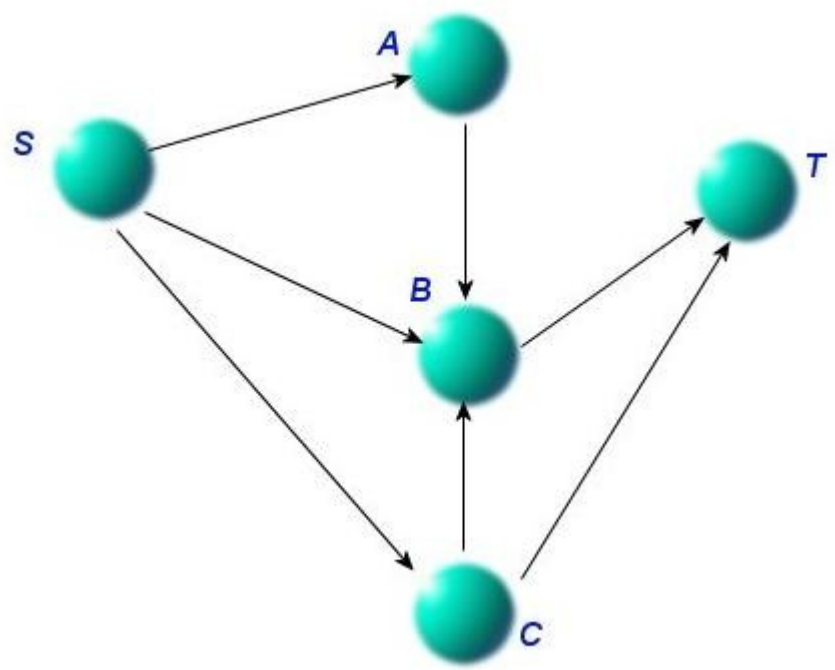

Figure 3.1 – Alternative routes example. There are four alternative routes from *S* to *T*. Three of them pass over the node *B*. If k = 1, two routes can be found. If k = 2, three routes can be found. If k = 3, four routes can be found.

3.3 Search Preferences

The system will allow users to enter two search preferences, which will be the input of the search algorithm.

The user should be able to exclude certain link types in his queries. For instance, a user should be able to exclude planes and intercity buses if he wants to travel by trains between cities. In order to provide this feature, we display a list of used link types to the user whenever a route is provided. User can select any number of these link types as unwanted types. After this selection user can request an alternative route. The search algorithm will this time discard a link if its type is in the excluded link types list. Discarding is a trivial issue. While processing the successors of a node in the search algorithm, types of the links that are connecting the node to its successors are checked.

In addition to excluded link types, the user should be able to select his cost preference out of two options. First option is the duration. Second option is the financial cost. So, the user should be able to specify if he is interested in a fast route that will take minimum amount of time to travel or a cheap route in which he will pay less. For each route search, user can select one option out of these two. In the search algorithm, in order to calculate the cost till the processed point, we add either durations together or the financial costs, according to the preference of the user. Since all links in the system contain information for both cost models, for any route search these two options are valid.

Throughout the thesis, whenever the cost of a link or the total cost of a route is mentioned, we are referring to one of these options. Since in both options a route or a link with a lower cost would be preferred over a higher cost route or link, it doesn't matter which option the user has chosen.

Chapter 4

Search Algorithm

In this chapter, we first present related work for graph search algorithms, mainly about the A* graph search since our algorithm's base is the A* search algorithm. Then we present the details of our algorithm. Afterwards we present the details of the h-value calculation together with our heuristics. After providing all the details of the algorithm, we explain the virtual link preference conditions. We conclude this chapter with an example execution scenario.

4.1 Related Work

BFS (Breadth First Search) is one of the most common graph search algorithms. Using the locations as vertices and links as edges, a graph representation is formed in memory and using BFS, a route from source to destination is searched.

BFS is an uninformed search algorithm which exhaustively searches all the nodes in the graph until it finds the desired route. It starts with the source node, and at each stage it visit one level up (source node is at level 0, successors of the source node are at level 1, etc...). On the first stage it visits all the vertices at level 1 (relative to the source node). In the second stage it visits all the vertices at level 2 and goes on like that. BFS can be considered as a "blind" search since it does not give any priority to the nodes. There are also versions of this algorithm for massive graphs [3].

DFS (Depth First Search) is very similar to BFS. Instead of processing a level after the processing of the previous level is finished, DFS processes a branch till its deepest node. There are versions [29] of this algorithm, which are handling the problems associated with duplicate nodes.

There are many algorithms [2, 5, 10] that guarantee finding the fastest path. Dijkstra's algorithm [11] guarantees to find the fastest path (route). All nodes' cost is initialized to a very large constant and source node's cost is initialized as 0. At each step, the node with the least cost is processed. Whenever the target node is reached, the cost of the target node is the minimum cost between the source and the target nodes. There are algorithms [6] that guarantee to find the lowest cost route in graphs with negative weight edges. There are algorithms [1] that find the lowest cost k-link route. There are also algorithms that are running on dual graphs [30].

We do not consider these approaches as a base to our solution since they do not take advantage of additional information in the graph. They are blind search strategies and not guided. Being able to use heuristics is a mandatory requirement for us. Researches like [14] present the performance advantages of guided search strategies over blind search strategies.

A* search algorithm [15] is accepted as one of the most efficient route finding algorithms for graphs. The original algorithm uses a function f(n), that gives the cost of going from source location to one of the target nodes through node n. The algorithm relies on heuristics. In the pseudo code (shown in Figure 4.1), successors(p) returns the nodes that can be reached by following a single link from p. It is assumed that the queue maintains an ordering by f-value automatically.

$$f(n) = g(n) + h(n)$$

g(n) is the cost of the route so far (cost between the source node and n) and h(n) is the estimated cost from n to the target node. In this h function, heuristics comes into play and used for estimating the remaining cost. According to the value of f(n) a route is given higher or lower priority.

With this algorithm, one can use several heuristics for prioritizing routes. The advantage is that different heuristics can be added to the algorithm as new optimization ways are discovered since the heuristic implementations are separated from the algorithm core.

```
A*(start,goal)
    var closed <- the empty set
    var q <- make_queue(start)
    while q is not empty
        var p <- remove_first(q)
        var x <- the last node of p
        if x in closed
            continue
        end if
        if x = goal
            return p
        end if
        add x to closed
        for each y in successors(x)
            enqueue(q, p, y)
        end for
    end while
    return failure
```

Figure 4.1 – Pseudo code of the A* search algorithm

IDA* [18] is a modified version of the A* algorithm. In this version, iterative-deepening approach [25] is applied to the A* algorithm. This algorithm suffers from the usage of only a limited amount of memory. So, it suffers from excessive node regeneration.

RBFS [19] is a recursive algorithm that attempts to mimic the operation of standard best-first search using only linear space. Its structure is similar to recursive depth-first search. Instead of continuing indefinitely down the current path, it keeps track of the f value of the best alternative route available from any ancestor of the current node. If the current node exceeds f-value limit, the recursion unwinds back to the alternative path. As the recursion unwinds, RBFS replaces the f-value of each node align the path with the best f-value of its children. By this, the algorithm remembers the f-value of the best leaf in the forgotten subtree. Similar to IDA*, RBFS also suffers from excessive node generation.

MA* [8] is the memory bounded version of A* algorithm. In contrast to IDA*, its memory limit is not fixed; the algorithm can take advantage of the whole available memory.

SMA* [26, 27] has some improvements over MA*. It proceeds just like A*. When the memory is full, it discards the node with the worst f-value from the queue and backups the f-value of the forgotten node to its parent. By this, the ancestor of a forgotten subtree knows the quality of the best route in that subtree. When all the nodes in the queue have f-values greater than the forgotten node's f-value, the node is regenerated.

SMAG* is a graph search extension of SMA*. This algorithm prunes a node whenever a lower cost route from source to that node is found. By this way, its entire subtree is removed from the search space and the node can be explored again.

A new algorithm which is similar to SMAG* is introduced in [31]. The difference of this approach is to propagate the change instead of pruning when a lower cost route is found to a previously explored node. The change is propagated to the node's descendants. In [14], a new approach which is using A* in combination with a technique based on landmarks and the triangle inequality is presented.

4.2 A*CD Algorithm Details

There are many approaches and algorithms for finding routes in graphs. Among these alternatives we have selected the A* search algorithm as the base. As explained in the previous section, A* algorithm's structure allows heuristics to be integrated to the search algorithm.

A* search guarantees to find the fastest route if the heuristic function is *monotone* [15, 31]. A heuristic function h is monotone if for each node n and successor node n',

$$h(n) \leq h(n') + c(n, n')$$

where c(n, n') is the cost of the link from n to n'. Our heuristic function does not meet this criteria so A*CD does not guarantee to find the fastest route.

A*CD starts from the source node and at each step successors of the current node are retrieved from the database. Upon successor retrieval, heuristics are applied to the successor nodes, some of them are eliminated and the others are added to the priority queue to be further processed.

In the original A* algorithm heuristics are applied to estimate the cost to the target node. Assume that we are processing a node that is corresponding to location *L*. Original A* algorithm takes successors of *L*, and for each node S in successors(*L*) it applies the heuristics and calculates an estimate; cost of the route with the lowest cost, which is passing through S

$$f(n) = g(n) + h(n)$$

where g(n) is the actual cost of the route so far and h(n) is the estimated cost to reach from the current node to the target node. In the original A* algorithm, f value is calculated for all nodes

S in successors(*L*) and the successor is added to the queue together with its f value. As the f value of a route decreases, it is more favorable since it means its cost estimate is lower. Lower cost means better route.

In our approach, calculation of g-value is done according to the costs of the links used so far. Assume that the algorithm is processing location *L*. This location has been retrieved from the queue in order to be processed. For each location that is in the queue, the cost to reach from the source location to that node is also stored. So, while processing successors of *L*, adding the link cost to the cost of *L* is enough to calculate g-value.

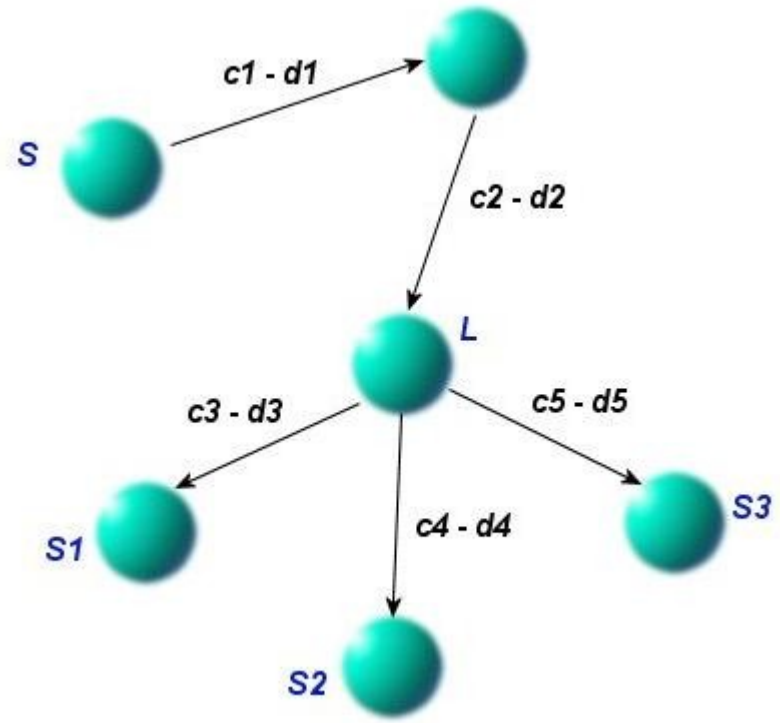

Figure 4.2 – An example scenario for g-value calculation.

For Figure 4.2, values near links indicate their cost. *c* values are the costs (financial) of the links and *d* values are the durations of the links. Assume that search is started from node S. L is the currently processed node; it has been retrieved from the queue. There are three successor nodes of L; S_1, S_2 and S_3. For each successor, a g-value must be calculated. If the user has selected duration as his point of interest, g-values will be as follows;

- $g(S_1) = d1 + d2 + d3$
- $g(S_2) = d1 + d2 + d4$
- $g(S_3) = d1 + d2 + d5$

If the user has selected financial cost as his point of interest, g-values will be calculated as follows;

- $g(S_1) = c1 + c2 + c3$
- $g(S_2) = c1 + c2 + c4$
- $g(S_3) = c1 + c2 + c5$

Calculation of h-value is done in three steps. In the first step the distance between the currently processed node and the target node is calculated by using the geographic coordinates of the nodes.

This value is then normalized in order to match its unit (kilometers) with the unit of the g-value (duration or financial cost, depends on the choice of the user).

This value is then increased or decreased by the heuristics [4, 7, 13, 20, 22]. According to the data existing in the system, the algorithm modifies the value of the estimated cost.

All heuristics are combined together to modify the estimated cost value. We do not have any constraint for the number of heuristics that are applied. They might be considered as rules [4] which are affecting the estimated cost.

An important advantage of this approach is its flexibility. As the system operates, there will be new ideas for collecting new information about locations and links. New heuristics, which are relying on the new data, can easily be integrated to this system.

While one heuristic increases the estimated cost, another one might decrease it. It is like saying, “This route uses very few numbers of hops and is cheap, but it takes too long”. By using this approach, we believe we are reflecting the advantages and disadvantages of routes to the algorithm in a suitable way.

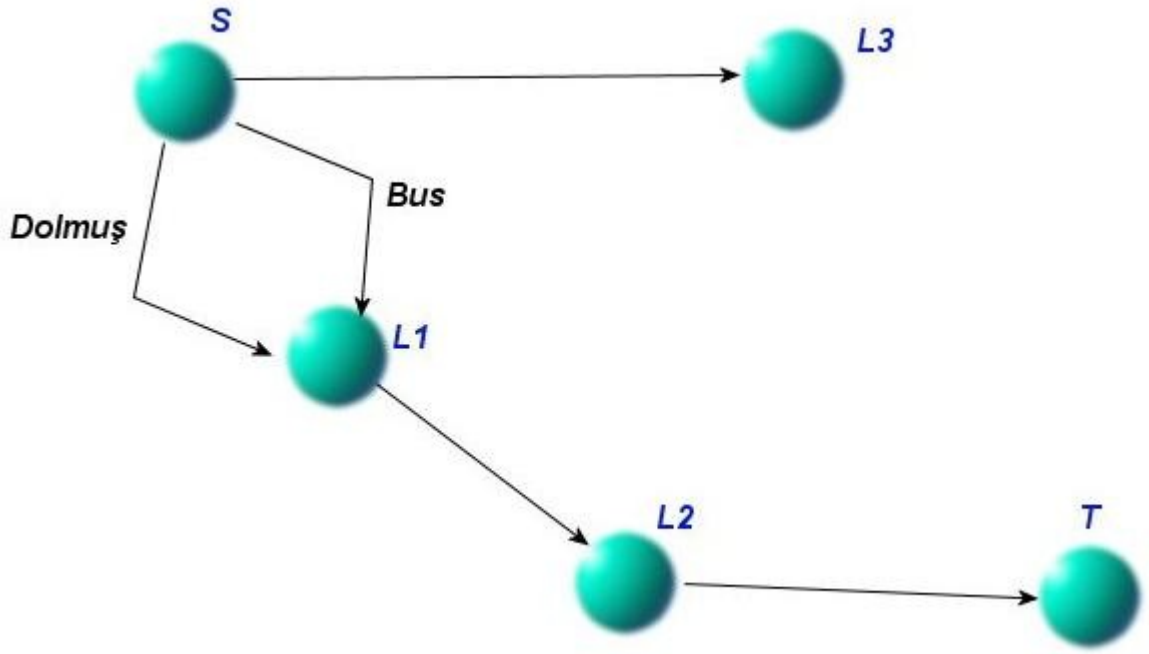

Figure 4.3 – Multiple routes reaching to the same node.

Another important thing to be noted on the algorithm is its support for supplying alternative routes. As indicated previously in Chapter 3, we want the algorithm to provide several alternative routes to the user. In the classical A* search algorithm structure, no such support exists. Whenever a node in successors set of the currently processed node is seen, the algorithm checks to see if the node has been previously added to the queue. If so, the successor node is skipped. Consider the graph in Figure 4.3. In this graph, there are two

alternative routes from S to T. Consider that Bus link is processed before Dolmuş link. In this case, Dolmuş link won't be processed by the algorithm, since the algorithm has already reached *L1* and *L1* has already been added to the queue (containing S as its ancestor and Bus link as its ancestor connecting link).

In order to modify this approach to support finding k-shortest routes, we are storing more than one instance of the same node in the queue. For the above example, two instances of node L1 will be stored in the queue with our approach. One instance contains Dolmuş as the ancestor connecting link and the other instance contains Bus as the ancestor connecting link.

The queue (defined at line 6) is used to store the nodes that should be processed. The nodes in this queue are ordered according to their f-value. For initialization, start node is added to the queue. Queue might contain more than one instance of the same node if more than one route has been found to that node.

The set "visitedNodes" is used to store nodes that have been reached in the search. This set might be considered as a permanent backup location for nodes. As indicated, the queue stores only nodes to be processed. So, if a node is processed it is removed from the queue. On the other hand, we need removed nodes' information (in order to track the route from the target to the source by visiting ancestor node of each node). This set is used to provide this information.

"excludedLinkTypes" is a set, in which elements are link types. This set is added to the algorithm in order to give the user a chance to exclude one or more link types from the search.

Line 22 is related with accepted target set. The "incompleteResults" set in the algorithm stores the found routes that are from source node to a node in the accepted target set. If no route between source and target is found, these results are displayed to the user.

In the original A* algorithm goal check is done inside the body of the for loop that is starting at line 26. Instead of this approach, we are doing this check inside the body of the while loop that is starting at line 8. By this, when the target node is reached, it is again processed and added to the priority queue. But this time its favorability is multiplied with a high constant (by heuristic 7). By this, if a node has more than one link to the goal target, an order between those two links can also be done by our system.

Also checks like the one at Line 2 of the "enqueue" method can be done by this approach. Consider the case in we have locations *S*, *P* and *T* with the following conditions

- *S* is the source node
- *T* is the target node
- There is a user entered link from *S* to *T*
- There is a type 2 virtual link from *S* to *P* (because of containment relationship)
- There is a type 1 virtual link from *P* to *T* (because of containment relationship and *S* – *T* link)

In these conditions, if original A* approach is used, as the algorithm is processing node *S*, it will find *P* – *T* link, and since it reaches us to the target, it will return the result immediately, which is an incorrect behavior. But when we process this link by our regular procedures, it is being eliminated because of the check at line 2 of the enqueue method.

```
1 A*CD(source, target, excludedLinkTypes)
2     var visitedNodes <- empty set
3     var acceptedTargets <- getAcceptedTargets(source, target)
4     var routes <- empty set
5     var incompleteResults <- empty set
6     var q <- make_queue(source)
7     var k <- number of routes to be found
8     while q is not empty
9         var node <- remove_first(q)
10        if (processedBefore(node))
11            continue
12        end if
13        addToVisitedSet(node)
14        if node == goal
15            addToFoundRoutes(routes, visitedNodes, node)
16            if (routes.Count == k)
17                return routes
18            else
19                continue
20            end if
21        end if
22        if node in acceptedTargets and L.getType() != virtual link
23            addToIncompleteResults(incompleteResults, visitedNodes, node)
24        end if
25        var routeFromSource <- createRouteFromSource(node)
26        for each Link L originating from node
27            if L.getType() in excludedLinkTypes
28                continue
29            end if
30            if L.getType() is virtual link and L.actualLink().getType()
31              in excludedLinkTypes
32                continue
33            end if
34            if L.getDestination() in routeFromSource.NodeSet
35                continue
36            end if
37            enqueue(node, L, routeFromSource)
38        end for
39    end while
40    add incompleteResults to routes set
41    return routes
```

Figure 4.4 – Pseudo code of A*CD algorithm

Whenever a node is retrieved from the queue, a check is done to see if the node has been processed before (line 10). In this check, value of k is important. If the node has been visited k times, it is considered to be processed before so it is skipped (pseudo code for that function is provided in Figure 4.7).

If the node turns out to be the target node, the route from source node to the target node is added to the found routes set (line 15). If the node turns out to be in the accepted target set, it is added to the incomplete results set (line 23). There is an important difference between these two cases. If the node is the target, the algorithm does not continue with the successors of the node. On the other hand, if the node is in the accepted target set, it is added to the incomplete results but in addition to that, successors of the node are also processed.

After the target checks, successors of the node are retrieved from the database. For each successor, excluded link type check is done. If the successor passes this check, the route till now (the route starting from the start node and ending at the currently processed node) is checked; if the route contains the successor, successor is skipped. After passing these checks, enqueue method is called.

```
1 enqueue(node, link)
2     if checkType2Type1VirtualLinkChain() is true
3         return
4     end if
5     var newNode <- link.getTarget()
6     newNode.ancestor <- node
7     newNode.ancestorConnectingLink <- link
8     var favorability <- applyAllHeuristics()
9     if (favorability < threshold)
10        return
11    end if
12    if (checkIndirectVirtualLinkLoop())
13       return
14    end if
15    var gVal <- node.distanceTillNow + link.cost
16    var fVal <- gVal + calculateHValue(node, newNode, link, favorability)
17    addToQueue(newNode, fVal)
```

Figure 4.5 – Pseudo code of enqueue method

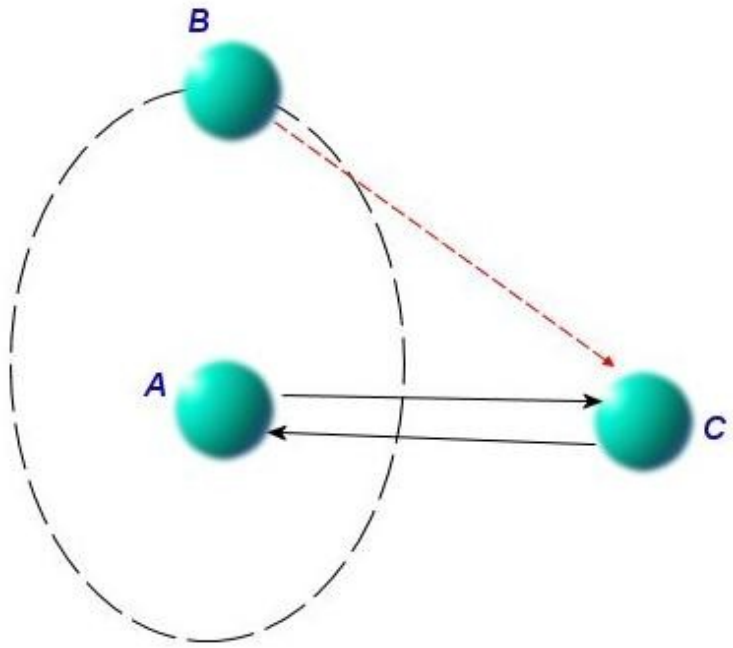

Figure 4.6 – Indirect virtual link loop. If node *B* is being processed, *B*-*C*-*A* route might occur if this check is not done. The virtual link from *B* to *C* is created because of the *B* contains *A* relationship and *A* to *C* link. So, in the graph it is not a loop but when the actual links in real life are considered, this condition is a loop.

```
1 processedBefore(node)
2   var tmpNodeArray <- visitedNodes.getNodeArray(node.Id)
3   if tmpNodeArray = null
4       return false
5   end if
6   if tmpNodeArray.Count < k
7       return false
8   end if
9   return true

1 addToVisitedSet(node)
2   var tmpNodeArray <- visitedNodes.getNodeArray(node.Id)
3   if tmpNodeArray = null
4       tmpNodeArray <- createEmptyArray()
5   end if
6   tmpNodeArray.addElement(node)
7   visitedNodes.addNodeArray(tmpNodeArray)

1 createRouteFromSource(node)
2     var route <- emptyRoute
3     var currentNode <- node
4     var link <- null
5     route.addNode(node)
6     while currentNode != start node
7         link <- currentNode.getAncestorConnectingLink()
8         currentNode <- currentNode.getAncestor()
9         route.addNode(currentNode)
10        route.addLink(link)
11    end while
```

Figure 4.7 – Pseudo codes of the helper methods

Enqueue method first checks for virtual link chains (line 2). Figure 4.6 shows an example virtual link chain. Then, the new node is created (line 5). This new node is the one that will be added to the queue (if it passes the checks). Its ancestor is initialized as the currently processed node and its ancestor connecting link is initialized as the currently processed link.

Than, at line 15, a favorability value is calculated. This is done by applying all heuristics one by one. If the favorability value turns out to be lower than the favorability threshold, *ft*, new node is not added to the queue.

Finally, the f-value is calculated by adding g-value with h-value.

4.3 Calculation of h-value and Heuristics

h(n) is the estimated cost of reaching from node n to target. We are calculating this value in three parts. First, we are calculating the geographical distance between node n and the target. After this, we are normalizing this value since the distance is in miles but we need a cost measure for summing h-value with g-value. Finally, we are applying several heuristic functions in order to calculate a favorability value r(n). According to the data entered by the users, this favorability value is increased or decreased. After this favorability value calculation is done, it's inverse (1 / r(n)) is multiplied by the normalized estimated cost in order to give the final h(n) value.

Calculating Geographical Distance

```
var x <- 69.1 * (Lattitude_t - Lattitude_s)
var y <- 69.1 * (Longitude_t - Longitude_s) * Cosine(Lattitude_t / 57.3)
var distance = SquareRoot(x^2 + y^2)
```

Figure 4.8 – Distance calculation

Normalization of the Distance

The cost estimate is based on the geographical distance between the new node and the target node. But this value, alone, is not enough for a proper estimate. The unit of this estimate is kilometers; it is the distance between the new node and the target node. On the other hand, unit of the cost till now is either a currency (financial cost) or time. So addition of these values with different units won't give us a proper overall f-value.

In order to normalize the value, we have developed two approaches. After developing and testing the first approach, we have realized that it has some disadvantages. Because of this, we have developed another approach.

Normalization – Approach 1

In this approach, we are normalizing the estimated cost by using three other values;

- Distance between the new node and the target
- Distance between the source and the new node
- Actual cost from the source node to the new node

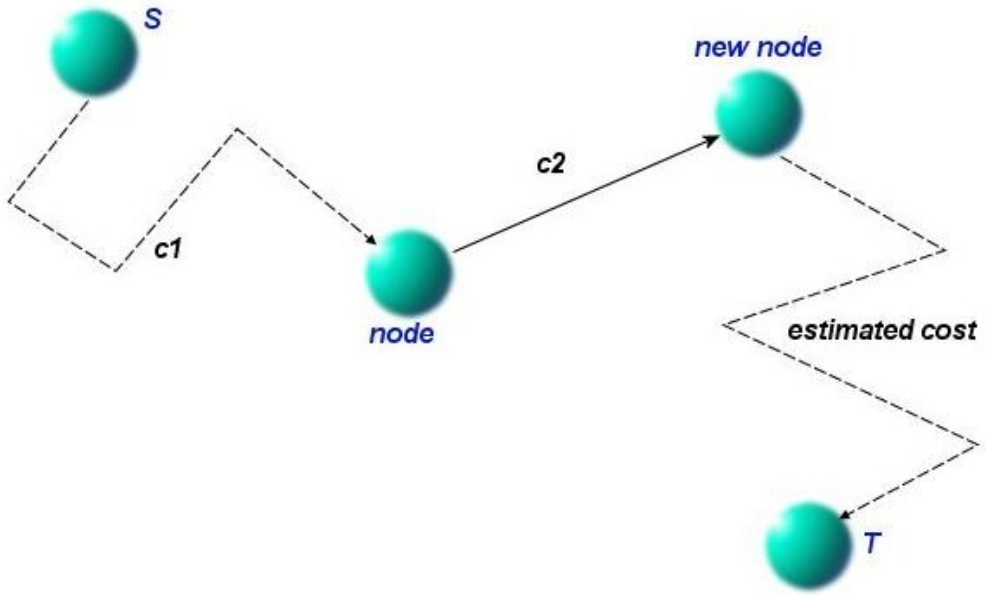

Figure 4.9 – Normalizing the distance. c1 is the actual cost between the source node and the currently processed node. There might be more than one link, only total cost of the route is important. c2 is the actual cost of the link connecting node and new node. Normalized estimated cost value is calculated according to c1, c2 and distance between new node and the target node.

The aim of this approach is to exploit the existing information to estimate a cost. We know the distance between the source node and the new node. We also know the cost between these nodes. Also, we know the distance between the new node and the target node. So, by a simple ratio calculation we can have a cost estimate between the new node and the target node.

distance(source, newNode) / (c1 + c2) = distance(newNode, source) / x

Although this approach seems as an applicable one, it has two main disadvantages. First of all, since type-2 virtual links (the links from the child nodes to the parent nodes) have a cost of 0, for nodes that are reachable by these links, the cost is underestimated.

Another disadvantage is the variable nature of the links. Consider that the query is between two very distant nodes (like two locations in different countries). Also consider that the route till now (route from the source node to the new node) contains only close locations to the source node. In this case, the estimate will be very different from the actual cost since links between close locations have different cost characteristics than links between distant locations. To be more specific, consider that the user has selected duration as the cost model. For close locations, x kilometers might take 2x minutes. On the other hand, for distant locations, if a plane will be used, 800x kilometers take x minutes.

Normalization – Approach 2

This approach provides a hard-coded conversion between miles and cost. The pseudo-code is given in Figure 4.10.

```
Normalize(distance)
    if distance < t1
        return distance * m1
    else if distance < t2
        return distance * m2
    else if distance < t3
        return distance * m3
    else
        return distance * m4
    end if
```

Figure 4.10 - Pseudo code of the applied normalization approach

The threshold values (t values) and multiplier values (m values) are different for duration and financial cost choices.

These are the hard-coded estimates. In a way, we are transferring our route expertise to the system by this normalization method. For example, if the distance is below 100 miles, we can say that the duration estimate will be distance * 1. Meaning each mile will take 1 minute to travel.

Of course, this approach has also disadvantages. Consider two distance values, $d_1 = t_1 - 1$ and $d_2 = t_1 + 1$. Although they have a very small difference, their cost estimate will be very different.

Also, for long distances, the multipliers should be arranged such that planes, trains, intercity buses, long distance ferries and such long distance links are all taken into consideration. Even with considering all such links, there will still be erroneous estimates since the multiplier would have been selected as an average of the several link types.

In spite of these disadvantages, this approach outperformed the first approach. Chapter 6 presents the results with this approach. When we have tested the system with the first approach, percentage of the queries, that returned the same route for both algorithms, A*CD and Dijkstra's, was below 50% for both point of interests (duration and financial cost). As you can see in that section, the percentages with the second approach are 65% for duration and 52% for financial cost.

In addition to the better results than the first approach, this approach has another important advantage over the first one. Its performance can be improved by tuning the constants; thresholds and multipliers. As the system is queried by the users, we will have many location pairs and actual route costs for them. So we will have a big data set, from which we can derive results to modify the constants.

Also, condition dependent applications can be applied by this approach. For example, if the normalization is for two locations that are residing in a crowded city, the estimated cost (result of the normalization) might be increased by multiplying it with a constant factor, which is greater than 1. The increase is for reflecting the effect of the traffic to the estimated cost.

Heuristics

Heuristics are used for calculating the favorability value. The favorability value is initialized as 1. Each heuristic updates this value by multiplying it (either with a value greater than 1, meaning the favorability is increased or with a value smaller than 1, meaning the favorability is decreased).

```
calculateFavorability()
    var favorability <- 1
    for each Heuristic h
        favorability <- applyHeuristic(h, favorability)
    end for
    return favorability
```

Figure 4.11 – Pseudo code of favorability calculation method

If the favorability value is under a certain threshold, the link is not processed and it is discarded (Line 9 of enqueue method).

In the heuristic explanations below, hc_n indicates a heuristic constant.

Heuristic 1

Goal: In the searches that have the source and destination close to each other, long distance links (such as planes) can be eliminated to reduce the search space.
How: Let d = distance between source and target locations.

- Multiply the favorability with 0 if the processed link is a Plane and $d <= hc_1$ kilometers

- Let d_2 = the distance between route source and route destination. If $d_2 > 5d$, multiply favorability by 0.
- If $d_2 > 2d$, multiply favorability with hc_2 where $hc_2 < 1$

Notes: This heuristic is for reducing the search space. An important percentage of the searches will be done for close source and target locations (in the same city, same district, same campus or even same building) according to our estimates. For such searches, including planes, increases the search space dramatically because with a plane hop, many new links and locations that resides near the plane's target location are introduced, which will slow down the system dramatically.

Heuristic 2
Goal: When the processed link gives nearer results to the destination, the successor should be given higher priority.
How: Let d_1 = dist(linkTarget, routeTarget), d_2 = dist(linkSource, routeTarget)

- If $d_1 / d_2 < hc_3$, multiply favorability by hc_4
- If $hc_3 < d_1 / d_2 < hc_5$, multiply favorability by hc_6
- $hc_3 < hc_5 < 1 < hc_6 < hc_4$

Notes: This heuristic is for giving high priority to links that seem to get us closer to the route destination. The heuristic is useful for both long distance searches and short distance searches. For long distance, a plane that is landing to the destination location's city will be promoted, since it gets us closer. For short distance searches, since the ratio is important, the heuristic is again applied successfully.

Heuristic 3
Goal: As the number of common parent locations between the processed link's target location and the route target location increases, give higher priority to the current route.

How: Iterate through parent lists of link's target and route target. For each common one, multiple favorability by hc_7 where $hc_7 > 1$.

Heuristic 4
Goal: Lower the priority of virtual links. They are less desirable and offered in the system for supplying information when no direct route can be found.

How: If the processed link is a virtual link, multiply favorability by hc_8 where $hc_8 < 1$.

Heuristic 5
Goal: As number of links increases priority should be decreased. Minimum number of hops is a general requirement for shortest routes.

How: Favorability is calculated by $1 - (hc_9 * \text{numberOfLinks})$ where hc_9 is a small constant which is close to 0.

Notes: The influence of this heuristic is small. It is useful when two routes have not been differentiated by the other heuristics. By this one, similar routes are ordered according to the number of links that they contain.

Heuristic 6
Goal: Move the links that are reaching us to the goal to the first place in the priority queue. As indicated before, at the end of the algorithm details section, our approach does not return the result immediately; it first processes it as a regular partial route. As indicated before the reason is to have an ordering if a node has two links to the target location.

How: Check current link's destination. If it is goal location, multiply the favorability with a high constant.

4.4 An Example Execution Scenario

Consider that the information (locations, links and relationships) in Figure 4.12 has been entered to the system by the users. In this graph, consider that a route starting from Bilkent and ending at *Kadıköy* is requested. Also assume that, k = 2, meaning 2 routes are requested.

The contents of the queue are given in Table 4.2. First value in the parentheses indicates the ancestor of the node, second value indicates the ancestor's instance number (as indicated before, same node might be added to the queue if more than one route has been found to that node). Asterisk near a node means that, the node has the lowest f-value. So it will be processed next. Red entries indicate the new nodes that have been added while processing the previous node.

At step five, first route to *Kadıköy* is found. Then the algorithm continues and at the next step, it finds the second route. Found routes are as follows;

- *Bilkent – AŞTİ – Harem – Taksim – Kadıköy*
- *Bilkent – AŞTİ – Harem – Beşiktaş – Kadıköy*

Step 1 Bilkent (-) *	Step 2 AŞTİ (Bilkent-1)* Ulus (Bilkent-1)	Step 3 Harem (AŞTİ-1)* ODTÜ (Bilkent-1) Ulus (Bilkent-1)	Step 4 ODTÜ (Bilkent-1) Ulus(Bilkent-1) Beşiktaş(Harem-1) Taksim(Harem-1)*
Step 5 ODTÜ (Bilkent-1) Ulus (Bilkent-1) Beşiktaş (Harem-1)* Beşiktaş-2 (Taksim-1) Kadıköy(Taksim-1) A. Airport(Taksim-1)	Step 6 ODTÜ (Bilkent-1) Ulus (Bilkent-1) Beşiktaş (Taksim-1) A. Airport(Taksim-1) Taksim-2 (Beşiktaş-1) Kadıköy (Beşiktaş-1)		

Table 4.1 – Queue contents

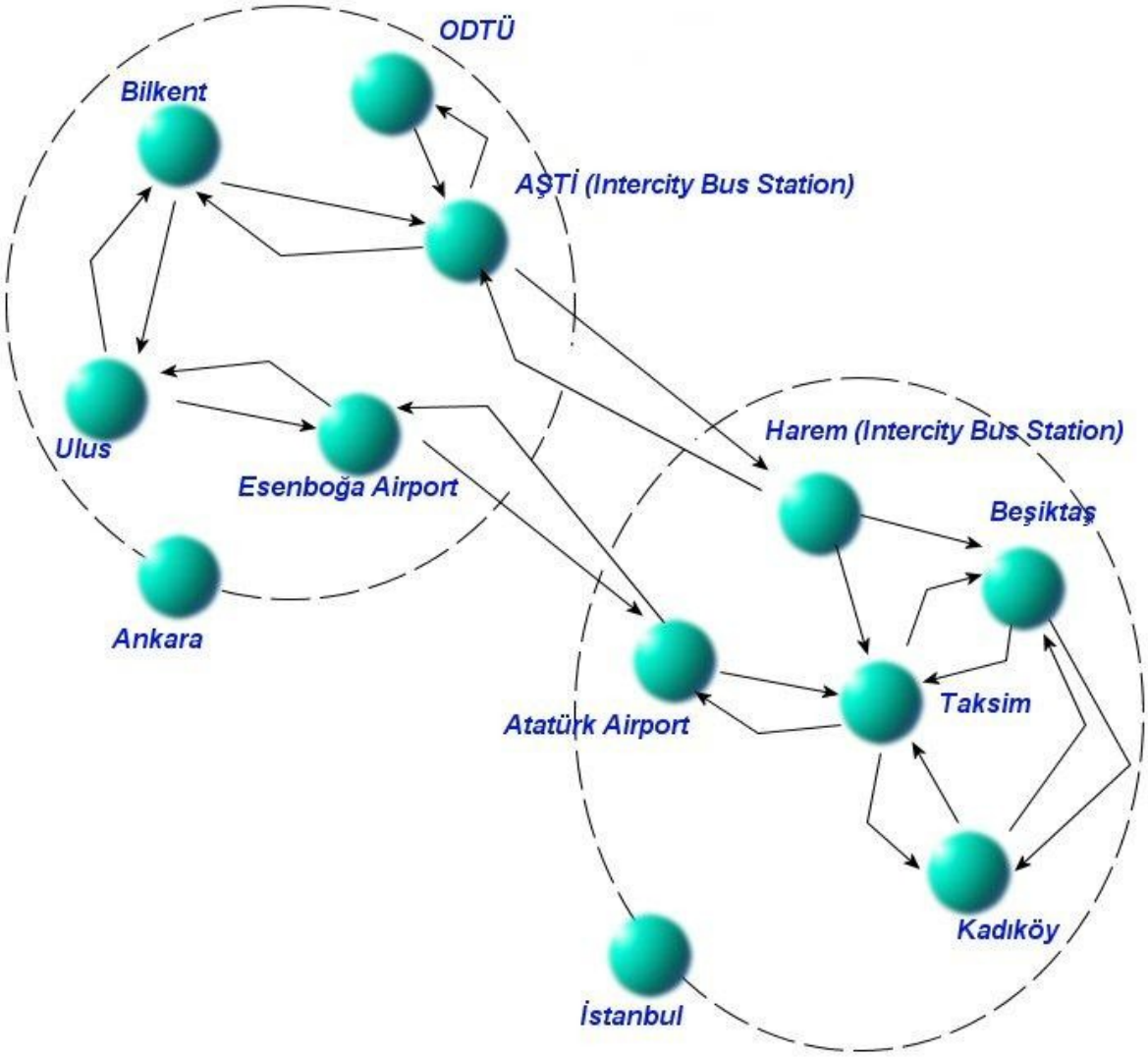

Figure 4.12 – Example graph. *Ankara* contains five locations in the upper left side. *Istanbul* contains five locations in the lower right side. Actual links entered by the users are indicated by arcs. Virtual links are not shown to simplify the example. We assume this data set has been entered by the users.

Although this is a simple graph and a simple example, it gives us some important hints about our search algorithm;

- Search continues in the destination city whenever a location in the destination city is reached. Because of heuristic 3, favorability value of *Harem* is increased. This decreases the estimated cost of reaching from *Harem* to the target location. So, the system continues with *Harem Bus Station* instead of *Ulus*, which resides in *Ankara*.
- Multiple instances of the same node can be added to the queue. In this example, both *Beşiktaş* and *Taksim* are added to the queue two times. Because of this property, each node stores both its ancestor and its ancestor's instance number. Each node knows that, "My ancestor is x^{th} instance of node y" (each node also stores its ancestor connecting link but we have skipped that detail for this example for the sake of simplification).

- The algorithm avoids cycles in the routes. For example, while processing *Harem Bus Station*, the algorithm skips the link starting from *Harem* and ending at *AŞTİ* because it detects that the route till *Harem* contains *AŞTİ* already.

4.5 Virtual Link Preference

We have presented the heuristic four, which is decreasing the favorability value of the nodes that are reached by virtual links. Preferring the routes without virtual links over the routes with virtual links is not a rule without exception. There might be scenarios in which a route with virtual link(s) might be preferred instead of a route without virtual links. Consider the following scenario.

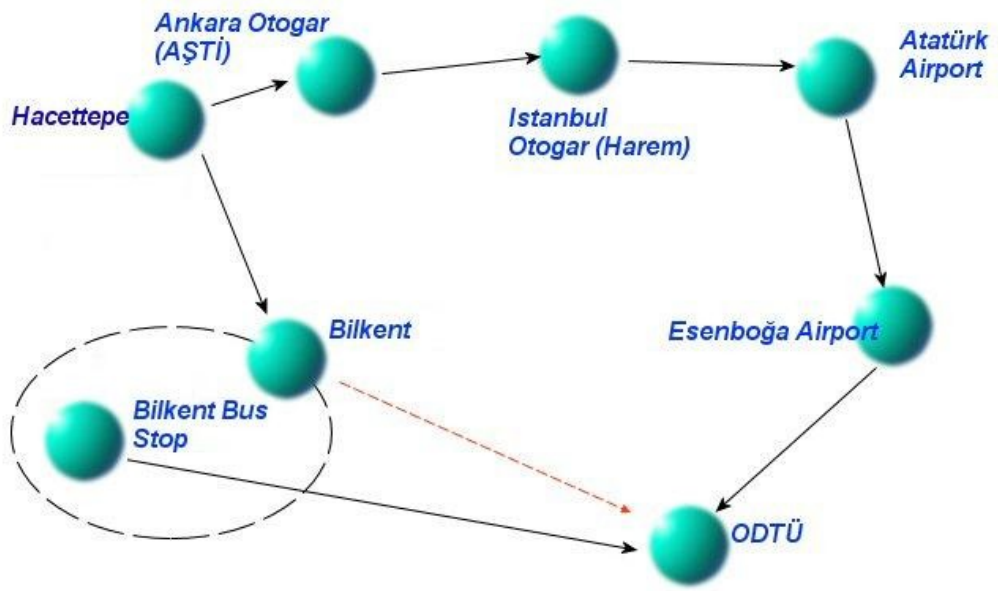

Figure 4.13 – Virtual link preference. Black arrows indicate links that have been entered by the users. Red dashed arrow from *Bilkent* to *ODTÜ* indicates a type-1 virtual link. We want a route from *Hacettepe* to *ODTÜ*.

The user requests a route from *Hacettepe* to *ODTÜ* in the scenario, which is given in Figure 4.13. There is a route which is not containing virtual links;

Hacettepe – AŞTİ – Harem – Atatürk Airport – Esenboğa Airport – ODTÜ

But this route visits Istanbul. Instead of this route, there is another route which is using a virtual link;

Hacettepe – Bilkent – ODTÜ

This route has a virtual link, *Bilkent* to *ODTÜ*. In this scenario, the route with the virtual link would probably be much more useful to the user since the other route travels 500 kilometers to another city (*Istanbul*) and then comes back to the city (*Ankara*) that is containing the source and the target locations.

Consider that we have two nodes, n_1 and n_2. Consider that, for n_2, a virtual link is being processed and for n_1 a link that is not virtual is being processed. Node n_2 will be preferred if

$$g(n_1) + nec(n_1) > g(n_2) + (nec(n_2) / hc_8)$$

where nec(n) stands for "Normalized Estimated Cost" of node n and hc_8 is the constant of heuristic four. In this formulation, it is assumed that all other heuristics calculate the same favorability value for n_1 and n_2.
For the scenario in Figure 4.13, consider that *Bilkent* and *Atatürk Airport* are being compared. Since g-value of Atatürk Airport will be much larger than the g-value of *Bilkent* (the actual cost of the route from *Hacettepe* to *Atatürk Airport* is greater than the actual cost of the route from *Hacettepe* to *Bilkent*), the above formulation will hold. So, the system will prefer the route with the virtual link and provide *Hacettepe – Bilkent – ODTÜ* route to the user.

Chapter 5

User Interface for Data Entry and Route Query

In this chapter, we provide the details of our system's user interfaces. In the first five sections we present the interfaces for data management. In section 5.6, we talk about how the data management operations are logged. In the last section we present the route query screen.

The following is a list of features related with data management. A user can,

- Enter a new location by specifying its name, country, city, address, type and explanation. Name, country, city and type are mandatory. Geographic coordinates cannot be entered during entry of a new location. Geographic coordinates are initialized with the same values that are used for the city selected.
- Enter a new link by selecting the source and the target locations, specifying the link's type, explanation, guessed duration, guessed financial cost and currency for guessed financial cost. All properties except explanation are mandatory. A user cannot enter a virtual link. As indicated before, virtual links are managed by the system.
- Enter a new relationship by selecting two locations and specifying the relationship type. As mentioned before, there are three kinds of relationships, containment, neighborhood and intersection.
- View all properties of a location.
- View all links that are originating from a location.
- View all links that are ending at a location.
- Update geographic coordinates of a location.
- Delete a location. In order to delete, the Delete hyperlink is used. Upon click, this hyperlink deletes the location together with its relationships and links (both incoming and outgoing). Countries cannot be deleted. There are no other constraints.
- Delete a link. Virtual links cannot be deleted. As indicated before, they are all managed by the system.
- Delete a relationship between two locations. There is no constraint for relationship deletion operation.

These features are explained in detail in the following sections.

5.1 Searching Locations

The collaborative data entry processes start with this feature. Users can search locations by their name. If no location with the entered name exists, they can create a new location. If one or more locations matching the query term are retrieved from the database, links are provided to the user to manage information related with the locations (details, relationships and incoming & outgoing links).

Figure 5.1 – Screenshot of location search

Users can search locations according to their names. The system displays the results to the user, together with links to manage information about the retrieved locations.

From this search screen users can go to location details page, location relationship management page and link management page.

“Set as Source” and “Set as Destination” links are related with route finding.

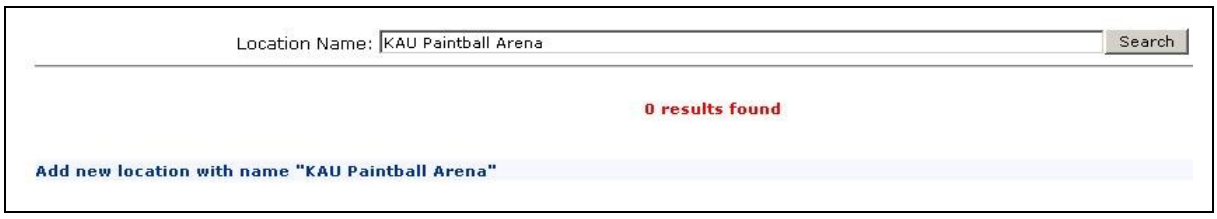

Figure 5.2 – Screenshot of location search. No result can be found.

Above screenshot shows the case in which no result can be found. In this case, a link is provided to the user to enter a new location.

5.2 Entering a New Location

By following the link from the location search screen, a user can reach to the new location entry screen.

The user can specify the name, country, city, address, type and explanation of the new location. Name, country, city and type fields are mandatory.

Figure 5.3 – Screenshot of new location entry.

5.3 Managing Location Information

Location information can be managed from the location details screen. As indicated before, this page is reachable from the location search screen.

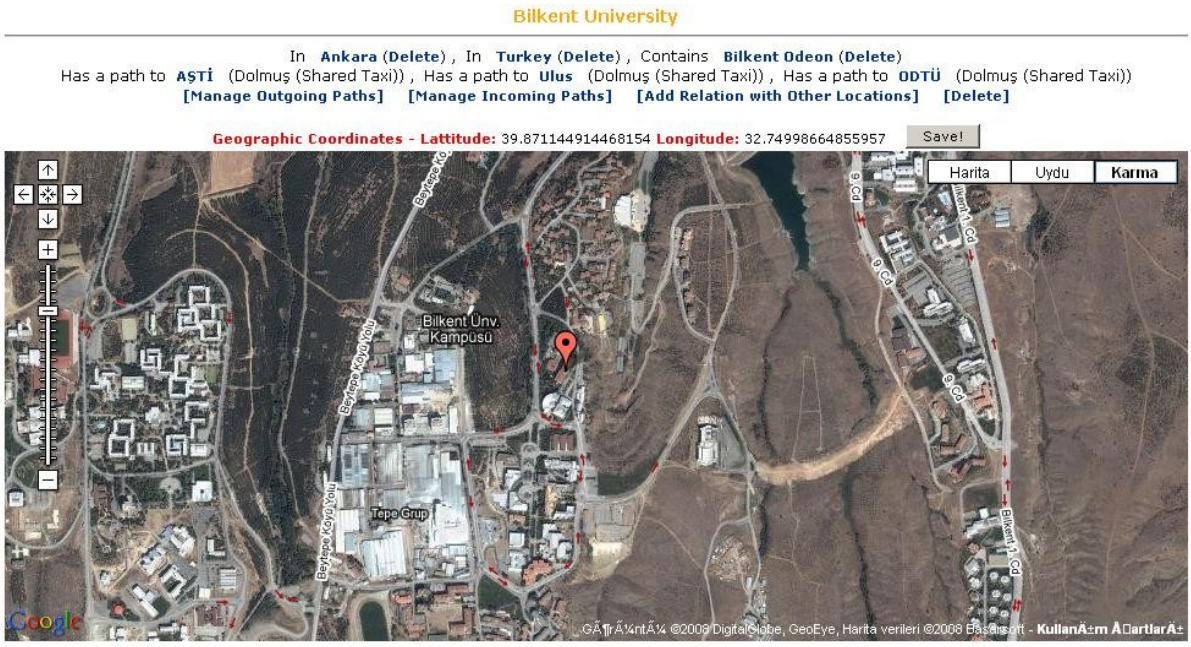

Figure 5.4 – Screenshot of location details page. This screenshot is for Bilkent University.

Location details page contains the following information,

- Location relationships. Delete hyperlinks near each relation provide the deletion feature of location relationships.
- Links originating from the location. There is a Delete hyperlink near each link. These hyperlinks provide the deletion feature of links. Countries cannot be deleted as indicated before. The system displays an error if the user tries to delete a county.
- Geographic coordinates of the location.
- A map, from which the user can change the geographical coordinates of the location. This map service is provided by Google.

This page also displays hyperlinks for managing location relationships of this location, managing outgoing links and managing incoming links. Also a delete hyperlink is provided for the deletion of the location.

As a location is deleted, links originating from it, links ending at it and all its relationships are also deleted.

5.4 Managing Location Relationships

Location relationships can be managed from the relationships page. This page is displayed for a specific location. The user can search the second location according to its name, to relate with. As the results are displayed, the user can add intersection, neighborhood or containment relationship.

Figure 5.5 – Screenshot of location relationships page.

In the above screenshot, location relationships page is opened for Bilkent University. In this page, the existing relationships are displayed to the user.

For adding new relationships, a search text box exists. As the user types a name to this field and clicks on the search button, locations are searched and retrieved according to their name. For each retrieved location, links for adding relationships are also displayed. So, user can relate any other location with the current location on screen.

5.5 Managing Links

For link management, there are two similar pages; one for managing incoming relationships and the other one for managing outgoing links. Only difference between them is that one displays incoming links and the other one outgoing links.

Screenshot in Figure 5.5 is the page in which outgoing links from Bilkent University is displayed. User can delete the links by using the “Delete” hyperlinks. Delete operation does not delete the reverse link if it exists.

Figure 5.6 – Screenshot of location relationships page.

From the search field, the user can enter a location name and search for locations to add new links. After the location is selected after search (using the checkbox), the link type, explanation, guessed duration (in minutes), guessed cost and its currency are entered. Also, user can denote that the link is also valid for reverse direction. This feature is for entering bi-directional links. On the data model, instead of a single bi-directional link, two uni-directional links are stored as indicated before.

5.6 Logging Mechanism

For all data entered by the users of the system, we are logging the actions. We are using a logging mechanism in which logs contain the following information;

1. **Location ID:** If the operation is related with a location, the ID of the location is stored.
2. **Link ID:** If the operation is related with a link, the ID of the link is stored.
3. **Relation ID:** If the operation is related with a location relation, the ID of the relation is stored.
4. **User ID:** ID of the user that has entered the information.
5. **Date:** Date of the operation.
6. **Old data:** This is stored as raw data in the database. The logging module forms and parses this information when necessary. If the operation is link delete, all information about the link (source location ID, target location ID, link type, guessed duration, etc…) is stored in this field. Similarly, for location delete and relationship delete operations, all data of the deleted entity is stored in this field. If the operation is geographical coordinates update for a location, old coordinates are stored in this field.
7. **New data:** This is stored as raw data in the database. The logging module forms and parses this information when necessary. For delete operations, this field is empty. For entry of new entities (location, link, location relationship), this field contains all information for the new entity. For geographical coordinate update of a location, this field contains the new coordinates.
8. **Log type:** Type of data entry (examples are new location entry, new link entry, new relationship entry, geographical information change, location deletion, link deletion and relationship deletion)

Below are the event types that are logged in the system.

1. Addition of a new location
2. Addition of a new link
3. Addition of a new location relationship
4. Deletion of a location
5. Deletion of a link
6. Deletion of a location relationship
7. Geographical information change of a location

Since the system does not have update features for locations, links and relationships, there is no event type other than the above seven event types.

5.7 Route Query

In order to query the system, the user should first select the source and the target locations. Then the cost model preference (duration or financial cost) should be entered. This indicates if the user is interested in a fast route or a cheap route.

User can also exclude certain link types and request alternative routes.

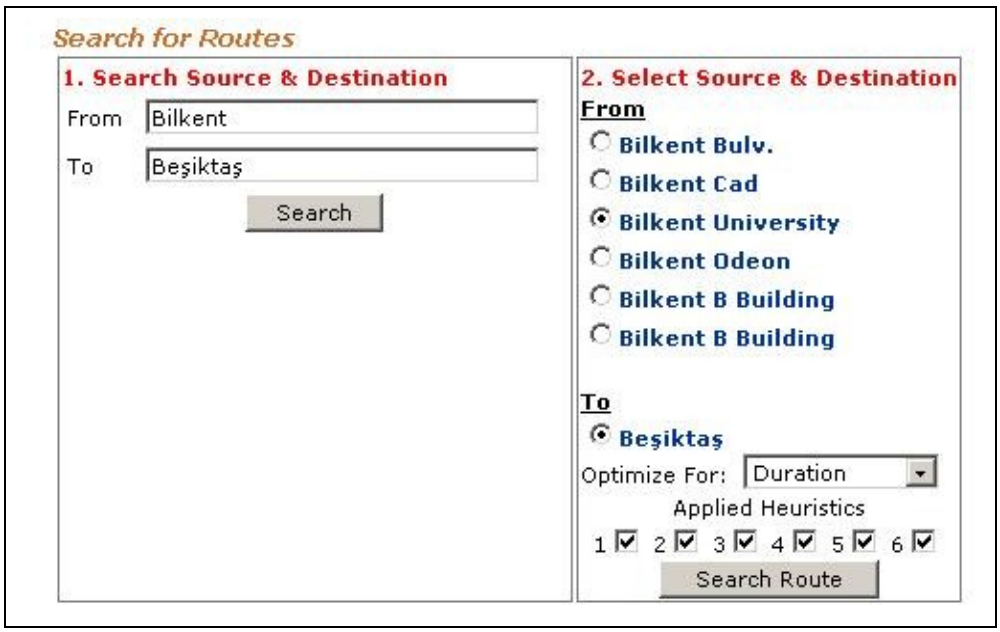

Figure 5.7 – Search results are displayed. User selects the cost model (duration or financial cost). The selection of the applied heuristics is provided for test purposes.

As the user clicks on the “Search Route” button, the system searches for a route between the selected locations and display the result to the user. The link types used in the provided route are displayed to the user together with check boxes near them so that the user can exclude those link types for the alternative route search.

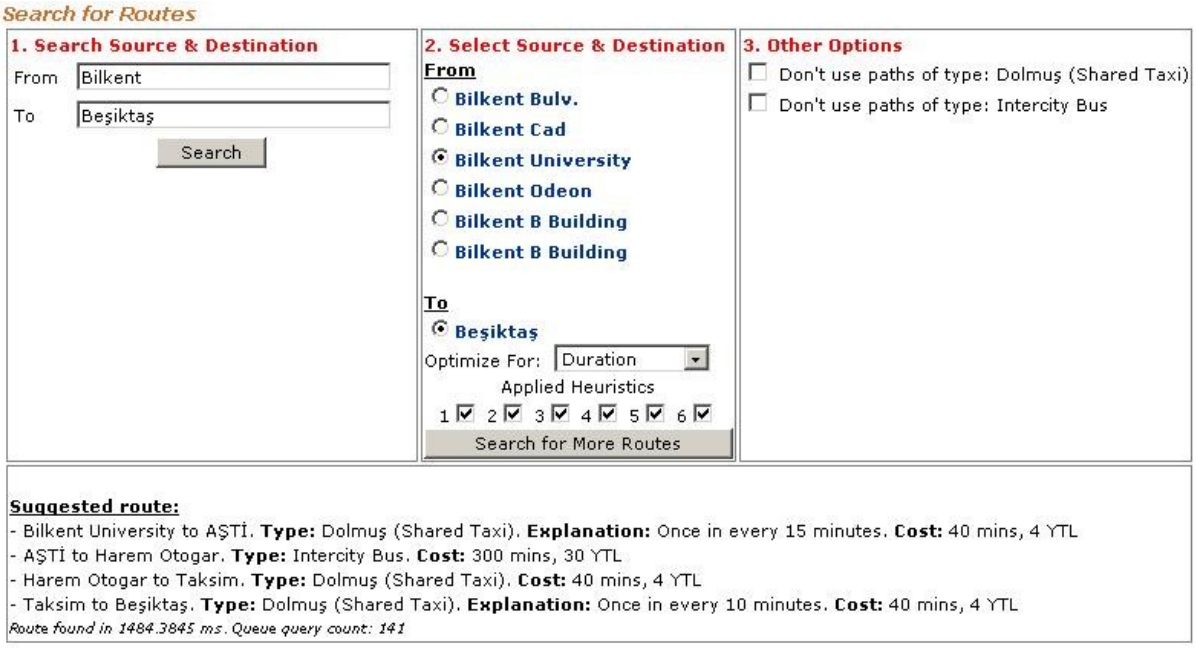

Figure 5.8 – A route is provided to the user. Used link types can be excluded from the alternative route search.

Chapter 6

Search Algorithm Evaluation

In this chapter, we first present the cases in which our algorithm misses an existing route. Then, we give the details of the data set that we are using for automated and manual test. Then, we present four example queries for which we query the system by disabling certain heuristics. By these four queries, we discuss the effects of the heuristics over the algorithm and the provided routes. Finally we present automated tests by which we are collecting valuable information for evaluating the performance of our search algorithm.

6.1 Missed Cases

In order to present the missed cases, we have the following assumptions;

- There exists a route from the selected source location to the selected target location. These locations are named *S* and *T*.
- "Excluded solutions" is an empty set. As explained before, this set is used to exclude solutions that are previously found. So we can assume it as empty since all searches will start with this set empty.
- "Excluded link types" is an empty set. We will assume that all links can be accepted in the resulting route.

Now, assume that, we do not have any skip code (continue lines) in the algorithm. Also assume that enqueue method always adds the current link to the queue. With these assumptions it is obvious that a route can be found by the algorithm. Because it starts from the source node, exploits all links originating from source and adds partial routes that are formed by adding the successor nodes to the source node to the queue. And it continues until queue is empty.

So, what we have to do in order to continue the proof is to attack the continue lines and the enqueue method.

We have five continue lines;

- **Line 11:** This skips the current partial route since the last location in the partial route is in the closed set. If we analyze the operations on the closed set, we can see that addition to this set is only executed at line 13. This means that, if algorithm skips the current partial route due to line 11, it should have visited line 13 before. This concludes that, the node has been processed before so it is not possible to miss a solution because of this line.

- **Line 28:** This line skips the current link if current link is an element of excluded link types. As indicated before, we can assume that this set is empty.
- **Line 32:** This line skips the current link if current link is a virtual link and the actual link that is causing this virtual link has a type that exists in excluded link types. This can also be ignored like Line 28.
- **Line 35:** Line 34 checks current link's destination location. If it exists in the current partial route the link can be skipped. This means that the route till now already contains the destination location of the current link, meaning that this destination location has already been processed. Routes formed by adding its successors to the partial route might still exist in the queue waiting to be processed. So, no successor of destination location can be missed.
- **Line 19:** This line is executed after the solution has been added to the found routes set so missing a solution because of this line is impossible.

Since it is impossible to miss a solution because of a discard in the algorithm, we have to check the enqueue function. As indicated before, it applies heuristic methods one by one and calculates a favorability value for the processed partial node. We have also indicated that if the calculated favorability value is under a certain threshold, *ft* (favorability threshold), the node will be discarded. So, we have to analyze the heuristics that are decreasing the favorability of the processed node.

- Heuristic 1
- Heuristic 4
- Heuristic 5

These three heuristics might decrease the favorability of the processed route. If the favorability value is dropped under a threshold, *ft* (favorability threshold), the processed node will be discarded. For values over *ft*, we do not have a problem since they will be added to the queue. So, we have to analyze the cases in which favorability can drop under *ft*. These are the cases in which, even if a route exists in the system from *S* to *T*, the algorithm cannot find it.

Case 1 - Discard Due to 0 Favorability: This case is caused by heuristic 1. If the distance between *S* and *T* is less than hc_1 kilometers and the processed link is a plane, favorability is set to 0. If the existing route in the system has such a link, this route will be missed.

Case 2 - Discard Due to 0 Favorability: This case is caused by heuristic 1.

- d_1 = distance between *S* and *T*
- d_2 = distance of the processed link
- $d_2 > 5 \times d_1$
- Existing route between *S* and *T* contains this link

In this case, the route will be missed.

Case 3 – Discard Due to Low Favorability: Such a case might occur if favorability value is not set to 0 but dropped under *ft*.

Heuristic 1 may decrease the favorability by multiplying it by hc_2. Heuristic 4 decreases the favorability if the processed link is a virtual link, by multiplying it by hc_8, which is smaller than 1. Heuristic 5 initializes the multiplier from 1 and subtracts hc_9 for each link in the route.

Now, consider that no heuristic increases the favorability. In that case, following situations will discard a valid route from *S* to *T*.

1. There exists a route *R* from source to destination. *R* contains at least one link *L* such that;
    a. *L* is a virtual link (meaning favorability will be multiplied by hc_8)
    b. hc_2 multiplier is applied by heuristic 1
    c. Let n = route's link count till *L*
    d. $1 \times hc_8 \times hc_2 \times (1 - (n \times hc_9)) < ft$
2. There exists a route *R* from source to destination. *R* contains at least one link *L* such that
    a. *L* is a virtual link
    b. Let n = route's link count till *L*
    c. $1 \times hc_8 \times (1 - (n \times hc_9)) < ft$
3. There exists a route *R* from source to destination. *R* contains at least one Link *L* such that
    a. hc_2 multiplier is applied by heuristic 1
    b. Let n = route's link count till *L*
    c. $1 \times hc_2 \times (1 - (n \times hc_9)) < ft$

As a summary to unhandled cases, we can conclude that, only favorability decrease features cause the system to miss some existing solutions. There are two cases in which favorability is set to zero and cause a solution to be missed.

Also, there are three situations which decrease the favorability below threshold but these are not very likely to occur.

There is another check in the beginning of the enqueue method. This check is at line 2 and guarantees that no virtual link chain is used in the solution. If processed link's target location exists in the visited locations of the processed node's route till source node, a loop is detected. So missing a solution because of this case is not possible.

Other than these cases, system is able to find a route from source to destination if such a route exists in the graph.

6.2 Example Data Set

In our example data set, we are using five sub data sets.

- Countries and cities
- Ankara bus network
- Istanbul bus network
- Istanbul ferry network
- Turkey domestic plane network (of a private company)
- Turkey intercity bus network (of a private company)

As a summary of all the added example data, we can say that three networks have been formed; Turkey, Ankara and Istanbul.

Ankara network contains only bus stops. There are a total of 271 bus stops. Number of links is around 1000.

Istanbul network contains bus stops and ferry bridges. There are around 300 locations and 600 links between these locations.

Turkey network consist airports and intercity bus terminals. There are around 125 locations and 325 links. This network connects locations in Ankara and Istanbul.

The details of the example data set are given in appendix B.

Statistics for example data set
Number of locations ~ 4000
Number of links ~ 2000

This data set is used for all the tests in the following sections.

6.3 Effects of Heuristics

In order to demonstrate the effects of the heuristics, we present example queries. For each example query, we get five alternative routes from the system by first disabling the tested heuristic(s) and then get five alternative routes from the system by enabling the tested heuristic(s).

There are two values for each suggested route; duration of search and number of nodes extracted from the queue, the queue query count.

Execution time in milliseconds might be considered as a good measure for comparison but there might be variations because of the other processes running in the operating system. On the other hand, queue query count is a good measure for comparing the performances of approaches. It indicates how many nodes have been extracted from the queue, giving us a comparable measure. Only problem of this measure might be the unbalanced nature of the edges. It is possible that some nodes contain more links than some other nodes. In this case, processing of a node with more links will take longer than the processing of a node with fewer links. The reason is the running time of each node processing, O(|L|) where |L| is indicating the number of links originating from the node. We are ignoring this unbalanced link count situation for the following queries.

The routes that the algorithm has found with and without the tested heuristics are provided in appendix C.

QUERY 1
In this example query, first the system is queried without heuristic 1, and five suggested routes are retrieved from the system. These results are compared with the five suggested routes from the system when all heuristics (including heuristic 1) are applied.

When the heuristic 1 is not applied, two suggested routes out of five contain locations that are very far away from both source and target locations. Although source and target location are close to each other and contained in the same city, these two routes pass from locations that are in another city (Istanbul), which is far away from the search city (Ankara). This is not a desired property of a fast route in our context.

QUERY 2

This example query is executed in order to demonstrate the importance of heuristic 2. As indicated before, heuristic 2 increases priorities of links that are getting the search closer to the target node. This is done by multiplying the priority with a constant greater than 1 whenever the ratio of distance(linkTarget, routeTarget) to distance(linkSource, routeTarget) is smaller than a threshold.

In order to demonstrate heuristic 2, we have turned off heuristic 3. Because with heuristic 3 turned on, the results were very close to each other. The reason of this is that, containment information also exists in the system for the example data set, together with geographical information. We did not want that location relationship information to affect the results and we wanted an isolated test for heuristic 2.

The results present a huge difference when number of queue pops is compared. The algorithm starts searching from the source node, “Çetin Emeç Bulv.”, which resides in Ankara. It finds a link to AŞTİ (Ankara bus terminal) after a while. When AŞTİ is processed, Harem Otogar (Istanbul bus terminal) is added to the queue since there is a link from AŞTİ to Harem Otogar. In the first execution, this newly found node is added to the queue without any decrease in its estimated cost. On the other hand, in the second execution, heuristic 2 decreases estimated cost by increasing the favorability value. With heuristic 2, algorithm sees that this link covers an important ratio of the distance between source and target.

So, in the first execution, a route can be found after 1399 queue pops. On the other hand, in the second execution it only takes 185 queue pops to find a route to the target. Alternative routes are also found with less number of pops from the queue in the second execution.

QUERY 3

This query is executed in order to demonstrate the effects of heuristic 3. Similar to query 2, in order to isolate the effects of heuristic 3, heuristic 2 is also turned off for this test. As indicated before, these two heuristics behave very similar since both containment information and precise geographical information are complete in our example data set.

Heuristic 3 takes advantage of location relationships, containments to be more specific. In this query, we are searching for a route from Hoşdere (in Ankara) to Kadıköy (in Istanbul).

In the first execution, heuristic 3 is not applied. It takes 31 queue pops to find a route to the target. On the other hand, in the second execution it takes only 9 queue pops to find a route to the target. In the second execution, whenever the algorithm finds a route to Atatürk Airport, which resides in Istanbul, it gives higher priority to this node since it contains a common parent (Istanbul) with the target node. This leads the algorithm to the target node quickly. As it can be checked from the results, this condition is also valid for alternative routes.

QUERY 4

This query is executed in order to demonstrate the effect of heuristic 4 on the algorithm. Heuristic 4 is used to give lower priority to virtual links. As explained before, virtual links and actual links are treated the same in the search algorithm. The algorithm does not differentiate between these two. Since virtual links main purpose is to provide intuitive connections, they seem unnecessary when there is a route from the source to the target, which is containing only actual links.

The routes with virtual links should be found after the routes without virtual links (this conditions has some exceptions, as explained in section 4.5). In order to accomplish this, we have defined heuristic 4.

As seen in the results, heuristic 4 and this demonstration are not related with running time of the algorithm. They are only related with the quality of the results. In this context, we are defining the quality as the completeness of the search result.

In the first execution, heuristic 4 is not applied. First suggested route contains two virtual links. Although this route seems as a valid answer from the system, it is using intuitive connections. In the first suggested route, the information for reaching from Hoşdere to Esenboğa Airport is missing. This situation is also valid for alternate routes in the first execution. On the other hand, the results for the second execution contain complete information. In the second execution, no route contains a virtual link (routes with virtual links exist but routes without them are preferred first).

6.4 Automated Tests

Our main purpose in these tests is to test A*CD algorithm's results' (routes') quality, by comparing their cost with the costs of the actual lowest cost routes. In order find the lowest cost routes, we have implemented the Dijkstra's 1 - *n* shortest path algorithm [11].

For the implementation of the Dijkstra's algorithm, we are not using any cache during the execution of several searches. We are retrieving the links and locations from the database for each execution. A*CD also runs without any cache.

We have selected one hundred random location pairs from the example data set for duration cost model and one hundred random location pairs for financial cost model. Logged information for each pair is as follows.

- Cost of the route provided by Dijkstra's algorithm
- Execution duration of Dijkstra's algorithm
- Queue query count of Dijkstra's algorithm
- Cost of the route provided by A*CD algorithm
- Execution duration of A*CD algorithm
- Queue query count of A*CD algorithm

Test result details for duration cost model are given in Table 6.1 and details for financial cost model are given in Table 6.2.

Percentage of queries that return same cost for both algorithms	%65
Minimum cost ratio (A* / Dijkstra)	1
Maximum cost ratio (A* / Dijkstra)	1.404
Average cost ratio (A* / Dijkstra)	1.03
Minimum queue query count ratio (A*CD / Dijkstra)	0.002
Maximum queue query count ratio (A*CD / Dijkstra)	4.29
Average queue query count ratio (A*CD / Dijkstra)	0.93
Minimum search duration ratio (A*CD / Dijkstra)	0.002
Maximum search duration ratio (A*CD / Dijkstra)	1.32
Average search duration ratio (A*CD / Dijkstra)	0.32
Number of unanswered queries (Dijkstra)	11
Number of unanswered queries (A*CD)	11

Table 6.1 – Test results for duration cost model.

Percentage of queries that return same cost for both algorithms	52%
Minimum cost ratio (A* / Dijkstra)	1
Maximum cost ratio (A* / Dijkstra)	1.75
Average cost ratio (A* / Dijkstra)	1.27
Minimum queue query count ratio (A*CD / Dijkstra)	0.003
Maximum queue query count ratio (A*CD / Dijkstra)	4.04
Average queue query count ratio (A*CD / Dijkstra)	1.67
Minimum search duration ratio (A*CD / Dijkstra)	0
Maximum search duration ratio (A*CD / Dijkstra)	1.467
Average search duration ratio (A*CD / Dijkstra)	0.43
Number of unanswered queries (Dijkstra)	9
Number of unanswered queries (A*CD)	9

Table 6.2 – Test results for financial cost model

Results for all queries and example query results are provided in appendix D.

6.4.1 Comments on Results

Missed Solutions (Routes)

In our example queries, no route has been missed between the source and the target nodes with A*CD algorithm. For the duration focused queries there are 11 pairs that do not have a route between them. For the financial cost focused queries there are 9 pairs that do not have a route between them. Since a route does not exist, Dijkstra's algorithm could not find routes between those pairs also. So we can say that, cases that are provided in the theoretical capability analysis did not occur with the example data set and random queries.

Percentage of Ideal Results by A*CD

Although the term ideal result is very relative in our context, we can consider the cost as a solid measure for the success of our algorithm. For the duration, %65 of the A*CD search results returned the same route with the Dijkstra algorithm. For the financial cost, %52 of the A*CD search results returned the same route with the Dijkstra algorithm. This means that, in more than half of the queries, the actual shortest path is returned by our algorithm.

Worst Case Cost
For the duration criteria, the worst result gave us a ratio of 1.4. This means that the route found by the A*CD algorithm has a cost which is, the cost of the actual shortest path multiplied 1.4. For the financial cost criteria, the worst case cost ratio is 1.75.

Queue Query Count Comparison
Queue query count average is less for A*CD algorithm when the criteria is duration. On the other hand, queue query count average is more for A*CD algorithm when the criteria is financial cost. But average A*CD queue query count value is affected dramatically by the pairs for which no route exist. For such pairs, A*CD queried the queue more than 2500 times with our example data set.

6.4.2 Limitations

Our data set contains information from transportation provider's systems. We convert this data to our graph model's format. So, structure-wise we do not have a problem with this data set. On the other hand, content-wise this data set might differ from a data set that would have been entered by the users collaboratively. As mentioned in Chapter 1, data quality cannot be guaranteed in collaborative systems; there might be missing or duplicate information. Since this is not the case for our data set, we could not demonstrate the intuitive connections that we have presented.

Chapter 7

Conclusion and Future Work

In this thesis, we have presented the design of a collaborative route providing system which can provide useful information to the users even if a route does not exist. The data of the system is entered by the users of the system. For storing the data, we have presented a data model which is containing locations, links between locations and relationships (containment, neighborhood, intersection) between locations. The data model is an extended unidirectional graph.

For the route finding purpose, we have presented a customized version of the A* search algorithm. This customized version has been named as A*CD (A* for Collaborative Data). The most significant customization is the calculation of the h-function in the A* search algorithm. h(*n*) estimates the cost of the route from node *n* to the goal node. We have presented a three-step calculation for this function.

- Calculate the geographical distance
- Normalize this distance
- Apply heuristics to increase or decrease the estimated cost

We have presented six heuristics, which are guiding the search process. In order to test the effects of the heuristics, we have executed example queries. For each query, we have executed the algorithm first without the tested heuristic(s) and then with the tested heuristic(s). There are two important results of these tests.

- A search with a certain heuristic can be seven times faster than the same search without the heuristic. We have used the queue query count of the execution as a measure.
- A heuristic can increase the usefulness of the results. For example, assume that a route between two locations in the same city is requested. It is possible that there are certain routes that are passing from locations in a distant city. A heuristic can insure that such routes are not provided to the user.

We have also presented the intuitive connections concept. Even if a route does not exist between the selected locations or only a route that is not sensible exists, the system can provide a route with missing links. The gap(s) between the disconnected locations are filled by the help of the relationships between locations. This has been accomplished by introducing a new type of link. These links are called virtual links. They are managed by the system and connect the related locations.

A*CD can also provide alternative routes, exclude certain link types in the searches according to user preferences and handle the problems associated with multiple stop transportation lines. We have used two cost models; duration and financial cost. For each route search, the user

can specify if he is interested in a fast route that will take minimum amount of time to travel or a cheap route in which he will pay less.

In order to evaluate A*CD algorithm's results' (routes') quality, we have compared their cost with the costs of the actual lowest cost routes. For 59% of the queries, A*CD algorithm returned the lowest cost route. On average, the ratio of the cost of the route that has been provided by the A*CD algorithm over the cost of the lowest cost route is 1.15.

Future work can be directed to improve the time and space complexities of the A*CD algorithm. In this thesis, we have not presented a significant improvement over the formal time and space complexities of the A* search algorithm. Another future work can be directed to develop feedback, trust and rollback mechanisms in order to improve the quality of the data that has been entered by the users.

The data dependency of our approach might be improved to a more flexible data model in a future work. Currently, we rely on the costs of the links and also the geographical coordinates of the locations. The f-function of the A*CD algorithm uses these data. A revised approach might classify this information as optional instead of mandatory.

As a final word, the A* search algorithm is a highly customizable search algorithm, which can be applied in collaborative contexts. Together with new heuristics and virtual links, a customized version of it can be developed to provide routes to the user in a graph with missing data in feasible running times.

Bibliography

[1] Aggarwal, A., Schieber, B. and Tokuyama T. Finding a minimum weight K-link path in graphs with Monge property and applications. *Proc. 9th Symp. Computational Geometry.* 189 – 197, 1993.

[2] Ahuja, R. K., Mehlhorn, K., Orlin, J. B. and Tarjan, R. E. Faster algorithms for the shortest path problem. *J. Assoc. Computing Machinery*. 37: 213 – 223, 1990.

[3] Ajwani, D., Dementiev, R, Meyer, U; and Osipov, V. Breadth first search on massive graphs. *9th DIMACS Implementation Challenge – Shortest Paths*, 2006.

[4] Arsanjani, A. Rule Pattern Language 2001: A Pattern Language for Adaptive Manners and Scalable Business Rule Design and Construction. *39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39),* 0370, 2001.

[5] Azevedo, J. A., Costa, M. E., Madeira, J.J. and Martins, E. Q. An algorithm for the ranking of shortest paths. *European Journal of Operational Research*, 69: 97 – 196, 1993.

[6] Bellman R.E. On a routing problem. *Quarterly Applied Mathematics*, 16: 87 – 90, 1958.

[7] Bulitko, V., Sturtevant, N., Lu, J. and Yau, T. Graph abstraction in real-time heuristic search. *Journal of Artificial Intelligence Research*, 30: 51-100, 2007.

[8] Chakrabarti, P., Ghosh, S., Achaya, A. and DeSarkar, S. Heuristic search in restricted memory. *Artificial Intelligence,* 47: 197-221, 1989.

[9] Chen, Y. L. An Algorithm for finding the k quickest paths in a network. *Computers and Operations Research*, 20: 59 – 65, 1993.

[10] Cormen, T., Leiserson, C., Rivest, R. and Stein C. Introduction to Algorithms, Second Edition. *MIT Press*, 2001.

[11] Dijkstra, E. W. A note on two problems in connection with graphs. *Numer. Math.*, 1:269--271, 1959.

[12] Eppstein, D. Finding the k shortest paths. *Proc. 35 th Symp. Foundations of Computer Science, IEEE*, 154-165, 1994.

[13] Ghosh, S. and Mahanti, A. Bidirectional Heuristic Search with Limited Resources, *Inform. Process. Letters* 40, 335-340, 1991.

[14] Goldberg, A.V. and Harrelson C. Computing the shortest path: A* search meets graph theory. *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, 156-165, 2005.

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

[16] Ikeda, T and Imai, H. Enhanced A* Algorithms for multiple alignments: Optimal alignments for several sequences and k-opt approximate alignments for large cases. *Theoretical Computer Science* 210: 341-374, 1999.

[17] Katoh, N., Ibaraki, T. and Mine, H. An efficient algorithm for K shortest simple paths. *Networks* 12(4): 411-427, 1982.

[18] Korf, R. Depth-first iterative deepening: An optimal admissable tree search. *Artifical Intelligence* 27: 97-109, 1985.

[19] Korf, R. Linear-space best-first search. *Artificial Intelligence* 62: 41-78, 1993.

[20] Korf, R. and Zhang, W. Divide and conquer frontier search applied to optimal seqeuence alignment. *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 910-916, 2000.

[21] Kumar, N., Ghosh, R. K. Parallel algorithm for finding first K shortest paths. *Computer Science and Informatics* 24 (3): 21-28 September 1994.

[22] Mero, L. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence* 23: 13-27, 1984.

[23] Minieka, E. The K-th shortest path problem. *ORSA/TIMS Joint National Mtg.* Vol 23, p. B/116, 1975.

[24] Pu, J., Manning, E., Shoja, G.C. and Srinivasan, A. A new algorithm to compute alternate paths in reliable OSPF (ROSPF). *Proceedings of PDPTA (the 2001 International Conference on Parallel and Distributed Processing Techniques and Applications)*, Las Vegas, June 25-28, 299-304, 2001.

[25] Reinefeld, A, and Marsland, T. Enhanced iterative deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16: 701-710, 1994.

[26] Russell, S. Efficient memory-bounded search methods. *In Proceedings of he Tenth European Conference on Artificial Intelligence* (ECAI-92), 1-5, 1992.

[27] Russell S., Norvig P. Artificial Intelligence. A Modern Approach. Second Edition. *Pearson Education, Inc., Upper Saddle River, New Jersey*, 2003.

[28] Santos, J. L. K shortest path algorithms. *9th DIMACS Implementation Challenge – Shortest Paths*, 2006.

[29] Taylor, L. A., and Korf R. Pruning Duplicate Nodes in Depth-First Search. *National Conference on Artificial Intelligence*, 1993.

[30] Winter, S. Route specifications with a linear dual graph. *Advances in Spatial Data Handling: Proc. 10th Int. Symp. Spatial Data Handling (SDH 2002)*, 2002.

[31] Zhou, R. and Hansen E.A. Memory-Bounded A* Graph Search. *15th International FLAIR Conference*, Pensacola, Florida, 2002.

Appendix A

Glossary

A* Search Algorithm: This algorithm is one of the most well known and widely used informed (heuristic) search algorithms. It uses an f-value for each visited node, which is indicating the cost of the best route from the start node to the target node, passing from the visited node.

A*CD – A* for Collaborative Data: The route search algorithm of the system is used to answer user queries for finding routes between locations. This algorithm is derived from A* search algorithm. It uses several heuristics and geographic information for estimating remaining cost. We have named the algorithm as A*CD (A* for Collaborative Data) since it is running on top of a data model that is designed for storing information from the users.

Accepted Target Set: Accepted target set contains nodes that are related with the target node such that if a route is found to an accepted node, that route can be provided as a solution to the user. The routes from the source node to nodes in this set are provided to the user if no route to the target node can be found.

Child Node: If two locations are related with a containment relationship, the contained node is called a child node.

Basic Route: A basic route is a sequence of user entered links. Each link in the basic route has the target location of the previous link as its source location. Assuming L_1, L_2 and L_3 are locations, the definition of a basic route is as follows;

- L_1 –> L_1 is a basic route
- L_1 –> L_2 is a basic route if there is a user entered link from L_1 to L_2
- L_1 –> L_3 is a basic route if there exists a location L_2 such that
    - L_1 –> L_2 is a basic route
    - L_2 –> L_3 is a basic route

Cost Model: While querying the system, users can select their point of interest for the cost function. There are two options available, financial cost and duration. If the user selects financial cost, the algorithm tries to optimize the result according to financial costs of links. If the user selects duration, the algorithm tries to optimize the result according to durations of links. The durations and financial costs of the links are entered by the users.

Distance between Locations: Distances between locations are defined in kilometers and are calculated according to geographic coordinates. This information forms a base for the estimated cost calculations. It is not the only information used for estimating costs.

Edge: An edge corresponds to a link in our data model.

Favorability Threshold (*ft*): Favorability threshold is the minimum value that a node can have as its favorability value, to be added to the priority queue. If a node's favorability value is under this threshold, the node is discarded.

Geographic Coordinates: Geographic coordinates define a location's geographic coordinates on the world. There are two values for coordinates; latitude and longitude. We are not using altitude.

Intersecting Location: If two locations are related with an intersection relationship, they intersect with each other.

Intuitive Connection: An intuitive connection is a method suggested by the system for reaching to a location from another location. The system can use location relationships to provide intuitive connections even if there is no basic route between the connected locations.

Iterative Deepening Search (depth-first): Iterative deepening search is a search strategy in which a series of independent depth-limited, depth-first searches are run. For each iteration, depth limit is increased. Iterative deepening search combines the completeness of breadth-first search with the depth-first search's space efficiency.

Link: A link is a connection between two locations. Links are unidirectional. They define a one-way connection from one location to another. Links can be entered by users.

Link Type: Link types define the type of the links. Types can be entered by users. Each link has a link type. Boats, walking, planes, trains, intercity buses are some examples. New link types can be entered by the users.

Location: A location defines a physical location in the world. Locations can be of any type; countries, cities, districts, buildings, offices, theatres or even rooms in buildings. Locations can be entered by users.

Location Relationship: A location relationship is used for relating two locations at a time. There are three kinds of relationships; containment, neighborhood and intersection. Relationships can be entered by users.

Missing Links Problem: Missing links problem is the problem of connecting two locations whenever there is no basic route between them or the basic route between them is not sensible. For example, the only basic route between two locations L_1 and L_2 might pass over another location L_3, which is thousands of kilometers away from L_1 although L_2 is only two kilometers away from L_1. Such a basic route is not sensible.

Neighbor Node: If two locations are related with a neighborhood relationship, they are neighbors of each other.

Node: A node corresponds to a location in our data model.

Parent Node: If two locations are related with a containment relationship, the container node is called a parent node.

Route: A route is a sequence of user entered links and intuitive connections. Each link / intuitive connection in the route has the target location of the previous link / intuitive connection as its source location. Assuming L_1, L_2 and L_3 are locations, the definition of a route is as follows;

- L_1 –> L_1 is a route
- L_1 –> L_2 is a route if there is a user entered link from L_1 to L_2
- L_1 –> L_2 is a route if there is an intuitive connection from L_1 to L_2
- L_1 –> L_3 is a route if there exists a location L_2 such that
    - L_1 –> L_2 is a route
    - L_2 –> L_3 is a route

Source Node: Source node is the node from which the requested route will be started. It indicates the starting point of the travel.

Target Node: Target node is the node at which the requested route will end. It indicates the ending point of the travel.

Unidirectional Graph: A unidirectional graph G is a pair (V, E). V is a set of nodes (vertices). E is a set of links (edges) between the vertices.

$$E \subseteq \{(u,v) \mid u, v \in V\}$$

User: A user is a registered user of the system. A user has the ability to enter data to the system and enter queries.

Virtual Link: A virtual link is a special type of link between two locations. Virtual links are entered by the system itself according to the location relationships and links that are entered by users. These links are used for providing intuitive connections to the user.

There are two types of virtual links. Type-1 virtual links are used whenever there exists an actual link from location *A* to location *B* and there exist another location *C*, which has a location relationship with *A* such that *C* contains *A*. In this case, a virtual link from *C* to *B* is added.

Type-2 virtual links are for connecting child locations to their parent. Whenever there exists a location relationship between two locations *A* and *B* such that *A* contains *B*, a virtual link from B to A is added.

Appendix B

Example Data Set

B.1 Bus Network in Ankara (EGO – Maintained by municipality)

Addition of this bus network is accomplished using “multiple stop line” concept. There are a total of 53 bus lines. Each line contains 15 to 100 stops. Below is a piece of the raw data that has been used. It contains 6 bus lines.

>>124-1
(1) OPERA HAREKET NOKTASI(2) İSTANBUL CAD 1 DURAK(3) HİPODRUM CAD 1. DURAK(4) HİPODRUM CAD 2. DURAK(5) HİPODRUM CAD 3. DURAK(6) KONYA YOLU 1. DURAK(7) KONYA YOLU 2. DURAK(8) KONYA YOLU 3. DURAK(9) KONYA YOLU 4. DURAK(10) ESKİŞEHİR YOLU 1. DURAK(11) ESKİŞEHİR YOLU 2. DURAK(12) ESKİŞEHİR YOLU 3. DURAK(13) ESKİŞEHİR YOLU 4. DURAK(14) ESKİŞEHİR YOLU 5. DURAK(15) ESKİŞEHİR YOLU 6. DURAK(16) ESKİŞEHİR YOLU 7. DURAK(17) TURKCELL DURAĞI(18) TESTAŞ GENEL MÜD. DURAĞI(19) ARŞİV GENEL MÜDÜRLÜĞÜ(20) ATOM ENERJİSİ KURUMU(21) TARIM KÖY İŞLERİ MİSAFİRHANESİ(22) BEYTEPE KOPRÜSÜ DURAĞI(23) ÜMİTKÖY KAVŞAĞI DURAĞI(24) ESKİŞEHİR YOLU 15. DURAK(25) ESKİŞEHİR YOLU 16. DURAK(26) ESKİŞEHİR YOLU 17. DURAK(27) ESKİŞEHİR YOLU 18. DURAK(28) ESKİŞEHİR YOLU 19. DURAK(29) ESKİŞEHİR YOLU(30) ESKİŞEHİR YOLU 21. DURAK(31) ESKİŞEHİR YOLU 22. DURAK(32) ESKİŞEHİR YOLU 23. DURAK(33) ESKİŞEHİR YOLU 24. DURAK(34) ESKİŞEHİR YOLU 25. DURAK(35) ESKİŞEHİR YOLU 26. DURAK(36) ESKİŞEHİR YOLU 27. DURAK(37) ESKİŞEHİR YOLU 28. DURAK(38) TEMELLİ 1. DURAK(39) TEMELLİ 2. DURAK(40) TEMELLİ 3. DURAK(41) TEMELLİ 4. DURAK(42) TEMELLİ 5. DURAK(43) TEMELLİ 6. DURAK(44) TEMELLİ 1. DURAK(45) ESKİŞEHİR YOLU 28. DURAK(46) ESKİŞEHİR YOLU 27. DURAK(47) ESKİŞEHİR YOLU 26. DURAK(48) ESKİŞEHİR YOLU 25. DURAK(49) ESKİŞEHİR YOLU 24. DURAK(50) ESKİŞEHİR YOLU 1. DURAK(51) ESKİŞEHİR YOLU 2. DURAK(52) ESKİŞEHİR YOLU 3. DURAK(53) ESKİŞEHİR YOLU 4. DURAK(54) ESKİŞEHİR YOLU 5. DURAK(55) ESKİŞEHİR YOLU 6. DURAK(56) MEKSİKA CAD 1. DURAK(57) MEKSİKA CAD 2. DURAK(58) MEKSİKA CAD 3. DURAK(59) MEKSİKA CAD 4. DURAK(60) MEKSİKA CAD 5. DURAK(61) ESKİŞEHİR YOLU 1. DURAK(62) ESKİŞEHİR YOLU ÜMİTKÖY KAVŞAĞI(63) HACETTEPE KAVŞAĞI(64) ESKİŞEHİR YOLU(65) TARIM BAKANLIĞI(66) ARSİV GENEL MÜDÜRĞÜLÜ(67) DİYANET İŞLERİ BAŞKANLIĞI(68) ATATÜRK HASTANESİ(69) ELEKTİRİK ETÜT MERKEZİ(70) ODTÜ(71) ESKİŞEHİR YOLU BALGAT YURDU DURAĞI(72) MTA DURAGI(73) ESKİŞEHİR YOLU ULUSOY DURAĞI(74) KONYA YOLU 11. DURAK(75) KONYA YOLU 3. DURAK(76) KONYA YOLU 12. DURAK(77) KONYA YOLU 13. DURAK(78) KONYA YOLU ANKARA ÜNV.DİŞÇİLİK FAK.DURAĞI(79) KONYA YOLU ÜST GEÇİT ALTI DURAĞI(80) HİPODRUM CAD 3. DURAK(81) HİPODRUM CAD 2. DURAK(82) HİPODRUM CAD 1. DURAK(83) OPERA HAREKET NOKTASI SONDURAK

>>160-1
(1) OPERA HAREKET NOKTASI(2) ULUS(3) OPERA(4) SIHHIYE(5) KIZILAY(6) ZİYA GÖKALP CAD. 1. DURAK(7) LİBYA CAD. 1. DURAK(8) LİBYA CAD. 2. DURAK(9) BÜLBÜL DERESİ CAD. 1. DURAK(10) BÜLBÜL DERESİ CAD. 2. DURAK(11) BÜLBÜL DERESİ CAD. 3. DURAK(12) BÜLBÜL DERESİ CAD. 4. DURAK(13) BÜLBÜL DERESİ CAD. 5. DURAK(14) BÜLBÜL DERESİ CAD. 6. DURAK(15) BAĞLAR CAD. 1. DURAK(16) BAĞLAR CAD. 2. DURAK(17) BAĞLAR CAD. 3. DURAK(18) YAVUZ EVLER SOK. 1. DURAK(19) YAVUZ EVLER SOK. 2. DURAK(20) YAVUZ EVLER SOK. 3. DURAK(21) SERHAT SOK. 1. DURAK(22) BAĞLAR CAD. LİSE DURAĞI(23) ZAFER TEPE 1. CAD. 1.

DURAK(24) ZAFER TEPE 1. CAD. 2. DURAK(25) ZAFER TEPE 1. CAD. 3. DURAK(26) ZAFER TEPE 1. CAD. 4. DURAK(27) ZAFER TEPE 1. CAD. 5. DURAK(28) BAĞLAR CAD.(29) BAĞLAR CAD.(30) BAĞCILAR CAD.(31) LİBYA CAD(32) LİBYA CAD.(33) ADNAN SAYGUN CAD. 1. DURAK(34) ADNAN SAYGUN CAD. 2. DURAK(35) OPERA SON DURAK

>>170-1
(1) OPERA HAREKET NOKTASI(2) ÇANKIRI CAD.OKUL DURAĞI(3) ÇANKIRI CAD.SSK DURAĞI(4) ETLİK CAD.KAYMAKAMLIK DURAĞI(5) TURGUT ÖZAL BULVARI BENZ.BULV.(6) TURGUT ÖZAL BULV.İŞ BANKASI DURAĞI(7) KONYA YOLU EMNİYET MÜD. DURAĞI(8) KONYA YOLU ETİLER SİTESİ DURAĞI(9) GAZİ ÜNİV.ECZACILIK FAKÜ.DURAĞI(10) SİLAHTAR CAD. OKUL DURAĞI(11) M E B DERS ALETLERİ YAPIM MERKEZİ DURAĞI(12) SİLAHTAR CAD. PARK DURAĞI(13) SİLAHTAR CAD.FİZİK TEDAVİ DURAĞI(14) SİLAHTAR CAD.OĞUZLAR SOK.DURAĞI(15) SİLAHTAR CAD.SÜT FABRİKASI DURAĞI(16) SİLAHTAR CAD.PTT DURAĞI(17) GÜVERCİNLİK CAD. TEKEL FABRİKASI DURAĞI(18) GÜVERCİNLİK CAD.FİŞEK FABRİKASI DURAĞI(19) GÜVERCİNLİK SON DURAK(20) GÜVERCİNLİK CAD.FİŞEK FABRİKASI DURAĞI(21) GÜVERCİNLİK CAD.TEKEL FABRİKA DURAĞI(22) SİLAHTAR CAD.PTT DURAĞI(23) SİLAHTAR CAD.SÜT FABRİKASI DURAĞI(24) SİLAHTAR CAD.OĞUZLAR SOK.DURAĞI(25) SİLAHTAR CAD.FİZİK TEDAVİ(26) SİLAHTAR CAD.PARK DURAĞI(27) M E B DERS ALETLERİ YAPIM MERKEZİ DURAĞI(28) SİLAHTAR CAD OKUL DURAĞI(29) BANDIRMA SOK.DHMİ DURAĞI(30) GAZİ ÜNİVERSİTESİ DURAGI(31) KONYA YOLU ANKARA ÜNV.DİŞÇİLİK FAK.DURAĞI(32) KONYA YOLU ÜST GEÇİT ALTI DURAĞI(33) KONYA YOLU TREN HATTI ÜST DURAĞI(34) KONYA YOLU HİPODRUM DURAĞI(35) TURGUT ÖZAL BULVARI SHELL BENZİNLİĞİ DURAĞI(36) TURGUT ÖZAL BUL.SANAYİ DURAĞI(37) ETLİK CAD.TCK DURAĞI(38) ETLİK CAD 1 . DURAK(39) ÇANKIRI CAD. SSK DURAĞI(40) ÇANKIRI CAD.OKUL DURAĞI(41) OPERA HAREKET NOKTASI

>>171-1
(1) OPERA HAREKET NOKTASI(2) OPERA DURAĞI(3) SIHHIYE DURAĞI(4) KIZILAY DURAĞI(5) BAKANLIK DURAĞI(6) KARAYOLLARI DURAĞI(7) MİLLİ KÜTÜPHANE DURAĞI(8) TURİZM BAKANLIĞI DURAĞI(9) HAZİNE MÜŞTAŞARLIĞI DURAĞI(10) ARMADA DURAĞI(11) MTA DURAĞI(12) BALGAT ÖĞRENCİ YURDU(13) ODTÜ DURAĞI(14) CEPA DURAĞI(15) TEDAŞ ETÜT İDARE DURAĞI(16) 6.CAD DURAĞI(17) 6.CAD DURAĞI(18) 6. CAD DURAĞI(19) 7 CAD. DURAĞI(20) MUSTAFA KEMAL DURAĞI(21) 67. SOK DURAĞI(22) 6. CAD OKUL DURAĞI(23) 6.CAD. MUSTAFA KEMAL DURAĞI(24) 49.SOKMUSTAFA KEMAL DURAĞI(25) 44. SOK.MUSTAFA KEMAL 3. DURAK(26) MUSTAFA KEMAL OKUL DURAĞI(27) 21.SOK MUSTAFA KEMAL DURAĞI(28) 3.CAD DURAĞI(29) 3. CAD DURAĞI(30) ESKİŞEHİR YOLU BALGAT YURDU DURAĞI(31) ESKİŞEHİR YOLU MTA DURAĞI(32) ESKİŞEHİR YOLU ULUSOY DURAĞI(33) ESKİŞEHİR YOLU DIŞİŞLER BAK. DURAĞI(34) İSMET İNÖNÜ BULV. CAMİ DURAĞI(35) TARIM KOOP MERĞKEZ BİRL.DURAĞI(36) KARA KUVETLER DURAĞI(37) MEŞRUTİYET CAD.DURAĞI(38) MİTHATPAŞA CAD. DURAĞI(39) SIHHIYE DURAĞI(40) OPERA DURAĞI(41) OPERA HAREKET NOKTASI

>>174-1
(1) OPERA HAREKET NOKTASI(2) OPERA DURAĞI(3) SIHHIYE DURAĞI(4) KIZILAY DURAĞI(5) BAKANLIK DURAĞI(6) KARAYOLLARI DURAĞI(7) MİLLİ KÜTÜPHANE DURAĞI(8) TURİZM BAKANLIĞI DURAĞI(9) HAZİNE VE DIŞ TİCARET MÜŞT.DURAĞI(10) ARMADA DURAĞI(11) MTA DURAĞI(12) BALGAT YURDU DURAĞI(13) ODTÜ DURAĞI(14) CESA DURAĞI(15) ELEKTRİK İŞLERİ ETÜT DURAĞI(16) ATATÜRK HASTANESİ DURAĞI(17) TURKCELL DURAĞI(18) TESTAŞ GENEL MÜD. DURAĞI(19) ARŞİV GENEL MÜDÜRLÜĞÜ(20) ATOM ENERJİSİ KURUMU(21) TARIM KÖY İŞLERİ MİSAFİRHANESİ(22) HACETTEPE BEY. KAMP.YOLU(23) HACETTEPE BEY. KAMP.YOLU(24) ANGORA EVLERİ 1. DURAK(25) DİCLE CAD. 1. DURAK(26) DİCLE CAD. 2. DURAK(27) DİCLE CAD. 3. DURAK(28) 13. CAD. 1. DURAK(29) 12. CAD. 1. DURAK(30) 13. CAD. 2. DURAK(31) 13. CAD. 3. DURAK(32) ANKARA.CAD. .. 1. DURAK(33) ANGORA CAD. 2. DURAK(34) ANGORA CAD. 3. DURAK(35) ANGORA CAD. 4. DURAK(36) SALTOĞLU BULV. 1. DURAK(37) SALTOĞLU BULV. 2. DURAK(38) SALTOĞLU BULV. 3. DURAK(39) SALTOĞLU BULV. 4. DURAK(40) HİTİT BULV. 1. DURAK(41) HİTİT BULV. 2. DURAK(42) HİTİT BULV. 2. DURAK(43) SALTOĞLU BULV. 5 DURAK(44) SALTOĞLU BULV. 6 DURAK(45)

KEDİ SEVEN CAD. 1. DURAK(46) KEDİ SEVEN CAD. 2. DURAK(47) KEDİ SEVEN CAD. 3. DURAK(48) ÖYKÜ CAD. 1. DURAK(49) SALTOĞLU BLV. 1.CAD(50) SALTOĞLU BLV. DURAK(51) SALTOĞLU BLV. DURAK(52) SALTOĞLU BLV. DURAK(53) SALTOĞLU BLV. DURAK(54) ANGORA CAD. 4. DURAK(55) ANGORA CAD. 3. DURAK(56) ANGORA CAD. 2. DURAK(57) ANKARA.CAD. .. 1. DURAK(58) 13. CAD. 3. DURAK(59) 13. CAD. 2. DURAK(60) 12. CAD. 1. DURAK(61) 13. CAD. 1. DURAK(62) DİCLE CAD. 3. DURAK(63) DİCLE CAD. 2. DURAK(64) DİCLE CAD. 1. DURAK(65) ANGORA EVLERİ 1. DURAK(66) HACETTEPE BEY. KAMP.YOLU(67) HACETTEPE KAVŞAĞI(68) ESKİŞEHİR YOLU(69) TARIM BAKANLIĞI(70) ARSİV GENEL MÜDÜRLÜĞÜ(71) DİYANET İŞLERİ BAŞKANLIĞI(72) ATATÜRK HASTANESİ(73) ELEKTİRİK ETÜT MERKEZİ(74) ODTÜ(75) ESKİŞEHİR YOLU BALGAT YURDU DURAĞI(76) MTA DURAGI(77) ESKİŞEHİR YOLU ULUSOY DURAĞI(78) ESKİŞEHİR YOLU DIŞİŞLER BAK. DURAĞI(79) İSMET İNÖNÜ BULV. CAMİ DURAĞI(80) TARIM KOOP MERĞKEZ BİRL.DURAĞI(81) KARA KUVETLER DURAĞI(82) MEŞRUTİYET CAD.DURAĞI(83) MİTHATPAŞA CAD. DURAĞI(84) SIHHIYE DURAĞI(85) OPERA DURAĞI(86) OPERA HAREKET NOKTASI SONDURAK

>>177-1
(1) OPERA HAREKET NOKTASI(2) SIHHIYE DURAĞI(3) NECATİBEY CAD. 1. DURAK(4) NECATİBEY CAD. 2. DURAK(5) NECATİBEY CAD. 1. DURAK(6) DEVLET SU İŞLERİ GENEL MÜDÜRLÜĞÜ(7) MİLLİ KÜTÜPHANE DURAĞI(8) TURİZM BAKANLIĞI DURAĞI(9) HAZİNE VE DIŞ TİCARET MÜŞT.DURAĞI(10) KONYA YOLU 10. DURAK(11) KONYA YOLU 9. DURAK(12) KONYA YOLU 8. DURAK(13) KONYA YOLU 7. DURAK(14) KONYA YOLU 6. DURAK(15) KONYA YOLU 5. DURAK(16) KONYA YOLU 4. DURAK(17) KONYA YOLU 3. DURAK(18) KONYA YOLU 2. DURAK(19) KONYA YOLU 1. DURAK(20) KONYA YOLU 4. DURAK(21) KONYA YOLU 2. DURAK(22) KONYA YOLU 1. DURAK(23) 218. SOK 1. DURAK(24) 218. SOK 2. DURAK(25) 218. SOK 3. DURAK(26) İLKOKUL SOK 1. DURAK(27) ANKARA HAYMANA YOLU 1. DURAK(28) ANKARA HAYMANA YOLU 2. DURAK(29) ANKARA HAYMANA YOLU 3. DURAK(30) HAYMANA YOLU DURAK(31) HAYMANA YOLU DURAK(32) HACILAR KÖYÜ DURAK(33) HACILAR KÖYÜ DURAK(34) HACILAR KÖYÜ DURAK(35) MERKEZ KENT SİTESİ(36) MERKEZ KENT SİTESİ(37) MARTI KÖY SİTESİ(38) MERKEZ KENT SİTESİ(39) MERKEZ KENT SİTESİ(40) HAYMANA YOLU DURAK(41) HACI HASAN KÖYÜ DURAK(42) HACI HASAN KÖYÜ DURAK(43) HACI HASAN KÖYÜ DURAK(44) HAYMANA YOLU DURAK(45) HAYMANA YOLU DURAK(46) ÇEVİK BİR CAD DURAK(47) ÇEVİK BİR CAD DURAK(48) ÇEVİK BİR CAD DURAK(49) ÇEVİK BİR CAD DURAK(50) KOPARAN KÖYÜ DURAĞI(51) KOPARAN KÖYÜ DURAĞI(52) KOPARAN KÖYÜ DURAĞI(53) KOPARAN KÖYÜ DURAĞI(54) ÇEVİK BİR CAD DURAK(55) ÇEVİK BİR CAD DURAK(56) ÇEVİK BİR CAD DURAK(57) ÇEVİK BİR CAD DURAK(58) HAYMANA YOLU DURAK(59) HAYMANA YOLU DURAK(60) HACILAR KÖYÜ DURAK(61) HACILAR KÖYÜ DURAK(62) MERKEZ KENT SİTESİ(63) MERKEZ KENT SİTESİ(64) MARTI KÖY SİTESİ(65) MERKEZ KENT SİTESİ(66) MERKEZ KENT SİTESİ(67) HAYMANA YOLU DURAK(68) HAYMANA YOLU DURAK(69) ANKARA HAYMANA YOLU 3. DURAK(70) ANKARA HAYMANA YOLU 2. DURAK(71) ANKARA HAYMANA YOLU 1. DURAK(72) İLKOKUL SOK 1. DURAK(73) 218. SOK 3. DURAK(74) ANKARA CAD 1. DURAK(75) ANKARA CAD 2. DURAK(76) ANKARA CAD 3. DURAK(77) KONYA YOLU 1. DURAK(78) KONYA YOLU 2. DURAK(79) KONYA YOLU 3. DURAK(80) KONYA YOLU 4. DURAK(81) KONYA YOLU 1. DURAK(82) KONYA YOLU 2. DURAK(83) KONYA YOLU 3. DURAK(84) KONYA YOLU 4. DURAK(85) KONYA YOLU 5. DURAK(86) KONYA YOLU 6. DURAK(87) KONYA YOLU 7. DURAK(88) KONYA YOLU 8. DURAK(89) KONYA YOLU 9. DURAK(90) KONYA YOLU 10. DURAK(91) HAZİNE DURAĞI(92) İSMET İNÖNÜ BULV. CAMİ DURAĞI(93) TARIM KOOP MERĞKEZ BİRL.DURAĞI(94) KARA KUVETLER DURAĞI(95) MEŞRUTİYET CAD.DURAĞI(96) MİTHAT PAŞA CAD(97) MİTHATPAŞA CAD. DURAĞI(98) SIHHIYE DURAĞI(99) OPERA DURAĞI(100) OPERA HAREKET NOKTASI SONDURAK

The lines starting with >> characters indicate the line number. Below these title lines, there exists stop names. Each stop name is started by (X) where X denotes the stop number.

The algorithm first adds the stops as locations to the system. It relates these locations with Ankara such that Ankara will contain the stops. If there exists a location with the same name as the stop name, and the location is contained in Ankara, location insertion is not done. This means that the stop has been previously added.

After the insertion of the stops, for each consecutive link pair in the line, two links are added. One link originates from one location and the other one originates from the other location of the pair. So, number of links that will be added for a bus line will be

$2 \times (n - 1)$

where n is the number of stops in the line. All these links are added with the same multiple stop line ID meaning they are links of the same multiple stop line.

This data is gathered from the web site of EGO (http://web.ego.gov.tr/map/mapView.asp)

Statistics for this data extraction
Number of bus lines: 53
Number of locations (stops) added: 271
Number of links added: Around 1000
Randomized duration of links: Between 20 and 40 (minutes)
Financial cost of links: 3 YTL

B.2 Bus Network in Istanbul (IETT – Maintained by municipality)

There are a total of 347 lines in this network. But different than the Ankara bus network, this information does not contain immediate stops. The raw information of this automatic data extraction method contains only start and end stops of the lines. So a total of n links, where n denotes the number of lines, is added to the database. Below is an example of the raw data. It contains 100 bus lines.

```
10-Kadıköy-Maltepe-Cevizli
10B-Kadıköy-AltBostancı
10E-Kadıköy-Esatpaşa
10M-Kadıköy-Ümraniye
10S-Kadıköy-AltBostancı
110-Kadıköy-Taksim
112-Bostancı-Taksim
11A-Altunizade-Alemdağ
11C-Üsküdar-Emniyet Mahallesi
11CT-Polonezköy Sapağı-Çavuşbaşı-Tepeüstü
11ÇB-Tepeüstü-Çavuşbaşı-Kavacık
11D-Altunizade-İnkılap Mahallesi
11E-Üsküdar-Esatpaşa
11G-Yenidoğan-Ümraniye
11H-Ortaçeşme-Ümraniye
11K-Altunizade-Kazım Karabekir Mahallesi
11M-Altunizade-Mustafa Kemal Mahallesi
11P-Altunizade-Sarıgazi Emek Mahallesi
11SM-Altunizade-Veysel Karani
11ST-Altunizade-Seyrantepe
11T-Üsküdar-Türkiş Blokları
11Ü-Üsküdar-Ünalan Mahallesi
11ÜS-Altunizade-Sultanbeyli
11Y-Üsküdar-Yavuztürk Mahallesi
12-Kadıköy-Üsküdar
120-Kadıköy-Mecidiyeköy
```

121A-Beykoz-Mecidiyeköy
121B-Kavacık-Mecidiyeköy
122B-Yenidoğan-Mecidiyeköy
122C-Ümraniye-Mecidiyeköy
122L-Ümraniye-Metro-1. Levent
122M-Ş. Şahinbey-Mecidiyeköy
122S-Sultanbeyli-4. Levent Metro
123U-Uğur Mumcu-Perpa
125-Kadıköy-Boğaziçi Üniversitesi
127-Kadıköy-Topkapı
128-Altbostancı-Mecidiyeköy
129-Altunizade-Mediyeköy
129K-Kozyatağı-Mecidiyeköy
129L-Kozyatağı-4. Levent Metro
129T-Kozyatağı-Taksim
12A-Kadıköy-Üsküdar
12C-Üsküdar-Polis Hastanesi
12H-Kadıköy-Harem
13-Kadıköy-Çakmak Mahallesi-Ataşehir
130-Kadıköy-Tuzla
130A-Kadıköy-Tuzla
131-Sultanbeyli-Ümraniye
131A-A. Yesevi Mahalesi-Ümraniye
131B-Çiftlik Mahallesi-Ümraniye
131C-Sultanbeyli-Ümraniye
131T-Taşdelen-İmes
131TD-Taşdelen-Ümraniye-Santral
131V-Veysel Karani-Yakacık-Kartal
131YD-Yenidoğan-Ümraniye
131YS-Yenidoğan-İmes
132-Kartal-Tepeören
132A-Kartal-Sultanbeyli
132C-Sultanbeyli-Yakacık-Kartal
132F-Kartal-Esenler
132K-Kartal-Sultanbeyli
132P-Kartal-Veysel Karani Mahallesi
132S-Kartal-Yenidoğan
132V-Kartal-Kaynarca-Velibaba Mahallesi
133-Kartal-Güzelyalı
133AP-Pendik-Akfırat
133D-Kartal-Tepeören
133F-Yeditepe Üniversitesi-Fındıklı Mahallesi
133G-Kartal-Gülensu
133K-Kartal-Kavakpınar
133P-Tepeören-Tuzla
133Ş-Kartal-Şifa Mahallesi
133T-Bostancı-Tuzla
134-Kartal-Aydos Hilal Konutları
134BK-Kartal-Altbostancı
134DK-Kartal-Esenkent
134GK-Kartal-Gümüşpınar Mahallesi
134K-Kartal-Kurfalı Mahallesi
134TK-Kartal-Topselvi
134UK-Kartal-Uğur Mumcu Mahallesi
134YK-Kartal-Yakacık
135-Kavacık-Poyrazköy
135A-Kavacık-Görele Mahallesi-Acarkent
135G-Tokatköy-Çavuşbaşı
135K-Ortaçeşme-Karlıktepe
136-Kavacık-Ali Bahadır
137-Beykoz-Riva-Cumhuriyet Köyü

```
138-Ümraniye-Ömerli-Hüseyinli Köyü
139-Harem-Şile
139A-Harem-Şile-Ağva
13A-Altunizade-Çakmak Mahallesi
13B-Kadıköy-Yenişehir
14-Kadıköy-Yenidoğan
142-Boğazköy Mahallesi-Yenibosna Metro
142A-Esenyurt-Yenibosna Metro
142F-Yeşilkent-Yenibosna Metro
143-Küçükçekmece-Zeytinburnu Metro
145-Marmara Evleri-Aksaray
145M-Beylikdüzü-Mecidiyeköy (Çift Katlı)
```

The first part of each line indicates the stop number. The second part indicates the start location of the line and the third part indicates the end location of the line.

This data is gathered from the web site of IETT (http://www.iett.gov.tr/)

Statistics for this data extraction
Number of bus lines: 347
Number of locations (stops) added: 287
Number of links added: 547
Randomized duration of links: Between 30 and 60 (minutes)
Financial cost of links: 2 YTL

B.3 Plane Network in Turkey (THY – A private corporation)

This information is gathered from the THY web site (http://www.thy.com.tr). Its raw data is simple;

```
Izmir
Bodrum
Dalaman
Denizli
Antalya
Konya
Bursa
Eskisehir
Samsun
Sivas
Kayseri
Trabzon
Kars
Agri
Erzincan
Erzurum
Elazig
Diyarbakir
Mardin
Batman
Kahramanmaras
Adiyaman
Malatya
Gaziantep
Sanliurfa
```

```
Adana
```

For each line in this raw data, a new location named X Airport is added to the data model. Afterwards, for each line in this data, 4 links are added to the database;

- A link from Esenboğa Airport (Ankara) to X Airport
- A link from X Airport to Esenboğa Airport (Ankara)
- A link from Atatürk Airport (Istanbul) to X Airport
- A link from X Airport to Atatürk Airport (Istanbul)

Statistics for this data extraction
Number of locations (airports) added: 26
Number of links added: 104
Randomized duration of links: Between 50 and 70 (minutes)
Financial cost of links: 100 YTL

B.4 Intercity Bus Network in Turkey (Ulusoy – A private corporation)

This information is gathered from the web site of Ulusoy (http://www.ulusoy.com.tr). The web site contains start and stop locations of each line. A piece from the raw data is given below as an example;

```
ADANA-ANKARA
ADANA-CORUM
ADANA-GIRESUN
ADANA-ISTANBUL
ADANA-MERZIFON
ADANA-ORDU
ADANA-RIZE
ADANA-SAMSUN
ADANA-TRABZON
ALANYA-ANKARA
ALANYA-ISTANBUL
ANKARA-ADANA
ANKARA-AKCAABAT
ANKARA-ALANYA
ANKARA-ANTALYA
ANKARA-ARAKLI
ANKARA-ARDESEN
ANKARA-ARHAVI
ANKARA-ARSIN
ANKARA-BELEK
ANKARA-BESIKDUZU
ANKARA-BODRUM
ANKARA-BULANCAK
ANKARA-C.BASI
ANKARA-CARSAMBA
ANKARA-CAYKARA
ANKARA-DENIZLI
ANKARA-EDIRNE
ANKARA-ESPIYE
ANKARA-EYNESIL
ANKARA-FATSA
```

```
ANKARA-FINDIKLI
ANKARA-G.YALI
ANKARA-GIRESUN
ANKARA-GORELE
ANKARA-HAVZA
ANKARA-HOPA
ANKARA-ISKENDERUN
ANKARA-ISTANBUL
ANKARA-IZMIR
ANKARA-KEMER
ANKARA-KESAP
ANKARA-MANAVGAT
ANKARA-MARMARIS
ANKARA-MERSIN
ANKARA-MERZIFON
ANKARA-MUGLA
ANKARA-OF
ANKARA-ORDU
ANKARA-PAZAR
ANKARA-PERSEMBE
ANKARA-PIRAZIZ
ANKARA-RIZE
ANKARA-SAMSUN
ANKARA-SERIK
ANKARA-SURMENE
ANKARA-TARSUS
```

Each line in the raw data indicates a bus line starting from an intercity bus station and ending at another intercity bus station.

The algorithm first adds two intercity bus stops for each line if they do not exist. If the line is X-Y, two locations named “X Otogar” and “Y Otogar” are added to the database. Afterwards, a link, started from X Otogar and ending at Y Otogar is added to the database.

Statistics for this data extraction
Number of locations (intercity bus stops) added: 97
Number of links added: 223
Randomized duration of links: Between 300 and 600 (minutes)
Financial cost of links: 50 YTL

B.5 Ferry Network in Istanbul (IDO - A private corporation)

This data set is used to geneerate the ferry network in Istanbul. Raw data for this set is small, it is as follows;

```
KARAKÖY - HAYDARPAŞA - KADIKÖY
KADIKÖY - EMİNÖNÜ
EMİNÖNÜ - ÜSKÜDAR
Üsküdar - Karaköy - Eminönü - Eyüp
KADIKÖY - BEŞİKTAŞ
BEŞİKTAŞ - ÜSKÜDAR
KADIKÖY - KABATAŞ
Eminönü - Kavaklar
ÇENGELKÖY - EMİNÖNÜ
ANADOLU KAVAĞI - RUMELİ KAVAĞI - SARIYER
```

```
KÜÇÜKSU - BEŞİKTAŞ
İSTİNYE-EMİRGAN-KANLICA-A.HİSARI-KANDİLLİ-BEBEK-ÇENGELKÖY
KÜÇÜKSU - TOKMAKBURNU
KABATAŞ - KADIKÖY - BOSTANCI - ADALAR
MALTEPE - ADALAR
SİRKECİ - HAREM
ESKİHİSAR - TOPÇULAR
ERDEK - MARMARA Adası - AVŞA Adası
SARAYBURNU - MARMARA - AVŞA
```

The algorithm processes each line individually. The stops are separated by ‘-‘ character. For each pair of stops, the script adds two links, connecting both stops to each other.

The algorithm adds missing locations as districts and relates these districts with Istanbul with a containment relationship.

Statistics for this data extraction
Number of locations (districts) added: 20
Number of links added: 60
Randomized duration of links: Between 20 and 40 (minutes)
Financial cost of links: 4 YTL

Appendix C

Example Query Results for Testing Heuristics

QUERY 1

Start Location: Bilkent
End Location: ODTÜ
Point of Interest: Duration

Execution 1: Without Heuristic 1 – All other heuristics are applied
Suggested route 1:
- Bilkent University to ODTÜ. **Type:** Dolmuş (Shared Taxi). **ID:** 272542
Route found in 156.25 ms. Queue query count: 14

Suggested route 2:
- Bilkent University to Ulus. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 39
- Ulus to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 125. **ID:** 336712
Route found in 390.625 ms. Queue query count: 36

Suggested route 3:
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 32
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **ID:** 308260
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 125. **ID:** 336712
Route found in 359.375 ms. Queue query count: 37

Suggested route 4:
- Bilkent University to Ulus. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 39
- Ulus to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 120. **ID:** 330956
Route found in 453.125 ms. Queue query count: 38

Suggested route 5:
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 32
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **ID:** 308260
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 120. **ID:** 330956
Route found in 421.875 ms. Queue query count: 39

Execution 2: With All Heuristics (Including Heuristic 1)
Suggested route 1:
- Bilkent University to ODTÜ. **Type:** Dolmuş (Shared Taxi). **ID:** 272542
Route found in 171.875 ms. Queue query count: 10

Suggested route 2:
- Bilkent University to Ulus. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes.
- Ulus to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 125. **ID:** 336712

Route found in 281.25 ms. Queue query count: 16

Suggested route 3:
- Bilkent University to Ulus. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 39
- Ulus to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 120. **ID:** 330956
Route found in 296.875 ms. Queue query count: 17

Suggested route 4:
- Bilkent University to Ulus. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 39
- Ulus to Diyanet İşleri Başkanlığı. **Type:** Public Transportation (Bus). **Explanation:** EGO - 120. **ID:** 330986
- Diyanet İşleri Başkanlığı to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 175. **ID:** 359202
Route found in 812.5 ms. Queue query count: 31

Suggested route 5:
- Bilkent University to Ulus. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 39
- Ulus to Diyanet İşleri Başkanlığı. **Type:** Public Transportation (Bus). **Explanation:** EGO - 120. **ID:** 330986
- Diyanet İşleri Başkanlığı to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 196-1. **ID:** 362351
Route found in 803.5 ms. Queue query count: 33

QUERY 2

Start Location: Çetin Emeç Bulv. (in Ankara)
End Location: Kadıköy (in Istanbul)
Point of Interest: Cost (Money)

Execution 1: All heuristics except 2 and 3 are applied
Suggested route:
- Çetin Emeç Bulv. to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 154-1. **ID:** 345349
- Ulus to Bilkent University. **Type:** Dolmuş (Shared Taxi). **ID:** 308258
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45
Route found in 4781.25 ms. Queue query count: 1399

Suggested route:
- Çetin Emeç Bulv. to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 154-1. **ID:** 345349
- Ulus to Bilkent University. **Type:** Dolmuş (Shared Taxi). **ID:** 308258
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Kadıköy. **Type:** Ferry. **Explanation:** Once in every 45 minutes. **ID:** 41
Route found in 4859.375 ms. Queue query count: 1401

Suggested route:
- Çetin Emeç Bulv. to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 154-1. **ID:** 345349
- Ulus to Bilkent University. **Type:** Dolmuş (Shared Taxi). **ID:** 308258
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Kabataş. **Type:** Public Transportation (Bus). **Explanation:** IETT - 22E. **ID:** 316596
- Kabataş to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315946
Route found in 4875 ms. Queue query count: 1408

Suggested route:
- Çetin Emeç Bulv. to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 161. **ID:** 346205
- ODTÜ to Bilkent University. **Type:** Dolmuş (Shared Taxi). **ID:** 272541
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45

Route found in 4750 ms. Queue query count: 1416

Suggested route:
- Çetin Emeç Bulv. to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 161. **ID:** 346205
- ODTÜ to Bilkent University. **Type:** Dolmuş (Shared Taxi). **ID:** 272541
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Kadıköy. **Type:** Ferry. **Explanation:** Once in every 45 minutes. **ID:** 41

Route found in 4812.5 ms. Queue query count: 1418

Execution 2: All heuristics 3 are applied

Suggested route:
- Çetin Emeç Bulv. to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 154-1. **ID:** 345349
- Ulus to Bilkent University. **Type:** Dolmuş (Shared Taxi). **ID:** 308258
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45

Route found in 2890.625 ms. Queue query count: 185

Suggested route:
- Çetin Emeç Bulv. to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 161. **ID:** 346205
- ODTÜ to Bilkent University. **Type:** Dolmuş (Shared Taxi). **ID:** 272541
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45

Route found in 2796.875 ms. Queue query count: 188

Suggested route:
- Çetin Emeç Bulv. to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 162. **ID:** 348742
- ODTÜ to Bilkent University. **Type:** Dolmuş (Shared Taxi). **ID:** 272541
- Bilkent University to AŞTİ. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 15 minutes. **ID:** 31
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45

Route found in 2875 ms. Queue query count: 191

Suggested route:
- Çetin Emeç Bulv. to Hoşdere Cad. **Type:** Public Transportation (Bus). **Explanation:** EGO - 107. **ID:** 327690
- Hoşdere Cad to AŞTİ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 167-1. **ID:** 353895
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45

Route found in 4671.875 ms. Queue query count: 1338

Suggested route:
- Çetin Emeç Bulv. to Hoşdere Cad.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 107. **ID:** 327720
- Hoşdere Cad. to AŞTİ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 167-1. **ID:** 353897
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33

- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45
Route found in 4859.375 ms. Queue query count: 1341

QUERY 3

Start Location: Hoşdere (in Ankara)
End Location: Kadıköy (in Istanbul)
Point of Interest: Duration

Execution 1: All heuristics except 2 and 3 are applied
Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45
Route found in 718.75 ms. Queue query count: 31

Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Kadıköy. **Type:** Ferry. **Explanation:** Once in every 45 minutes. **ID:** 41
Route found in 781.25 ms. Queue query count: 33

Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Eminönü. **Type:** Ferry. **Explanation:** İDO. **ID:** 315932
- Eminönü to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315930
Route found in 859.375 ms. Queue query count: 39

Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Karaköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315933
- Karaköy to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315925
Route found in 843.75 ms. Queue query count: 41

Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Karaköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315933
- Karaköy to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - İDO2. **ID:** 316872
Route found in 859.375 ms. Queue query count: 42

Execution 2: All heuristics except 2 are applied
Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582

- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45
Route found in 734.375 ms. Queue query count: 9

Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Kadıköy. **Type:** Ferry. **Explanation:** Once in every 45 minutes. **ID:** 41
Route found in 718.75 ms. Queue query count: 11

Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Eminönü. **Type:** Ferry. **Explanation:** İDO. **ID:** 315932
- Eminönü to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315930
Route found in 703.125 ms. Queue query count: 17

Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Karaköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315933
- Karaköy to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315925
Route found in 703.125 ms. Queue query count: 19

Suggested route:
- Oran Sitesi to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 145. **ID:** 343817
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Karaköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315933
- Karaköy to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - İDO2. **ID:** 316872
Route found in 796.875 ms. Queue query count: 20

QUERY 4

Start Location: Hoşdere (in Ankara)
End Location: Kadıköy (in Istanbul)
Point of Interest: Duration

Execution 1: All heuristics except 4 are applied
Suggested route:
- Hoşdere to Ankara. **Type:** Virtual Path. **Explanation:** Virtual Path (Hoşdere resides in Ankara). **ID:** 326900
- Ankara to Atatürk Airport. **Type:** Virtual Path. **Explanation:** Virtual path through Esenboğa Airport (by Plane (Domestic) - THY) (Virtual path through Esenboğa Airport (by Plane (Domestic) - THY)). **ID:** 363583
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45
Route found in 328.125 ms. Queue query count: 5

Suggested route:
- Hoşdere to Ankara. **Type:** Virtual Path. **Explanation:** Virtual Path (Hoşdere resides in Ankara). **ID:** 326900

- Ankara to Atatürk Airport. **Type:** Virtual Path. **Explanation:** Virtual path through Esenboğa Airport (by Plane (Domestic) - THY) (Virtual path through Esenboğa Airport (by Plane (Domestic) - THY)). **ID:** 363583
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Kadıköy. **Type:** Ferry. **Explanation:** Once in every 45 minutes. **ID:** 41

Route found in 359.375 ms. Queue query count: 7

Suggested route:
- Hoşdere to Ankara. **Type:** Virtual Path. **Explanation:** Virtual Path (Hoşdere resides in Ankara). **ID:** 326900
- Ankara to Atatürk Airport. **Type:** Virtual Path. **Explanation:** Virtual path through Esenboğa Airport (by Plane (Domestic) - THY) (Virtual path through Esenboğa Airport (by Plane (Domestic) - THY)). **ID:** 363583
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Eminönü. **Type:** Ferry. **Explanation:** İDO. **ID:** 315932
- Eminönü to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315930

Route found in 437.5 ms. Queue query count: 13

Suggested route:
- Hoşdere to Ankara. **Type:** Virtual Path. **Explanation:** Virtual Path (Hoşdere resides in Ankara). **ID:** 326900
- Ankara to Atatürk Airport. **Type:** Virtual Path. **Explanation:** Virtual path through Esenboğa Airport (by Plane (Domestic) - THY) (Virtual path through Esenboğa Airport (by Plane (Domestic) - THY)). **ID:** 363583
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Karaköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315933
- Karaköy to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315925

Route found in 406.25 ms. Queue query count: 15

Suggested route:
- Hoşdere to Ankara. **Type:** Virtual Path. **Explanation:** Virtual Path (Hoşdere resides in Ankara). **ID:** 326900
- Ankara to Atatürk Airport. **Type:** Virtual Path. **Explanation:** Virtual path through Esenboğa Airport (by Plane (Domestic) - THY) (Virtual path through Esenboğa Airport (by Plane (Domestic) - THY)). **ID:** 363583
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Karaköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315933
- Karaköy to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - İDO2. **ID:** 316872

Route found in 421.875 ms. Queue query count: 16

Execution 2: All heuristics are applied (including heuristic 4)

Suggested route:
- Hoşdere to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 189-1. **ID:** 361646
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Kadıköy. **Type:** Dolmuş (Shared Taxi). **ID:** 45

Route found in 1171.875 ms. Queue query count: 22

Suggested route:
- Hoşdere to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 189-1. **ID:** 361646
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Kadıköy. **Type:** Ferry. **Explanation:** Once in every 45 minutes. **ID:** 41

Route found in 1203.125 ms. Queue query count: 24

Suggested route:
- Hoşdere to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 189-1. **ID:** 361646
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35

- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Eminönü. **Type:** Ferry. **Explanation:** İDO. **ID:** 315932
- Eminönü to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315930

Route found in 1312.5 ms. Queue query count: 30

Suggested route:

- Hoşdere to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 189-1. **ID:** 361646
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Karaköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315933
- Karaköy to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315925

Route found in 1312.5 ms. Queue query count: 32

Suggested route:

- Hoşdere to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 189-1. **ID:** 361646
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Beşiktaş. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 44
- Beşiktaş to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315943
- Üsküdar to Karaköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315933
- Karaköy to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - İDO2. **ID:** 316872

Route found in 1296.875 ms. Queue query count: 33

Appendix D

Automated Test Results

D.1 Duration: Example Query Results (Routes)

SEARCH 1
From: Çiftlik Mahallesi - 110357
To: Konya Yolu Hipodrum - 113464
Cost Choice: Duration

DIJKSTRA
Suggested route:
- Çiftlik Mahallesi to Ümraniye. **Type:** Public Transportation (Bus). **Explanation:** IETT - 131B. **ID:** 316490
- Ümraniye to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 10M. **ID:** 363616
- Kadıköy to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 46
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Konya Yolu Hipodrum. **Type:** Public Transportation (Bus). **Explanation:** EGO - 148. **ID:** 344814

Search Duration (ms): 5578,125
Queue Query Count: 391
Total Cost: 271

A*CD
Suggested route:
- Çiftlik Mahallesi to Ümraniye. **Type:** Public Transportation (Bus). **Explanation:** IETT - 131B. **ID:** 316490
- Ümraniye to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 10M. **ID:** 363616
- Kadıköy to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 46
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Konya Yolu Hipodrum. **Type:** Public Transportation (Bus). **Explanation:** EGO - 148. **ID:** 344814

Search Duration (ms): 1500
Queue Query Count: 177
Total Cost: 271

SEARCH 2
From: Meciyeköy - 110522
To: Celal Bayar Bulv. - 113561
Cost Choice: Duration

DIJKSTRA
Suggested route:
- Meciyeköy to Göktürk. **Type:** Public Transportation (Bus). **Explanation:** IETT - 48. **ID:** 316720
- Göktürk to Topkapı. **Type:** Public Transportation (Bus). **Explanation:** IETT - 48A. **ID:** 363886
- Topkapı to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 83. **ID:** 316794
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Konya Yolu. **Type:** Public Transportation (Bus). **Explanation:** EGO - 148. **ID:** 344802
- Konya Yolu to Celal Bayar Bulv.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 101.

Search Duration (ms): 5562,5
Queue Query Count: 560
Total Cost: 300

A*CD
Suggested route:
- Meciyeköy to Göktürk. **Type:** Public Transportation (Bus). **Explanation:** IETT - 48. **ID:** 316720
- Göktürk to Topkapı. **Type:** Public Transportation (Bus). **Explanation:** IETT - 48A. **ID:** 363886
- Topkapı to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 83. **ID:** 316794
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Tarım Koop Merğkez Birl.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 125. **ID:** 336760
- Tarım Koop Merğkez Birl. to Celal Bayar Bulv.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 101. **ID:** 327141
Search Duration (ms): 1234,375
Queue Query Count: 61
Total Cost: 309

SEARCH 3
From: Beytepe Köyü - 113618
To: Atatürk Hastanesi - 113420
Cost Choice: Duration

DIJKSTRA
Suggested route:
- Beytepe Köyü to Atatürk Hastanesi. **Type:** Public Transportation (Bus). **Explanation:** EGO - 175. **ID:** 359362
Search Duration (ms): 5718,75
Queue Query Count: 17
Total Cost: 36

A*CD
Suggested route:
- Beytepe Köyü to Atatürk Hastanesi. **Type:** Public Transportation (Bus). **Explanation:** EGO - 175. **ID:** 359362
Search Duration (ms): 46,875
Queue Query Count: 2
Total Cost: 36

SEARCH 4
From: Mediyeköy - 110353
To: Güvercinlik Cad. Tekel Fabrikası - 113455
Cost Choice: Duration

DIJKSTRA
Suggested route:
- Mediyeköy to Altunizade. **Type:** Public Transportation (Bus). **Explanation:** IETT - 129. **ID:** 363646
- Altunizade to Birlik Mahallesi. **Type:** Public Transportation (Bus). **Explanation:** IETT - 9. **ID:** 316818
- Birlik Mahallesi to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 14B. **ID:** 363698
- Kadıköy to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 46
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Konya Yolu Ankara Ünv.dişçilik Fak.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 148. **ID:** 344804
- Konya Yolu Ankara Ünv.dişçilik Fak. to Güvercinlik Cad. Tekel Fabrikası. **Type:** Public Transportation (Bus). **Explanation:** EGO - 170-1. **ID:** 354616
Search Duration (ms): 5625
Queue Query Count: 538
Total Cost: 308

A*CD
Suggested route:

- Mediyeköy to Altunizade. **Type:** Public Transportation (Bus). **Explanation:** IETT - 129. **ID:** 363646
- Altunizade to Birlik Mahallesi. **Type:** Public Transportation (Bus). **Explanation:** IETT - 9. **ID:** 316818
- Birlik Mahallesi to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 14B. **ID:** 363698
- Kadıköy to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 46
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Çankırı Cad.okul. **Type:** Public Transportation (Bus). **Explanation:** EGO - 146. **ID:** 344106
- Çankırı Cad.okul to Güvercinlik Cad. Tekel Fabrikası. **Type:** Public Transportation (Bus). **Explanation:** EGO - 170-1. **ID:** 354734

Search Duration (ms): 2218,75
Queue Query Count: 309
Total Cost: 321

SEARCH 5
From: AŞTİ - 110203
To: İhlas Marmara Evleri - 110552
Cost Choice: Duration

DIJKSTRA
Suggested route:
- AŞTİ to Simon Bolivar Blv. **Type:** Public Transportation (Bus). **Explanation:** EGO - 167-1. **ID:** 353898
- Simon Bolivar Blv to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 152. **ID:** 345029
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Atakent Mahallesi. **Type:** Public Transportation (Bus). **Explanation:** IETT - 89T. **ID:** 363979
- Atakent Mahallesi to Bakırköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 98T. **ID:** 316855
- Bakırköy to İhlas Marmara Evleri. **Type:** Public Transportation (Bus). **Explanation:** IETT - 76Y. **ID:** 363948

Search Duration (ms): 5453,125
Queue Query Count: 517
Total Cost: 261

A*CD
Suggested route:
- AŞTİ to Simon Bolivar Blv. **Type:** Public Transportation (Bus). **Explanation:** EGO - 167-1. **ID:** 353898
- Simon Bolivar Blv to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 152. **ID:** 345029
- Ulus to Esenboğa Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 35
- Esenboğa Airport to Atatürk Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363582
- Atatürk Airport to Taksim. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272524
- Taksim to Otogar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 83O. **ID:** 363960
- Otogar to Bakırköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 98O. **ID:** 316853
- Bakırköy to İhlas Marmara Evleri. **Type:** Public Transportation (Bus). **Explanation:** IETT - 76Y. **ID:** 363948

Search Duration (ms): 5421,875
Queue Query Count: 1652
Total Cost: 261

SEARCH 6
From: Harem - 110380
To: Konya Yolu Üst Geçit Altı - 113426
Cost Choice: Duration

DIJKSTRA
Suggested route:
- Harem to Pendik. **Type:** Public Transportation (Bus). **Explanation:** IETT - 16A. **ID:** 316566
- Pendik to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 16. **ID:** 363730
- Kadıköy to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 46
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Konya Yolu Üst Geçit Altı. **Type:** Public Transportation (Bus). **Explanation:** EGO - 148. **ID:** 344806

Search Duration (ms): 5421,875

Queue Query Count: 397
Total Cost: 245

A*CD
Suggested route:
- Harem to Pendik. **Type:** Public Transportation (Bus). **Explanation:** IETT - 16A. **ID:** 316566
- Pendik to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 16. **ID:** 363730
- Kadıköy to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 46
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Konya Yolu Üst Geçit Altı. **Type:** Public Transportation (Bus). **Explanation:** EGO - 148. **ID:** 344806

Search Duration (ms): 1734,375
Queue Query Count: 261
Total Cost: 245

SEARCH 7
From: Şirintepe - 110449
To: Güzelyalı - 110362
Cost Choice: Duration

DIJKSTRA
Suggested route:
- Şirintepe to Kabataş. **Type:** Public Transportation (Bus). **Explanation:** IETT - 27E. **ID:** 316607
- Kabataş to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315946
- Kadıköy to Kartal. **Type:** Public Transportation (Bus). **Explanation:** IETT - 21A. **ID:** 316589
- Kartal to Güzelyalı. **Type:** Public Transportation (Bus). **Explanation:** IETT - 133. **ID:** 316496

Search Duration (ms): 5765,625
Queue Query Count: 151
Total Cost: 118

A*CD
Suggested route:
- Şirintepe to Kabataş. **Type:** Public Transportation (Bus). **Explanation:** IETT - 27E. **ID:** 316607
- Kabataş to Kadıköy. **Type:** Ferry. **Explanation:** İDO. **ID:** 315946
- Kadıköy to Kartal. **Type:** Public Transportation (Bus). **Explanation:** IETT - 21A. **ID:** 316589
- Kartal to Güzelyalı. **Type:** Public Transportation (Bus). **Explanation:** IETT - 133. **ID:** 316496

Search Duration (ms): 562,5
Queue Query Count: 47
Total Cost: 118

SEARCH 8
From: Eyüp - 110601
To: Tedaş Etüt İdare - 113478
Cost Choice: Duration

DIJKSTRA
Suggested route:
- Eyüp to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315936
- Üsküdar to Beşiktaş. **Type:** Ferry. **Explanation:** İDO. **ID:** 315944
- Beşiktaş to Taksim. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 43
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Sıhhıye. **Type:** Public Transportation (Bus). **Explanation:** EGO - 183. **ID:** 360613
- Sıhhıye to Tedaş Etüt İdare. **Type:** Public Transportation (Bus). **Explanation:** EGO - 171-1. **ID:** 355839

Search Duration (ms): 5468,75
Queue Query Count: 467
Total Cost: 220

A*CD
Suggested route:
- Eyüp to Üsküdar. **Type:** Ferry. **Explanation:** İDO. **ID:** 315936
- Üsküdar to Beşiktaş. **Type:** Ferry. **Explanation:** İDO. **ID:** 315944

- Beşiktaş to Taksim. **Type:** Dolmuş (Shared Taxi). **Explanation:** Once in every 10 minutes. **ID:** 43
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Eskişehir Yolu Balgat Yurdu. **Type:** Public Transportation (Bus). **Explanation:** EGO - 120. **ID:** 330992
- Eskişehir Yolu Balgat Yurdu to Tedaş Etüt İdare. **Type:** Public Transportation (Bus). **Explanation:** EGO - 171-1. **ID:** 355761

Search Duration (ms): 1937,5
Queue Query Count: 502
Total Cost: 229

SEARCH 9
From: Gürpınar - 110517
To: Küçükköy - 110494
Cost Choice: Duration

DIJKSTRA
Suggested route:
- Gürpınar to Yenibosna Metro. **Type:** Public Transportation (Bus). **Explanation:** IETT - 458. **ID:** 316714
- Yenibosna Metro to Eminönü. **Type:** Public Transportation (Bus). **Explanation:** IETT - 82. **ID:** 316792
- Eminönü to Küçükköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 38E. **ID:** 363841

Search Duration (ms): 5515,625
Queue Query Count: 41
Total Cost: 110

A*CD
Suggested route:
- Gürpınar to Yenibosna Metro. **Type:** Public Transportation (Bus). **Explanation:** IETT - 458. **ID:** 316714
- Yenibosna Metro to Eminönü. **Type:** Public Transportation (Bus). **Explanation:** IETT - 82. **ID:** 316792
- Eminönü to Küçükköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 38E. **ID:** 363841

Search Duration (ms): 234,375
Queue Query Count: 15
Total Cost: 110

SEARCH 10
From: Opera - 113401
To: Alacatlı Cad. - 113648
Cost Choice: Duration

DIJKSTRA
Suggested route:
- Opera to Alacatlı Cad.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 125. **ID:** 336936

Search Duration (ms): 5453,125
Queue Query Count: 199
Total Cost: 40

A*CD
Suggested route:
- Opera to Alacatlı Cad.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 125. **ID:** 336936

Search Duration (ms): 375
Queue Query Count: 2
Total Cost: 40

D.2 Duration: Values to Be Evaluated

Dijkstra Cost	Dijkstra Search Duration	Dijkstra Queue Query Count	A*CD Cost	A*CD Search Duration	A*CD Queue Query Count	Fastest Route Hop Count
271	5578,125	391	271	1500	177	7
300	5562,5	560	309	1234,375	61	8
36	5718,75	17	36	46,875	2	1
308	5625	538	321	2218,75	309	9
261	5453,125	517	261	5421,875	1652	8
245	5421,875	397	245	1734,375	261	7
118	5765,625	151	118	562,5	47	4
220	5468,75	467	229	1937,5	502	8
110	5515,625	41	110	234,375	15	3
40	5453,125	199	40	375	2	1
161	5609,375	226	161	1281,25	203	4
279	5437,5	468	283	1562,5	227	8
27	5671,875	83	28	234,375	2	1
0	5421,875	140	0	46,875	7	0
0	5609,375	676	0	6671,875	2842	0
197	5578,125	193	207	890,625	115	6
0	5375	351	0	15,625	5	0
47	5937,5	48	66	687,5	7	2
0	5437,5	205	0	15,625	5	0
119	5656,25	194	119	296,875	8	3
53	5468,75	174	53	531,25	12	2
201	5609,375	227	201	562,5	74	5
100	5375	74	100	281,25	6	3
145	5515,625	120	145	546,875	30	4
229	5500	280	229	1671,875	391	5
104	5515,625	113	106	312,5	8	3
191	5437,5	251	191	859,375	147	6
217	5593,75	427	228	5000	1100	7
242	5578,125	511	252	1406,25	125	7
46	5784,375	113	48	796,875	4	2
264	5546,875	528	264	5406,25	1672	7
125	5406,25	42	132	171,875	12	3
151	5796,875	196	151	656,25	17	3
0	5500	668	0	6765,625	2866	0
0	5468,75	697	0	2281,25	1446	0
407	5562,5	605	517	4312,5	1564	4
351	5531,25	39	369	343,75	4	3
237	5468,75	253	237	1000	239	6
0	5531,25	477	0	15,625	1	0
144	5406,25	119	144	734,375	64	5
248	5562,5	445	256	1578,125	187	8
213	5671,875	346	215	1250	123	6
265	5484,375	343	275	1359,375	201	7

47	5734,375	136	47	437,5	6	2
240	5515,625	435	240	4593,75	1219	7
171	5625	368	171	484,375	10	5
218	5546,875	368	218	4406,25	898	6
186	5593,75	319	186	2656,25	909	4
271	5671,875	527	278	1593,75	227	8
224	5625	428	224	4656,25	959	7
57	5515,625	245	57	546,875	8	2
40	5687,5	133	40	640,625	12	2
110	5546,875	138	110	484,375	31	4
119	5703,125	117	119	281,25	8	3
60	6390,625	220	60	2250	27	2
259	5609,375	435	259	1468,75	162	7
225	5656,25	368	225	4421,875	1121	7
137	5640,625	92	137	671,875	57	4
128	5515,625	125	128	234,375	15	3
42	5718,75	111	52	281,25	4	2
0	5640,625	245	0	31,25	2	0
42	5656,25	49	55	187,5	4	2
261	5593,75	423	263	1500	177	7
22	5671,875	15	27	125	2	2
190	6125	329	192	3890,625	723	5
40	5796,875	46	44	296,875	4	2
125	5484,375	115	125	234,375	5	3
298	5671,875	455	302	921,875	48	8
231	7093,75	492	231	4781,25	146	7
160	5546,875	192	160	703,125	52	4
215	5609,375	358	215	1265,625	143	6
283	5515,625	318	290	1828,125	313	7
213	5671,875	330	213	750	38	6
23	5562,5	15	23	140,625	2	2
43	5890,625	86	60	343,75	4	2
0	5578,125	698	0	7390,625	2866	0
52	5500	126	60	343,75	8	2
43	6140,625	124	43	500	7	2
212	6046,875	319	212	1843,75	545	5
244	5859,375	497	244	5125	1421	7
256	5953,125	505	256	5109,375	1269	7
193	5843,75	136	193	375	28	5
232	6218,75	407	232	5109,375	1319	6
274	6390,625	493	274	2531,25	337	8
220	5968,75	400	220	843,75	46	6
227	5671,875	381	234	4468,75	1355	7
70	5687,5	249	70	750	37	3
0	5578,125	709	0	2875	1380	0
0	5843,75	711	0	6765,625	2866	0
40	6421,875	28	40	668,75	4	2

50	5640,625	249	57	1125	32	2
220	5953,125	355	220	4734,375	1265	8
202	5765,625	513	220	2437,5	205	7
114	6515,625	164	114	2000	127	4
251	6406,25	479	270	2593,75	291	8
69	5796,875	260	82	1156,25	55	3
31	5765,625	35	35	93,75	2	1
218	5625	405	222	4375	950	7
211	5578,125	391	211	4906,25	1158	7
267	6937,5	284	267	4640,625	546	5

D.3 Financial Cost: Example Query Results (Routes)

SEARCH 1
From: Sultan Murat Mahallesi - 110420
To: Yeşilbağlar - 110427
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Sultan Murat Mahallesi to Üsküdar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 15S. **ID:** 363725
- Üsküdar to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 12. **ID:** 363635
- Kadıköy to Yeşilbağlar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 16Y. **ID:** 316573
Search Duration (ms): 5703,125
Queue Query Count: 50
Total Cost: 6

A*CD
Suggested route:
- Sultan Murat Mahallesi to Üsküdar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 15S. **ID:** 363725
- Üsküdar to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 12. **ID:** 363635
- Kadıköy to Yeşilbağlar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 16Y. **ID:** 316573
Search Duration (ms): 265,625
Queue Query Count: 22
Total Cost: 6

SEARCH 2
From: Ankara Otogar - 110231
To: Sultançiftliği - 110470
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Ankara Otogar to Istanbul Otogar. **Type:** Intercity Bus. **Explanation:** Ulusoy. **ID:** 316107
- Istanbul Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 308255
- Taksim to Dikilitaş. **Type:** Public Transportation (Bus). **Explanation:** IETT - 43. **ID:** 316710
- Dikilitaş to Eminönü. **Type:** Public Transportation (Bus). **Explanation:** IETT - 26. **ID:** 316604
- Eminönü to Sultançiftliği. **Type:** Public Transportation (Bus). **Explanation:** IETT - 336E. **ID:** 363805
Search Duration (ms): 5625
Queue Query Count: 484
Total Cost: 61

A*CD

Suggested route:
- Ankara Otogar to Ulus. **Type:** Dolmuş (Shared Taxi). **ID:** 308257
- Ulus to Kuğulu Park. **Type:** Public Transportation (Bus). **Explanation:** EGO - 114-1. **ID:** 328882
- Kuğulu Park to AŞTİ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 167-1. **ID:** 353907
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Dikilitaş. **Type:** Public Transportation (Bus). **Explanation:** IETT - 43. **ID:** 316710
- Dikilitaş to Eminönü. **Type:** Public Transportation (Bus). **Explanation:** IETT - 26. **ID:** 316604
- Eminönü to Sultançiftliği. **Type:** Public Transportation (Bus). **Explanation:** IETT - 336E. **ID:** 363805

Search Duration (ms): 5218,75
Queue Query Count: 1572
Total Cost: 102

SEARCH 3
From: Topselvi - 110372
To: Sultan Murat Mahallesi - 110420
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Topselvi to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 16B. **ID:** 363732
- Kadıköy to Üsküdar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 12. **ID:** 316469
- Üsküdar to Sultan Murat Mahallesi. **Type:** Public Transportation (Bus). **Explanation:** IETT - 15S. **ID:** 316560

Search Duration (ms): 5578,125
Queue Query Count: 137
Total Cost: 6

A*CD
Suggested route:
- Topselvi to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 16B. **ID:** 363732
- Kadıköy to Üsküdar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 12. **ID:** 316469
- Üsküdar to Sultan Murat Mahallesi. **Type:** Public Transportation (Bus). **Explanation:** IETT - 15S. **ID:** 316560

Search Duration (ms): 437,5
Queue Query Count: 27
Total Cost: 6

SEARCH 4
From: Esenkent - 110369
To: Selimiye Cad - 113560
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Esenkent to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 21C. **ID:** 363756
- Kadıköy to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 110. **ID:** 316451
- Taksim to Istanbul Otogar. **Type:** Dolmuş (Shared Taxi). **ID:** 308254
- Istanbul Otogar to Ankara Otogar. **Type:** Intercity Bus. **Explanation:** Ulusoy. **ID:** 316195
- Ankara Otogar to Ulus. **Type:** Dolmuş (Shared Taxi). **ID:** 308257
- Ulus to Selimiye Cad. **Type:** Public Transportation (Bus). **Explanation:** EGO - 152. **ID:** 345036

Search Duration (ms): 5546,875
Queue Query Count: 459
Total Cost: 67

A*CD
Suggested route:
- Esenkent to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 21C. **ID:** 363756
- Kadıköy to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 110. **ID:** 316451
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Selimiye Cad. **Type:** Public Transportation (Bus). **Explanation:** EGO - 152. **ID:** 345036

Search Duration (ms): 2875
Queue Query Count: 1443

Total Cost: 117

SEARCH 5
From: Gözcübaba - 110566
To: Ümitköy 8. Cad. - 113653
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Gözcübaba to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 8A. **ID:** 363981
- Kadıköy to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 110. **ID:** 316451
- Taksim to Istanbul Otogar. **Type:** Dolmuş (Shared Taxi). **ID:** 308254
- Istanbul Otogar to Ankara Otogar. **Type:** Intercity Bus. **Explanation:** Ulusoy. **ID:** 316195
- Ankara Otogar to Ulus. **Type:** Dolmuş (Shared Taxi). **ID:** 308257
- Ulus to Ümitköy 8. Cad.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 125. **ID:** 336788
Search Duration (ms): 5687,5
Queue Query Count: 421
Total Cost: 67

A*CD
Suggested route:
- Gözcübaba to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 8A. **ID:** 363981
- Kadıköy to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 110. **ID:** 316451
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Ümitköy 8. Cad.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 125. **ID:** 336788
Search Duration (ms): 2812,5
Queue Query Count: 1359
Total Cost: 117

SEARCH 6
From: Filistin - 113550
To: Kavakpınar - 110366
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Filistin to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 114-1. **ID:** 328869
- Ulus to Ankara Otogar. **Type:** Dolmuş (Shared Taxi). **ID:** 308256
- Ankara Otogar to Istanbul Otogar. **Type:** Intercity Bus. **Explanation:** Ulusoy. **ID:** 316107
- Istanbul Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 308255
- Taksim to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 110. **ID:** 363617
- Kadıköy to Kartal. **Type:** Public Transportation (Bus). **Explanation:** IETT - 21A. **ID:** 316589
- Kartal to Kavakpınar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 133K. **ID:** 316499
Search Duration (ms): 5984,375
Queue Query Count: 544
Total Cost: 69

A*CD
Suggested route:
- Filistin to Kuğulu Park. **Type:** Public Transportation (Bus). **Explanation:** EGO - 112-1. **ID:** 328744
- Kuğulu Park to AŞTİ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 167-1. **ID:** 353907
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 110. **ID:** 363617
- Kadıköy to Kartal. **Type:** Public Transportation (Bus). **Explanation:** IETT - 21A. **ID:** 316589
- Kartal to Kavakpınar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 133K. **ID:** 316499
Search Duration (ms): 6000
Queue Query Count: 1634
Total Cost: 97

SEARCH 7
From: Ballıkumcu Köyü - 113543

To: Kısırkaya - 110410
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Ballıkumcu Köyü to Konya Yolu Ankara Ünv.dişçilik Fak.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 196-1. **ID:** 363443
- Konya Yolu Ankara Ünv.dişçilik Fak. to Ulus. **Type:** Public Transportation (Bus). **Explanation:** EGO - 148. **ID:** 344805
- Ulus to Ankara Otogar. **Type:** Dolmuş (Shared Taxi). **ID:** 308256
- Ankara Otogar to Istanbul Otogar. **Type:** Intercity Bus. **Explanation:** Ulusoy. **ID:** 316107
- Istanbul Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 308255
- Taksim to Sarıyer. **Type:** Public Transportation (Bus). **Explanation:** IETT - 25T. **ID:** 363768
- Sarıyer to Kısırkaya. **Type:** Public Transportation (Bus). **Explanation:** IETT - 152. **ID:** 316548
Search Duration (ms): 5750
Queue Query Count: 441
Total Cost: 70

A*CD
Suggested route:
- Ballıkumcu Köyü to ODTÜ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 196-1. **ID:** 362385
- ODTÜ to Tarım Koop Merğkez Birl.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 111. **ID:** 328094
- Tarım Koop Merğkez Birl. to AŞTİ. **Type:** Public Transportation (Bus). **Explanation:** EGO - 167-1. **ID:** 353893
- AŞTİ to Harem Otogar. **Type:** Intercity Bus. **ID:** 363578
- Harem Otogar to Taksim. **Type:** Dolmuş (Shared Taxi). **ID:** 33
- Taksim to Sarıyer. **Type:** Public Transportation (Bus). **Explanation:** IETT - 25T. **ID:** 363768
- Sarıyer to Kısırkaya. **Type:** Public Transportation (Bus). **Explanation:** IETT - 152. **ID:** 316548
Search Duration (ms): 4875
Queue Query Count: 1405
Total Cost: 98

SEARCH 8
From: Bulgurlu Mahallesi - 110336
To: Hacılar Köyü - 113509
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Bulgurlu Mahallesi to Üsküdar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 11L. **ID:** 363626
- Üsküdar to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 12. **ID:** 363635
- Kadıköy to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 110. **ID:** 316451
- Taksim to Istanbul Otogar. **Type:** Dolmuş (Shared Taxi). **ID:** 308254
- Istanbul Otogar to Ankara Otogar. **Type:** Intercity Bus. **Explanation:** Ulusoy. **ID:** 316195
- Ankara Otogar to Ulus. **Type:** Dolmuş (Shared Taxi). **ID:** 308257
- Ulus to Meşrutiyet Cad.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 189-1. **ID:** 361613
- Meşrutiyet Cad. to Hacılar Köyü. **Type:** Public Transportation (Bus). **Explanation:** EGO - 177-1. **ID:** 360047
Search Duration (ms): 5578,125
Queue Query Count: 545
Total Cost: 72

A*CD
Suggested route:
- Bulgurlu Mahallesi to Üsküdar. **Type:** Public Transportation (Bus). **Explanation:** IETT - 11L. **ID:** 363626
- Üsküdar to Kadıköy. **Type:** Public Transportation (Bus). **Explanation:** IETT - 12. **ID:** 363635
- Kadıköy to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 110. **ID:** 316451
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Opera. **Type:** Public Transportation (Bus). **Explanation:** EGO - 114-1. **ID:** 328846
- Opera to Hacılar Köyü. **Type:** Public Transportation (Bus). **Explanation:** EGO - 177-1. **ID:** 359735
Search Duration (ms): 3000

Queue Query Count: 1392
Total Cost: 122

SEARCH 9
From: Sokullu Mehmet Paşa 2. Durak - 113614
To: Rafet Canıtez Cad. - 113599
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Sokullu Mehmet Paşa 2. Durak to Dikmen Cad. **Type:** Public Transportation (Bus). **Explanation:** EGO - 140. **ID:** 343605
- Dikmen Cad to Rafet Canıtez Cad.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 185. **ID:** 361603
Search Duration (ms): 5484,375
Queue Query Count: 120
Total Cost: 6

A*CD
Suggested route:
- Sokullu Mehmet Paşa 2. Durak to Güvenpark. **Type:** Public Transportation (Bus). **Explanation:** EGO - 140. **ID:** 343565
- Güvenpark to Rafet Canıtez Cad.. **Type:** Public Transportation (Bus). **Explanation:** EGO - 185. **ID:** 361225
Search Duration (ms): 343,75
Queue Query Count: 4
Total Cost: 6

SEARCH 10
From: Karlıktepe - 110378
To: 3. Etap - 113604
Cost Choice: Financial Cost

DIJKSTRA
Suggested route:
- Karlıktepe to Ortaçeşme. **Type:** Public Transportation (Bus). **Explanation:** IETT - 135K. **ID:** 363680
- Ortaçeşme to Ümraniye. **Type:** Public Transportation (Bus). **Explanation:** IETT - 11H. **ID:** 316458
- Ümraniye to Üstbostancı. **Type:** Public Transportation (Bus). **Explanation:** IETT - 19D. **ID:** 316579
- Üstbostancı to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 202. **ID:** 316588
- Taksim to Istanbul Otogar. **Type:** Dolmuş (Shared Taxi). **ID:** 308254
- Istanbul Otogar to Ankara Otogar. **Type:** Intercity Bus. **Explanation:** Ulusoy. **ID:** 316195
- Ankara Otogar to Ulus. **Type:** Dolmuş (Shared Taxi). **ID:** 308257
- Ulus to Hazine. **Type:** Public Transportation (Bus). **Explanation:** EGO - 125. **ID:** 336774
- Hazine to 3. Etap. **Type:** Public Transportation (Bus). **Explanation:** EGO - 111. **ID:** 328590
Search Duration (ms): 5750
Queue Query Count: 572
Total Cost: 74

A*CD
Suggested route:
- Karlıktepe to Ortaçeşme. **Type:** Public Transportation (Bus). **Explanation:** IETT - 135K. **ID:** 363680
- Ortaçeşme to Ümraniye. **Type:** Public Transportation (Bus). **Explanation:** IETT - 11H. **ID:** 316458
- Ümraniye to Üstbostancı. **Type:** Public Transportation (Bus). **Explanation:** IETT - 19D. **ID:** 316579
- Üstbostancı to Taksim. **Type:** Public Transportation (Bus). **Explanation:** IETT - 202. **ID:** 316588
- Taksim to Atatürk Airport. **Type:** Public Transportation (Bus). **Explanation:** HAVAŞ. **ID:** 272525
- Atatürk Airport to Esenboğa Airport. **Type:** Plane (Domestic). **Explanation:** THY. **ID:** 363580
- Esenboğa Airport to Ulus. **Type:** Public Transportation (Bus). **ID:** 308264
- Ulus to Güvenpark. **Type:** Public Transportation (Bus). **Explanation:** EGO - 114-1. **ID:** 328854
- Güvenpark to 3. Etap. **Type:** Public Transportation (Bus). **Explanation:** EGO - 111. **ID:** 328600
Search Duration (ms): 3140,625
Queue Query Count: 1435
Total Cost: 124

D.4 Financial Cost: Values to Be Evaluated

Dijkstra Cost	Dijkstra Search Duration	Dijkstra Queue Query Count	A*CD Cost	A*CD Search Duration	A*CD Queue Query Count	Fastest Route Hop Count
6	5703,125	50	6	265,625	22	3
61	5625	484	102	5218,75	1572	5
6	5578,125	137	6	437,5	27	3
67	5546,875	459	117	2875	1443	6
67	5687,5	421	117	2812,5	1359	6
69	5984,375	544	97	6000	1634	7
70	5750	441	98	4875	1405	7
72	5578,125	545	122	3000	1392	8
6	5484,375	120	6	343,75	4	2
74	5750	572	124	3140,625	1435	9
9	5593,75	273	9	3609,375	767	3
12	5421,875	224	12	1125	308	6
69	5734,375	557	97	6406,25	1653	7
4	5500	21	4	187,5	3	2
74	5921,875	576	124	3109,375	1420	9
70	5703,125	585	120	3312,5	1436	7
72	6203,125	502	122	3046,875	1424	8
4	5500	46	7	78,125	3	2
6	5765,625	123	6	484,375	4	2
69	5625	475	119	2812,5	1411	7
10	6796,875	46	10	562,5	11	5
68	6187,5	538	118	3125	1444	6
3	6453,125	114	3	578,125	2	1
6	6640,625	79	6	203,125	8	3
8	5500	176	8	1140,625	90	4
6	5781,25	129	6	875	20	2
6	5484,375	119	6	468,75	26	3
6	5671,875	31	6	484,375	4	2
74	5750	558	124	3203,125	1420	9
69	5468,75	384	119	2937,5	1443	7
0	5609,375	454	0	31,25	5	0
70	5718,75	612	120	3343,75	1462	7
65	5484,375	355	93	3765,625	351	5
6	6093,75	231	6	1312,5	46	2
12	5609,375	289	12	1734,375	283	4
72	5453,125	481	100	5343,75	1501	8
69	5562,5	552	97	5921,875	1648	7
8	5437,5	101	8	250	19	4
67	5875	386	95	4781,25	1404	6
6	5453,125	209	6	1156,25	111	3
8	5656,25	196	8	1000	112	4
108	5562,5	624	108	593,75	17	3
6	5703,125	53	6	281,25	22	3
69	6343,75	366	119	2953,125	1439	6
6	7609,375	97	6	984,375	21	3
8	5546,875	153	8	500	33	4
6	5875	181	6	750	11	2
0	5968,75	438	0	15,625	2	0

70	5812,5	544	120	2953,125	1360	7
6	5781,25	153	6	578,125	4	2
68	5593,75	343	96	3796,875	573	6
10	5609,375	254	10	1343,75	234	5
70	5687,5	525	120	3015,625	1372	7
74	5671,875	596	124	3453,125	1442	9
0	5515,625	260	0	0	1	0
72	6828,125	559	122	3359,375	1408	8
9	5843,75	246	9	2140,625	106	3
67	5765,625	373	95	4906,25	1404	6
72	5734,375	578	122	3171,875	1385	8
0	5718,75	3	0	0	1	0
67	5750	487	117	2859,375	1371	6
0	5937,5	717	95	7296,875	2818	0
70	6156,25	593	120	3515,625	1383	7
69	5921,875	545	94	6125	1492	7
6	5812,5	131	6	578,125	4	2
10	5781,25	245	10	1593,75	475	5
6	5812,5	72	6	250	4	2
69	5968,75	576	97	6046,875	1674	7
3	5765,625	61	3	343,75	2	1
70	5890,625	417	95	5265,625	1359	7
8	5843,75	163	8	687,5	63	4
6	5718,75	128	6	359,375	18	3
8	5578,125	148	8	265,625	19	4
70	5890,625	470	95	5062,5	1430	7
72	5828,125	482	97	5265,625	1428	8
0	5796,875	709	0	7468,75	2866	0
69	5890,625	577	97	8640,625	1674	7
69	6031,25	365	119	2968,75	1411	7
67	5765,625	424	95	5171,875	1407	6
0	5859,375	463	0	15,625	4	0
12	5828,125	242	12	1265,625	257	5
67	6375	407	117	3046,875	1435	6
6	5812,5	51	6	187,5	4	2
0	5734,375	36	0	46,875	5	0
76	6328,125	598	126	3250	1435	9
76	6390,625	568	126	3453,125	1404	10
6	6406,25	227	6	2203,125	83	2
72	6265,625	537	100	6000	1539	8
67	6328,125	366	117	3578,125	1419	6
0	6125	677	0	2625	1383	0
3	6218,75	59	3	156,25	2	1
12	6937,5	277	12	2218,75	768	6
12	6406,25	272	12	1375	133	5
70	6453,125	408	95	6015,625	1383	7
6	5796,875	222	6	1218,75	21	2
6	5812,5	88	6	703,125	5	2
6	7187,5	64	6	515,625	4	2
0	6375	711	0	8156,25	2838	0
67	7437,5	431	95	6015,625	1407	6
69	7156,25	565	97	6781,25	1572	7