FREIE UNIVERSITÄT BOZEN
LIBERA UNIVERSITÀ DI BOLZANO
FREE UNIVERSITY OF BOZEN · BOLZANO

TECHNISCHE UNIVERSITÄT WIEN
VIENNA UNIVERSITY OF TECHNOLOGY

EUROPEAN MASTER THESIS

An Application of Heuristic Route Search Techniques for a Scalable Flight Search System

written by

Yansen Darmaputra

under supervision of

Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler

Database and Artificial Intelligence Group
Institute for Information Systems
Faculty of Informatics
Technische Universität Wien
Vienna, Austria

June 2008

Abstract

Most of the flight search engines that are currently available in the Internet can be considered as meta-search engine. They forward the route query to other websites such as online travel agents, major airlines, and low-cost carriers. The result from each source is collected and aggregated before presented to user. There are several weaknesses for this approach.

First, not all available routes can be found. Take low-cost airlines (which emphasize on direct flights) as example. If the airline doesn't have a direct flight from X to Y, then trying to search for route from X to Y, would give no result, although it has route from X to Z and Z to Y. Second, they can not mix flights between airlines in different alliances. An airline would never promote flights from other airline, except if the other airline is in the same alliance or if there exists some codeshare agreements.

In this thesis we propose a *mashup* solution for this problem. A mashup application does not own the data. It uses data from other resources (called *content provider*) to create a new application with new feature and functionality that is not offered by any of the content provider. The web data extraction technology from Lixto Visual Developer is used to *wrap* data.

The flight search problem is treated as graph search problem with airports as the nodes and the pair of airports where exists direct flight between them as the edges. We consider the scalability of the current flight search engines by introducing *hub identification heuristic*. Instead of analyzing and evaluating all possible routes to reach the destination, this heuristic gives hint on which hub airports that are possibly containing the best routes (in terms of shortest flight duration). Hence, the system can limit the search by only considering a fraction from all possible routes. Performing the search this way also ensures system scalability and increases the system's responsiveness by shortening the query procesing time.

The term *interesting route* is defined as a list of routes which match with the user's preference. An interesting route for one may not be interesting for the other. Therefore, three sorting criteria are used for evaluating and ranking the search results. The search is also very customizable, for example user can choose airline preferences (e.g. only searching from low-cost airlines), transit time preference (e.g. transits between 1-3 hours), which airports/routes to be chosen for transits/stops (e.g. avoid non-Schengen airports), and searching for flights from close airports (e.g. include other airports within particular distance from the original departure airport).

Acknowledgements

I would like to thank every one in general who has helped and given continuous support for me in writing this Thesis, either directly or indirectly, in forms of physical and moral support, encouragement, suggestion, and prayers.

Thank you to Prof. Reinhard Pichler as the supervisor of this Thesis. This work could not be done without contributions from members of Database and Artificial Intelligence (DBAI) group, especially from Robert Baumgartner, Bernhard Krüpl, and Wolfgang Holzinger who have shared and given many ideas, thoughts, and insightful discussions during the development of this Thesis.

Special thanks to Lixto Software GmbH who has provided the academic trial license for Lixto Visual Developer software that is used in this Thesis for creating wrappers. Thank you to Gerald Ledermüller from Lixto that provide short introduction on Lixto Visual Developer and prepare the tutorials and documentations.

I would like also to say my gratitude to all lecturers and professors in Free University of Bozen-Bolzano, the university where I spent the first year of this European Master in Computational Logic (EMCL) program, for giving me the basic foundations so that I am able to write this thesis.

I am very grateful for the financial support provided through Erasmus Mundus scholarship. Thank you to Prof. Steffen Hölldobler (TUD), Prof. Enrico Franconi (FUB), Prof. Sergio Tessaris (FUB), Prof. Alexander Leitsch (TUW), Prof. Susana Munoz Hernandez (UPM), and Prof. Luis Monis Perreira (UNL) as the coordinators of the EMCL program, who have given me the chance to pursue this education in Europe.

Contents

Abstract i

Acknowledgements ii

List of Figures vi

List of Tables viii

1 Introduction 1
- 1.1 Motivation . . . 3
- 1.2 Contribution . . . 5
- 1.3 Organization of Thesis . . . 7

2 Preliminaries and Backgrounds 9
- 2.1 Types of Airlines . . . 9
- 2.2 Several Types of Flight Search Engine . . . 12
- 2.3 Advantages of Mixing Airlines . . . 16
- 2.4 Geocoding . . . 17
- 2.5 PostGIS . . . 19
- 2.6 Maps API . . . 19

3 Extracting Data from the Web 21
- 3.1 The Problems with the Web . . . 21
- 3.2 Semantic Web . . . 23
  - 3.2.1 Resource Description Framework (RDF) . . . 24
  - 3.2.2 RDF Schema (RDFS) . . . 26
  - 3.2.3 Web Ontology Language (OWL) . . . 26
  - 3.2.4 Challenges of Semantic Web . . . 27
- 3.3 Techniques for Web Data Extraction . . . 27
- 3.4 Lixto Visual Developer . . . 28
  - 3.4.1 Editor and Navigation . . . 29
  - 3.4.2 Data Model . . . 30
  - 3.4.3 Patterns and Filters . . . 31

4 System Design and Data Preparation 32
- 4.1 System Architecture . . . 32
- 4.2 Data Requirement . . . 33

4.3 Business Logic . . . 35
4.3.1 Web and Wrapper Layer . . . 35
4.3.2 Database Layer . . . 36
4.3.3 User Interface Layer . . . 38
4.4 Adding Geometry Column in Table Airports . . . 39
4.5 Wrapping Airports Data . . . 40
4.6 Data Cleaning of Airports Time Zone Value . . . 41
4.7 Wrapping Airlines Website . . . 42
4.7.1 Choosing the Airlines to be Wrapped . . . 42
4.7.2 Filtering Search Results . . . 43

5 Scalable Algorithm 46
5.1 Complexity for Flight Search . . . 46
5.1.1 Complexity for Direct Routes . . . 46
5.1.2 Complexity for Transit Routes . . . 47
5.2 Route Determination . . . 48
5.2.1 No Transit vs. One Transit vs. Two Transit . . . 49
5.2.2 Transit Time Analysis . . . 49
5.3 Hub Identification Heuristics . . . 51
5.3.1 Approaches for Hub Identification . . . 52
5.3.2 Optimality of the Heuristic . . . 54
5.4 Searching from Close Airports . . . 57
5.5 Evaluation Function . . . 57
5.6 Scalability . . . 58
5.6.1 Limit The Search to Two Transit . . . 58
5.6.2 Table Design . . . 58
5.6.3 Limiting the Number of Possible Routes . . . 58
5.6.4 Limiting the Number of Flight Combinations . . . 59

6 Experimental Results 61
6.1 Random Schedule Generator . . . 61
6.2 Search Parameter . . . 63
6.2.1 Common Search Parameter . . . 63
6.2.2 Advanced Search Parameter . . . 64
6.3 Performing the Search . . . 65

7 Conclusions 67
7.1 Summary . . . 67
7.2 Future Works . . . 69
7.2.1 Validate the Interesting Route Output . . . 69
7.2.2 Extend the Close Airport Feature . . . 69
7.2.3 Cost to Switch Between Airports . . . 70
7.2.4 Extend the Travel Domain . . . 71

A Wrapping Airports Data 72
A.1 Navigation Sequence . . . 72

A.2 General XPath Expression 74
A.3 Configuring XPath for Iterators 75

Bibliography **76**

List of Figures

2.1 Order screen of a low-cost carrier, showing the total fare components . . . 10
2.2 Lounge and seat assignment service offered by a low-cost carrier 10
2.3 Screenshot of AirNinja 12
2.4 Screenshot of Low Cost Airline Guide 12
2.5 Star Alliance search page 13
2.6 Star Alliance search results 13
2.7 Working diagram of flight meta-search engine 14
2.8 Screenshot of Momondo 14
2.9 Skyscanner results for flight from Vienna to multiple destinations in Spain during July 2008 15
2.10 Skyscanner results for flights from Vienna to Barcelona during July 2008 . 15
2.11 Map of European airports 17
2.12 Great-circle distance 18
2.13 PostGIS extension in PostgreSQL relational database 19
2.14 Google Map example for possible routes from Barcelona to Berlin 20

3.1 Google search results for Jurassic Park book price 21
3.2 Amazon search results for books about "Web" 22
3.3 Difficulties of aggregating search results 23
3.4 RDF example for airline's schedule 25
3.5 Lixto VD services overview 29
3.6 Main screen of Lixto VD 30
3.7 Example of data model 30
3.8 Using patterns and filters in Lixto VD 31

4.1 System architecture 32
4.2 Flight statistics for Emirates' flight number EK344 34
4.3 Flight statistics for Easy Jet 34
4.4 Flight statistic for Vienna-Amsterdam route 34
4.5 Business logic of the system 35
4.6 Database schema of the system 37
4.7 Lixto data model for airports 40
4.8 World Airport Codes screenshot 40
4.9 Choosing only airlines with select box and JavaScript 43
4.10 Air Berlin's direct flight and flight with transits 44
4.11 Lixto data model for connections 44
4.12 Wrapping Ryan Air 45
4.13 Wrapping Air Berlin 45

5.1 Graph representation of the flight database 47
5.2 Search complexity for one-transit and two-transit routes 48
5.3 Possible transits from London Stansted to Nuremberg 50
5.4 Transavia route map, with hub at Amsterdam 51
5.5 Route search techniques by using hub 52
5.6 Identifying hub by number of available connections 53
5.7 Identifying hub by using geographical coordinate 54
5.8 Identifying hub by total journey distance 55
5.9 Start page of the system 56
5.10 Flowchart for searching from close airports 57
5.11 Limiting the airlines included in the search by putting bounds on price and reliability index 59
5.12 Limiting the airlines included in the search by putting bounds on transit time 59

6.1 Least squares function for distance less than 12,000 km 62
6.2 Least squares function for distance greater than 12,000 km 62
6.3 Common search parameters 63
6.4 Advanced search parameters 64
6.5 List of possible routes from London Stansted to Lisbon 66
6.6 Search results for flights from London Stansted to Lisbon 66

7.1 Summary diagram of the thesis 67
7.2 Future work for the system 69
7.3 Extending the close airport feature 70
7.4 Measuring the journey time by using car with Google Map 71

A.1 Navigation sequence in Lixto VD 72
A.2 Calling pageclass start2 from "iatalist" 73
A.3 Calling pageclass start3 from "airportslist" 73
A.4 Getting XPath expression of an element 74
A.5 XPath for elements of "iatalist" 74
A.6 XPath for elements of "airportslist" 75
A.7 Putting general XPath expression for "iatalist" 75

List of Tables

2.1 Number of reachable airports of by adding one-stop 16

4.1 Schema of Table Airports 37
4.2 Schema of Table Airlines 37
4.3 Schema of Table Connections 38
4.4 Schema of Table Schedules 38
4.5 Schema of Table Distances 38

5.1 Transit time comparisons 50

Chapter 1

Introduction

Travel industry is clearly one where Internet has a strong impact on customer behavior and business models. In Europe, online travel sales accounted for about 10% of the total market in 2005. Internet transactions are showing a strong growth rate and are expected to reach 20% of overall market in 2010. In air travel sector, the online transaction is even higher, accounting for over 50% of the European online market.

According to recent research by the European Commission [19], more and more European tour operators are taking reservations online. About 36% of tour operators, 62% of hotels, and 40% of package deal operators have internet booking capabilities. Of those, 28% receive a quarter of their bookings via this channel.

There are clear advantages of doing reservations through the Internet. People can book a reservation 24 hours a day, 7 days a week. Online booking saves a lot of time. One can now reserve and pay for a flight to Paris (comparing schedules and prices), book an ongoing rail ticket to Toulouse, select a hotel on the basis of exterior and interior views and customer reviews, negotiate over the room price and bed size, and order flowers and chocolates to be delivered upon arrival, all without leaving one's desk. Different offers can also be compared with complete transparency.

Online travel agencies are taking over from traditional travel agencies. They have the advantage of being able to access multiple real-time reservation systems, to group product offerings, and to provide preferential pricing through strategic alliances with various online reservation system firms. It is obvious that online services provide less consultation, but clients don't have to go into a travel agency. Both sides win, while clients can review the various offers at their leisure, the companies can cut distribution costs.

Online services are also feeding a new trend called *dynamic packaging* in which people switch from the traditional all-inclusive packages that have been offered for decades by

travel companies. Rather, they can create their own packages as their wish, choosing the flight, hotel and local services themselves - a process made easier on the Internet.

The convenience and availability of information makes the traditional travel agents (still struggling with text-based direct reservation systems, directories of hotels, and the problems of making multiple telephone calls, maybe to countries with several hours of time difference) redundant. The only way in which travel agencies can compete is by focusing on the ever-decreasing proportion of consumers who are not computer-literate or who would rather pay someone to make reservations for them.

Low-cost airlines, modeled on the phenomenally successful Southwest Airlines in the United States, are characterized by high utilization of aircraft, one-way pricing and extra charges, such as for food and drinks. The Internet has helped fuel the boom in low-cost airlines such as Easy Jet, Ryan Air, and Air Berlin, which rely almost entirely on online reservations. For these airlines, online booking means extra revenue, eliminating giving away commissions to the travel agents.

Europe's low-cost airlines have grown at a tremendous rate in recent years as attractive ticket prices incited more people to fly. According to a recent article at International Herald Tribune [26], low-cost carriers are growing at 9 percent a year. The arrival of low-cost airlines across Europe has created an explosion of new resorts in places like Murcia (Spain), catering to budget tourists. It is extraordinarily easy and cheap to get to Murcia from nearly anywhere in Britain - and from many small cities in Germany, the Netherlands and Norway, for that matter.

Passenger statistics at Murcia's San Javier Airport over the past decade illustrate the magnitude of the problem. Arrivals increased from 88,608 in 1995, to 848,037 in 2004, to a staggering 1,905,182 last year - more than a 20-fold increase in a little more than a decade. Norwegian, Scandinavia's largest low-cost airline, flies to Murcia from Bergen, Oslo, Stavanger, and Trondheim, with prices starting at about €100. Demand for low-cost flights is galloping so fast that private investors are building a new international airport in Corvera, 20 minutes from Murcia.

It is not only low-cost carriers who are expanding their business, major airlines do that as well. People are becoming more mobile each day, so that the need for traveling is also increasing. Just few months ago in October 2007, Singapore Airlines launched the first commercial flight with Airbus A380, the new double-decker jumbo jet, from Singapore to Sydney and return. This A380 can carry up to 525 people in three-class configuration (economy, business, and first class) or 853 people in all-economy configuration. All of these facts constitutes the fact that online airline business will be one of the dominating Internet business in the future.

1.1 Motivation

There are a lot of flight search engines currently available in the Internet. If one tries to search in Google using keywords such as "cheap flights" or "flight booking", he/she would get numerous number of online travel agents from where the flight query can be posed. Most of these search engines share the same characteristic, that is they behave as *meta-search engine* that simply forward the query to several other websites and aggregate the result from each website. In flights domain, meta-search engine has some inherent weaknesses.

The first weakness is that current flight search engine has not yet offered all available routes. We take an example from one of the low-cost airlines, that is Easy Jet. In Easy Jet's website, there are two select boxes, one for choosing departure airport, and the other one is for choosing destination airport. The website uses JavaScript so that the list of destinations changes dynamically based on the choice of departure airport. By choosing Barcelona as departure airport, we get the list of destinations that reachable from Barcelona. Since Rome is not in the list of destinations, then it seems that we can not fly with Easy Jet from Barcelona to Rome. This statement is true if we only consider direct flights. However, if we consider also transit routes, then it is actually possible to go from Barcelona to Rome with Easy Jet.

By browsing through the destinations reachable from Barcelona, we find that Easy Jet has flight from Barcelona to Paris and from Paris to Rome. Hence, it is logical to infer that one can fly from Barcelona to Rome by using only Easy Jet's flight. The problem is that not everyone has the patience to look for such kind of route. Aside from frequent flyer or loyal customer of Easy Jet who may know about the existence of this route, common people would consider another airline that advertise this route, possibly taking direct flight from a major airline for which the ticket price is higher than combining two Easy Jet's flight.

There is no entity to be blamed for this result. Low-cost airlines are known for emphasizing only direct flights. They never advertise transit destinations (destinations that can be reached by transiting at other airport), even if it is possible to make connecting flight. They leave it to the travelers to find the existing transit connection by themselves. In the other side, the customer doesn't have time and patience to search for all possible combinations. By giving the departure and destination airport, they want the system to figure out itself all the possible alternatives and present them with the good ones.

We do the same search for Barcelona-Rome route at several online travel agents. The result is that this Easy Jet route is never found. To be noted, we ensure that Easy Jet has flights on the date, for which the transit time is 2 hours and 5 minutes (this transit

time is short so that the route should not be eliminated by the travel agents). This example shows that there is a need for a search engine that can find the transit routes from low-cost airlines.

The second weakness is the concept of *mixing airlines* (for transit routes). Having transit means that the whole journey can be broken into several *flight legs* which are independent one another. Independent means that each leg can be served by any airline (from any alliance). However, rarely we found the flight search engine that uses this concept. There are some search engines that show transit routes composed from different airlines, but it is limited to airlines in the same alliance. This mixing airlines concept is important since it may result in more flexible schedules, more flight alternatives, and possibility of getting cheaper fare.

The third weakness with current flight search engine is on the evaluation and ranking of search results. Most of them simply categorize the best flights are the ones with cheap price and short journey time. This evaluation may fit with most people, however not all travelers are satisfied with this setting.

It needs to be noted that there are different types of travelers. A flight plan that is good for one, may not be good for others. The backpackers, who are tight on budget, may consider that low-cost flights are the best. They don't mind to have very short transit (which may make them rush to catch the connecting flight) or very long transit (which may make them spend overnight at airport), as long as they can get the bargain price and travel with minimum budget. Business travelers may have different preferences. Price is the last constraint of their decision. They prefer to fly with reliable airlines (who have excellent on-time performance), having no or minimum number of transit, and short journey time. There are others, in the middle of these two extremes, who satisfy enough to fly with not-so-comfortable airline and to pay not-so-cheap price, as long as they can arrive at their destination.

It would be nice if we have a search engine that can be customized such that we can search, for example, only from low-cost or reliable airlines. It is also nice if one can mention that his/her tolerance of transit time, for example to find connecting flights with departure time at most 3 hours from the previous leg arrival time. By having this customizable evaluation setting, the engine only looks for transit routes which satisfy this constraint. Hence, each traveler can find best flights of his/her version.

The final motivation of this thesis concerns about the scalability of current flight search engines. As the number of airlines growing in the future, then the number of resources that needs to be considered by the search engine would also grow in direct proportion.

By observing on how current engines perform their search, it raises a question whether these engines are scalable to follow the growing trend of airline industry.

1.2 Contribution

In this thesis, we propose a *mashup* model for a flight search system. Mashup are application that combines data from multiple resources (called *content provider*) to create a new application which provides new and distinct functionality that is not offered by any of the content provider.

The flight search problem is transformed to graph search problem with airports as the nodes and the pair of airports where exists direct flight between them as the edges. The list of nodes and edges is extracted from the Web by using a wrapper generation technology from Lixto Visual Developer. For the edges, we only consider direct routes from the airlines that we want to consider in our search. The wrapped data is put in local repository (database) so that the data retrieval and query processing can be performed faster since all the needed resources reside locally. The query processing part is handled by creating a rule system that governs how the search is performed.

We are only interested in direct routes so that we can freely mix and match which flight and airline to choose for each flight leg. By this way, we have a solution to the *mixing airline* problem. By having direct routes, the system can also infer that if Easy Jet flies from Barcelona to Paris and from Paris to Rome, then it is possible to go from Barcelona to Rome, by Easy Jet, by connecting through Paris. Hence, we can find the transit routes offered by low-cost airlines.

Several leading-edge technologies that are used in this system:

- **Lixto Visual Developer** to extract data from the Web.
- **PostgreSQL** for the database and **PostGIS** as its spatial extension (later in Chapter 2, it will be explained why we need this spatial extension).
- **PLpgSQL**, the proprietary language of PostgreSQL (which is similar to Oracle's PL/SQL) to build the rule system.

By performing the search process locally (not real-time), several issues need to be further explained.

First issue is about the data validity. How can the system assures that the data is up-to-date and that it suggests correct routes (meaning that the flight schedule is exist and

correct if the query is done directly in the respected airline website). We observe that airline schedules are weekly-based, and do not change very often. The schedule has some validity period, for example in Europe, there are different summer (where routes to new summer destinations such as beach are opened) and winter schedule (where routes to the winter destinations such as ski places are opened). Therefore, to provide an up-to-date data, the wrapping has to be performed in a timely basis to preserve the data in the local repository to be always actual.

Second issue is about the ticket price. Ticket price are determined based on many factors such as seat availability, route traffic (fat or thin route), and discount/promotion. Ticket price can change at any time, and it can't be predicted when the change happens. Therefore, ticket price query should be done in real-time. Wrapping ticket price in advance would not be useful since the price when the wrapping is performed with the price when the query is posted may be different. Moreover, to wrap the price data for all possible connections for all possible routes is tedious, if not impossible task.

We define *price index* approach to differentiate between low-cost and major airlines. Price index gives information, not on the real price, but on the likeliness of price, based on the airline. This approach is used based on obeservation that in real world, the price hierarchy between airlines exists. It is more vivid in major airlines. A general observation on Economy-class flights from Europe to Asia, the middle-east airlines such as Emirates, Qatar, and Etihad Airways have cheaper price compared to Lufthansa and KLM. But, Lufthansa and KLM have cheaper price than British Airways and Air France. The price index has range value from 1 to 10 where 1 indicates the cheapest and 10 indicates the most expensive price. One possible implementation is to assign low-cost carriers with price index value from 1 to 5, while for major airlines it is from 6 to 10.

In this system, we also embed the concept of *searching from close airports*. This concept aims to give alternative routes originating from other airports in particular radius from departure airport. To know the distance of an airport to other airports we need *geocoding* technology. This technology is embedded in PostGIS spatial extension of PostgreSQL. Lixto Visual Developer is used to wrap the geographical coordinate (latitude and longitude) of each airport so that the distance can be computed.

In existence of many possibilities to reach a destination, not all routes would give good results. For example, considering possible one-transit routes to fly from Vienna to Frankfurt, then Dubai is a possible transit since from Vienna we can fly to Dubai, and from Dubai we can fly to Frankfurt. However, no one will choose the Vienna-Dubai-Frankfurt route since there exists other transit airports inside Europe that give better time performance and cheaper price.

We define the *hub identification heuristic* for dynamic hub identification. Instead of analyzing and evaluating all possible routes to reach the destination, this heuristic gives hint on which hub airports that are possibly containing the best routes (in terms of shortest flight duration). Hence, the system can limit the search by only considering a fraction from all possible routes. Performing the search this way also ensures system scalability and increases the system's responsiveness by shortening the query procesing time. In Chapter 5, this heuristic will be discussed in detail where the optimality of the heuristic will be shown.

We also define the term *interesting route* for our search results. Interesting route is defined as a list of routes which match with the user's preference. It can be sorted based on three criteria: *time*, *reliability*, and *price* to give users full flexibility on defining which routes are most interesting for them.

1.3 Organization of Thesis

The thesis is organized as follows.

Chapter 1. Introduction. This chapter gives a general overview of the problem domain, describes the motivation, and defines the contribution of this thesis.

Chapter 2. Preliminaries and Background. We begin the chapter by providing an overview of major and low-cost airlines, and their differences. We also evaluate the functionality offered by some flight search engines and discuss the advantages of mixing airlines. The last part of the chapter discuss the geocoding technology, PostGIS, and Maps API.

Chapter 3. Extracting Data from the Web. This chapter exposes some state of the art technologies to extract data from the Web. We begin by showing some problems to pull data from the current Web. Next, Semantic Web vision and some techniques to perform web data extraction is discussed. The last part of the chapter describe the *wrapper generation technology* from Lixto.

Chapter 4. System Design and Data Preparation. This chapter comprises two parts. The system design part focuses on the system architecture and business logic while the data preparation part focuses on the wrapping and data cleaning effort to populate the database. Some wrapping examples using Lixto VD are shown in this chapter.

Chapter 5. Scalable Algorithm. This chapter begins with discussion on the complexity for flight search. The next part discuss the route determination, hub identification heuristics and the approaches that we use to ensure the scalability of our system.

Chapter 6. Experimental Results. This chapter presents step by step tutorial on how to perform the search. It also shows various search features embedded in the system.

Chapter 7. Conclusions. This chapter contains the summary and future works in this area.

Chapter 2

Preliminaries and Backgrounds

2.1 Types of Airlines

There are several types of airline in the airline world today. **Major airlines** or *full-service airlines* are the ones which offer many convenience to their passenger such as single check-in, smooth transit, and advanced baggage handling so that the baggage appears at final destination as if by magic. In the other hand, **low-cost airlines**, also known as *no-frills* or *discount carriers*, are the ones which offer generally low fares in exchange for eliminating many traditional passenger services. A recent article at International Herald Tribune [7] introduces another type, **hybrid carriers** which blend low-cost traits with those of traditional or full-service carriers in the pursuit of business travelers.

The high level of service offered by major airlines are complicated and costly. It requires interline and code share agreements, integrated processes, systems for bookings of connected tickets, and distribution of revenues. Checking passengers and baggage through to final destination calls for baggage handling services at each stop. All of these add up to higher ticket prices. There is nothing low-cost about the full-service connection.

Low-cost airlines tend to focus on short haul routes (of generally less than 1,500 km). To achieve the low operating costs, this type of carrier needs to have as many seats on its aircraft as possible, to fill them as much as possible, and to fly the aircraft as often as possible. They usually use uncongested secondary airports and not offering anything other than point-to-point services. Significant cost savings can be made by selling directly to customers via the Internet and call centres and by using electronic ticketing. By not selling via travel agents, low cost airlines avoid travel agency commissions and also avoid computer reservation system fees.

Low-cost airlines generate their revenue from selling various services to passengers such as *travel insurance* (for those without personal travel insurance), *express/priority boarding* (for those who want hassle-free boarding before the other passengers, to avoid the boarding crowds and choose the seat of their preference in case of non-seated flight where passengers may occupy any seat). They also charge for the foods and drinks offered on board. Some discounts may be given if the meal is ordered prior to the flight, for example when booking the flight.

Figure 2.1 displays the screenshot of an order page from a low-cost airline, showing multiple fee components that constitute the total fare. Sometimes, the sum of the other fees may be higher than the ticket price itself. Figure 2.2 displays the screenshot from another low-cost airline which offers lounge and seat assignment service.

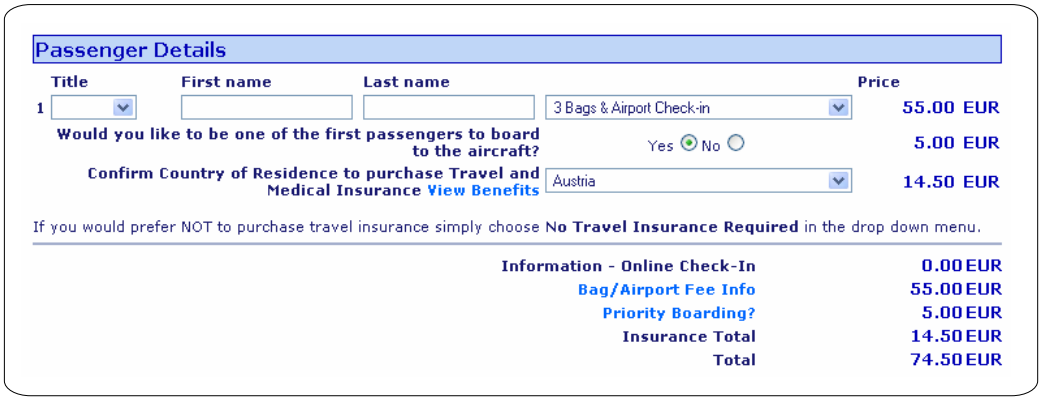


FIGURE 2.1: Order screen of a low-cost carrier, showing the total fare components

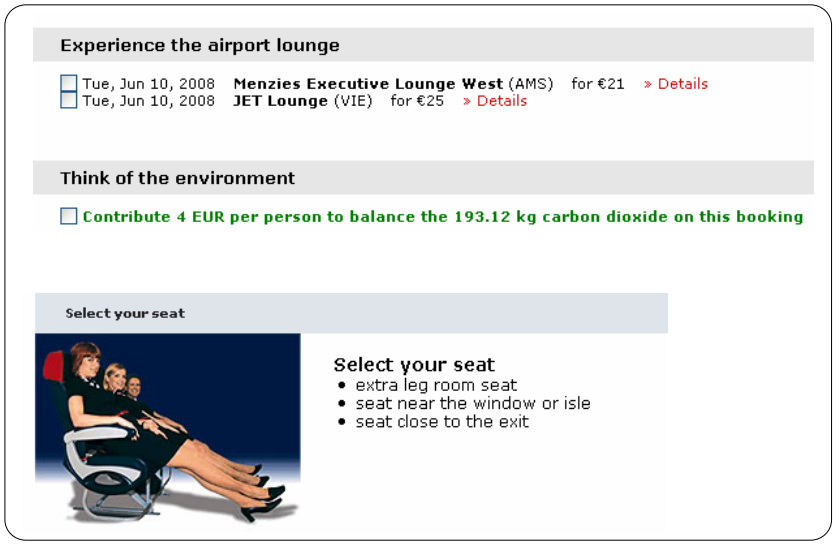


FIGURE 2.2: Lounge and seat assignment service offered by a low-cost carrier

Some comparisons between major airlines and low-cost airlines are shown below.

Price. The ticket price of major airlines is usually multiple times of low-cost airlines for the same route.

Network Topology. Major airlines use *hub-and-spoke* while low-cost airlines use *point-to-point* (direct flights). Therefore, when delay happens and the passenger misses the connecting flight, usually there is no compensation from low-cost airlines since they sold the flights separately, not as a package of several point-to-point connections.

Baggage Handling. Major airlines offers *check-in through*, for which the baggage is only checked-in once at departure. The baggage is handled by the airline so that it is routed to the final destination without passenger needs to collect it at each stops. Low-cost airlines emphasize the use of direct flights. Baggage is not automatically transferred from one flight to another, even if both flights are from the same airline company.

Passenger Class. Major airlines have different passenger class, such as Economy, Business, and First class. Low-cost airlines only have single passenger class, but passengers may have different services (depends on the extra services that the passengers purchase).

Seat Assignments. Major airlines have seat assignments and passenger may choose their seat on check-in for free. Low-cost airlines usually do not have seat assignments, but passengers who want to sit in a particular seat can purchase for reserved seat, by paying some extra amount.

Booking process. Major airlines sell their tickets through various channels, such as travel agents, city counter, and airline's website. Low-cost airlines emphasize direct booking, either through Internet or call center.

Airports and Flight Time. Major airlines use major airports and operates in various time of the day. Low-cost airlines usually only use secondary or less-congested airports and fly in early morning or late night to avoid air traffic delays at day time due to congestion. Secondary airports tend to charge airlines less for using their services. Since they are less busy, delays due to congestion are less.

Aircraft type. Major airlines use various types of aircraft, possibly from several manufacturers. Low-cost airlines use single aircraft type for some reasons. First, pilots and cabin crew can operate on any aircraft in the fleet, reducing training cost of the crew. Second, they can do bulk purchase of aircraft components at lower price, reducing the maintenance cost of the aircraft.

2.2 Several Types of Flight Search Engine

Based on its business purpose, there are two types of flight search engine: independent and commercial. **Independent flight search engines** do not sell tickets. They merely perform the search, then redirect potential clients to the respected airline website to continue with booking process. Some examples of independent search engine are Skyscanner[1], Momondo[2], and Dohop[3]. **Commercial engines** are usually owned by travel agents which have agreements with the airlines to allow clients to book directly in their website. Some examples of commercial search engine are Opodo[4], Expedia[5], Travelocity[6], and Orbitz[7].

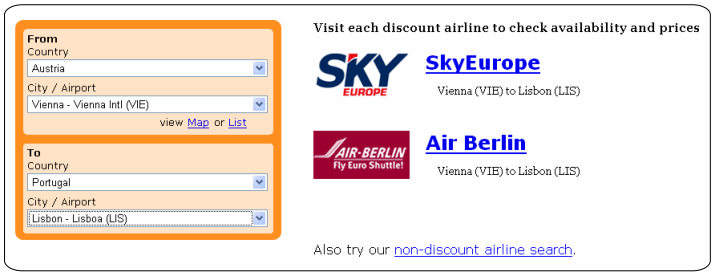


FIGURE 2.3: Screenshot of AirNinja

FIGURE 2.4: Screenshot of Low Cost Airline Guide

There are numerous number of low-cost airlines nowadays. The names such as CoastAir[8], Corendon[9], and Sterling[10] may never be heard by most people. Even if they know that those names correlates with airline, then which routes served by the airline may not be a common knowledge.

[1] `http://www.skyscanner.net`
[2] `http://www.momondo.com`
[3] `http://www.dohop.com`
[4] `http://www.opodo.com`
[5] `http://www.expedia.com`
[6] `http://www.travelocity.com`
[7] `http://www.orbitz.com`
[8] `http://www.coastair.no`
[9] `http://www.corendon.com`
[10] `http://www.sterling.dk`

There exists search engines which are built to help people to find which low-cost airlines serves the route of their journey. Its input is simply the departure and destination airport. It does not provide any information on flight's availability, schedule, or price. Figure 2.3 and 2.4 shows the screen from AirNinja[11] and Low-Cost-Airline-Guide[12] for routes from Vienna to Lisbon.

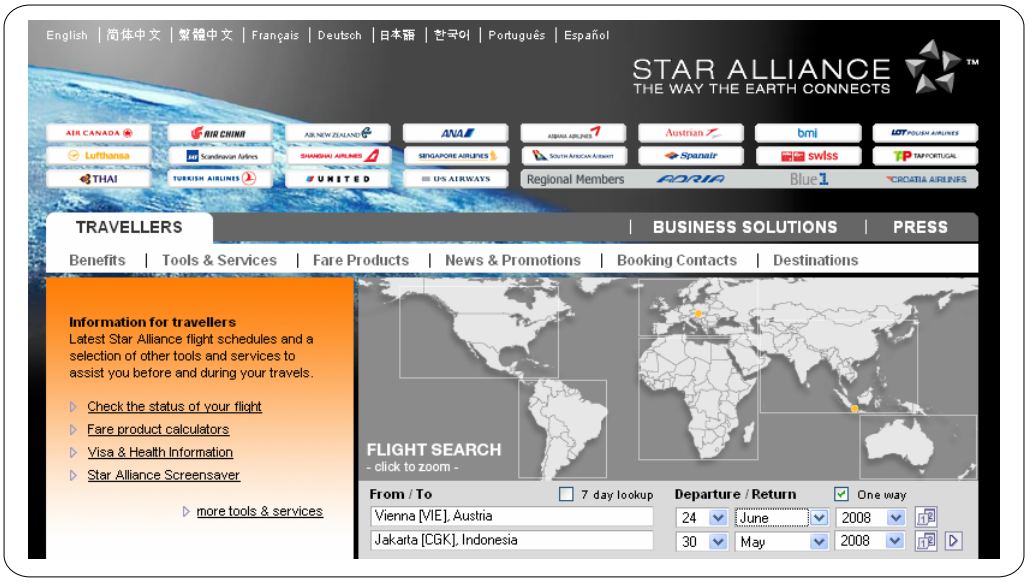


FIGURE 2.5: Star Alliance search page

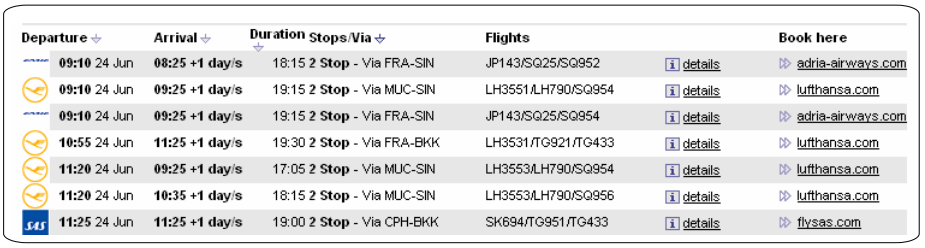


FIGURE 2.6: Star Alliance search results, showing offered flights from the alliance airlines: Lufthansa, Adria Airways, and SAS Scandinavian

Every airlines and airline alliances website usually have search engine that only searches for flights that are served by themselves and codeshare flights that are served together with, or solely by their partner/alliance. The search process is fast and the prices are correct because all data are retrieved directly from their back-end system. Figure 2.5 and 2.6 shows the search page and search results from Star Alliance, an alliance formed by Lufthansa, Austrian Airlines, Singapore Airlines, United Airlines, and several other airlines.

[11] `http://www.airninja.com`
[12] `http://www.low-cost-airline-guide.com`

A **flight meta-search engine** is a flight search engine that sends user query to several other low-cost carriers, major airlines, and online travel agents. The results are aggregated into a single list and displayed according to their source. Meta-search engine can generate more comprehensive search results from several sources and save user's time from having to search in multiple engines separately. Most flight search engines that are currently available in the Internet can be categorized into this type. Figure 2.7 shows how flight meta-search engine works.

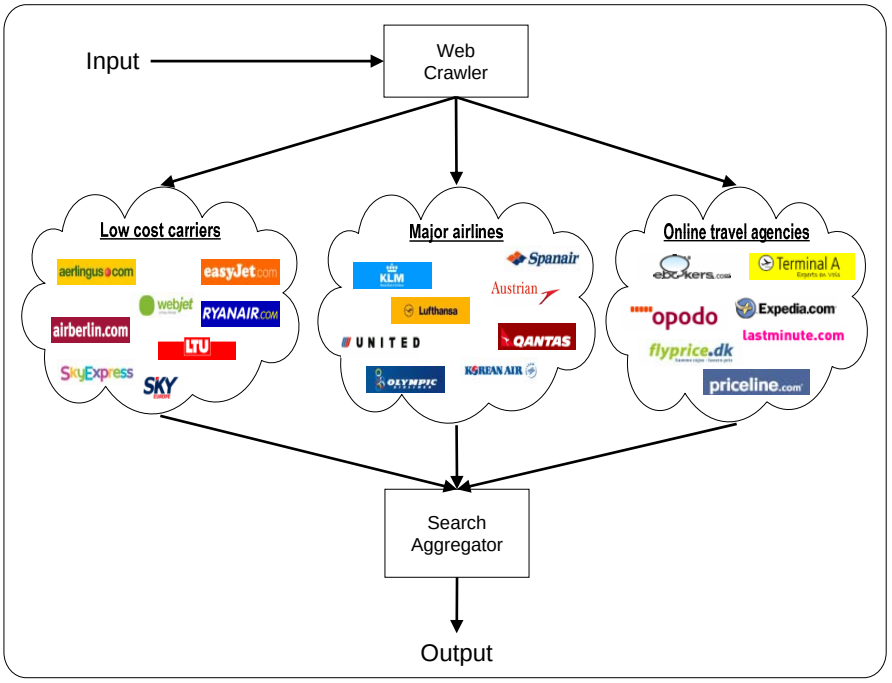


FIGURE 2.7: Working diagram of flight meta-search engine

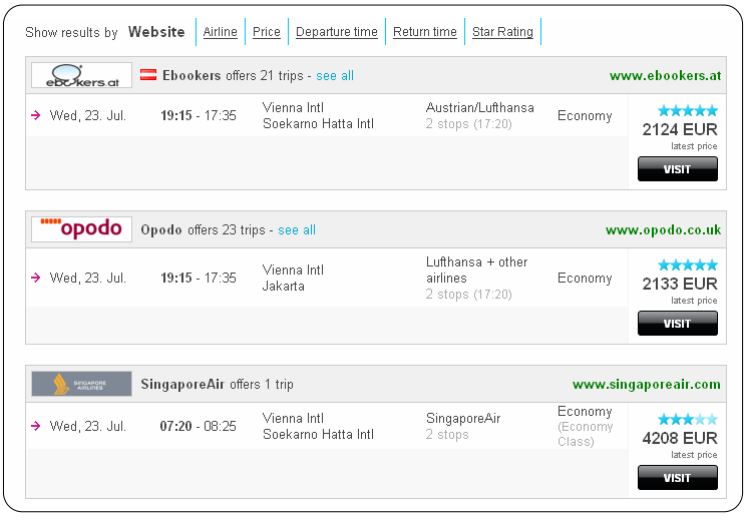
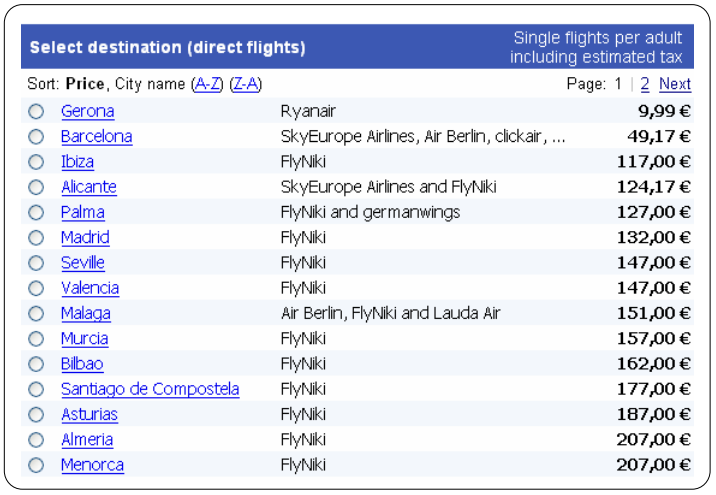


FIGURE 2.8: Screenshot of Momondo showing the websites from where the price is obtained

Figure 2.8 shows a search result from Momondo, a travel search engine that claims to search 492 travel sites (as of 10 June 2008) at once and find the best fares. Momondo does their search by using web-crawlers which need to fill in information (departure, destination, departure dates) in order to access information (timetables and prices).

From all the search engines that we discuss above, they have one thing in common, that is the departure date has to be fixed. For travelers with flexible date, they may want to check the price for different departure dates so that they can go on the date when lowest price is offered. Skyscanner is built for this purpose. Not only the dates are flexible (during whole month or whole year), but the destination are also flexible. User can check the prices for many destinations in a country. Figure 2.9 and 2.10 illustrates this idea.



Select destination (direct flights)		Single flights per adult including estimated tax
Sort: **Price**, City name (A-Z) (Z-A)		Page: 1 | 2 Next
Gerona	Ryanair	**9,99 €**
Barcelona	SkyEurope Airlines, Air Berlin, clickair, ...	**49,17 €**
Ibiza	FlyNiki	**117,00 €**
Alicante	SkyEurope Airlines and FlyNiki	**124,17 €**
Palma	FlyNiki and germanwings	**127,00 €**
Madrid	FlyNiki	**132,00 €**
Seville	FlyNiki	**147,00 €**
Valencia	FlyNiki	**147,00 €**
Malaga	Air Berlin, FlyNiki and Lauda Air	**151,00 €**
Murcia	FlyNiki	**157,00 €**
Bilbao	FlyNiki	**162,00 €**
Santiago de Compostela	FlyNiki	**177,00 €**
Asturias	FlyNiki	**187,00 €**
Almeria	FlyNiki	**207,00 €**
Menorca	FlyNiki	**207,00 €**

FIGURE 2.9: Skyscanner results for flight from Vienna to multiple destinations in Spain during July 2008

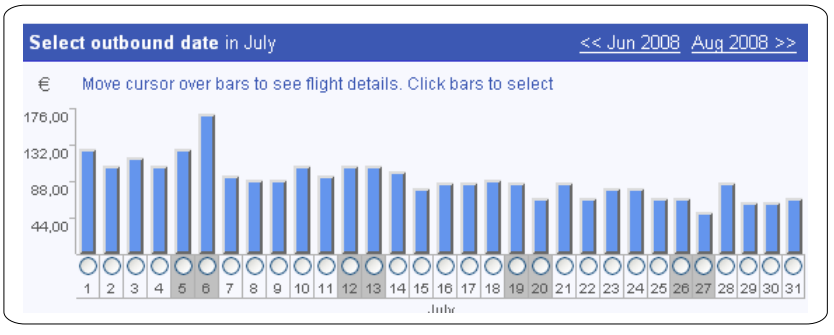


FIGURE 2.10: Skyscanner results for flights from Vienna to Barcelona during July 2008

2.3 Advantages of Mixing Airlines

Low-cost carriers only focus their business in certain regions. They do not cover a large area of operations as major airlines. A research that was carried out by Dohop Analytic, reveals that adding one transit to low-cost routes brings a notable increase to the number of additional reachable airports.

Table 2.1 shows the increase for four low-cost airlines, by connecting with another airline with less than 4 hours in transit, respecting minimum connection times for each airport. The travel period for this research is first week of September 2007.

TABLE 2.1: Number of reachable airports of by adding one-stop (table is reproduced from [8], with the addition of last column)

Airline	Direct Airports	Addition (one stop)	Scale factor
Ryan Air	170	311	1.83
Easy Jet	70	550	7.86
Iceland Express	9	145	16.11
German Wings	65	549	8.45

Interline travel is simply vital to the global traveler. Connections allow a much wider choice and flexibility in terms of airlines and schedules. Departing from London Luton airport, travelers can choose from 66 airports using non-stop flights but 587 additional ones via one connecting flight [8]. If all European low-cost airlines would connect with other airlines, it would improve the total efficiency air travel significantly. It could reduce the average duration of journeys due to less waiting in airports and shorter flights.

Another possible advantages of mixing airlines is to get cheaper fare. To illustrate this idea, we search for flights from Vienna to Jakarta through Frankfurt on a particular date. For Vienna-Frankfurt route, there are several flights from major and low-cost airlines such as by Austrian Airlines, Lufthansa, Adria Airways, and Niki. For Frankfurt-Jakarta route, the only direct flight is by Lufthansa.

Based on the route findings, then it is clear that the price is determined by the first leg flight since there is only one choice for the second leg (which is Lufthansa with price €2400). We check the price of each airline serving the first leg route and get the result as follows. Lufthansa and Austrian Airlines offer price around €600. Adria Airways offers €150. But, Niki, only offers €50. Therefore by combining Niki-Lufthansa, the traveler can earn €550 savings (compared to completely using Lufthansa for the whole journey), a deal that doesn't need to be thought twice by budget travelers.

2.4 Geocoding

Each airport has a coordinate, expressed as latitude and longitude, in earth geographical system. The coordinate can be mapped to see the location of each airport to the world map as in Figure 2.11.

We can calculate distance between two points in the earth, given their coordinates. Earth surface distance is defined as the shortest distance between any two points on the surface of a sphere measured along a path on the surface of the sphere (as opposed to going through the sphere's interior). Figure 2.12 shows the great-circle distance between point p and q with red line. A method to calculate earth surface distance is by using great-circle distance method [31].

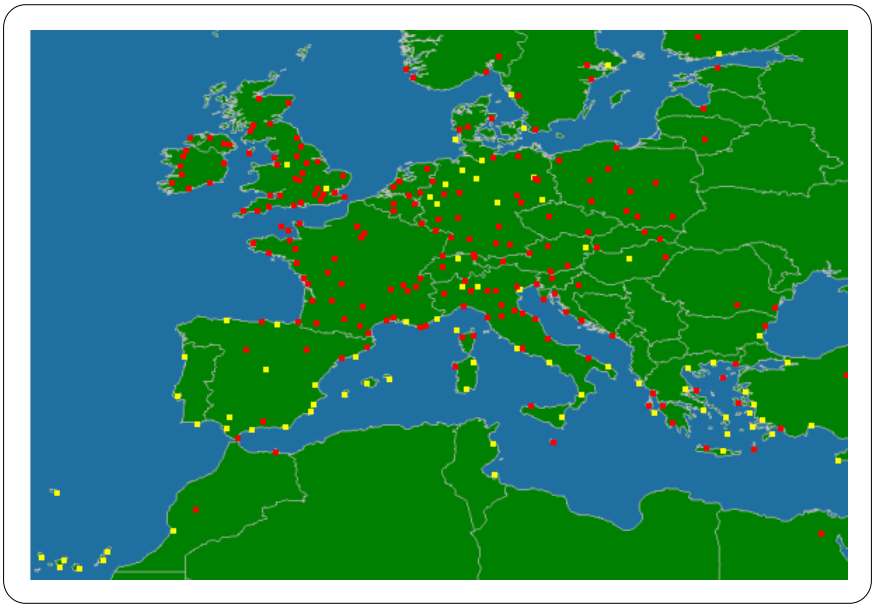


FIGURE 2.11: Map of European airports

Spherical geometry is different from ordinary Euclidean geometry. The distance between two points in Euclidean space is the length of a straight line from one point to the other. On earth sphere, there are no straight lines. In non-Euclidean geometry, straight lines are replaced with *geodesics*. Geodesics on the sphere are the great circles (circles on the sphere whose centers are coincident with the center of the sphere).

Between any two points on a sphere which are not directly opposite each other, there is a unique great circle. The two points separate the great circle into two arcs. The length of the shorter arc is the great-circle distance between the points. Between two points which are directly opposite each other, called antipodal points, there are infinitely many great circles, but all great circle arcs between antipodal points have the same length, that is half the circumference of the circle, or πr, where r is the radius of the sphere.

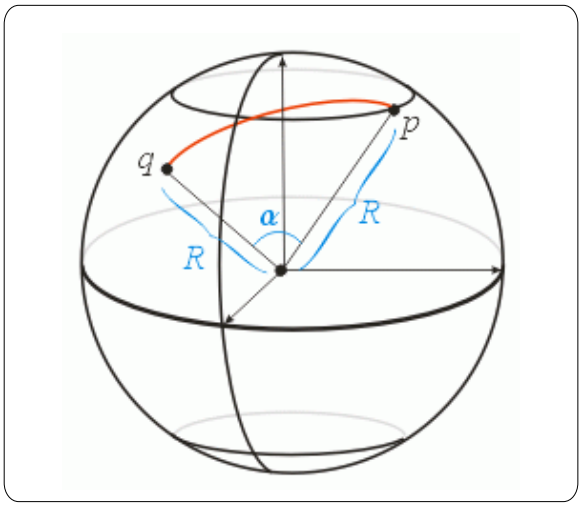


FIGURE 2.12: Great-circle distance

One use of geocoding in this thesis is to calculate distance between airports. This distance is useful to know which other airports located in particular radius from an airport. Therefore, we can implement the *search from close airports* feature, which is useful for the following purposes:

- Flight availability.
  In high/peak season, it is very useful to consider flights from nearby airports if all flights from the preferred departure airport is fully-booked.

- Lower fare.
  A less-congested airport may have alternative flights with lower fare. This is due to the fact that most of low-cost airlines use secondary airport. A comment from a reader in an online article [27] reveals this fact. The reader searched for flights from Orlando, Florida to Copenhagen, and it costs $ 1100 with three stops. But, if he flew from Stanford, Florida (located 20 miles from Orlando airport) to the same destination, there exist a flight plan with only two stops with price under $ 700.

- Existence of better routes.
  Currently, there is no direct flight serving route from London Stansted to Lisbon. By includeing other airports in radius of 100 km from Stansted, the system can find London Gatwick (96 km), London Heathrow (73 km), and London Luton (46 km), from where there exists direct flight to Lisbon.

2.5 PostGIS

PostGIS is an open source spatial extension to PostgreSQL relational database. PostGIS adds support for geographic objects to the PostgreSQL object-relational database. It is developed by Refractions Research, a company from British Columbia, Canada as a project in open source spatial database technology. The architecture of PostGIS is shown in Figure 2.13.

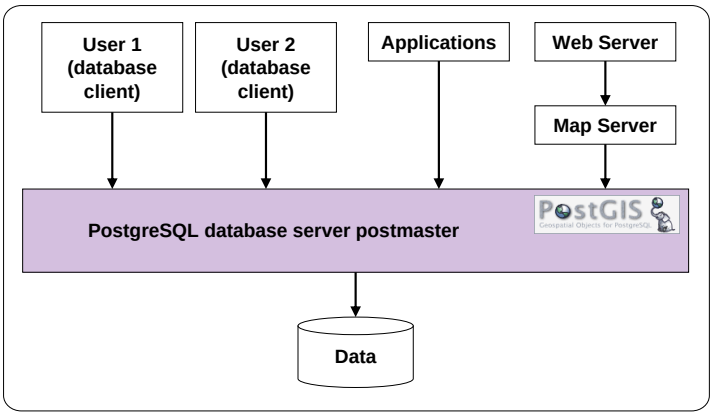


FIGURE 2.13: PostGIS extension in PostgreSQL relational database

In effect, PostGIS spatially enables the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS) like Oracle Spatial extension. For managing large volumes of read/write spatial data, using a spatial database can improve access speed, ease management overhead and guarantee data integrity.

PostGIS has been certified as "Simple Features for SQL" compliant by the Open Geospatial Consortium. PostGIS was first released in 2001, and is now used around the world as a high-performance server for spatial objects. It features a spatially-enabled query planner, highly concurrent R-Tree spatial index, and hundreds of spatial analysis and processing functions that allow for GIS-style data analysis right inside the database.

2.6 Maps API

As part of our customizable system approach, we let user to choose the routes from which they want to perform the search. This feature may be useful if user wants to avoid some airports possibly due to transit visa regulations, airport comfortability, or merely personal preferences. However if the user doesn't have ample geographical knowledge, the user may get lost and choose transit airports that is far away from the destination.

By displaying the route map, the user can get hints on the journey's path before making their decision.

Currently, there are two free mapping API providers that can be utilized, Google Maps API and Yahoo! Map. There are some differences between them which are listed below.

Geocoding support. Yahoo provides geocoding support while Google accept only latitude and longitude. The Geocoding Web Service from Yahoo allows user to find the specific latitude and longitude for an address.

Map generation. Google Map can be embedded in any website, while for Yahoo, the map is only generated in Yahoo site.

Technology. Google Map is based on JavaScript and support Ajax. Yahoo Map use XML (based on geoRSS 2.0) API and does not support Ajax.

Purpose and usage. Google Map can be used for commercial purposes, but should be freely available to end user. Usage is not restricted up to an upper bound. Yahoo Map can be used for commercial purposes, but should obtain written permission, no usage restriction.

We choose to use Google Map API so that the map can be embedded in the local web server. Since Google does not provide geocoding, then it needs to be supplied with latitude and longitude of the airports, which is obtained through web data extraction. Figure 2.14 shows the example of possible routes from Barcelona to Berlin using Google Map API.

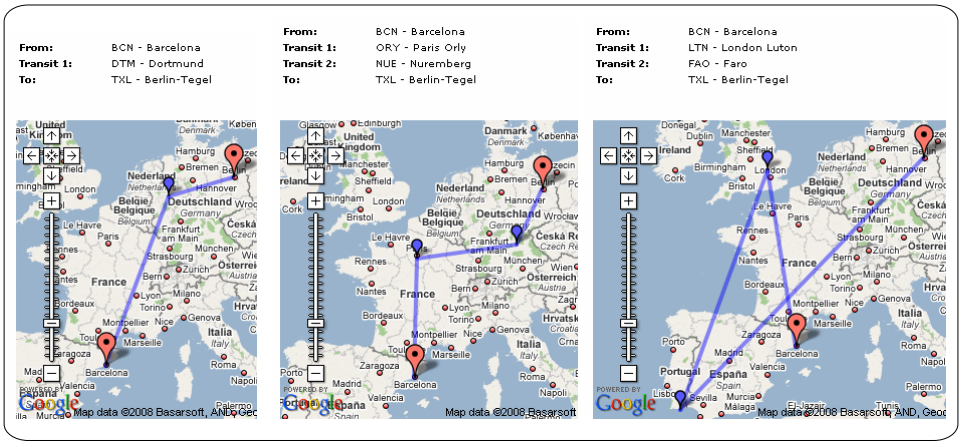


FIGURE 2.14: Google Map example for possible routes from Barcelona to Berlin

Chapter 3

Extracting Data from the Web

3.1 The Problems with the Web

Web is the biggest database in the world. Billions of diverse documents are put online. However, it is unstructured and lacks of query techniques to pull data from it. Below, we present several problems that are faced by the current Web.

The first problem is on **retrieving documents**. For example, we are interested to know the price of Jurassic Park book by Michael Crichton. Posing this query in Google, we get the results as in Figure 3.1. The results doesn't show exactly which link provides the price information.

FIGURE 3.1: Google search results for Jurassic Park book price

The second problem is on **extracting information**. Figure 3.2 illustrates this problem by searching for books about "Web" in Amazon[1]. We take the top four search results and analyze them. There are several prices for each book, so it is not clear which price is exactly the right one. The first three books have the right context, but the fourth book is not about Web. The computer can't be blamed for displaying the fourth book. It does not know what is the semantic of "Web" about.

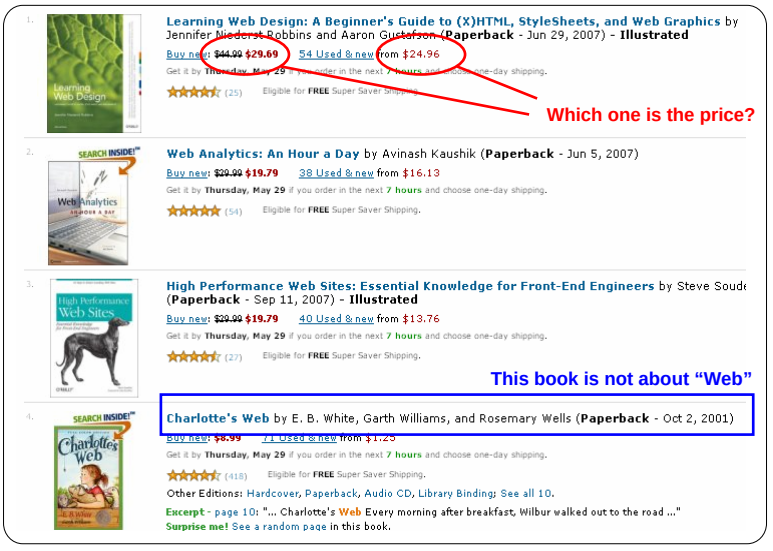


FIGURE 3.2: Amazon search results for books about "Web"

The third problem is on **combining/aggregating information**. Figure 3.3 shows two search results for "Semantic Web Primer" book from Amazon and Barnes&Noble[2] (BN). Human eyes can quickly capture the picture of the book cover and infer that the two books are the same. However, for computers, pictures is just a sequence of bytes without semantics. The book title is different. BN includes ISBN number of the book, while Amazon does not. Hence, it is difficult for computer to decide whether this two books corresponds to same book.

How if the user wants to know which store has the cheapest price, which price should be used for comparison? There exist different price schemas. The computer does not have basis to determine the price for this user since there is no information on which kind of user performs this search (member or non member) and what preference does he/she has (new or used books). A more difficult question is, which store has the cheapest total price, including shipping charge? This question can't be answered without employing *deep web navigation* technique to retrieve the shipping charge.

[1] `http://www.amazon.com`
[2] `http://www.barnesandnoble.com/`

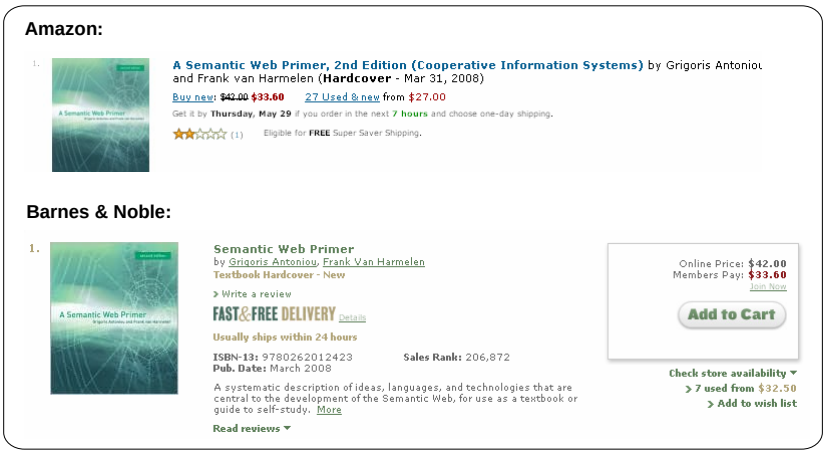


FIGURE 3.3: Difficulties of aggregating search results

3.2 Semantic Web

The Semantic Web vision was conceived by Tim Berners-Lee, the inventor of the World Wide Web. It can be simply defined as the Web with a meaning. The idea of representing information in structured form so that computers can understand it and then solve complex problems was one of the keystones of the Semantic Web vision.

- SkyEurope has a flight with number NE3612.
- Flight NE3612 flies from Vienna to Amsterdam.
- Flight NE3612 departs at 6:30 and arrives at 8:25.

Statements such as above are easy to be understood by people. But how can they be understood by computers? This is what Semantic Web is all about. Describing things in a way that computers applications can understand. Semantic Web describes the relationships between things (like A is a part of B and Y is a member of Z) and the properties of things (like size, weight, age, and price)

The Semantic Web is an evolving extension of the World Wide Web in which the semantics of information and services on the web is defined, making it possible for the Web to understand and satisfy the requests of people and machines to use the Web content. Information is stored in a machine-readable format so that computers able to handle information in more useful ways by processing the meanings within documents instead of simply the documents themselves. Provided with the data semantics, then computers can find, extract, share, re-use information, and potentially even reason with

it. In the future, it will realize the Web as a universal medium for data, information, and knowledge exchange.

With Semantic Web, the information is structured, but it does not mean that the computer can necessarily solve complex problems. These are two completely different things. Just because you have a map, does not mean that you know the best way to get from point A to point B. Having a map is necessary, but it is not sufficient, you need the algorithm to find the best path. There is a big difference between asking what is the capital of France and what is the cheapest airfare today to fly from New York to Paris.

The goal of Semantic Web is to collect data in a useful way, like a large database. Semantic Web will allow businesses to manipulate external, heterogeneous Web data in much the same way they do internally. Its most immediate use may be as a tool to solve data integration problems. It relies on structured sets of information and inference rules that allow it to understand the relationship between different data resources. The computer doesnt really understand information the way a human can, but it has enough information to make logical connections and decisions.

The Semantic Web would allow manipulation across multiple, heterogeneous databases. This capability could, for instance, allow an electronic airline reservation service to automatically interact with a personal calendar program to arrange a flight that fits a user's schedule, even if there was no pre-established interface between the two pieces of software.

Some elements of the Semantic Web include Resource Description Framework (RDF), a variety of data interchange formats (e.g. RDF/XML, N3, Turtle, N-Triples), and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain. The Semantic Web is a simple, but potentially poweful idea. Just as the Web was implemented using URLs, HTTP and HTML, the Semantic Web is built with URIs, HTTP and RDF.

3.2.1 Resource Description Framework (RDF)

An official W3C recommendation, RDF is an XML-based standard for describing resources that exist on the Web, intranets, and extranets. RDF builds on existing XML and Uniform Resource Identifier (URI) technologies. URI is an identifier for resources, and not location on the Web. It is not necessarily to be started with "http". For a book, the suitable URI perhaps is in the form of ISBN number.

RDF documents are complicated and hard to be read by human without the help from RDF parser software. To tackle this issue, triple data model is constructed as `<subject, predicate, object>` to represent the same data in RDF in simple notations which is more human-readable. A subject can be formed by a resource (identified by a URI) or blank node. Predicate corresponds to property. Object can be a resource, blank node, or literal value.

Figure 3.4 shows an example of RDF for airline's schedule which expresses the following facts:

- SkyEurope's flight number NE3612 flying from Vienna to Amsterdam, departs on 6:30 and arrives at 8:25.
- SkyEurope's flight number NE3613 flying from Amsterdam to Vienna, departs on 8:55 and arrives at 10:55.
- KLM's flight number KL 1849 flying from Amsterdam to Vienna, departs on 20:30 and arrives at 22:20.

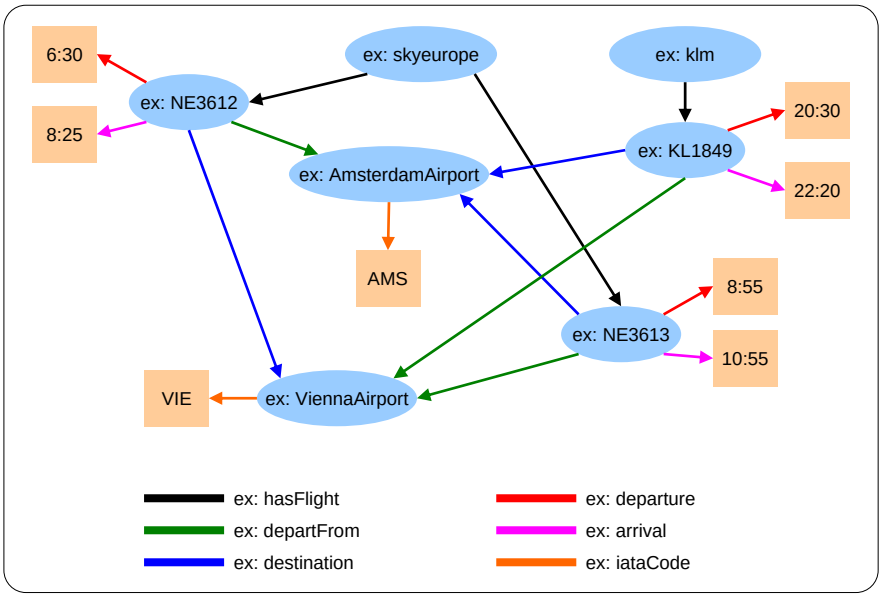


FIGURE 3.4: RDF example for airline's schedule

By creating triples with subjects, predicates, and objects, RDF allows machines to make logical assertions based on the associations between subjects and objects. And since RDF uses URIs to identify resources, each resource is tied to a unique definition available on the Web. However, while RDF provides a model and syntax (the rules that specify the elements of a sentence) for describing resources, it does not specify the semantics (the meaning) of the resources. To truly define semantics, we need RDFS and OWL.

3.2.2 RDF Schema (RDFS)

RDFS is used to create vocabularies that describe groups of related RDF resources and the relationships between those resources. From previous example, using RDFS, we can say that "Sky Europe" has type of "Airline" and "NE3612" has type of "FlightNumber". Further, we can restrict that "hasFlight" property has domain of class "Airlines" and range of class "FlightNumber".

Using the same triples paradigm defined by RDF, RDFS triples consist of classes, class properties, and values that define the classes and relationships between the resources within a particular domain. In an RDFS vocabulary, resources are defined as instances of classes. A class is a resource, and any class can be a subclass of another. This hierarchical semantic information is what allows machines to determine the meanings of resources based on their properties and classes.

Overall, RDFS is a simple vocabulary language for expressing the relationships between resources. Building upon RFDS is OWL, which is a much richer, more expressive vocabulary for defining Semantic Web ontologies.

3.2.3 Web Ontology Language (OWL)

OWL is the third W3C specification for creating Semantic Web applications. Building upon RDF and RDFS, OWL defines the types of relationships that can be expressed in RDF using an XML vocabulary to indicate the hierarchies and relationships between different resources. In fact, this is the very definition of ontology in the context of the Semantic Web: a schema that formally defines the hierarchies and relationships between different resources. Semantic Web ontologies consist of a taxonomy and a set of inference rules from which machines can make logical conclusions.

Since taxonomies (systems of classification) express the hierarchical relationships that exist between resources, we can use OWL to assign properties to classes of resources and allow their subclasses to inherit the same properties. OWL also utilizes the XML Schema datatypes and supports class axioms such as subClassOf, disjointWith, etc., and class descriptions such as unionOf, intersectionOf, etc. Many other advanced concepts are included in OWL, making it the richest standard ontology description language available today.

All the detailed relationship information defined in an OWL ontology allows applications to make logical deductions. Its important to note that OWL has three sub languages, each with increasing complexity: OWL Lite, OWL DL, and OWL Full. OWL DL

includes OWL Lite, and OWL Full includes OWL DL and OWL Lite. Developers choose which OWL dialect to use based on the level of complexity and level of detail required by their semantic model.

When RDF resource descriptions are associated with an ontology defined somewhere on the Web, intranet, or extranet, its possible for machines to retrieve the semantic information associated with each resource. Its in this way that URIs, XML, RDF, RDFS, and OWL combine to make the Semantic Web a reality.

3.2.4 Challenges of Semantic Web

Semantic Web is not a very fast growing technology. Since the beginning, it has been associated with artificial intelligence. It was developed by people with academic background in logic and artificial intelligence. For traditional developers it is not very easy to understand. The problem for representing billions of existing web documents as RDF is a rather daunting, if not impossible task.

A more difficult aspect of building Semantic Web is the creation of ontologies. This process requires efforts by diverse communities, such as the medical, insurance and finance industries, to develop common vocabularies that systems will use to recognize what's in a Web document. Fortunately, creating ontologies doesn't require a global coordinated effort. If words are used differently, such as "title" in insurance vs. the "title" of a book, services will be able to map those differences to allow interoperability.

3.3 Techniques for Web Data Extraction

The Semantic Web that we just discussed, in practical, can still be considered as merely a vision. There are some progress into realizing it, but in general we have not been able to pull data, such as airline schedules by using Semantic Web. Therefore, in this section we discuss about some other techniques for web data extraction.

We need to differentiate between **data discovery** (retrieving documents) and **data extraction** (extracting information). Data discovery deals with navigating a web site to arrive at the pages containing the data you want, and data extraction deals with actually pulling that data off.

A simple data discovery might be as simple as requesting a single URL. For example, going to the homepage of a news site and extract out the latest news headlines. However, in real-life scenarios, most of the time the case is not that simple. Password-protected

sites, cookies, JavaScript, Session IDs, Web forms iterations, traversing series of pages, following all the detail links, and dynamic changes on websites are the obstacles.

In the data extraction phase, we already arrived at the page containing the data, so that the only thing left is to pull it out of the HTML. The simple extraction can involve creating a series of regular expressions that match the pieces of the page we want (e.g., URLs and link titles). However, regular expressions can be a bit complex to deal with. Screen scraper can simplify the process by hiding most details behind the scenes.

Regular expressions is an easy approach when the number of page to be scraped is small and all the data is contained in one page. Regular expressions are supported in most programming languages, even in VBScript. Various implementations of it don't vary too much in their syntax. For someone that has been familiar with a programming language, then regular expressions can be a quick solution to wrap a Web page. An example which shows regular expression extraction method using PHP is the Yellowpages Scraper Tutorial[3].

Screen scraping is a technique in which a computer program extracts data from the display output of another program. The program doing the scraping is called a screen scraper. The output being scraped is intended for final display to a human user, rather than as input to another program, and is therefore usually neither documented nor structured for convenient parsing. Screen scraping is generally considered an ad-hoc, inelegant technique, often used only as a last-resort when no other mechanism is available. Aside from the higher programming and processing overhead, output displays intended for human consumption often change structure frequently. Humans can cope with this easily, but computer programs will often crash or produce incorrect results.

Compared to regular expression, screen scraping abstracts the most complicated stuff away. User can do some pretty sophisticated things in most screen scraping applications without knowing anything about regular expressions, HTTP, or cookies. Once the user master a particular a screen scraping applications, the amount of time required to scrap a site is dramatically reduced.

3.4 Lixto Visual Developer

Lixto Visual Developer (VD) is a wrapping software tool which is developed by Lixto GmbH[4]. Lixto VD allows its user to define wrappers, which visually access data in a structured way, as well as configuring the necessary web connectors.

[3] `http://www.scrapingpages.com/`
[4] `http://www.lixto.com`

A *wrapper* is understood as a program that allows for automatic and flexible extraction of information from regular documents such as Web pages. The process of creating a data extraction program, e.g. generating XML based on relevant data taken from HTML is usually referred to as *wrapper generation* [4]. Lixto VD is a methodology and tool for visual and interactive wrapper generation. It allows wrapper designers to create so-called XML companions to HTML pages in a supervised way.

Lixto VD is a developer tool for daily working business concerns. It provides businesses with effective, user-friendly, and time critically viable wrapping, integration, and delivery of information all in the same product. Such a combination of information processing is called a service in the VD and corresponds conceptually to a particular solution.

The Lixto VD allows a Service Designer to create services that wrap his desired information from multiple sources into a defined data model. With Lixto VD, web data extraction programs can be developed interactively. The user only needs to markup relevant parts of a web page and the program will automatically generate generalized extraction rules that identify and extract the relevant content from the web page. This method is very intuitive and increase productivity.

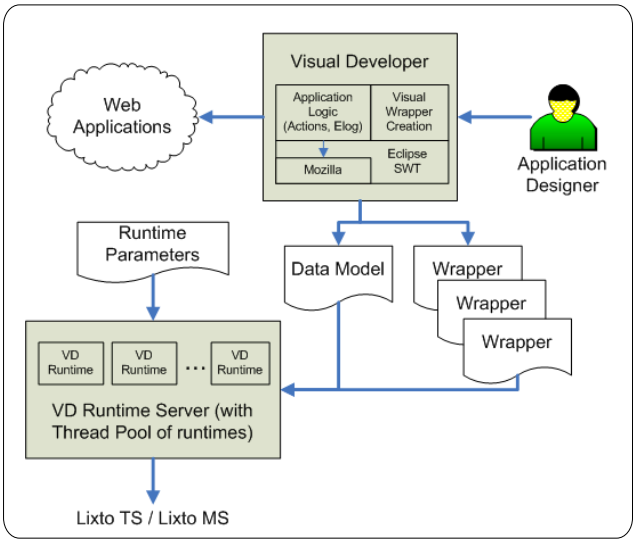


FIGURE 3.5: Lixto VD services overview [20]

3.4.1 Editor and Navigation

VD uses the Eclipse IDE as a framework and the Mozilla browser. Both of them are both well recognized as standard and guarantee a sound basis for the further development of the Lixto VD. A navigation is a consumer oriented human-computer interaction, for which a sequence of human input commands such as mouse clicks, keyboard entry is

recorded by the VD and can be replayed at any given time to reproduce the desired information. Recording a navigation is especially useful since some data on the Web does not have an adress (URL), but requires a series of actions to access it. Figure 3.6 shows parts of Lixto VD main editor.

FIGURE 3.6: Main screen of Lixto VD

3.4.2 Data Model

Before doing wrapping, the wrapper designer should already know what kind of data he wants and in what structure. In Lixto VD, the data model corresponds to the structure of XML output. Figure 3.7 shows an example of data model for wrapping airports data and its corresponding XML output. For merging data from different sources, using data model is a good approach for integrating data.

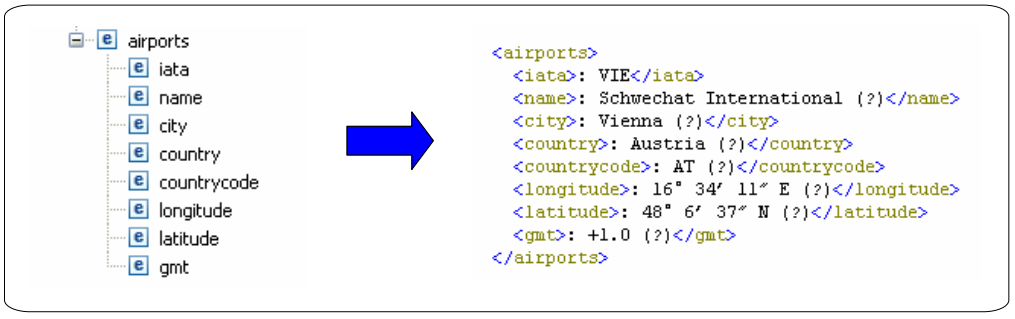


FIGURE 3.7: Example of data model

3.4.3 Patterns and Filters

To extract data, the position of the data or group of data needs to be located. Patterns can be regarded as the names for data and filters are the procedures to get the data. In figure 3.8, the patterns are `news`, `title`, and `links`. The filter are the XPath expression shown in the bottom of the figure. The filter characterize the location for `news`. Lixto VD provides four kinds of filtering method: XPath, Text, Script, and Tokenization.

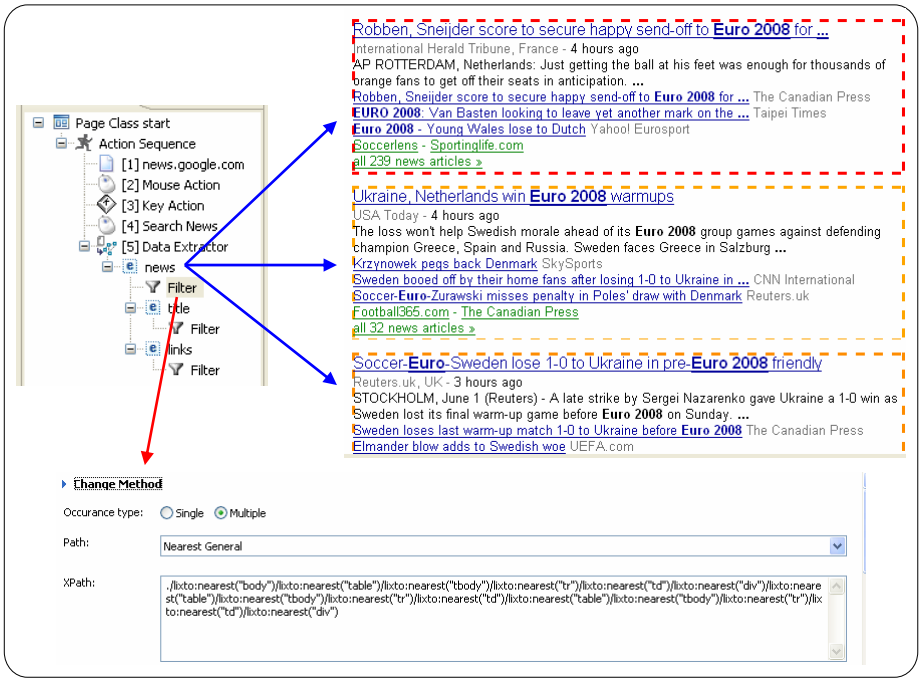


FIGURE 3.8: Using patterns and filters in Lixto VD

Chapter 4

System Design and Data Preparation

4.1 System Architecture

The following client-server architecture (Figure 4.1) is used for the flight search system. Clients access the application through Apache Web Server which is connected to the back-end PostgreSQL database. The route map in the application is embedded using JavaScript so that clients are also connected to Google Map server and retrieve the map directly from there.

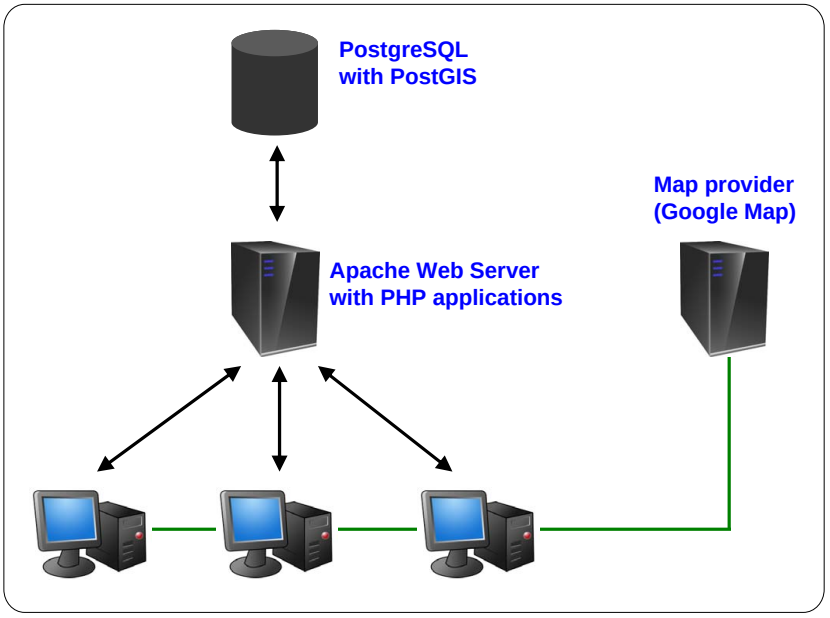


FIGURE 4.1: System architecture

4.2 Data Requirement

In this section, we analyze the data that we need and from where to obtain them.

- **Coordinate** of each airport in form of latitude and longitude. The coordinate is needed to locate the position of the airport in the map and to calculate the distance between one airport to another using the geocoding functions. This data can be obtained from World Airport Codes[1] (shown in Figure 4.8).

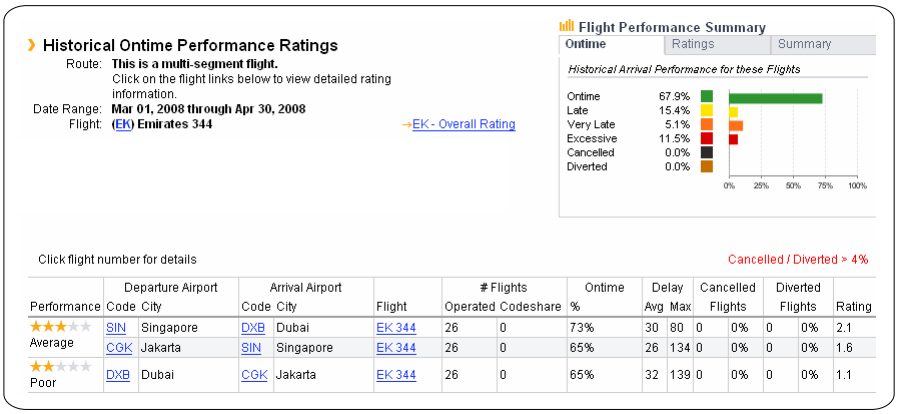
- **Time zone information** of each airport. We need this data to calculate the flight duration since the time displayed in the flight schedule is the local time at each airport. This data can also be obtained from World Airport Codes.

- **Price**. The best approach to get price data is by querying it at real time. But by doing real-time price query, there are two aspects that need to be considered. First, the system's responsiveness would degrade since the search process (which is fast since it is done locally) needs to wait until the price query is finished. Second, the system needs to incorporate an algorithm on how to rank the search results when there is failure of retrieving the price data (perhaps because of modification in the web page structure or temporary server unavailability).
  Therefore, to model the price, we use *price index*, with value from 1 to 10. Smaller price index means that the airline offers flights at lower fare. One possible implementation is to assign low-cost carriers with price index value from 1 to 5, while for major airlines it is from 6 to 10.

- **Reliability**. This factor correlates to the performance of the airline. There are many different meanings that can be incorporated into reliability term, for example on-time performance, less cancelled flights, or less incidents. The idea of using reliability factor in the search originates from FlightStats[2] which provides flight statistics based on flight number (Figure 4.2), airline (Figure 4.3), and route (Figure 4.4).
  To model reliability, we use the same approach as price. We create *reliability index* with value from 1 to 10. Higher reliability index implies that the airline is more reliable (can be considered as having great on-time performance).

- **Routes and schedules**. This data is composed from departure and destination airport (represented in three letters IATA code), departure time, and arrival time. This information can be obtained from each airline website.

[1] `http://www.world-airport-codes.com/`
[2] `http://www.flightstats.com`



Historical Ontime Performance Ratings

Route: **This is a multi-segment flight.**
Click on the flight links below to view detailed rating information.
Date Range: **Mar 01, 2008 through Apr 30, 2008**
Flight: **(EK) Emirates 344**

→EK - Overall Rating

Flight Performance Summary

Ontime | Ratings | Summary

Historical Arrival Performance for these Flights

Ontime	67.9%
Late	15.4%
Very Late	5.1%
Excessive	11.5%
Cancelled	0.0%
Diverted	0.0%

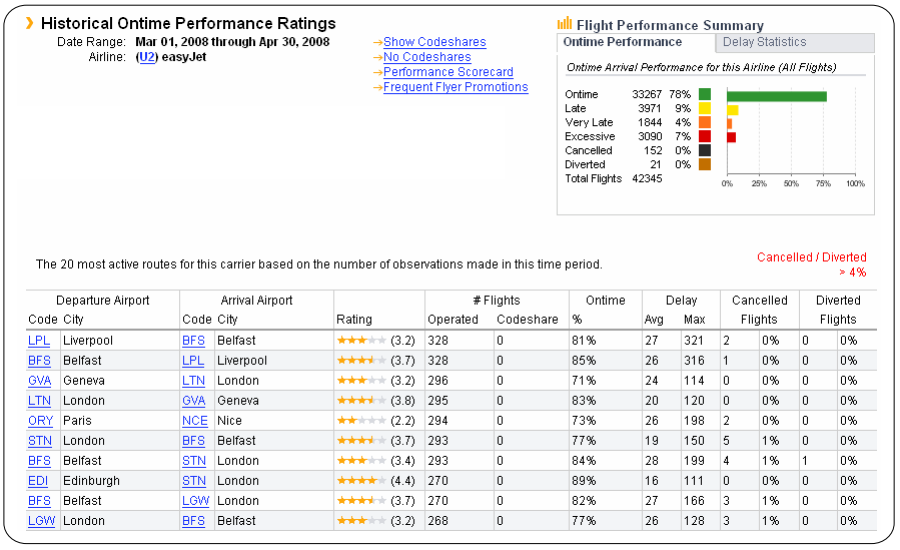
0% 25% 50% 75% 100%

Click flight number for details

Cancelled / Diverted > 4%

Performance	Departure Airport Code	City	Arrival Airport Code	City	Flight	# Flights Operated	Codeshare	Ontime %	Delay Avg	Max	Cancelled Flights		Diverted Flights		Rating
Average	SIN	Singapore	DXB	Dubai	EK 344	26	0	73%	30	80	0	0%	0	0%	2.1
	CGK	Jakarta	SIN	Singapore	EK 344	26	0	65%	26	134	0	0%	0	0%	1.6
Poor	DXB	Dubai	CGK	Jakarta	EK 344	26	0	65%	32	139	0	0%	0	0%	1.1

FIGURE 4.2: Flight statistic from Emirates' flight number EK344



Historical Ontime Performance Ratings

Date Range: **Mar 01, 2008 through Apr 30, 2008**
Airline: **(U2) easyJet**

→Show Codeshares
→No Codeshares
→Performance Scorecard
→Frequent Flyer Promotions

Flight Performance Summary

Ontime Performance | Delay Statistics

Ontime Arrival Performance for this Airline (All Flights)

Ontime	33267	78%
Late	3971	9%
Very Late	1844	4%
Excessive	3090	7%
Cancelled	152	0%
Diverted	21	0%
Total Flights	42345	

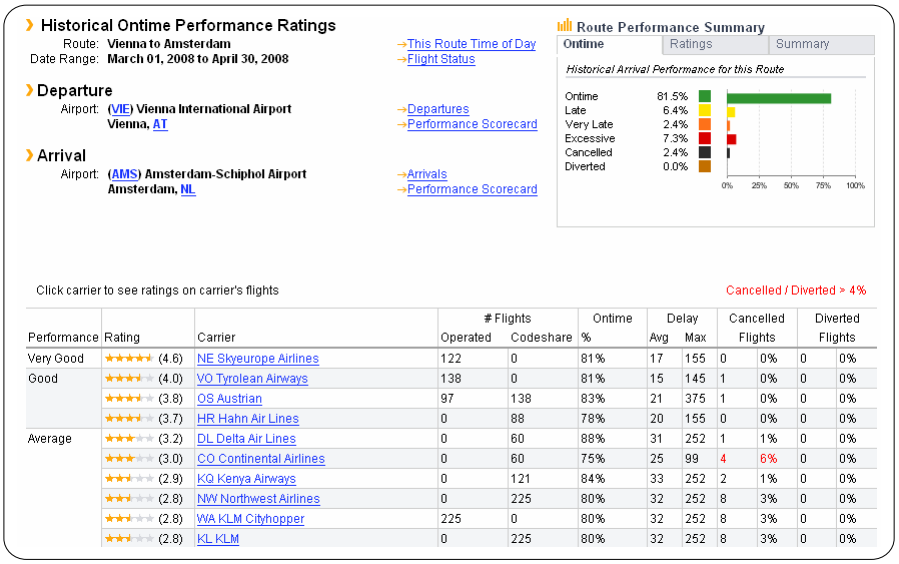
0% 25% 50% 75% 100%

The 20 most active routes for this carrier based on the number of observations made in this time period.

Cancelled / Diverted > 4%

Departure Airport Code	City	Arrival Airport Code	City	Rating	# Flights Operated	Codeshare	Ontime %	Delay Avg	Max	Cancelled Flights		Diverted Flights	
LPL	Liverpool	BFS	Belfast	(3.2)	328	0	81%	27	321	2	0%	0	0%
BFS	Belfast	LPL	Liverpool	(3.7)	328	0	85%	26	316	1	0%	0	0%
GVA	Geneva	LTN	London	(3.2)	296	0	71%	24	114	0	0%	0	0%
LTN	London	GVA	Geneva	(3.8)	295	0	83%	20	120	0	0%	0	0%
ORY	Paris	NCE	Nice	(2.2)	294	0	73%	26	198	2	0%	0	0%
STN	London	BFS	Belfast	(3.7)	293	0	77%	19	150	5	1%	0	0%
BFS	Belfast	STN	London	(3.4)	293	0	84%	28	199	4	1%	1	0%
EDI	Edinburgh	STN	London	(4.4)	270	0	89%	16	111	0	0%	0	0%
BFS	Belfast	LGW	London	(3.7)	270	0	82%	27	166	3	1%	0	0%
LGW	London	BFS	Belfast	(3.2)	268	0	77%	26	128	3	1%	0	0%

FIGURE 4.3: Flight statistic for Easy Jet



Historical Ontime Performance Ratings

Route: **Vienna to Amsterdam**
Date Range: **March 01, 2008 to April 30, 2008**

→This Route Time of Day
→Flight Status

Departure

Airport: **(VIE) Vienna International Airport**
Vienna, AT

→Departures
→Performance Scorecard

Arrival

Airport: **(AMS) Amsterdam-Schiphol Airport**
Amsterdam, NL

→Arrivals
→Performance Scorecard

Route Performance Summary

Ontime | Ratings | Summary

Historical Arrival Performance for this Route

Ontime	81.5%
Late	6.4%
Very Late	2.4%
Excessive	7.3%
Cancelled	2.4%
Diverted	0.0%

0% 25% 50% 75% 100%

Click carrier to see ratings on carrier's flights

Cancelled / Diverted > 4%

Performance	Rating	Carrier	# Flights Operated	Codeshare	Ontime %	Delay Avg	Max	Cancelled Flights		Diverted Flights	
Very Good	(4.6)	NE Skyeurope Airlines	122	0	81%	17	155	0	0%	0	0%
Good	(4.0)	VO Tyrolean Airways	138	0	81%	15	145	1	0%	0	0%
	(3.8)	OS Austrian	97	138	83%	21	375	1	0%	0	0%
	(3.7)	HR Hahn Air Lines	0	88	78%	20	155	0	0%	0	0%
Average	(3.2)	DL Delta Air Lines	0	60	88%	31	252	1	1%	0	0%
	(3.0)	CO Continental Airlines	0	60	75%	25	99	4	6%	0	0%
	(2.9)	KQ Kenya Airways	0	121	84%	33	252	2	1%	0	0%
	(2.8)	NW Northwest Airlines	0	225	80%	32	252	8	3%	0	0%
	(2.8)	WA KLM Cityhopper	225	0	80%	32	252	8	3%	0	0%
	(2.8)	KL KLM	0	225	80%	32	252	8	3%	0	0%

FIGURE 4.4: Flight statistic for Vienna-Amsterdam route

4.3 Business Logic

The business logic of the system is shown in Figure 4.5. Each part is shown with the technology that is used to develop that particular part.

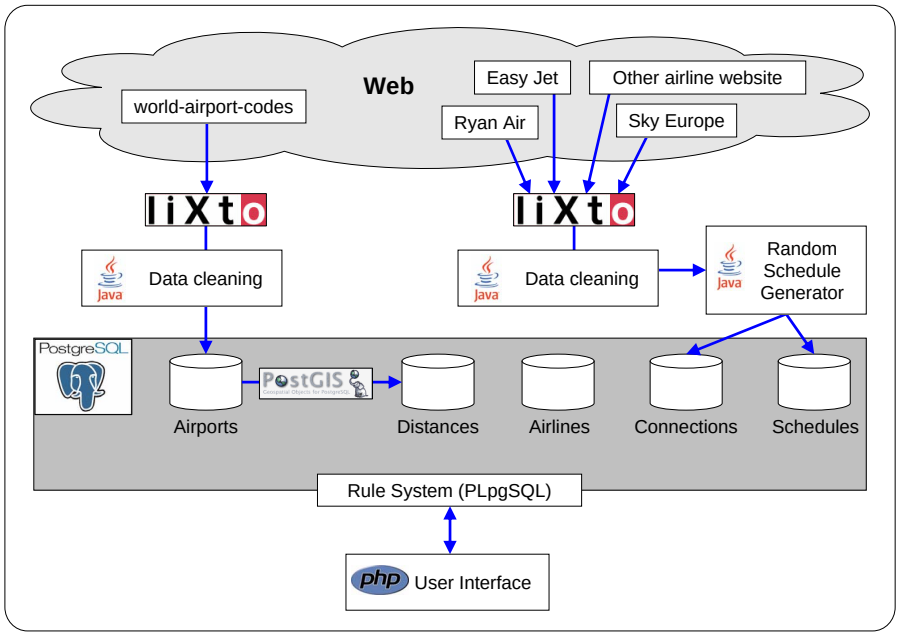


FIGURE 4.5: Business logic of the system

The system can be divided into four layers: web, wrapper, database , and user interface layer. Each layer is detailed below.

4.3.1 Web and Wrapper Layer

In the previous section we have analyzed the Web resources from which we can obtain our data. A Lixto wrapper is created for each of the resources, that is one wrapper for the airport data, and one wrapper for each airline website.

It is often that the wrapped data still contains some unnecessary annotations, such as punctuation marks, field names, and extra spaces. Therefore, we employ data cleaning process to ensure that the data only contains the exact information that we want. Besides data cleaning, some preprocessing steps may also be needed to ensure that the data complies with the database schema. For example, later when we wrap the airport's geographical coordinate, the latitude and longitude is expressed in degrees, minutes, and seconds. We need to convert this form into a single decimal degree to fit with the table schema.

Lixto wrapper generates output in XML format. We use Java's Simple API for XML (SAX) technology to read the XML output and do the data cleaning process. Standard Java technology is used to preprocess the data to be in the correct format. After having the clean data in correct format, there are two options to populate the database, either using Java Database Connectivity (JDBC) technology or exporting the data into comma-separated values (CSV) text file, which can be imported by PostgreSQL.

In Figure 4.5, we also see that the wrapped data from the airline websites is inputted to the Random Schedule Generator. In short, this generator is used to create arbitrary schedules for the airlines. It will be described in more detail in Chapter 6.

4.3.2 Database Layer

We need a database that supports geocoding feature. Currently, there are several possible database options. The evaluation of some leading database providers is provided below.

Oracle provides geocoding through Oracle Locator and Oracle Spatial, however it is only available in non-free Oracle Editions (Standard and Enterprise). **MySQL** has already announced their support for spatial extensions since MySQL version 5.0.16 (November 2005), but until now, the extension has not been mature enough and still under development. **SQL Server** has not provided any spatial data support. They plan to support it in the future release of SQL Server 2008.

With the above facts, **PostgreSQL** with PostGIS spatial extension is the most suitable choice. It is one of the leading relational database management system currently available. It is free, open source software which allows anyone to use the software on any number of servers without restrictions on number of users, connections, CPUs, or size of data set. It can be optimized for many situations, ranging from read-only information websites to multi-user e-commerce systems with high transactional load. Whether the site needs a single shared database server with a few tables, or multiple database servers storing millions of records, PostgreSQL is able to scale with the demands.

Figure 4.6 shows the database schema of the system. Table 4.1 to 4.7 show the detailed schema for each table.

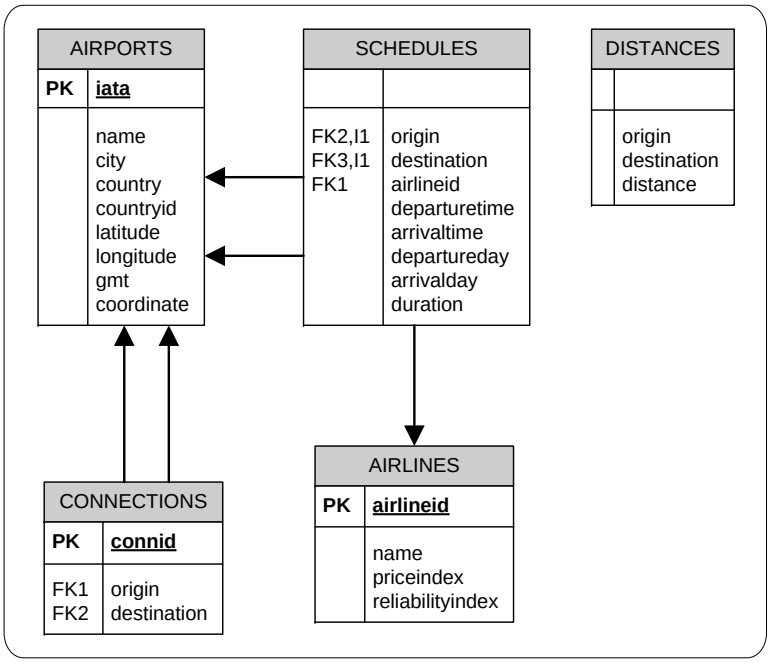


FIGURE 4.6: Database schema of the system

TABLE 4.1: Schema of Table Airports

Field name	Data type	Notes
iata	char(3)	Primary key
name	varchar(80)	Airport name
city	varchar(40)	City of airport
country	varchar(40)	Country of airport
countryid	char(2)	Country in 2-letters code
latitude	numeric	In degrees
longitude	numeric	In degrees
gmt	numeric	GMT offset of the airport
coordinate	point	Point is PostGIS data type

TABLE 4.2: Schema of Table Airlines

Field name	Data type	Notes
airlineid	serial	Primary key
name	varchar(40)	Airline name
priceindex	integer	Range value is 1 to 10
reliabilityindex	integer	Range value is 1 to 10

Table 4.3: Schema of Table Connections

Field name	Data type	Notes
connid	serial	Primary key
origin	char(3)	Foreign key reference to Airports.iata
destination	char(3)	Foreign key reference to Airports.iata

Table 4.4: Schema of Table Schedules

Field name	Data type	Notes
origin	char(3)	Foreign key reference to Airports.iata
destination	char(3)	Foreign key reference to Airports.iata
airlineid	integer	Foreign key reference to Airlines.airlineid
departuretime	time	Local time at departure airport
arrivaltime	time	Local time at arrival airport
departureday	integer	1=Monday, 2=Tuesday, ..., 7=Sunday
arrivalday	integer	1=Monday, 2=Tuesday, ..., 7=Sunday
duration	integer	In minutes

Table 4.5: Schema of Table Distances

Field name	Data type	Notes
origin	char(3)	Foreign key reference to Airports.iata
destination	char(3)	Foreign key reference to Airports.iata
distance	numeric	In kilometers

4.3.3 User Interface Layer

For the user interface, we choose to use PHP which provide server-side processing so that the page can be generated dynamically on-the-fly. PHP is an open source technology that can be used to create applications that run for free on any Apache web server.

PHP has many built in functions, libraries and classes. In this way PHP programming language allows a quicker development of web applications. PHP applications can be connected to various databases. PHP is a flexible and portable language. Applications programmed in PHP are easy to be implemented or ported on many operating systems, such as Windows, Linux, MacOS and Solaris. In addition to being cross-platform, it has built-in functions for connecting to various popular database, including general ODBC connections.

4.4 Adding Geometry Column in Table Airports

To create table `Airports` with structure as in Table 4.1, two steps are needed because the `coordinate` column is a geometry column. At first, we create the first eight fields using SQL CREATE TABLE statement. Afterwards, the `coordinate` column is generated using the following PostGIS statement:

```
SELECT AddGeometryColumn( 'public', 'airports', 'coordinate', 32661,
'POINT', 2 );
```

where:

- `public` is the schema where AIRPORTS is located.
- `airports` is the table to be added with this new column.
- `coordinate` is the column name.
- `32661` is the SRID[3] which refers to the WGS 84[4] system , used by DoD for all its mapping, charting, surveying, and navigation needs.
- `POINT` is the geometry data type that we need to hold geometric data representing pair of [longitude, latitude].
- `2` is the dimension of POINT.

The `coordinate` column is populated by executing the following statement:

```
UPDATE airports SET coordinate = transform(PointFromText('POINT(' ||
longitude || ' ' || latitude || ')',4269),32661);
```

where:

- `transform` is the PostGIS built-in function to convert data from one SRID to another, where in the example above, it is converted from SRID 4269 (longitude/latitude measurement) to SRID 32661 (WGS 84 system).
- `PointFromText` is another PostGIS built-in function which recognize a text in the format of Well Known Text (WKT) and converts it to POINT data type.
- `||` is the concatenation operator in PostgreSQL.

[3]Spatial Reference Identity
[4]World Geodetic System of 1984

4.5 Wrapping Airports Data

Figure 4.8 shows the screenshot of WAC. In the top bar, we have alphabets from A to Z. Each alphabet contains the link to the detail page, containing all airports with IATA started with the letter. The figure shows airports with IATA starting with A.

These alphabets are the *first iterator*. Below the first iterator is the list of airports which are the *second iterator*. For each airport, by clicking on the airport name, WAC opens the detail page containing airports data that we need. Appendix A contains full details on the wrapping of airports data. Figure 4.7 shows the data model that is used in the wrapper.

FIGURE 4.7: Lixto data model for airports

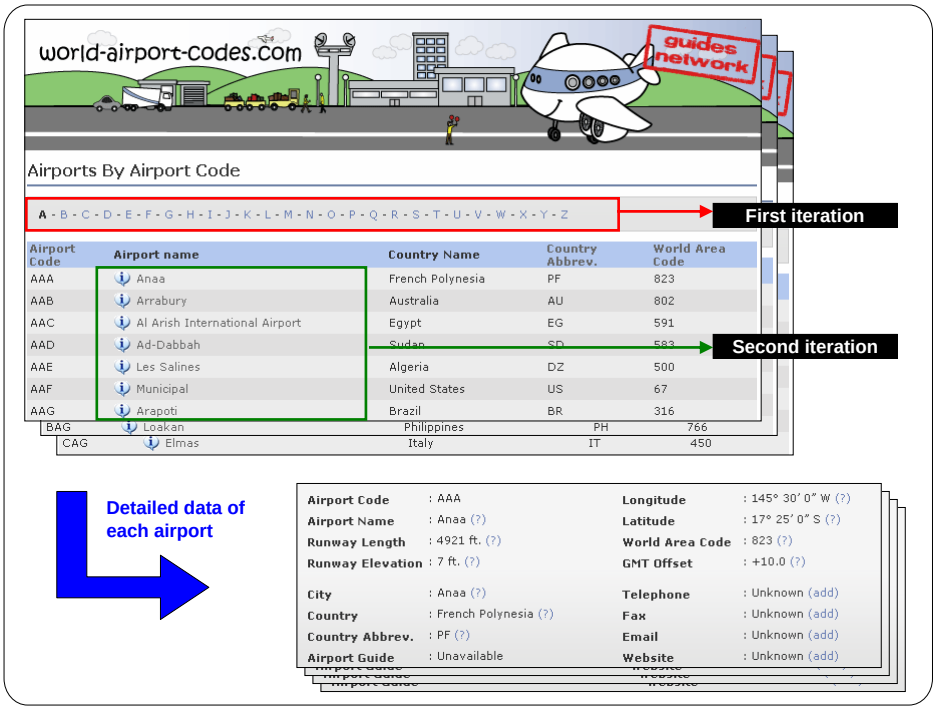


FIGURE 4.8: World Airport Codes screenshot

The sample XML output from the wrapper is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  ...
  <airports>
    <iata>: VIE</iata>
    <name>: Schwechat International (?)</name>
    <city>: Vienna (?)</city>
    <country>: Austria (?)</country>
    <countrycode>: AT (?)</countrycode>
    <longitude>: 16 34 11 E (?)</longitude>
    <latitude>: 48 6 37 N (?)</latitude>
    <gmt>: +1.0 (?)</gmt>
  </airports>
  ...
</document>
```

4.6 Data Cleaning of Airports Time Zone Value

In the data that we get from WAC, we found the existence of some invalid time zone data. Some airports have simply its minus and plus sign reversed, however some others are completely located in wrong time zone. Therefore, aside from the simple data cleaning effort, such as cleaning the unnecessary annotations, in here we need a more complex data cleaning effort.

The following heuristics are used for the data cleaning:

- Most of the records have its `countryid` field in ISO 3166-1 alpha-2 format (e.g. AT for Austria, DE for Germany).
- Most countries are located in single time zone.

The major steps of the data cleaning process are as follows:

1. Create another table `Airports2` as the clone of table `Airports` (schema and records).
2. Delete all records in `Airports`.

3. Wrap time zone data from Wikipedia , from here we can get pairs of [countryid, TZname], for example [AD, Europe/Andorra].
4. Put data from step 3 into a new table `CountryTZ`.
5. Wrap time zone data from Tiscali website , from here we can get pairs of [TZname, GMT], for example [Europe/Andorra, +1].
6. Put data from step 5 into a new table `TZGMT`.
7. Join table `CountryTZ` and `TZGMT` by matching TZname column.
8. Update gmt in `Airports2` with gmt from the join table for all countries with single time zone.
9. Move (insert into `Airports` and delete from `Airports2`) all airports in single time zone countries (which is now already in valid gmt) from `Airports2` to `Airports`.
10. Now, table `Airports2` only contains airports located in multi-time zone countries.
11. Check each airport in `Airports2`, if it has same absolute gmt value with one of the gmt value in the join table in its country.
    - If yes, then update its gmt with gmt from the join table and move it to table `Airports`.
    - If no, then we can't assign time zone to this airport because we don't know in which time zone exactly the airport is located.

    This step may be inaccurate because it only relies from the heuristic assuming that most of the records have its minus and plus sign reversed.
12. In the end, we have table `Airports` containing records with valid gmt content field and table `Airports2` containing airports with unknown time zone.

4.7 Wrapping Airlines Website

4.7.1 Choosing the Airlines to be Wrapped

Some airline websites use text box to input the city/airport name, some others use select box where user simply picks the city/airport name from a list. For those which use text box, to find the routes offered by that airline we need to try to enter all possible pairs of IATA codes that we have obtained from wrapping airports data.

Even with 100 airports, then there are 9900 possible pairs that has to be tested, where most of the pairs are not served by the airline. The fact is that there are more than 9000 airports all over the world (so there are more than 80 million pairs that need to be tried), where some of them are used only for national, chartered, or private flights. Hence, trying all possible pairs is not practical since the number of possible pairs is too large.

Therefore, we choose to wrap only the airlines that use select box. Doing this is much simpler since the options are limited. The airlines only give possibility to choose from the airports which they served. Limiting our scope on website with select box, some use JavaScript, some do not. With JavaScript, each change in departure airport selection will change the list of possible destination airports. This is also desired since we do not waste time on searching for two airports where no flights between them are served by the airline. Figure 4.9 shows an example from an airline website that use select box with JavaScript.

FIGURE 4.9: Choosing only airlines with select box and JavaScript

As in wrapping airports data, by using select box and JavaScript, we can maintain a list of departure airport as *first iterator*, then browse through the list of possible destinations as *second iterator*. The detail page is shown after the form is submitted (deep web navigaton). All of these actions, such as mouse click and submitting form are supported by Lixto VD.

4.7.2 Filtering Search Results

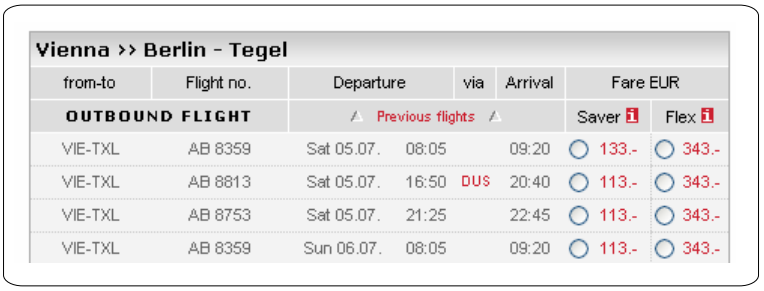
We are only interested in direct routes. By only having direct routes, we can freely mix and match the routes and airlines. Some airlines show both direct routes and transit

routes in their search results. For example, between Vienna and Berlin, the search results from Air Berlin shows both direct flights and flights with transit at Dusseldorf (Figure 4.10).

There are two options for eliminating this example of non-direct routes. We can use complex XPath expression to choose only flights where the `via` column is empty. Another option is eliminating it in the data cleaning and preprocessing step when we process the XML output.

The data integrations may also pose a problem because an airport may have different name on each airline. For Milan Bergamo (BGY) airport, Ryan Air name it as *Milan (Bergamo)*, Myair name it as *Milano Orio al Serio*, and Sky Europe name it as *Milan - Bergamo (BGY)*. Fortunately, almost all airlines embed the IATA code of the airport in the `value` attribute of the select box element. Hence, we know exactly which airport is mentioned by the airline although there are variations in airport names.

In Figure 4.11, the common data model for connections is shown. Figure 4.12 and 4.13 show the wrapping steps for Ryan Air and Air Berlin with their corresponding fragment of XML output. The wrapping details is not described here.



Vienna >> Berlin - Tegel

from-to	Flight no.	Departure	via	Arrival	Fare EUR	
OUTBOUND FLIGHT		Previous flights			Saver	Flex
VIE-TXL	AB 8359	Sat 05.07. 08:05		09:20	133.-	343.-
VIE-TXL	AB 8813	Sat 05.07. 16:50	DUS	20:40	113.-	343.-
VIE-TXL	AB 8753	Sat 05.07. 21:25		22:45	113.-	343.-
VIE-TXL	AB 8359	Sun 06.07. 08:05		09:20	113.-	343.-

FIGURE 4.10: Air Berlin's direct flight and flight with transits

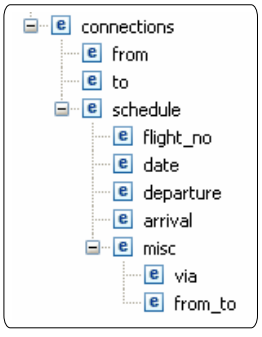


FIGURE 4.11: Lixto data model for connections

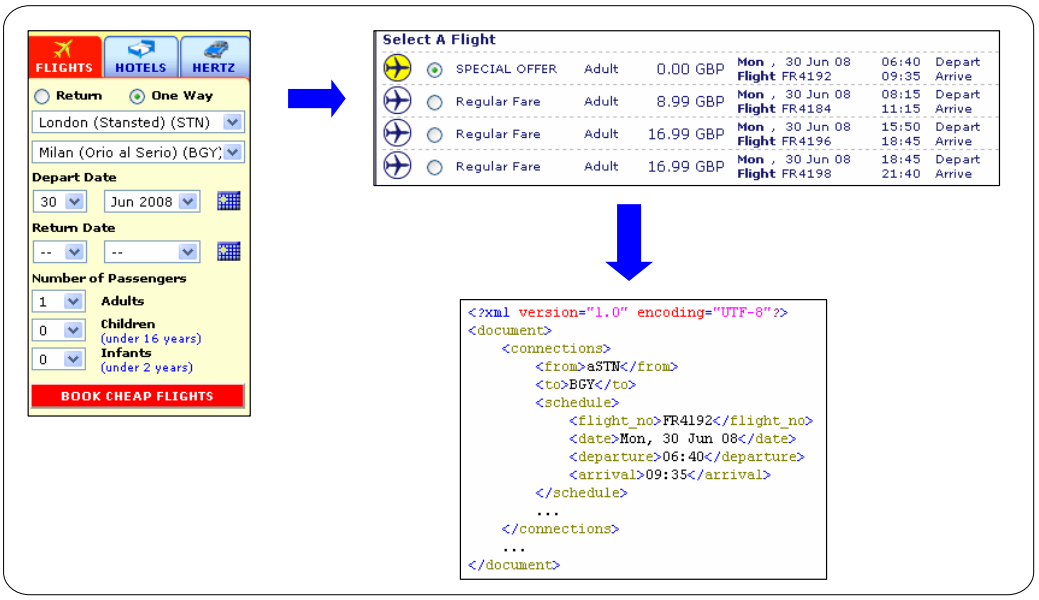


FIGURE 4.12: Wrapping Ryan Air

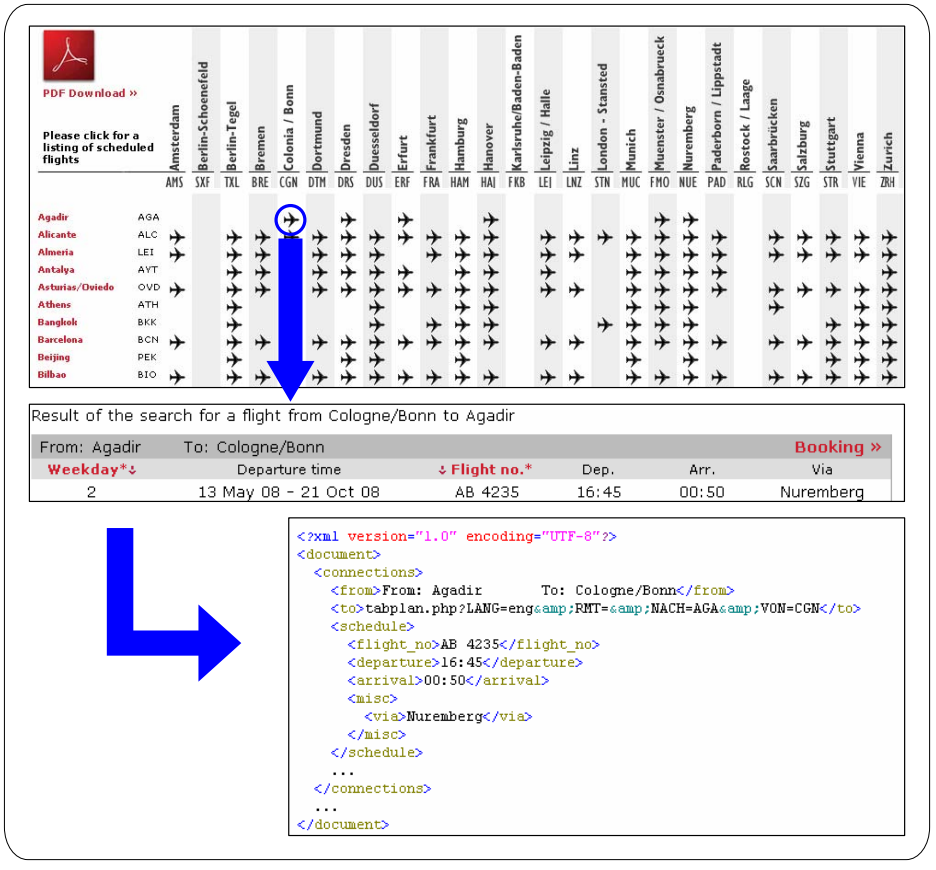


FIGURE 4.13: Wrapping Air Berlin

Chapter 5

Scalable Algorithm

5.1 Complexity for Flight Search

A graph $G = (V, E)$ is a finite set V of nodes and a set E of edges, which are pairs of nodes. Our flight database can be viewed as a directed graph, with airport as the nodes and the pair of airports where exist direct flights between them as the edges. Two airports can be connected by several flights (either from a single airline or several airlines). Therefore, we may have several edges between two nodes in the graph. In this case, our graph is called *multigraph*.

In our database design, we differentiate between table `Connections` which contains the unique origin-destination airport pairs, and table `Schedules` which contains the schedule details of each flight, such as departure and arrival time, departure and arrival day, day of operations, and flight duration.

Figure 5.1 shows the relationship between the information inside the tables (viewed in simplified version by omitting the field name) and the corresponding graph representation. In table `Schedules`, the rightmost column indicates the number of available flights on a particular day for a particular route (to be noted, the numbers of available flights on each day may be different).

5.1.1 Complexity for Direct Routes

For direct routes (no transit), the system simply browse the list of records in table Connections. If it finds a matching record, then there exists direct flight from the origin to destination. The search complexity is linear to the size (number of records) of table `Connections`.

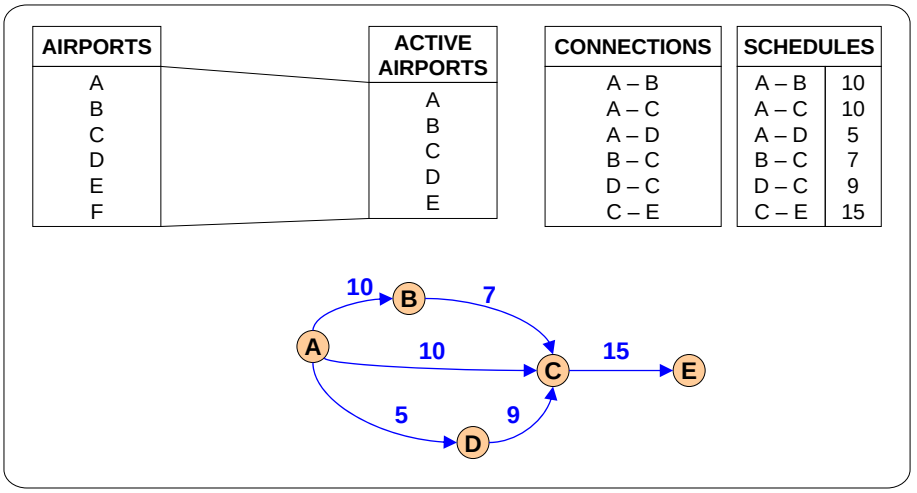


FIGURE 5.1: Graph representation of the flight database (table is represented in simple format, not displaying the fields name)

5.1.2 Complexity for Transit Routes

For transit routes, assume that we have graph $G = (V, E)$ where $V = \{A_1, A_2, \ldots, A_k\}$ and the graph is in full-mesh topology, so that $\forall x \in V, y \in V$, where $x \neq y$ we have $(x, y) \in E$. We want to know how many route possibilities we have for going from A_i to A_j.

For one-transit case, from A_i we have $(k-1)$ edges to other nodes. However, we can't go directly to A_j, since we would arrive at the destination without transiting (our goal is to find one-transit routes). We can go to $(k-2)$ nodes, where from each we can go to A_j. Therefore, there are $(k-2)$ possible routes for one-transit case. For two-transit case, with the same reasoning, there are $(k-2).(k-3)$ possible routes from A_i to A_j.

The number of possible routes is different from the search complexity to find them. In our database, we do not retain the list of nodes. What we retain in table `Connections` is the ordered pairs of airports where there exists direct connection between them. The transit routes can be found by using table joins. For finding one-transit routes, we join two table `Connections`. Assuming the table size is r, then the join table would be of size r^2. For finding two-transit routes, we join three table `Connections`, so that the size of the join table is r^3. From the join table, we filter so that we retrieve only the records with desired departure and destination airport.

We can conclude that for general graph, the search complexity for determining possible routes is $O(r^{s+1})$ where r is the size of table `Connections` and s is the number of transits. In other words, it is $O(r^2)$ for one-transit route and $O(r^3)$ for two-transit route.

The number of possible routes is a factor in the total complexity for flight search. The other factor is **the number of edges between two nodes**. Figure 5.2 shows illustration for this problem. The p in the graph represents the number of edges between two nodes.

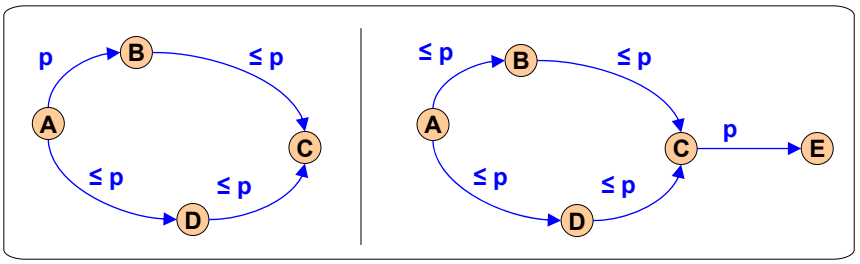


FIGURE 5.2: Search complexity for one-transit and two-transit routes

For one-transit case in the left part of the figure, it is shown that there are p edges between A and B, where p is the biggest number compared to other edges in the same graph. Hence, there are at most p^2 combinations for route from A to C, through B. For route from A to C, through D, the number of combinations may be smaller that p^2, but we can say that it is upper-bounded by p^2. For two-transit case in the right part of the figure, with the same reasoning, there are at most p^3 combinations from A to E for each possible route (either A-B-C-E or A-D-C-E).

Therefore, we conclude that in general case, the search complexity for searching posssible flight combinations for each possible route is $O(p^q)$ where p is the maximum number of edges in any particular leg among all possible routes, and q is the number of transits. It is $O(p^2)$ for one-transit route and $O(p^3)$ for two-transit route.

Summarizing from the explanations above, the **total flight search complexity** is $O(n^2)$ for one-transit and $O(n^3)$ for two-transit where $n = p.r$.

5.2 Route Determination

In flight search system of any scale, direct route is the simplest one to search, there is no variation on finding their existence. Route determination is interesting to be discussed for transit routes.

From the complexity side, the algorithm of finding transit routes is polynomial, hence it is already *tractable*. However, what is desired mostly from a search engine is its responsiveness. The system has to be able to give search results in a reasonable time for human users. Performing exhaustive search by finding and evaluating all possible

combinations is the best way to find the best routes, but the user may not be willing to wait for the long search time.

We observe that not every possible routes give good search results. For example, a possible one-transit route from Vienna to Frankfurt is through Singapore, a route which would never be taken by anyone. So, we need an algorithm that can eliminate this kind of route. The system doesn't need to find all possible flight plan, but it has to be able to return a part of the best routes (if not the best route itself) in the search results. Below we discuss on some considerations in finding the heuristic for optimal search.

5.2.1 No Transit vs. One Transit vs. Two Transit

In this section we analyze whether we can optimize the search based on the number of transits of the route. In the system, there are three criteria to sort the search results: Time, Price, and Reliability. Users can also determined how many search results that they desire.

If **Time** is the first criterion, then direct routes must be in the top ranks of the search results. Undoubtedly, direct routes have shorter journey time compared to transit routes. To illustrate, for short-time flights where flight duration is less than 3 hours, then having transit already adds at least one hour to the total journey time. An optimization action that can be performed is to stop the search when the number of available direct flights already exceeds the number of search results desired by user. To search further for transit routes is useless since the transit routes would never appear in the search results.

If the first criterion is Price or Reliability, then the number of transits doesn't give hint on how to perform optimization. With **Price** as the first criterion, there is a chance where total price of transit routes is cheaper from the price of direct routes. For example, two flights from low-cost airlines can have cheaper price than one direct flight by major airlines. If **Reliability** is the first criterion, then several flights from reliable airlines must have higher reliability index value compared to a direct flight with unreliable airlines.

5.2.2 Transit Time Analysis

In this section, we take a look on the transit time factor in route determination. For this purpose, we search for routes from London Stansted (STN) to Nuremberg (NUE) on a particular date. There are several possible transits shown in Figure 5.3. Simply observing the route map, Düsseldorf and Amsterdam, unlike Berlin and Munich, seems to be good transits since it is located in the middle, between the origin and destination.

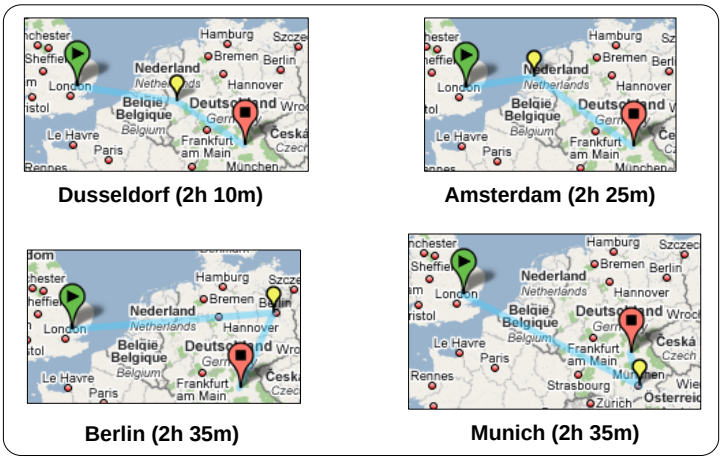


FIGURE 5.3: Possible transits from London Stansted to Nuremberg, the number in the brackets are the journey time without transit (only the total flight duration)

The question is, from the four possible transit airports shown in the figure, which one is the best? There are some possible answers, depending on the criteria that we use to evaluate each route. Table 5.1 shows the time details for each route.

TABLE 5.1: Transit time comparisons from London Stansted to Nuremberg, through Düsseldorf (DUS), Amsterdam (AMS), Berlin (TXL), and Munich (MUC)

	DUS	AMS	TXL	MUC
STN - x	1h 15m	1h 10m	1h 40m	1h 50m
Transit time	2h 25m	3h 30m	2h 35m	4h 10m
x - NUE	0h 55m	1h 15m	0h 55m	0h 45m
Total journey time	4h 35m	5h 55m	5h 10m	6h 45m
Flight duration (without transit)	2h 10m	2h 25m	2h 35m	2h 35m

If we solely consider the flight duration time, then the order is Düsseldorf, Amsterdam, Berlin, and Munich. But if we consider total journey time (flight duration + transit time), then the decision really depends on the transit time in each airport. In another case where the transit time in Berlin is 1 hour, but the transit time in Düsseldorf is 3 hours, then routes through Berlin are better options than Düsseldorf.

The above example try to show that transit time can not be estimated (unlike the flight duration which can be estimated by creating a function of distance vs. time - this function is used in the random schedule generator). We don't know which transit airport has the shortest transit time, except if we review all possible transit airports (exhaustive search), but once again, this is infeasible in terms of processing time. The best effort is by choosing some hubs, then check for total journey time for route through those hubs. There is a chance that the real best route is missed due to misjudgement

of hub airports. However, this is the trade-off that can not be avoided in exchange for a scalable and responsive system.

5.3 Hub Identification Heuristics

An airport may be regarded as hub if it is used as a transfer point to get passengers to their intended destination. Many hubs of the airlines are situated at airports in the cities of the respective head offices. Some airlines use only a single hub, while some others use multiple hubs. Figure 5.4 shows Transavia route map where Amsterdam is the hub.

FIGURE 5.4: Transavia route map, with hub at Amsterdam

Since we support the idea of mixing airlines, then when we mention "hub", the hub can be the traditional airline hubs (such as Frankfurt, Dubai, and Amsterdam) or other airports which have high-traffic (for a particular route) or served by many airlines. In real-world, this new hub concept exists. A recent article from Travel Daily News [29] discuss about the investment by Düsseldorf International Airport to build up its infrastructure in anticipation of additional hub functions.

The hub identification heuristic idea is depicted in Figure 5.5. By using the heuristic, the system no longer browse through each possible routes and search all possible combinations from them. Instead, it chooses some good hubs then try to find flight combinations from routes going through these good hubs.

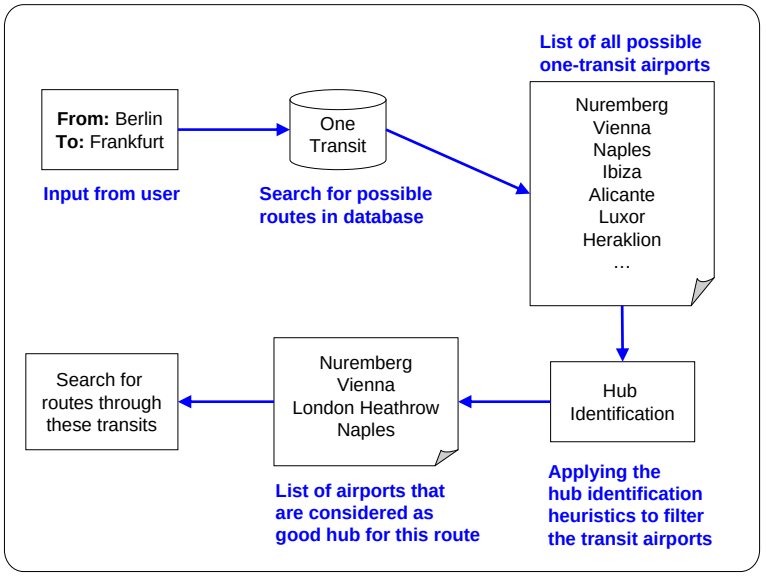


FIGURE 5.5: Route search techniques by using hub

There are two possible outcomes for using this heuristic. First, if the heuristic misjudges a hub which is actually a good hub, then we may miss the optimal search results. Second, not every search results that we get can be claimed as the best route (such as the shortest-journey-time route or the lowest-fare route) since we only search from a fraction of all possible routes.

Therefore, it is interesting to find an approach for this hub identification heuristic. The approach needs to behave dynamically and can adapt for various cases on various graph model. The process of finding the good hubs needs to have short processing time since this processing time also contributes to total processing time of the search process.

5.3.1 Approaches for Hub Identification

In this section we discuss about several approaches for identifying hub. To be noted, a hub that is good for one route, may not be good for other routes.

First approach is to list the hubs based on current knowledge, for example by using the traditional airline hubs. This approach has several weaknesses. It is not fully dynamic because it needs some human maintenance for updating the list when there are changes (perhaps airline bankruptcy or airport renovation). If the list is not regularly updated, then it is possible that one time it is outdated and not relevant anymore. This approach can be enhanced, for example by adding the information for each routes, which hub is relevant for them. However listing all possible hub for all possible pairs of airport is a very tedious and not efficient task.

Second approach is to identify the hub dynamically, by counting the number of airports reachable from that airport. The larger the number of reachable airports, then the hub quality is better. However, this approach suffer from irrelevant hub problem. By this approach, the hub ranking for any route is the same. For example, Dubai is a hub airport of Emirates. It is reachable from many airports and can reach many airports. Therefore, it is always on the top ranks of hub. But, for routes between two airports in Europe, Dubai is not a good hub.

Third approach is also dynamic, that is by considering the number of available connections from the origin airport to transit airport and from the transit airport to destination airport. The problem with this approach is that what method (addition or multiplication) do we use for saying transit A is better than transit B (Figure 5.6). Second problem is which connections to be considered in this approach? Is it only the connections that meet with user criteria (for example, departure between 8:00 and 10:00, transit time between 1-3 hours)? If we use this technique, then it it means that we need to evaluate all possible routes, only to make the hub ranking. Hence, we don't need the hub anymore because the best route should already be able to be identified from the process of hub identification.

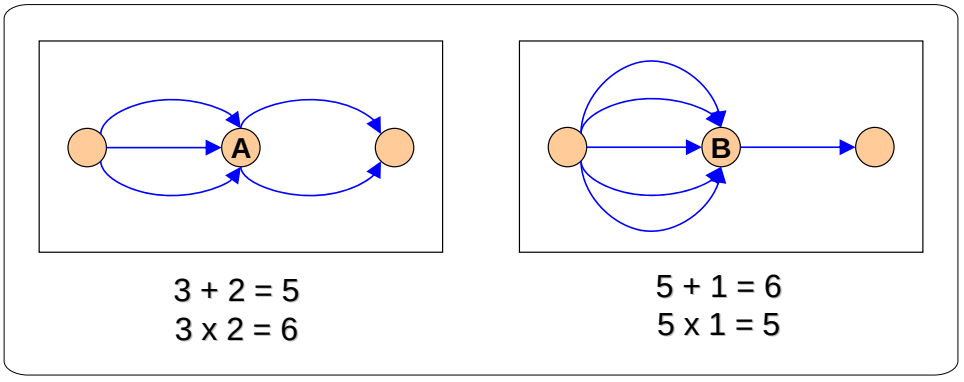


FIGURE 5.6: Identifying hub by number of available connections

Fourth approach is by ranking the hub based on the minimum journey time (flight duration + transit time). But this means that all possible hubs need to be evaluated first before the ranking is performed. Then the best routes should already be found before the hub is found.

Fifth approach is by dynamically consider the geographical position of the possible hubs with respect to the origin and destination airport. From the latitude and longitude of origin and destination airport, we can calculate the location of the middle point, then create virtual circle with origin and destination airport as the diameter. The radius of

the virtual circle is the distance between the middle point and the origin/destination airport.

Airports that are located inside the virtual circle between origin and destination has better ranks than the one outside. We can know whether an airport is located inside the virtual circle or not by computing the distance from the middle point to the airport. If the distance is less than the radius, then it is inside the circle. One problem for this approach is that there exist some pairs of airport where no direct connection are available, and all possible transit airports are located outside the virtual circle. This idea is illustrated in Figure 5.7.

FIGURE 5.7: Identifying hub by using geographical coordinate, the left picture has many transit airports in between, while the right has no transit airports in between

This approach is also prone for intercontinental routes. For this kind of routes, we will have a virtual circle with a very large value of radius, and we need to consider enormous number of airports which are located inside this virtual circle, as the hub.

Sixth approach is similar with fifth approach and it even covers the fifth approach by solving the case where there are no possible transit airports in between. This approach calculates the total distance from the origin to transit and transit to destination. From simple physics law, we know that distance is directly proportional to time. Hence, minimizing the total distance means also minimizing the flight duration (to be noted, flight duration is the time which is spent for flying and doesn't consist of transit time). Therefore, this approach is optimal, in terms of minimizing flight duration.

5.3.2 Optimality of the Heuristic

Using the sixth approach (*total distance approach*), the distance between pairs of airport can be pre-computed beforehand. The geographical coordinate of an airport will be likely not changing (if never). Hence, we create another table, `Distances` that contains

distance between each pair of airport. By putting index on the table, the retrieval of the records in table `Distances` is faster.

Suppose that we are looking for routes from Berlin (TXL) to Frankfurt (FRA). Figure 5.8 shows the possible one-transit routes, sorted by their total journey distance. The shortest route uses Nuremberg (NUE) as hub. The question is, how many possible routes that the system need to consider to be included in the search? We can put a hard limit such as only include 10 shortest-distance routes in the search. The problem is, if the best route is actually through hub that is on position 11 of the list, then we would miss the best route.

In the following discussion we discuss the approach that we use, based on the transit time input parameter from user. In our system, user can configure the minimum and maximum transit time (Figure 5.9). We also show the optimality of our hub identification heuristic.

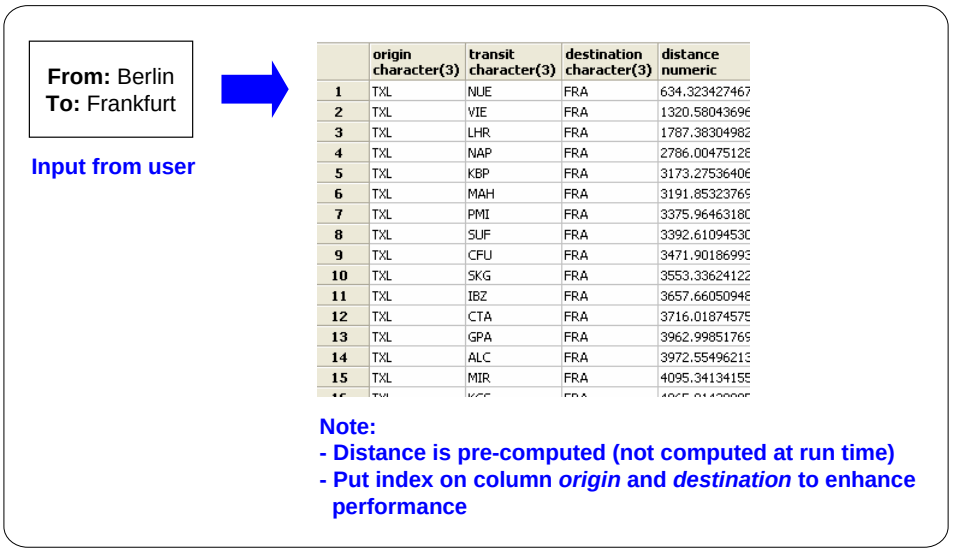


FIGURE 5.8: Identifying hub by total journey distance

To use this approach, first we need to determine the constant value of travelled distance per hour (how many kilometers can be travelled in an hour). Assume that we take the value of 1000, which means that in 1 hour the aeroplane can travel up to 1000 km. Assume also that the user wants the transit time between 1 and 3 hours.

Referring back to Figure 5.8, the route through Nuremberg takes 634 km. If the transit time in Nuremberg is 1 hour, then for sure this is the best route (having the minimum total journey time) since we have shown that our heuristic approach guarantees minimum flight duration. Let's now consider if the transit time in Nuremberg is 3 hours (the maximum transit time allowed by the user). In this case, for the same total journey time

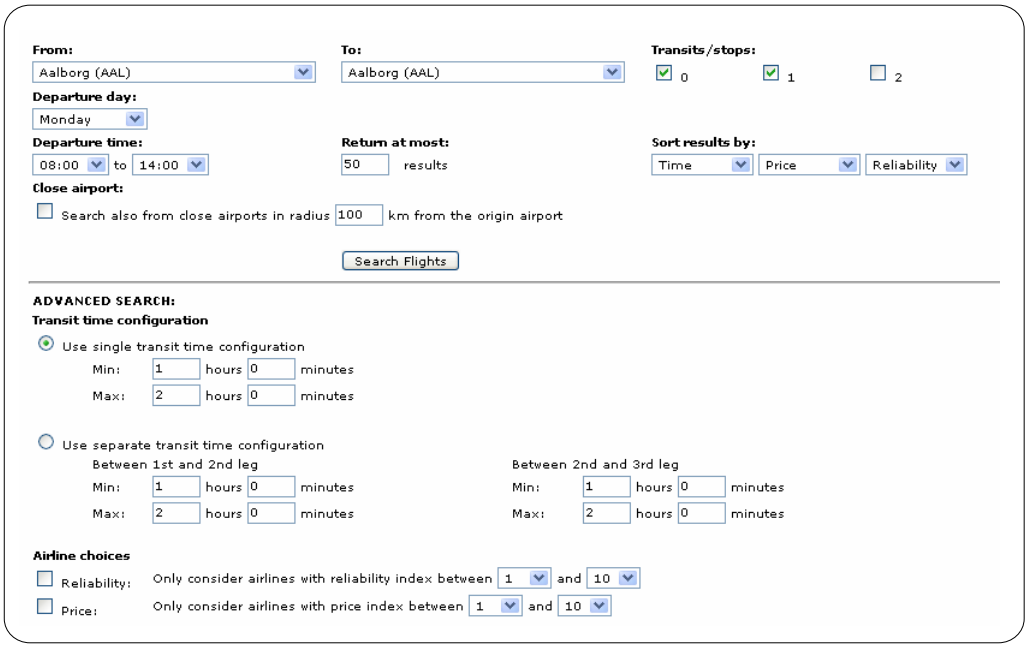


FIGURE 5.9: Start page of the system

as the one offered by Nuremberg, the aeroplane can travel for another 2 hours (equal to 2000 km, based on our constant value of 1000) through another hub X, provided that the transit time in X is 1 hour. If the transit time in X is more than 1 hour, then route from Nuremberg must have better total journey time performance.

How about the possible hubs with total distance larger than 2634 km? In this case, these hubs would not give the best route in terms of total journey time since even when the transit time at these hubs is 1 hour (minimum), the total journey time is already exceeding the route offered through Nuremberg with 3 hours (maximum) transit time. This approach answers the question on how many possible routes needs to be evaluated. In this example, it means that we only need to consider routes with distance at most 2634 km, which means only the top three routes from Figure 5.8.

This approach may pose a problem if the maximum transit time parameter is configured to have a big value (for example, 7 hours which equals 7000 km) or there are many possible routes with close distance one to another. In these cases, the system becomes not scalable due to many routes that needs to be checked. Therefore, we also put an upper hard limit on number of possible routes to be checked. Eventhough we may miss the real best route due to this upper limit, but the heuristic would find routes that are close to the real best route.

5.4 Searching from Close Airports

Figure 5.10 shows the flowchart of searching from close airports feature. If user choose this feature then the system adds the list of close airports to the list of possible departure point. The possible routes from all departure point are then presented to user (Figure 6.5). User can modify which routes to consider by selecting/unselecting the check box which is provided to the left of each route.

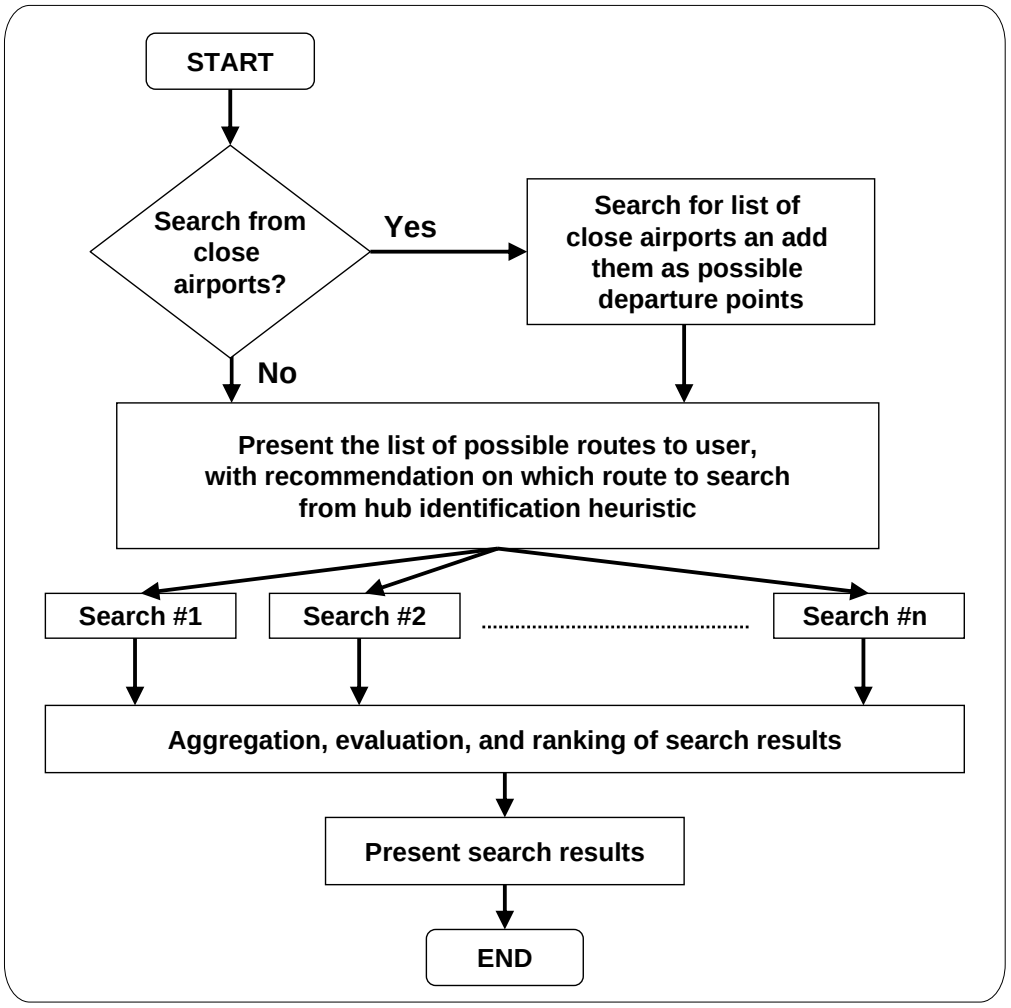


FIGURE 5.10: Flowchart for searching from close airports

The search process for each route is performed independently. The search results from each route is then aggregated before presented based on the selection criteria that have been choosed by user.

5.5 Evaluation Function

In this section, we describe the evaluation function for each sorting criteria that is provide in the system.

- **Time** means the total journey time which is the sum of flight duration with transit time (if any).
- **Reliability** is the average reliability of the airlines involved in the route.
- **Price** is the sum of product between distance with price index (computed for each flight leg).

The detail page as in Figure 6.6 shows the details time, price, and reliability for each flight leg.

5.6 Scalability

In this section we present the approaches that are implemented in the system to ensure its scalability.

5.6.1 Limit The Search to Two Transit

We need to limit the search to two transit for two reasons. First, SQL is not a deductive database system. It means that it can only search for routes from limited-specified predetermined number of transits. Second, by limiting to two transit, we limit the complexity to $O(n^3)$. We observe that in real world, two transit can already cover a large part of all possible pair of airports.

5.6.2 Table Design

We split our tables that contain the routes information into two tables, `Connections` and `Schedules`. Table `Connections` is simply used for knowing the existence of direct routes and to find one-transit and two-transit routes. Table `Schedules` contains the exact schedule of each flight, including airline ID, departure and arrival time, and day of operations. By separating the tables, we avoid the extra processing time to process `DISTINCT` query from table `Schedules` to retrieve possible connections.

5.6.3 Limiting the Number of Possible Routes

This is done by using the hub identification heuristic which only considers several routes through hubs that are expected to contain the best or close-to-the-best route.

5.6.4 Limiting the Number of Flight Combinations

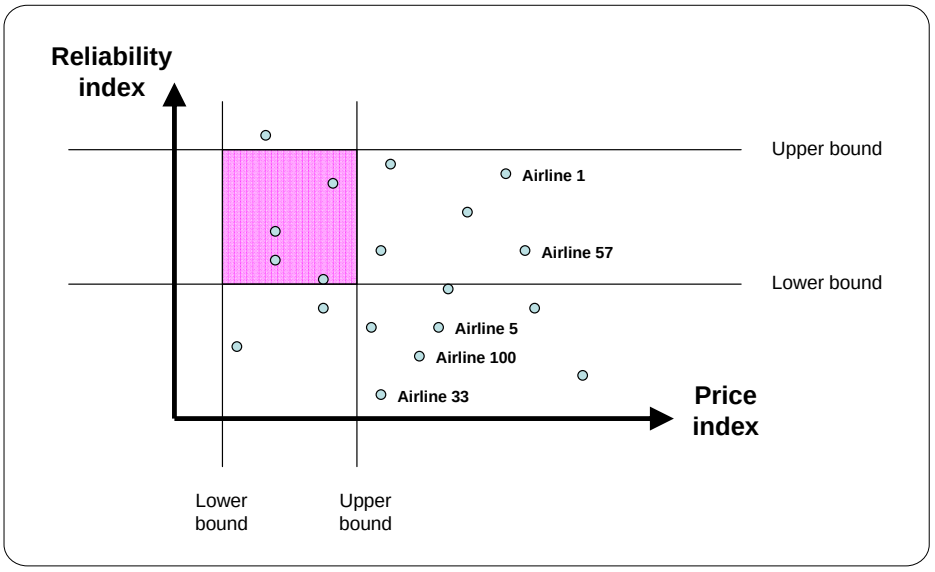


FIGURE 5.11: Limiting the airlines included in the search by putting bounds on price and reliability index

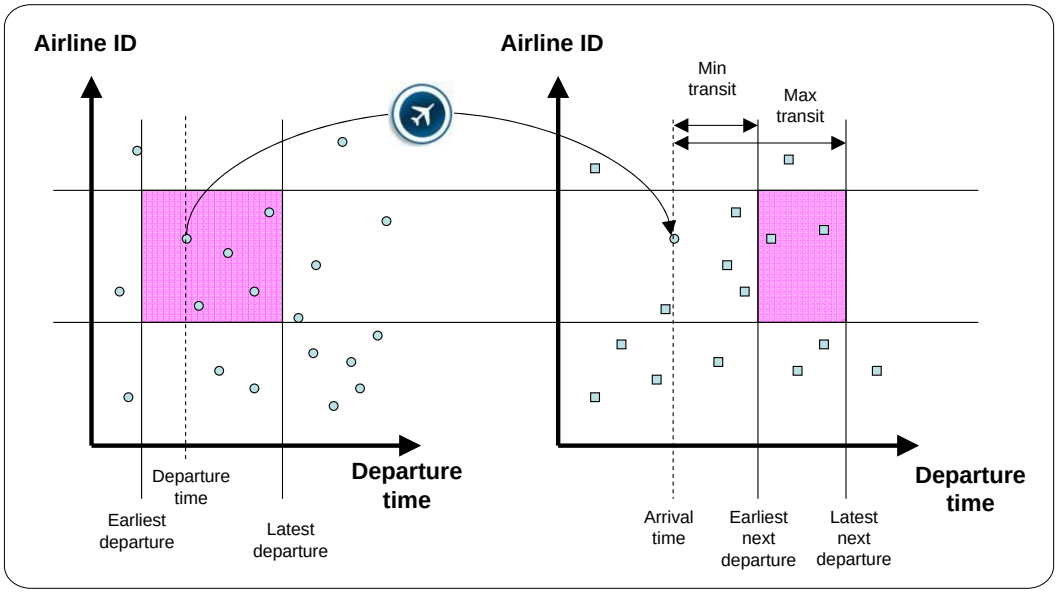


FIGURE 5.12: Limiting the airlines included in the search by putting bounds on transit time

In our complexity discussion previously, the second factor that affects total complexity is the number of edges between two airports. The more number of edges, then the more combinations that need to be checked. There are two techniques that we use to limit the number of combinations.

First, we limit the number of airlines considered in a route (Figure 5.11) by putting lower and upper bound of price and reliability index. We can ignore flights from airlines that do not match the price and reliability criteria posed by the user. For example, user that wants to only search from low-cost carriers can configure to only search for airlines with price index less than 5. Only airlines meeting the constraint are included in the route search. In this way, the number of combinations can be reduced.

Second, we put bounds on minimum and maximum transit time for connecting flights. We provide configuration option both for one-transit case and two-transit case. Figure 5.12 shows the idea. The system does not consider all flights during that day, but only flights within the transit time range. Hence, the number of combinations is also reduced.

Chapter 6

Experimental Results

6.1 Random Schedule Generator

In chapter 4 we have discussed about wrapping airlines where we collect the information on departure airport, destination airport, departure time, and arrival time. We wrap ten airline websites to get the data for the knowledge base of the generator.

From this data, we can calculate the distance between the two airports (by using geocoding functions) and the flight duration (by using the GMT time zone offset value at each airport). Hence, if we plot the distance and flight duration in 2-dimensional plane, we can obtain the function of time versus distance as shown in Figure 6.1 and 6.2.

We use least squares method to measure the equation function. By observing the points in the plane, the most appropriate least squares method to be implemented is the linear method. By having the time as function of distance, then we can measure the flight duration, given the departure and destination airport.

We use this information to create a random schedule generator which functions to generate arbitrary schedule for our virtual airlines so that we can test the system with a lot of airlines. The generator is build by using Java technology, with the following configurations:

- Maximum flights per day is limited to 5.
- The earliest departure time for the first flight is 6 AM.
- The latest departure time for the first flight is 10 PM.
- In case where there are more than 1 flight per day, per airline, the departure time is distributed evenly, for example 2 hours between each flight.

- Since we are using mathematical function, then the result is exact. For the same pair of departure-destination airport, the flight duration is always the same. In real-world, for same route, different airlines may have different flight duration. Therefore, we add some random time interval offset to the function so that there exisst variation of flight durations between different airlines.
- Each day has 75% chance that the flight operates on that day. In a week, the flight has to operate on at least one day.

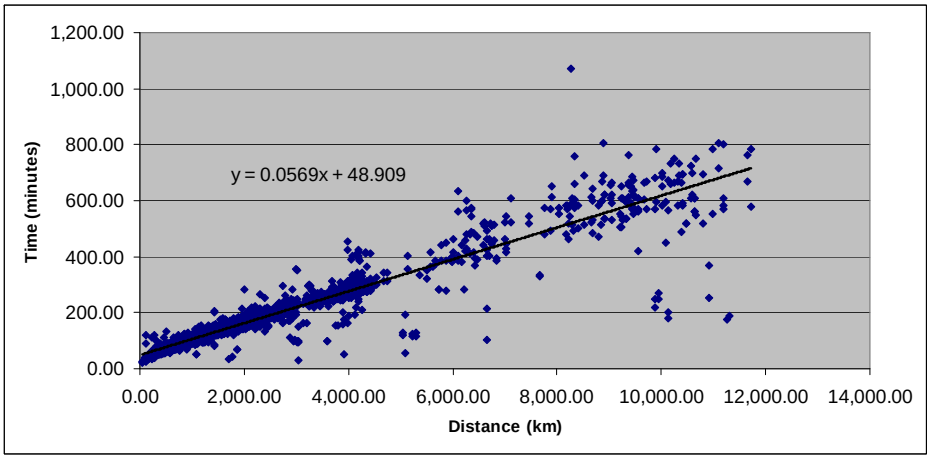


FIGURE 6.1: Least squares function for distance less than 12,000 km

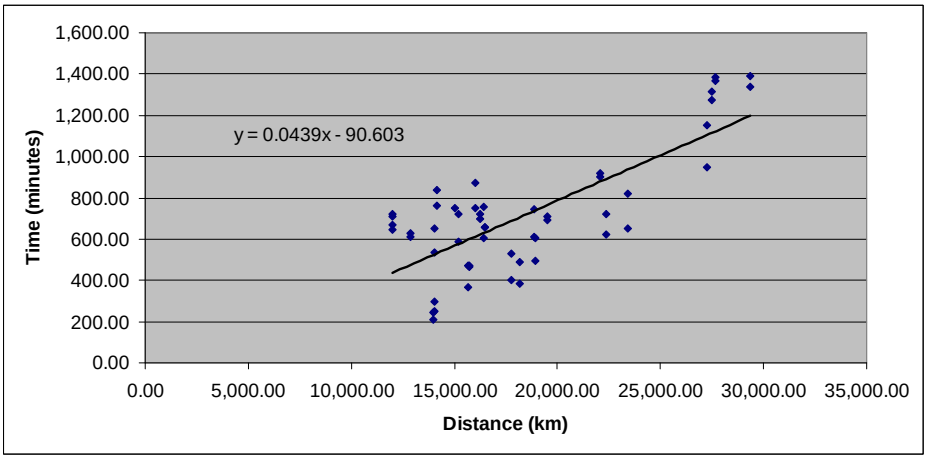


FIGURE 6.2: Least squares function for distance greater than 12,000 km

Once we have created this generator, then we can create various graph model (e.g. hub and spoke, full mesh, partial mesh, etc) for testing the system. We can also, for example, wrap the direct routes from other airlines (without wrapping their departure and arrival time) then use this data as input to the generator to generate random schedule.

6.2 Search Parameter

6.2.1 Common Search Parameter

Common search parameter contains the fields that needs to be inputted by user as the basic requirements to start the search. These parameters are:

- Departure airport.
- Destination airport.
- Departure day.

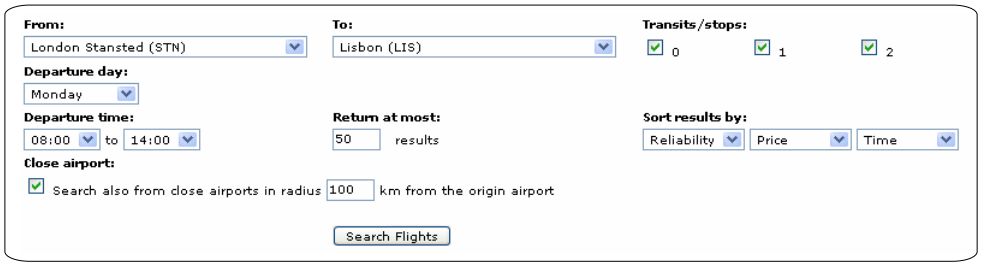


FIGURE 6.3: Common search parameters

There are also some additional parameters, which is optional whether to be configured or not by the user. If it is not configured, then default values are used. Figure 6.3 shows the user interface for common search parameters.

- Range of departure time.
  To consider only flights departing between the specified hours. Default value is from 8:00 to 14:00.
- Number of maximum results to be returned.
  Default value is 50 results.
- Type of routes to be searched (0 or 1 or 2 transit).
  Default value is 0 and 1 transit.
- Criteria for sorting the results.
  Default value is Time, Price, and Reliability.

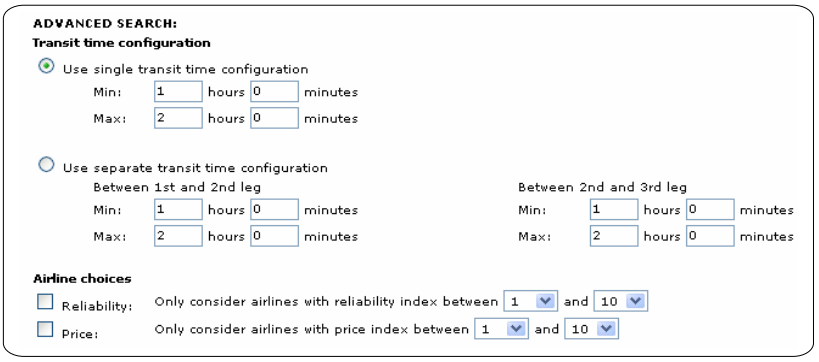


FIGURE 6.4: Advanced search parameters

6.2.2 Advanced Search Parameter

Advanced search parameters are the parameters that can be configured for fine-tuning the search. Figure 6.4 shows them. There are several parameters:

- Single transit time configuration.
  This transit time configuration are used only for routes with transit. By using single transit time configuration, the same transit time constraint is used for transit between first and second leg and for transit between second and third leg.

- Separate transit time configuration.
  By using separate transit time configuration, then transit time constraint between first and second leg and between second and third leg is different. For example, if the user arrives at first transit at night (for example, 11 PM), and wants that the next flight to be in the morning, then he can configure the minimum transit time between first and second leg to be 7 hours (so that the next departure is at least 6 AM on the next day). Between second and third leg, he can configure the transit time to be between 1 and 2 hours.

- Airline choices.
  By configuring this parameter, user can choose, for example to only consider flights from low-cost airlines (by limiting the price index value, for example between 1 and 5). They can also try to search from low-cost and reliable airlines (by limiting both price and reliability index).

6.3 Performing the Search

In this section, we show the steps for doing the search for routes from London Stansted to Lisbon. The start page of the search is already shown at Figure 6.3 and 6.4. Summarizing the parameters:

- Departure day: Monday.
- Departure time: between 8:00 and 14:00.
- Search for 0, 1, and 2 transit routes.
- The results are sorted based on Reliability, Price, and Time.
- Search also for routes from other airports which is within 100 kilometers distance from London Stansted.

We post the search, and get the list of possible routes as shown in Figure 6.5. In the leftmost of each route, there is a checkbox where user can choose whether to include the route in the search or not.

This checkbox is also useful if user wants to avoid certain transit airports or has some preference on the routes. Since we mention to search also from close airports, then the figure also shows routes from London Heathrow, Gatwick, City, and Luton in the options. The link to display the route map is also shown. By clicking on the link, user can view the route map.

The last page shown in Figure 6.6, contains the search results, sorted by user criteria. Since our first criterion is Reliability, then in the results, flights from reliable airlines dominate the top rank. If the first criterion is Time, then direct flights would be in top ranks. If the first criterion is Price, then direct flights from low-cost airlines would be in top ranks.

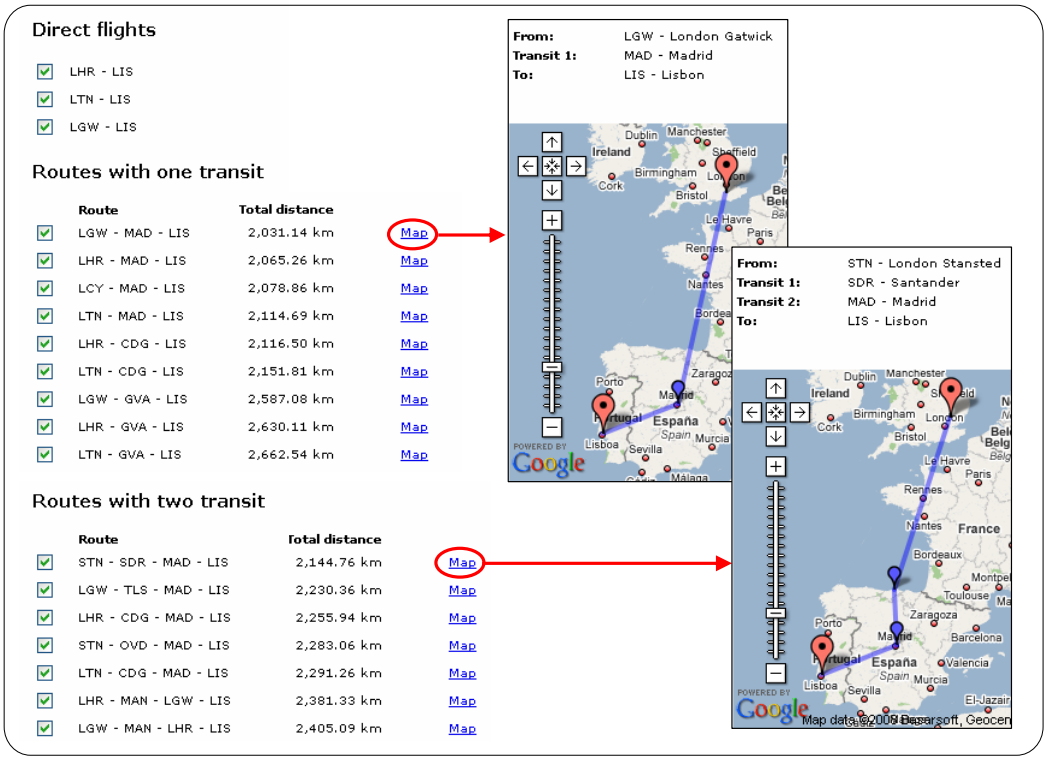


FIGURE 6.5: List of possible routes from London Stansted to Lisbon

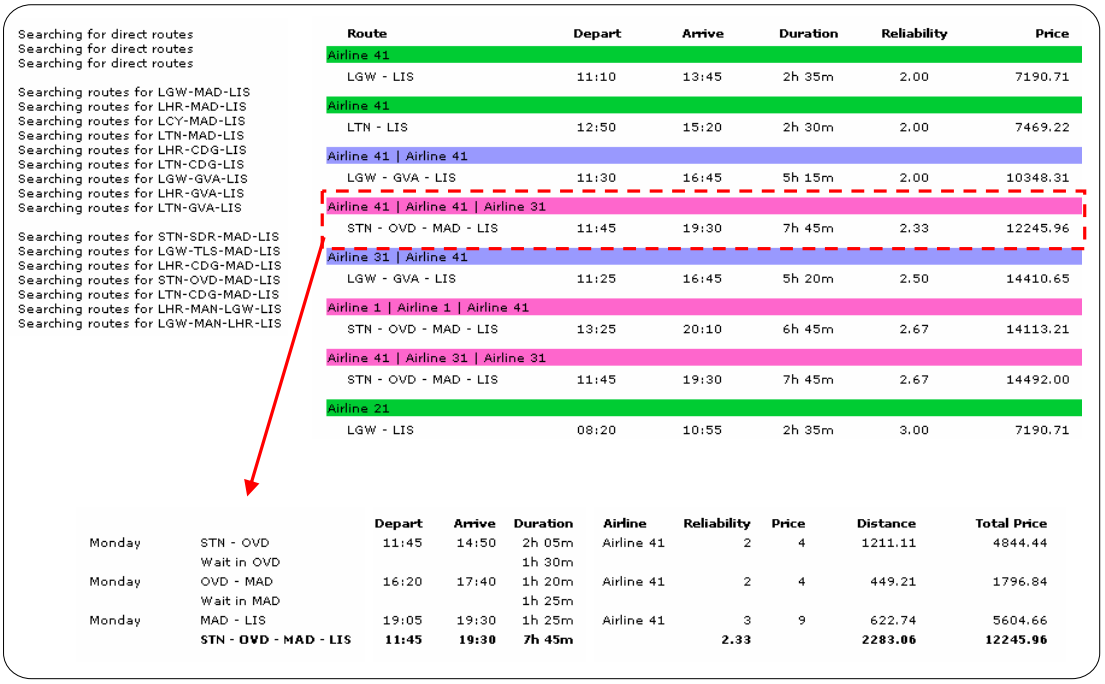


FIGURE 6.6: Search results for flights from London Stansted to Lisbon, sorted by Reliability, Price, and Time

Chapter 7

Conclusions

7.1 Summary

Figure 7.1 shows the relation of the topics discussed in this thesis. The goal of the thesis is to present the mashup model of a scalable flight search system that addresses the weaknesses of the flight meta-search engine.

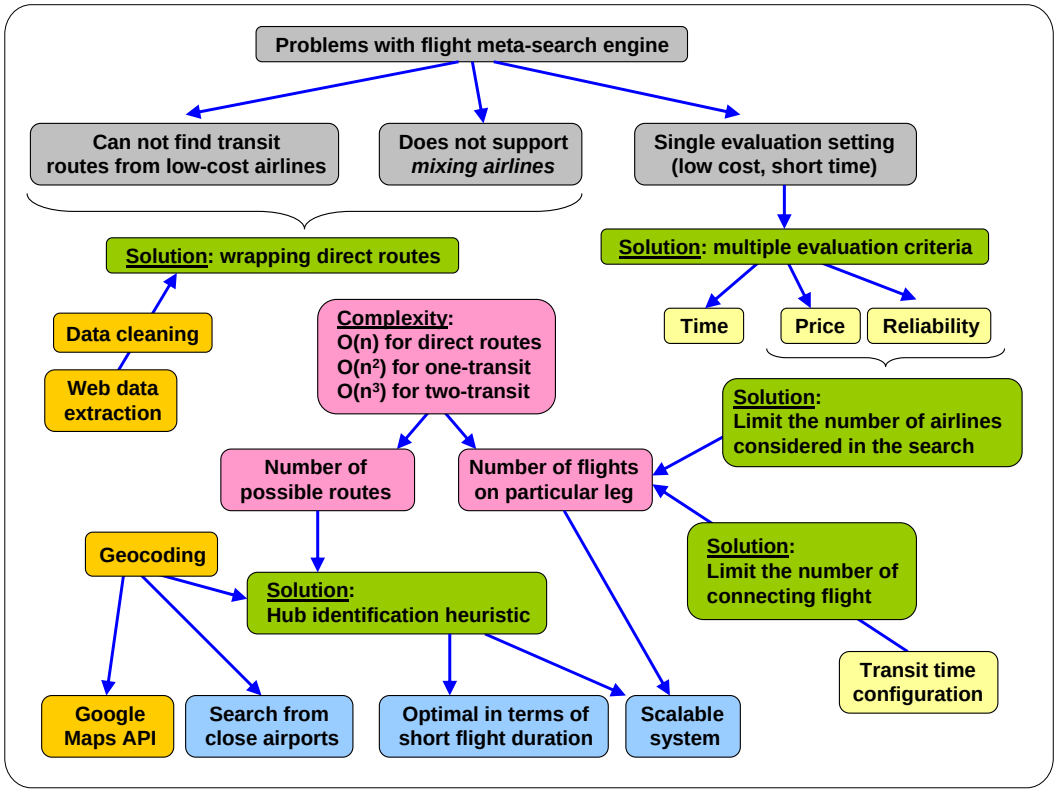


FIGURE 7.1: Summary diagram of the thesis

We start from the problems of flight meta-search engine. The first two weaknesses: incapability of finding transit routes from low-cost airlines and incapability of finding routes by mixing airlines, can be solved by wrapping direct routes from the airline websites by using web data extraction technology from Lixto. In the explanation, we have shown some examples of wrapping by using Lixto Visual Developer. We also show the data cleaning effort to ensure that the data which are populated to database are in correct format and do not contain unnecessary annotations. The third weakness: single evaluation setting, is solved by employing three evaluation criteria for evaluating and ranking the search results. Hence, each user can customize the search to fit with his/her criteria and finding the best routes of their own version.

By having direct routes, the flight search problem can be regarded as graph search problem with airports as the nodes and direct flight connection between two airports as the edges. Since we know all the direct routes, then given the departure and destination airport, we can find all route possibilities: direct, one-transit, and two-transit. For transit routes, we independently search the flights for each leg so that the first two weaknesses are addressed. The respected complexity for finding each route is shown in the diagram. The variable n in the diagram is composed by two other factor: the number of possible routes and the number of flights on particular leg.

To reduce the number of possible routes, we introduce hub identification heuristic which able to give indication on which routes should be traversed to check the flight combination for that route. This heuristic works dynamically so that it may give different hub recommendations for each pair of departure and destination airport. It is also able to set its own limit on how many routes to search by analyzing the possibility of whether the route has possibility of containing the best routes.

By using this heuristic, we avoid exhaustive search for all possible flight combination from all possible routes. Hence, this heuristic realizes our goal of system scalability. The heuristic is created by using geocoding approach (total journey distance approach). In the explanation, we have shown that the heuristic is optimal in terms of flight duration, so that it always finds the routes with the shortest flight duration. For the total journey time, in case of high availability of possible routes and number of flights to reach a destination, this heuristic may miss the route with shortest total journey time (flight duration added with transit time). However it would still find routes that are close to the route with shortest total journey time.

To reduce the number of flights on particular leg, we limit the number of airlines considered in the search by providing user the advanced input parameter, for example to choose only for flights from low-cost and reliable airlines. Airlines that don't meet the criteria for price index and reliability index are not considered in the search. The transit

time configuration where user can input the range of minimum and maximum transit time also limits the number of possible connecting flights. Hence, by providing customized search to user, the system can narrow the search by putting user constraint in the search process. If user doesn't put any constraint on the price and reliability, the system is still scalable since hub identification heuristic is always used in any case.

7.2 Future Works

This section will give idea on several possible future works that can be done for continuing the contribution that has been achieved from this thesis.

7.2.1 Validate the Interesting Route Output

Figure 7.2, shows another system, *Data Retriever* that functions to validate the interesting route output from our *Flight Search System*. Hence, what user gets as search results is valid schedules, complete with availability, and real-time price.

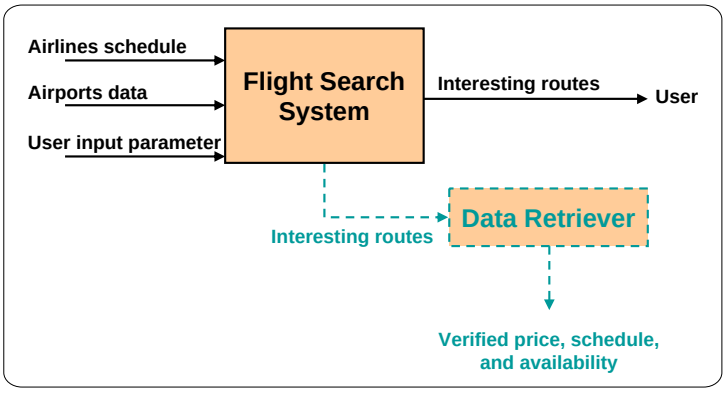


FIGURE 7.2: Future work for the system

7.2.2 Extend the Close Airport Feature

In this thesis, we only provide the feature of *searching from close airport* from the initial point of departure (left part of Figure 7.3). The right part of Figure 7.3 shows the possible extension. If A and B are the good hubs from the output of hub identification heuristic, then we also search for routes through other airport that are close to A and B. The destination itself can be several airports. Simply observing, the complexity may blow up by applying this feature. However, in real-world implementation this feature may be desired.

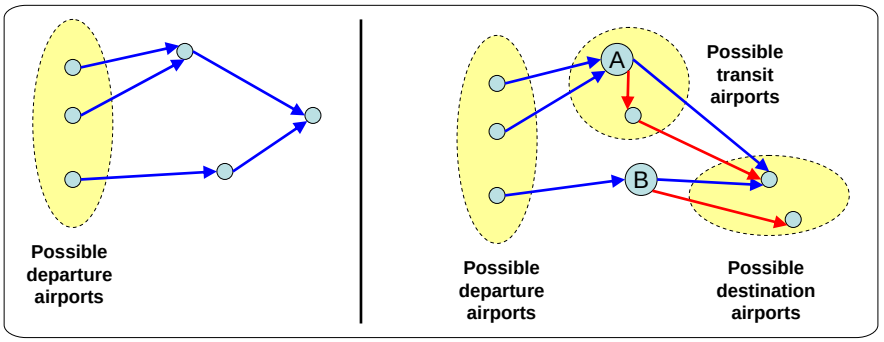


FIGURE 7.3: Extending the close airport feature, (the left picture is the one used in this thesis, the right picture is the possible extension)

We give the real-world example as follows. Currently there is no direct flight from Lisbon to Rome Ciampino. However, there are many direct flights from Lisbon to Rome Fiumicino. In our current system, the system will find all the possible routes from Lisbon to Rome Ciampino, but will never suggest the route from Lisbon to Rome Fiumicino. By applying this new feature, then the route to Rome Fiumicino would be suggested and user can decide whether the alternative routes pay-off the switching of destination airport.

Another example that involves transit route is as follows. Suppose someone wants to fly from Vienna to Singapore. This person is a budget traveler, and was having good experience with Easy Jet so that he decides to use Easy Jet again. Easy Jet has only one destination for flights departing from Vienna, that is London-Luton. Speaking for intercontinental flights, London Heathrow has more options compared to London Luton. Therefore, considering connecting flights from London Heathrow would be a useful feature.

7.2.3 Cost to Switch Between Airports

In relation with the close airport feature, until now we always omit the effort that is needed to switch between airports. There are several possible resource from where to get the cost. First, geocoding functions can be used to obtain the distance between the airports, then employing a function of cost versus distance. Second, to get the cost from Google Maps (Figure 7.4) from which we can put the start and end address, and get the journey time (by using car).

FIGURE 7.4: Measuring the journey time by using car with Google Map

7.2.4 Extend the Travel Domain

Travel domain is composed from many sub domains, flight is one example of sub domain. Hence, there is possiblity to extend our flight search system with other applications in hotel/hostel sub domain or train sub domain so that the dynamic packaging concept can be realized.

For these sub domains, the challenge will be on aggregating the results and identifying the same records. In flights domain, each airport has an unique 3-letter IATA code so that we always know exactly which airport we are dealing with. However, in train stations/stops and hotel/hostel, there are no such identifier. Therefore, the concept of *record linkage* may need to be implemented here.

Appendix A

Wrapping Airports Data

This Appendix contain details on wrapping airports data from World Airport Codes website.

A.1 Navigation Sequence

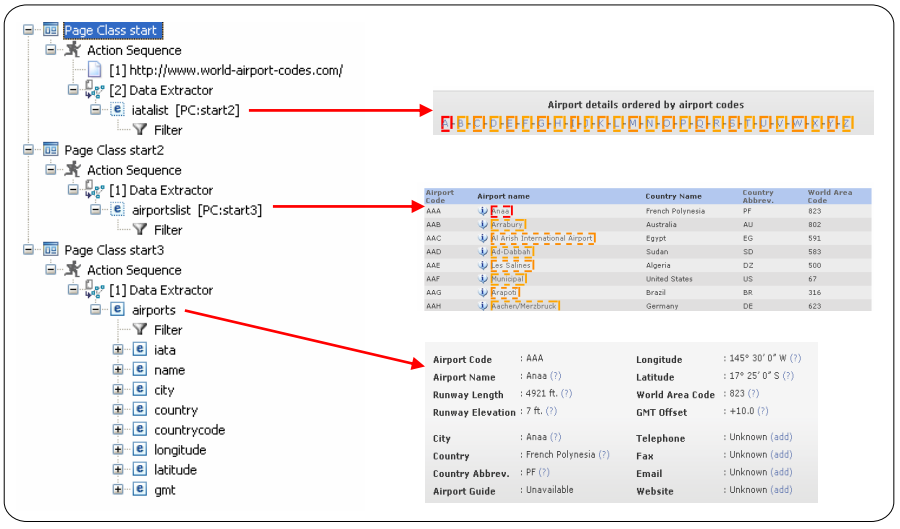


FIGURE A.1: Navigation sequence in Lixto VD

In the wrapper, we build three page class. Each page class corresponds to a page structure in the website. Below, we describe what is done by each action sequence in Figure A.1:

- Pageclass start

1 URL action, instructing the browser to go to website of WAC.

2 Data extractor, containing a pattern named "iatalist". For each element in "iatalist", the wrapper continues to pageclass start2 (Figure A.2).

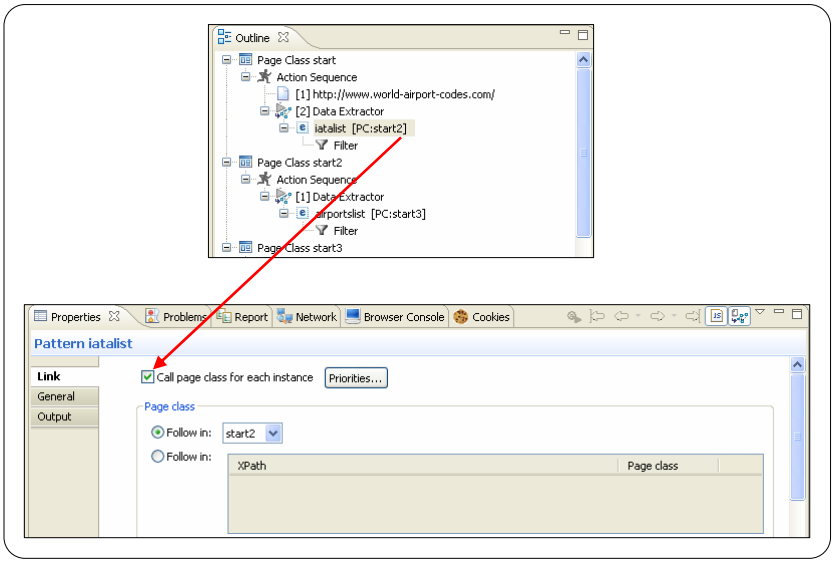


FIGURE A.2: Calling pageclass start2 from "iatalist"

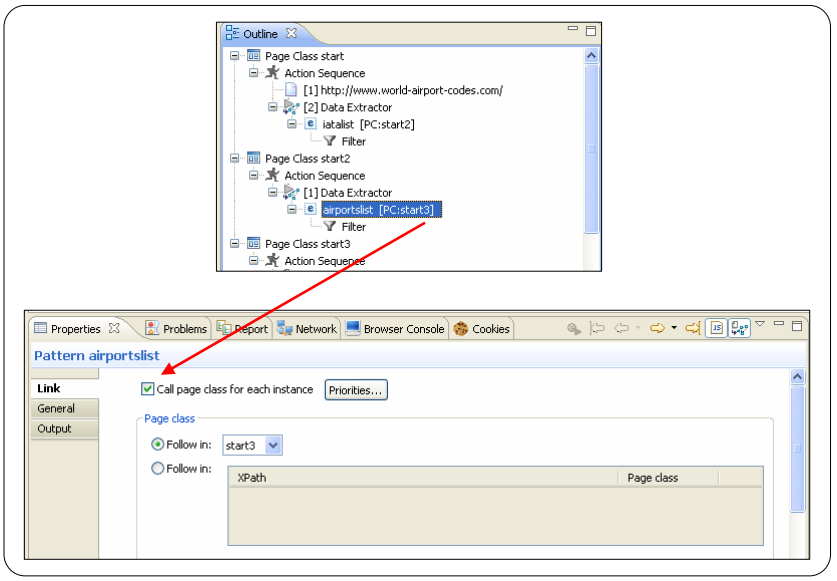


FIGURE A.3: Calling pageclass start3 from "airportslist"

- Pageclass start2

1 Data extractor, containing a pattern named "airportslist". For each element in "airportslist", the wrapper continues to pageclass start3 (Figure A.3).

- Pageclass start3

  1 Data extractor, containing a pattern named "airports" (using Lixto data model). Inside this pattern, there exist the other sub-patterns.

A.2 General XPath Expression

The XPath expression of any element in the HTML page can be obtained by creating a pattern, then choose the element using the filter. Figure A.4 shows how it is done.

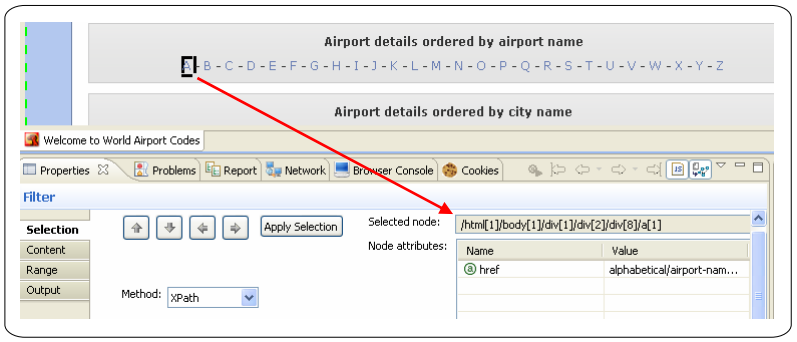


FIGURE A.4: Getting XPath expression of an element

In the "iatalist" and "airportslist" patterns, we use XPath filtering to recognize the elements. Figure A.5 and A.6 illustrates how to build the general XPath expression.

By observing the figure, we get the general XPath expression for "iatalist" is:
`/html[1]/body[1]/div[1]/div[2]/div[7]/a`.
For "airportslist", the general XPath expression is:
`.../tbody[1]/tr/td[3]/a[1]`.

FIGURE A.5: XPath for elements of "iatalist"

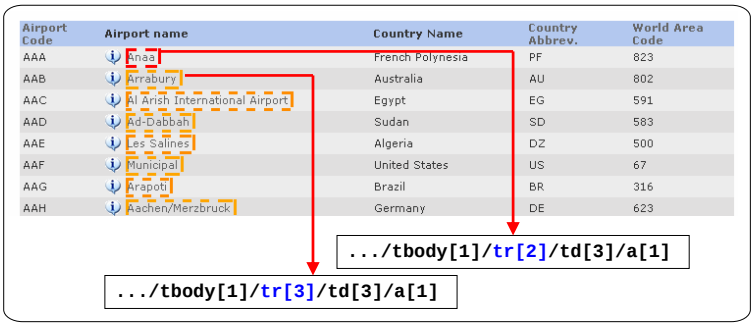


FIGURE A.6: XPath for elements of "airportslist"

A.3 Configuring XPath for Iterators

Finally, the general XPath can be put in the "Content" tab of the "iatalist" filter as shown in Figure A.7.

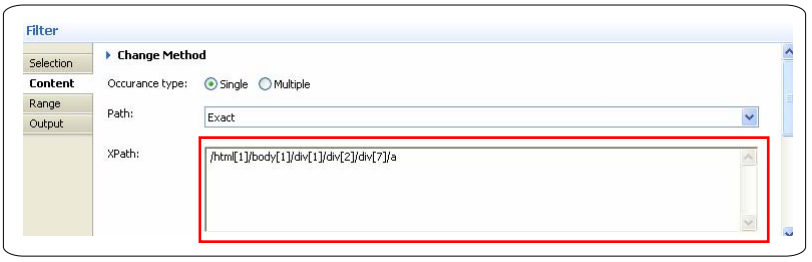


FIGURE A.7: Putting general XPath expression for "iatalist"

Bibliography

[1] A. Ankolekar, M. Krötzsch, T. Tran, and D. Vrandečić. The two cultures: Mashing up web 2.0 and the semantic web. *Web Semant.*, 6(1):70–75, 2008. ISSN 1570-8268. doi: http://dx.doi.org/10.1016/j.websem.2007.11.005.

[2] R. Baumgartner, M. Ceresna, G. Gottlob, M. Herzog, and V. Zigo. Web information acquisition with lixto suite: A demonstration. *icde*, 00:747, 2003. ISSN 1063-6382. doi: http://doi.ieeecomputersociety.org/10.1109/ICDE.2003.1260855.

[3] R. Baumgartner, M. Herzog, and G. Gottlob. Visual programming of web data aggregation applications, 2003. URL `citeseer.ist.psu.edu/baumgartner03visual.html`.

[4] R. Baumgartner, M. Ceresna, and G. Ledermuller. Deepweb navigation in web data extraction. In *CIMCA '05: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06)*, pages 698–703, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2504-0-02.

[5] C. Bernardoni, G. Fiumara, M. Marchi, and A. Provetti. Declarative web data extraction and annotation. In *WLP*, pages 137–144, 2006.

[6] M. Ceresna. Interactive generation of html wrappers using attribute classification. In *Proceedings of the First International Workshop on Representation and Analysis of Web Space*, page 137142, Prague-Tocna, Czech Republic, 2005.

[7] R. Collis. No frills? full service? meet the hybrid carrier. Online, May 2008. URL `http://www.iht.com/articles/2008/05/30/travel/trfreq30.php`. International Herald Tribune, THE FREQUENT TRAVELER, last accessed on 5 June 2008.

[8] Dohop. Connecting low-cost airlines. Online, August 2007. URL `http://blog.dohop.com/?p=44`. Last accessed on 1 June 2008.

[9] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1):1–16, 2007. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2007.9.

[10] S. Flesca, G. Manco, E. Masciari, E. Rende, and A. Tagarelli. Web wrapper induction: a brief survey. *AI Commun.*, 17(2):57–61, 2004. ISSN 0921-7126.

[11] G. Gottlob. Web data extraction for business intelligence: The lixto approach. In *BTW*, pages 30–47, 2005.

[12] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The lixto data extraction project: back and forth between theory and practice. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–12, New York, NY, USA, 2004. ACM. ISBN 158113858X. doi: http://doi.acm.org/10.1145/1055558.1055560.

[13] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web. *Commun. ACM*, 50(5):94–101, 2007. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1230819.1241670.

[14] M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998. URL `citeseer.ist.psu.edu/article/hernandez98realworld.html`.

[15] W. Holzinger, B. Kruepl, and R. Baumgartner. Exploiting semantic web technologies to model web form interactions. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 1145–1146, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: http://doi.acm.org/10.1145/1367497.1367698.

[16] T. Hornung, K. Simon, and G. Lausen. Mashing up the deep web - research in progress. In *4th International Conference on Web Information Systems and Technologies (WEBIST) 2008*, pages 58–66, Funchal, Madeira - Portugal, May 4-7, MAY 2008.

[17] M. Kayed and K. F. Shaalan. A survey of web information extraction systems. *IEEE Trans. on Knowl. and Data Eng.*, 18(10):1411–1428, 2006. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2006.152. Member-Chia-Hui Chang and Member-Moheb Ramzy Girgis.

[18] S. Kuhlins and R. Tredwell. Toolkits for generating wrappers – a survey of software toolkits for automated data extraction from web sites. In M. Aksit, M. Mezini,

and R. Unland, editors, *Objects, Components, Architectures, Services, and Applications for a Networked World*, volume 2591 of *Lecture Notes in Computer Science (LNCS)*, pages 184–198, Berlin, Oct. 2003. International Conference NetObjectDays, NODe 2002, Erfurt, Germany, October 7–10, 2002, Springer. URL `citeseer.ist.psu.edu/kuhlins02toolkits.html`.

[19] H. Leggatt. European travel industry doing more business online. Online, March 2007. URL `http://www.bizreport.com/2007/03/european_travel_industry_doing_more_business_online.html`. Last accessed on 1 June 2008.

[20] Lixto. *Lixto Visual Developer Tutorial 5.0*, chapter 1, page 8. Lixto GmbH, 2008.

[21] D. Merrill. Mashups: The new breed of web app. Online, October 2006. URL `http://www.ibm.com/developerworks/xml/library/x-mashups.html`. Last accessed on 10 June 2008.

[22] J. Myllymaki. Effective web data extraction with standard XML technologies. In *World Wide Web*, pages 689–696, 2001. URL `citeseer.ist.psu.edu/myllymaki01effective.html`.

[23] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994. ISBN 0201530821.

[24] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 129–138, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4.

[25] E. Rahm and H. Do. Data cleaning: Problems and current approaches. *Data Engineering Bulletin*, 23:3–13, 2000.

[26] E. Rosenthal. Low-cost airfares, big-time carbon footprint. Online, May 2008. URL `http://www.iht.com/articles/2008/05/30/news/spain.php?page=1`. International Herald Tribune, last accessed on 5 June 2008.

[27] G. Trapani. Where to search for low airfares online. Online, November 2007. URL `http://lifehacker.com/software/travel/where-to-search-for-low-airfares-online-319404.php`. Last accessed on 1 June 2008.

[28] M. Tvarozek and M. Bieliková. Personalized view-based search and visualization as a means for deep/semantic web data access. In *WWW*, pages 1023–1024, 2008.

[29] M. Verikios. Dusseldorf international to invest some 300 million euros in the future. Online, March 2008. URL `http://www.traveldailynews.com/pages/show_page/24747`. Travel Daily News, last accessed on 1 June 2008.

[30] L. Wei, X. Meng, and W. Meng. Vision-based web data records extraction. In *WebDB*, 2006.

[31] Wikipedia. Great-circle distance. Online, 2007. URL `http://en.wikipedia.org/wiki/Great_circle_distance`. Last accessed on 5 June 2008.

[32] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 66–75, New York, NY, USA, 2005. ACM. ISBN 1-59593-046-9. doi: http://doi.acm.org/10.1145/1060745.1060760.

[33] H. Zhao, W. Meng, and C. Yu. Mining templates from search result records of search engines. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 884–893, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-609-7. doi: http://doi.acm.org/10.1145/1281192.1281286.