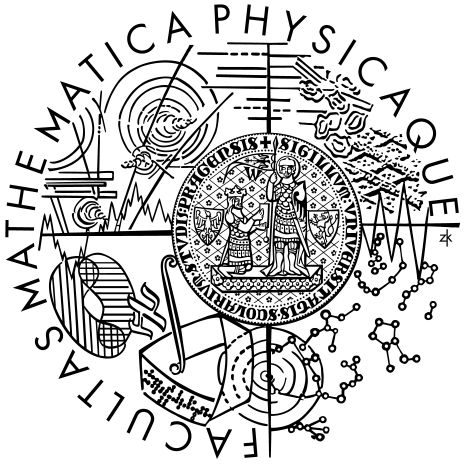
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Michal Bečka

Plánování spojů ve veřejné dopravě na mobilních zařízeních

Katedra softwarového inženýrství

Vedoucí diplomové práce: *RNDr. Viliam Holub, Ph.D.*

Studijní program: *Informatika, softwarové systémy, systémové architektury*

2010

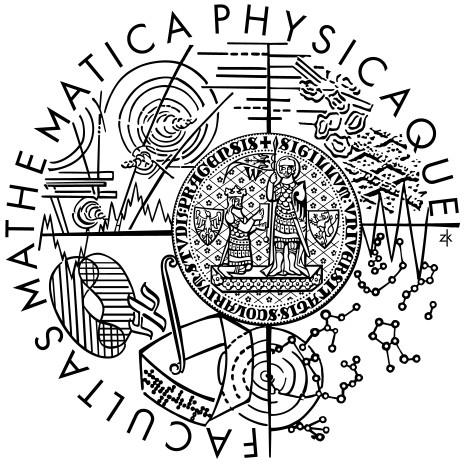
Rád bych poděkoval všem, co mě podporovali během psaní této diplomové práce, především za jejich trpělivost při dlouhých dnech mého psaní a studia materiálů. Zvláště bych rád poděkoval svému vedoucímu RNDr. Viliamu Holubovi, Ph.D. za výběr tématu a trpělivé vedení mé práce.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne *16. 4. 2010* *Michal Bečka*

Charles University in Prague
Faculty of Mathematics and Physics

Diploma thesis



Michal Bečka

Mass transport routing

Department of Software Engineering

Supervisor: *RNDr. Viliam Holub, Ph.D.*

Branch of study: *Informatics, software systems, system architectures*

2010

I would like to thank all that supported me during my work on this thesis for their patience in the long days of my writing and study. I would especially like to thank my supervisor RNDr. Viliam Holub, Ph.D. for the selection of the subject and patient supervision of my work.

I declare that I have written my diploma thesis on my own and with exceptional use of quoted sources. I agree with lending of my work.

In Prague on *16th of April 2010* *Michal Bečka*

Table of Contents

Název práce: *Plánování spojů ve veřejné dopravě na mobilních zařízeních*

Autor: *Michal Bečka*

Katedra: *Katedra softwarového inženýrství*

Vedoucí diplomové práce: *RNDr. Viliam Holub, Ph.D.*

E-mail vedoucího: *Viliam.Holub@mff.cuni.cz*

Abstrakt: *Plánovaní spojů přístupné z mobilního zařízení poskytuje cestovní informace v pohybu. Tato práce studuje různé oblasti takových mobilních aplikací, včetně historie mobilních zařízení a jejich limitujících vlastností. Nejdříve zkoumáme spojení s telefonními sítěmi a Internetem pro možnosti umístění tohoto nástroje, přičemž doporučíme nejvhodnější možnosti. Potom studujeme vývojové prostředí, složené z mnoha operačních systémů, pro zjištění kompatibility aplikace při zvolení programovacího jazyka. Další část práce pokrývá možné parametry při hledání tras, od základního zadání místa a času po možnosti optimalizace jiných vlastností cesty. Následuje porovnaní s již existujícími vyhladávači. Dále zkoumáme zdroj a formát vstupních dat o dopravě spolu s jejich dostupností. Nápady, jak je možné implementovat navržené parametry, ilustrujeme na příkladu vyhledávacího algoritmu. Cílem je podat obsáhlý přehled tématu přičemž jsou studována možná vylepšení.*
Klíčová slova: *plánovač spojů, mobilní zařízení, cestovní informace*

Title: *Mass transport routing*

Author: *Michal Bečka*

Department: *Department of Software Engineering*

Supervisor: *RNDr. Viliam Holub, Ph.D.*

Supervisor's e-mail address: *Viliam.Holub@mff.cuni.cz*

Abstract: *A journey planning tool accessible from a mobile device provides travel information on the move. This work studies various aspects of mobile journey planners, including the history and limitations of mobile devices to deal with. First we analyze the connection with phone networks and the Internet for possible choices to place the tool, while recommending the most suitable ones. Then we study the developer environment consisting of various operating systems to show how compatible an application will be while selecting a programming language. Another part covers possible parameters for the journey search, from setting of basic location and time to optimizing other journey attributes. Following is an evaluation of journey planners already available. Then we investigate the source and form of input traffic data along with their availability. Finally we illustrate the ideas how the proposed parameters can be implemented on an example search algorithm. The purpose is to provide a comprehensive overview of the subject while researching possible improvements.*
Keywords: *journey planner, mobile device, traffic information*

1. Introduction

A journey planner is a service that became common in recent years. Since usage of mobile devices spread quickly, it is not surprising these two technologies combined their strengths. Now people can find travel information while actually traveling. In this work, we will have a look at how such journey planner can be implemented and what difficulties it has to overcome.

What is a journey planner? Sometimes called a trip planner, it is an application that can find a way to travel by public transport from one place to another at specified time. Apart from these basic parameters, there can be several more, like what type of transport the user wants to use, for example a train, a bus, a tram or just some of them. User can also specify the type of transport in more detail, like the quality of the vehicle, or disability support. If he wants to specify the route, he can add a list of stations he want to pass through and other parameters.

After all the parameters are entered, user submits them and waits for the results. Results will include the best trip found, or several trips. Alternatively the user can get the best result at first and if it is not enough for him, he can request some more trips, either preceding ones or some following ones. He can also choose to modify the parameters of the search, in case he misspelled something, he wants to specify his search or just to find something a little different.

One unusual possible aspect of the search that will be discussed in this work will be a trip reliability.

The work consists of the following sections. Mobile devices are covered first, with their history and capabilities, followed by how a journey planner can be placed on them with development tool choices. Notable existing journey planners are evaluated. Another part covers search parameters user can specify, followed by the sources and form of input traffic information. Another part explains search algorithm implementation. The work concludes with the tasks and properties of the user interface.

The purpose of this thesis is to provide deeper understanding of the current situation of journey planning on mobile devices, investigating some new interesting features on the way.

2. Overview of mobile devices

Mobile device is a small computing device, that can fit into a pocket. Usually, a mobile device is equipped with a display for graphic output and a miniature keyboard or a touch screen for input. Different types of mobile devices include pagers, digital cameras, e-book readers, navigation systems, but for the purpose of this work only relevant types of devices will be considered. These are cell phones, PDAs, smartphones and other types of mobile computers capable of running a journey planner tool, or at least accessing it.

2.1. Brief history

2.1.1. Cell phones

Cell phones have a long history, as they were developed during the 20th century. They are devices offering long range communication over wireless telephone networks. First cell phones used analog signals to transport voice only, they were bulky devices that were limited by range, availability to general public and mobility, since they were often mounted on a car. These are sometimes referred to as the “0G” generation. In 1979 the first commercially automated cellular network started in Japan, beginning the “1G”, the first generation of wireless telephone technology standards. Although it used digital signals to manage the calls, the voice transmission was still encoded in an analog signal. The second generation “2G” that launched at 1991 was fully digital and paved the way for the first data services, notably the SMS text messaging. Eventually data services evolved to support a full Internet access. The third generation “3G” launched in 2001, providing mainly a significant increase in data transfer speeds and data services enabled by those speeds, like mobile television, video conferencing and many more. For more detailed information there are various articles, such as [1].

2.1.2. Mobile computers

Different development in the mobile market saw the introduction of hand-held computers, generally called PDA – personal digital assistants, or palmtops. These were the inevitable result of computer miniaturization, when developers took different path from desktops, notebooks and laptops. Instead of focusing on packing more power into the computer of the same size, they retained the power and reduced the computer size. Mobile computers started to appear widely in 1990s and over time included more and more classic computer tools. Often such devices were customized for a specific purpose, like management in warehouses, various record keeping, access control, security or navigation.
PDAs specifically served, as its name suggests, as a personal assistant, providing functions to take notes, a calendar and an address book. Early in the development mobile computers connected to the internet, providing email, fax and later complete web browsing. Current PDAs are mostly fully functional personal computers. See [2] for more information.

2.1.3. Smartphones

Different paths of cell phones and hand-held computers soon intersected and now there are many devices offering features from both sides. These devices are called smartphones. The term smartphone is not strictly defined, smartphones are sometimes considered a type of PDA. Typically a smartphone runs its operating system and provides advanced computer features as well as cell phone capabilities. Standalone PDAs reached their peak around the year 2003 and their market share declined since then in favor of smartphones. Today (2010) about 98% of PDAs sold are smartphones [2].

2.2. Hardware capabilities

The most important features of a mobile device in respect to the application like a journey planner are the memory size and the network connection. The memory size is important for applications running a journey planner on the device. In contrast applications running on remote servers rely on the network connection. Today PDAs can have even several gigabytes

of memory, or at least several hundred of megabytes, with a possibility to add additional memory by an SD card or a memory stick. But devices on the market vary greatly, from these extreme capacities to very little or no free capacity. When there is not enough memory to accommodate a tool, mobile devices still offer data services to remote locations, that can be used by a tool.

2.3. Summary

This chapter showed that mobile devices are relatively new with rapid development in multiple directions. Even when a single mobile device does not have a long lifetime, it is still enough for the development to make devices obsolete before they stop working. That means that the differences in mobile devices in use range greatly. Developers of mobile applications therefore have to provide support not just for what devices are best selling at the time.
With basic hardware covered, the next chapter can study how a mobile application can be placed or accessed from the device.

3. Journey planner architecture

A mobile device usually connects itself either to the Internet, or to another remote location like a cellular network. This provides the journey planner tool with two main locations to occupy. It can be located on the device itself, completely at a remote location, like the Internet, or at both places, splitting the functionality in two parts. When the tool has at least some part of it at a remote location, it has to use some of the data services the device provides. In this chapter we will look at the advantages and disadvantages of different data services for the use by the journey planner. They will be compared by several criteria and evaluated.

3.1. Data services

3.1.1. List of data services available

First let's list the data services that have a potential to be used by the tool. In addition, a data service will be listed in a role it would take in the tool architecture. If it would be used only as a presentation layer with no tool part on the mobile device, or if it would be used as a means of communication between two parts of the tool, partially hidden from the user.

SMS service (presentation layer)

SMS service (internal communication) – Originally part of the GSM standard, it has expanded into other mobile technologies. An SMS service is now the most widespread way to send short text messages from a mobile device. A tool must use additional service, an SMS gateway, to handle the messages between itself and the user. This allows the tool to be placed on the Internet.

Email (presentation layer)

Email (internal communication) - An Internet text message service. Both email and SMS are text messaging services, but the main difference between email and SMS relevant to the tool is that SMS connects to the cellular network, while email connects to the Internet. Therefore there is no need for an additional service like the SMS gateway.

Web page connection – A service allowing the tool to be completely placed on the Internet, while it is accessed by a web browser on a mobile device. With web browsing capability naturally comes standard internet connection, so the web page browsing does not have to be considered from the internal communication angle. For less powerful mobile devices a web page access is often done through WAP – Wireless access protocol. It allows the mobile device to connect to a WAP gateway that acts as a proxy between wireless network and the Internet and is transparent to the user. WAP browser works with WML – Wireless markup language, adapted for the lack of resources on a mobile device. WAP gateway translates HTML content into WML. Notable alternative to WAP is the Japanese i-mode. More powerful devices use XHTML MP, or even full HTML.

Internet connection – A pure IP based connection is ideal for the internal tool communication, enabling communication between tool parts in its own native way. Since „2G" networks, General packet radio service (GPRS) is available to users, allowing IP protocol access to the Internet.

No data service – The tool would be located completely on the mobile device.

An adaptation of the tool would be to use a voice communication with either a live person handling search queries, or a voice recognition.

Other services may include MMS – a service allowing to send multimedia messages, but such service is not suitable for specific data transmissions. At most it can be used to provide a nice looking search result, but it is not suitable to send any information from the user.

While it is possible to use different data services at once, one for transmitting and other for receiving, it would only introduce unnecessary complications and compatibility issues. Therefore in this chapter it is presumed that the tool uses only one data service for a single search.

3.1.2. Criteria for comparison

Availability – One of the most important criteria is what percentage of mobile devices the data service can be used on. Other aspect is the relation of its availability with the availability of other services. For example if those services are usually exclusive or are often available at the same time.

Ease of use – It is important whether the user can understand how the service works. This can be easily forgotten by the developer, because he can spend a lot of time around his product, understanding it perfectly, and forgetting how the first time user will see it. People are very diverse in all their aspects, especially in their technical understanding. Some can grasp the workings of a tool immediately, many others can get stuck on some technical quirk. If it is not obvious how the service works, many give up on the service, thinking it is not worth the trouble. When user requires directions, not to mention when he is on the move with the mobile phone, he does not have the time to study the tool. Even when there is time to study it, users are not known for reading manuals.

This chapter is about the data service used. But ease of use would seem to apply mainly for the user interface, the presentation layer of the tool. But the data service selected often cannot be completely hidden for the user. For example, when using the web page service, user is aware that that there is an additional requirement of being connected to the Internet to use the service. Another difficulty is the price of using the service. Even though these examples are overlapping with the criteria of availability and price, in the ease of use they have their influence on how complex or difficult the tool seems. That is why a structure of the tool is important.

Initial cost of obtaining the data service – First of the criteria covering money, this one includes the cost of making the service available at all from the developer side. From the user perspective the data service is already available, the initial cost is none. One way or another, the service will eventually reach an Internet server that is common to all the data services, so the cost of setting up a server with a tool is also ignored, only the data service alone is considered.

Per use cost of the data service – Includes the cost of a single journey query from both the user and the developer perspective. It may vary between service providers.

3.1.3. Analysis

SMS service (presentation layer)

Availability – An SMS service is available on virtually every cell phone and smartphone. On pure PDAs it is still possible to send an SMS through some Internet applications, like ICQ. In relation to other services, SMS can be viewed as the universally available basic function.

Ease of use – Great advantage of SMS messaging universality is that almost everybody can write an SMS message without a problem, with no training necessary. The trouble is the format of a message. A tool on the other side has to parse the information to interpret individual parameters of a query and that requires a specific encoding. A very low portion of users are patient enough to learn the encoding. It may seem unlikely, but many users are also confused by the simplest encoding types and conventions, which a programmer learns even during his first experience with command line arguments. Last significant problem is the availability of help or a manual to teach such encoding. When using only an SMS service, a mobile device will not have a good way to explain it and all will depend on external information.

Initial cost of obtaining the data service – The developer has to use the SMS gateway service. The initial price consist mainly of renting a phone number, which can cost at most 100 USD for setup, and 25 USD as monthly fees, but can be acquired cheaper, it depends on the provider.

Per use cost of the data service – One outgoing SMS message costs on average 0.11 USD [3]. Service providers usually offer discount plans bringing the price lower, without such plans price per SMS is mostly between 0.10 – 0.20 USD. In addition there is a cost per message at the SMS gateway. For commercial gateway it can be around 0.015 USD [4], an alternative is to use your own gateway, such as an open source project Kannel [5].

A reply message can cost around 0.05 USD [4], or in case of own gateway at most a minor switching fee.

SMS service (internal communication)

Availability – The availability of the SMS itself is the same as in the SMS service (presentation layer). However to use the SMS as an internal communication, part of the tool must be on a mobile device, excluding simple cell phones. PDAs, smartphones and newer cell phones allow such tools.

Ease of use – Since the communication is internal to the tool, it is almost completely hidden from the user. User's only concern is the price of the SMS communication as well as a little delay between the query and a reply.

Initial cost of obtaining the data service – The same as in the SMS service (presentation layer), increased by a possible fee for having part of the tool on the mobile device.
Per use cost of the data service – The same as in the SMS service (presentation layer).

Email (presentation layer)

Availability – Unlike SMS, email is not available in pure cell phones, it is limited to devices with more program functions, PDAs and smartphones.
Ease of use – Email provides more freedom than an SMS, allowing longer texts. Problems of encoding persist, but mobile devices supporting email generally provide better ways to help and explain the tool. Additional concern is the need for an active Internet connection, as opposed to ever present SMS connection.
Initial cost of obtaining the data service – Registering an email address, which is either free or for a small fee.
Per use cost of the data service – Cost per email is tied to the cost of data used, depends on the provider, usually much cheaper than an SMS message.

Email (internal communication)

Availability – The same as in the Email (presentation layer). As in SMS (internal communication), part of the tool must be on a mobile device, excluding simple cell phones, where email services are not common anyway. So it is a smaller drawback than for an SMS service.
Ease of use – As an internal communication it is hidden from the user. It is cheaper than the SMS service, which provides less concern for the user. Disadvantage is an additional concern that an active Internet connection is needed, as opposed to ever present SMS connection.
Initial cost of obtaining the data service – The same as in the email (presentation layer), increased by a possible fee for having part of the tool on the mobile device.
Per use cost of the data service – Also the same.

Web page connection

Availability – Like email, it is not available on pure cellphones, but mostly on PDAs and smartphones. Availability of a web page is strongly tied with email (presentation layer). When a web page is accessible, email is too, but not the other way around.

Ease of use – Web browsing allows to present a user friendly interface, making it easy to enter search parameters. Active Internet connection is needed.
Initial cost of obtaining the data service – Registering a domain name for a small fee.
Per use cost of the data service – Cost is tied to the cost of data used, depends on the provider. With more advanced machines it is cheaper than for a limited WAP browser.

Internet connection

Availability – The same as in email (internal communication), mobile device must support part of the tool. When a mobile device supports part of the tool and sending emails, it is bound to support IP connection as well.
Ease of use – It is hidden from the user completely, apart from the need for an active Internet connection. Lacks the restriction of an email format, making it the most flexible way to share information between the tool parts.
Initial cost of obtaining the data service – A possible fee for having part of the tool on the mobile device.
Per use cost of the data service – Cost is tied entirely to the cost of data used, depends on the provider and a purchased data plan.

No data service

Availability – Could be located only on mobile devices supporting more complex applications, mainly several megabytes of memory for data storage.
Ease of use – The tool could be customized to the device, providing any description and help necessary. No limitation is present by data services. However, it will be shown later that the tool might work with dynamic information, not to mention that transit schedules change periodically. To update the tool's data user would have to reinstall or update the tool manually or rely on some data service anyway to supply new data. This limits the tools capabilities and creates the risk of out of date results.
Initial cost of obtaining the data service – A possible fee for having the tool on the mobile device.
Per use cost of the data service – None.

3.1.4. Analysis interpretation

After analyzing advantages and disadvantages of several solutions, they can be compared as a whole. When developing a tool, it is possible to choose just one approach and use it, or combine several approaches for better universality. Single solutions appear suitable as follows:

1. ***Web page connection*** – Provides the best combination of availability and ease of use. It is also the easiest to implement.
2. ***Internet connection*** – Closely second, provides only a little bit less availability with similar ease of use. Implementation might be a little harder, but gains more flexibility on the mobile device.
3. ***SMS service (presentation layer)*** – Leading advantage of this solution is its availability. The most universal solution of all lacks mainly a user base patient enough to understand and use it.
4. ***Email (presentation layer)*** – Has slightly different availability compared to an SMS service (presentation layer), but in the end covers much less number of mobile devices. Suffers all its disadvantages apart from being a little cheaper.
5. ***No data service*** – Compared to the Internet connection solution, the constraints on updated data and dynamic features together with increased hardware requirements outweigh the freedom from data services. Falls to SMS and email (presentation layer) in much reduced availability.
6. ***SMS service*** (internal communication) – Requires the mobile device to support applications, and such a device will rarely lack any better means of communication with the Internet, be it in price or technical limitations.
7. ***Email (internal communication)*** – Has practically the same availability as the Internet connection, provides good features. But its fatal drawback is its uselessness, because on mobile devices supporting applications, where email is available for internal communication, so is the internet connection. Compared to it email does not provide any benefits at all.

The methods are not mutually exclusive, the tool can incorporate several of them, making it more universal and flexible. When a part of the tool is located on the Internet, several data

services can connect mobile devices to the same server. Therefore it is purely on the developer how sophisticated he wants the tool to be.

When choosing multiple solutions together it is desirable for them to complement each other's weak points. For this purpose it is recommended to combine either web page or the Internet connection with an SMS service (presentation layer). Together they provide the greatest availability, web page's or the Internet connection's ease of use with SMS coverage as a backup.

Next the focus of the work will be shifted to the developer environment and programming languages available for the tool.

3.2. Programming environment

Part of the tool located on the Internet server can be programmed in any language of choice, as servers are not restricted by hardware limitations like mobile devices are. But the tool part on the mobile device, if there is any, must make use of what is available. There are many manufacturers of mobile devices supporting applications, including Nokia, Motorola, Blackberry and Apple. More important then manufacturer of the device is the underlying operating system. Different operating systems in turn support different programming languages and application restrictions. This chapter will cover the differences an propose the best language to use.

3.2.1. Notable programming languages

Java ME

Java follows the philosophy of “write once, run everywhere”, to provide the best compatibility across wide range of devices. It is achieved by implementing a layer between the programs and the operating system called Java virtual machine. This layer hides all the differences between various systems and instead introduces an unified environment. That makes it possible for one program to run on various operation systems, provided the Java virtual machine is installed.

It is easier to hide all the differences of desktop and laptop computers, but when it comes to mobile devices, their severe hardware constraints make it impossible to support the full

features of the Java environment. For this purpose there is a cut down version of the Java environment, a subset of its features called Java ME. When we look at the versions of Java, there are three basic ones – Java EE – enterprise edition, the most comprehensive one, that enables powerful features (for servers and such). Then there is the SE – standard edition, that provides the standard features and is the most widely used, will full features one can expect form normal PCs. The last is the Java ME – mobile edition, designed for mobile devices. Java ME provides a subset of features found in the SE, and that in turn provides a subset of features in EE. That should guarantee, that the applications written in the Java ME subset will run in the superset versions. However, there are some features that are specific to the mobile environment, so Java ME contains some features, that are not found in the SE version. Using these features will make the application incompatible with the SE, but that is necessary only for the part of the application specific for the mobile device anyway. The general features, those that are found in the SE version too, are the same, there are no different versions of it. This makes the application compatible, apart from the mobile specific part. The application logic can be safely run in the SE version, just with some changes in the mobile specific code. In short, what can be compatible, is compatible.

The main difference from the virtual machines on normal PCs lies in configurations – CLDC for less powerfull devices, or CDC for more powerfull once. A configuration decribes the basic set of libraries and features a virtual machine on the device has. On top of this layer sits another layers, on CLDC sits MIDP – rich set of Java APIs for the use by applications. For more information see [6].

C/C++

C and C++ family of languages. The main difference between different devices is not so much in syntax, but in the available libraries and APIs. Thus it lacks the compatibility of the Java ME environment.

3.2.2. List of mobile operating systems

Notable operating systems, listed by a market share of smartphones equipped with the operating system in the year 2009, according to a study by Canalys [7]:

1. 47,2% – Symbian OS – Currently used by Fujitsu, Nokia, Samsung, Sharp, and Sony Ericsson, in the past also by BenQ, LG, Mitsubishi and Motorola. Currently being succeded by Symbiam platform, which is fully open source. Programming languages include C++, Java ME, OPL, Web/WAP scripting.
2. 20,8% – RIM BlackBerry OS – Used by Research In Motion in their BlackBerry devices. Supports Java ME.
3. 15,1% – iPhone OS – Used by Apple Inc. Main programming language is Objective-C.
4. 8,8% – Windows Mobile, Windows Phone – Used by Microsoft, these systems are part of the Windows Embedded family [8]. Development tools are available at [9]. Programming languages include Visual C++, Visual C#, Visual Basic.
5. 4,7% – Android – Used by Google. Supports Java ME.
6. Palm operating systems – Used by Palm Inc. Programming languages include C, C++, Pascal.
7. Linux – Used by Motorola in China, by DoCoMo in Japan.

3.2.3. Programming language choice

When choosing a programming language what matters most is the scope of the tool. If it is a little project the choice of Java ME is obvious. It is portable and third party applications are easy to use on that platform. But when the tool is supposed to be commercially successful and widely available, specific languages do not matter. The company developping such application will try to cover as wide range of devices as possible, using whatever language is available at each one. Good example is making an application for iPhone OS. It would be written in the native language and distribution agreements would be handled at corporate level.

4. Existing journey planners

With the widespread use of the Internet, journey planners gained popularity. Nearly every transport operator provides some kind of journey planning, either for its own services or combined with some of its competitor's, so listing all of them would be impossible. Instead only notable examples will be shown.

A journey planner is not to be confused with a route planner. Example of a route planner would be a car navigation. The difference is that a journey not only covers a route, but also time of travel, taking into account individual transport connections and exchanges between them. A route is just a connection of places to go through.

4.1. Google Transit

Google Transit [1] is becoming the most widespread journey planner tool in the world. Launched in 2005 as a part of Google Maps, its goal is to cover journey planning all over the world. Despite being relatively new, it has grown much, mainly because it is backed by the corporate giant the Google is. What also made it popular is being incorporated into an existing technology, the Google Maps, allowing the user to work with it seamlessly. In classic journey planners user works with them as with a standalone application, while integration in interactive maps can provide several search parameters automaticaly, making it easier to use.

Architecture of Google Transit it tied to the Google Maps. Originally only available on the Internet, in 2006 Google Maps for mobile was released. First it was intended to run on Java-based phones or mobile devices, providing many features of the web site, but in time it was adopted to other platforms. Finally in 2007 the Google Transit feature was added to Google Maps.

Of the discussed architectures it fits the Internet connection, having part of the tool on the device and part on the Internet, connected most likely by an internal Internet connection.

Google Maps are supported by every major mobile operating system, including Symbian OS, RIM BlackBerry OS, iPhone OS, Windows Mobile, Android and others.

4.2. Transport Direct Portal

Transport Direct Portal [11] is a comprehensive journey planner in the United Kingdom, covering England, Wales and Scotland. Many modes of transport are combined into this tool. In 2005 it introduced access from mobile devices. From the architecture point of view, it is accessible as a website, customized to be viewed from a mobile device. Setting up on the mobile device involves at most creating a bookmark. Since there is no part of the tool on the mobile device, the compatibility remains wide, available wherever web browsing is available.

4.3. Idos

Local to Czech republic [12], it provides journey planning for many modes of transportation, including trains, buses and city public transports. The Department of transport payed for its development and now it is managed by the CHAPS company. It does not have any notable competitors in the country.

Although it is possible to view the web page from a mobile device, a notable architecture solution is a standalone application [13] that can be run from a mobile device. It is unusual that the application is not free of charge. The installation of the application itself is free, but for the use of the data files with transit information user has to pay a periodical licence fee.

The only operating system to support the standalone application is Windows Mobile. In the past it was available for Pocket PC, but the support was discontinued.

4.4. Comparison

Provided examples were chosen for their significant differences. Google Transit is an example of a global tool with wide area coverage, using the services of both a mobile device and the Internet as much as possible, backed by a powerful commercial software company. Transport Direct Portal is an example of a service with significant cooperation of different areas of transport, focusing mainly on the journey planner itself on a web page, with mobile access not being it main priority. It is backed by a nation and following significant research on the subject. Idos is a journey planner for local use with little cooperation between areas of travel, enabling it custom-tailoring for is task. Mobile device standalone application does not seem to be a serious effort for a mobile device tool.

5. Transit search parameters

Now that the information about structure of a mobile device tool and examples were provided, it is time to focus more on the features of a journey planner. When a person requires a service providing travel planning, the end result should be to provide sufficient information for him to follow. To achieve this the planner first needs some input, and based on that information. This chapter will have a look at what a typical user would require. Those parameters will be evaluated later when studying how well they can be implemented by a search algorithm.

5.1. Basic search parameters

In this section we discuss basic search parameters to specify a trip.

5.1.1. Departure and destination

When looking for a route in transport, user usually has a clear idea of the origin and the destination of the trip. In this application, as well as in most of the journey planners, it is not possible to choose any place at will. A valid place is usually a transport station, with connections to the transportation network. Furthermore this station has to be known to the application and it has to have the data about it and its transport connections. But this is the concern of how complete the data are and not the immediate concern of the user. So the data about stations should be as complete as possible and the only thing user has to think of is to identify the closest stations to his departure and destination places.

There are ways these parameters can be expanded though, using different services. Such as finding a connection between places other than transport stations or giving an estimate of the time to get to the first station. That would however require additional services, like information about roads, or GPS coordinates. Complex applications like Google Transit provide this information though. As it is part of Google Maps, it has access to roads and other ways of travel, so it can chart a route from almost anywhere. Even if a tool does not support these external services, it is a good idea to make the journey planner prepared for them, so they can be easily incorporated in the future.

These parameters sometimes does not have to be entered, but sometimes they can be supplied by external services. A mobile device provides a possibility of supplying the origin based on the location of the device. Google Transit provides this service under the name My Location. It does not even have to support GPS, it can estimate user location based on unique footprints of nearby cellphone towers providing reception to the device.
When user is not looking for a specific destination, but for example for a nearest museum, other service can provide the location, which can be supplied directly to the planner.
Proposed parameters:

- Origin
- Destination

5.1.2. Time

Another basic parameter to define the trip is time. There are two main ways how to define it, by the departure time or the arrival time. Unlike places however, it is not so straightforward. It is highly unlikely that the time user specifies will be the exact time of either departure or arrival, so it cannot be fixed like places are.
The ways to specify the time are several, like looking for a connection departing or arriving before or after a specified time. To decide which ones to implement, first consider the situations user is in when he uses them.

- The first most likely case, user wants to get somewhere and be there at a specified time. This means to search for connections arriving before specified time.
- The another most likely case, user want to get somewhere as soon as possible. That means to depart after specified time, with the specified time being the first moment he can.
- Another case, user want to arrive somewhere after specified time, when for example an accommodation or another service is made ready for him at that time.
- The last combination of before/after and departure/arrival time is to depart before the specified time. That can be the case opposite of the previous one, when an accommodation or another service is no longer available.

The first two choices are the ones used by all the journey planners. The last two ones are interesting, but the necessary result can be obtained by traditional search by either by making

a query with a different time, until a satisfying result is returned, or in case there are several consecusive results returned at once, by just selecting a convenient result from the list.
Like an origin of the trip, time parameter has its default value. Current time can be set as th time of departure, specifying immediate travel query.
Proposed parameters:

- Time.
- Specification if the supplied time is a time of arrival, or time of departure.

5.2. Advanced parameters

Now we have all the basic information needed to produce search results: The departure place, the destination place, and the departure or destination time.
Without any more parameters, the obvious way to judge the quality of the desired connection is the time it takes. But there are several more things that can determine the quality of trips. For example it is the number of exchanges on the way. There are also other parameters that can influence the result, like restrictions on the type of transport or the delay expected. In this chapter we will look at these parameters more closely and how are they related to each other.

5.2.1. Trip duration

The most important property of the trip will be its duration. While other parameters do not have to be optimized, this one always will. The shorter the trip, the better. The way how it is tied to the specified time was described in the previous chapter. However, other parameters will be optimized at the expense of this one. Since it is always optimized, no additional parameter is necessary.

5.2.2. Number of exchanges

Another parameter is the number of exchanges. A connection will consist of a trip from the departure to the destination, and it does not have to be by one vehicle only. Different parts of the journey can be traveled by different vehicles, with necessary exchanges on the way.

One way to limit the number of exchanges it to specify the maximum, so any trip with more than that number will never be considered as a valid result. This has a disadvantage, that there is no optimization among trips that fit into this criterion. Other way is to try to minimize the number of exchanges. This optimization would work between all the trips. However, it is much harder to define how aggressive it should be, since this would definitely work at the expense of the total traveling time. How to choose between a shorter trip with more exchanges and longer trip with less exchanges? One way to do it is for the user to define, how much time he is willing to sacrifice in favor of one less exchange. Most of the journey planners do not use this approach and use the maximum limit. But this approach can be useful.

Proposed parameters:

- Maximum number of exchanges.
- Number of travel minutes to sacrifice for one less exchange.

5.2.3. Exchange duration

Exchanges also take time and this time is another parameter to consider. Most of the travel planners let the user define the minimum time for the exchange to take place. That is certainly quite simple and elegant, but there is a room for improvement.

Some planners also let the user input a maximum time the exchange can take. That seems to be irrelevant. The application will try to minimize the time of the trip including the time spent by exchanges. So by setting a maximum for an exchange will make the searcher choose another trip, where user will spent more time in the traveling than he would save by the shorter exchange. However, this would be useful if the user did not want to spend time in unpleasant places, as some stations can be. Other use would be when no result returned is better than returning bad results. So even this parameter is worth considering.

One time limit for all exchanges at once may not be desirable. Some exchanges can take much longer that others, for example from one train to another it is much faster than from one train to an airplane. There is a possibility to specify different limits for different properties of vehicles or station. But that is different for specific transits, so this would have to be implemented with specific requirements in mind. Customizing the application to specific needs will be covered later.

Proposed parameters:

- Minimum exchange duration.
- Maximum exchange duration.

5.2.4. Exchange location constraints

Apart from time constraints for exchanges, there are also place constraints to consider. In that case the exchanges would be limited to specified places. It wouldn't make sense to require exchanges at all of them, since straight train would be faster. And if the user wanted to stop at all of those stations anyway, it would mean that he does not just want to go from the departure point to the destination, but to visit other places, and for that more separate trip searches would be more appropriate.
Also the place constraints do not have to be limited to exchanges, user just might want the route to go through specific places. This is handy when the searcher provides unwanted routes that are not familiar to the user, or otherwise inconvenient.
Opposite way is to specify what stations the user does not want to go through.
Proposed parameters:

- Stations where only at them can the trip make an exchange.
- Stations the trip must pass, independent on if there is an exchange or not.
- Stations where where not to make an exchange.
- Stations the trip must not pass.

5.2.5. Walking

Sometimes it is possible to walk between close stations. This form of transport is somewhere between an exchange and another transport mode, typically used between stations that are too close to each other for other means of transport. It is desirable to select user's average walking speed, so there is enough time for an exchange for an elderly person, or that there is not much time wasted if user is in a hurry and doesn't mind to run. One more advantage of walking is that in some rare cases it would be faster to walk somewhere than to use public transportation, probably because the public transportation can take a long detour.

Proposed parameters:

- Maximum walking distance user is willing to traverse.
- Average walking speed.

5.2.6. Reliability

Transport connections are not always reliable. The projected trip might not be available when the user follows it. When deciding the reliability of a trip, let's first have a look at how a trip can fail, what are the sources of unreliability.

The first one would be a missed exchange. This can be caused by a delay of a previous connection, or when the next connection would not go at all, because of some accident or a breakdown.

Other thing to consider is a delayed arrival of the whole trip. Unlike delays between exchanges, this concerns a delay of a vehicle after the last exchange. Although it does not share the danger of a missed exchange, there may be other obligations user can have after the trip, that can also make these delays part of a reliability property. But the end result is satisfied, the user ended in the destination, even if late, so for the rest of the work it will be ignored.

So now we have established that the main reliability issues are in a failure of an exchange due to delays and a failure of a connection due to breakdown. These two values can be represented by a probability value in percents.

The trip does not just have to succeed of fail completely. When something happens, it may be possible to avoid the obstacle by taking an alternate route to the destination. That too influences reliability. When there are more alternate routes to the destination for some parts of the journey, naturally the trip is more reliable, than if there was just one connection available. However for the purpose of this work we will consider only two outcomes, success or failure.

When the connection fails and another route must be taken, it is possible to enter another search query to find the route. It does not have anything to do with the search algorithm itself, this functionality can be completely cover in the user interface, in the presentation functionality. If the user can just enter the new search, it does not have anything to do even with the application itself. But there may some support in the user interface to ease this, so the user does not have to enter all the parameters again.

So, how the user should be able to specify the level of reliability? Since it is about probability, it can be in percents how much does he want the whole trip to be reliable. This is straightforward, but might not be practical in the search algorithm, as will be explained in the algorithm chapter. Another way is to specify reliability of each exchange in percents. It would be easier for the search algorithm, but it lacks the elegant nature of the whole trip number. However it is a good way to avoid spikes in unreliability when the normal risks are not what concerns the user, just a presence of an increased risk.

Alternative to fixed limits is to provide more flexible constraint tied to the traveling time. It would be how much longer the user is willing to travel for a unit of reliability, for example how many minutes he would sacrifice at an exchange for a percent in reliability.

To make it even simpler, there can be one single time variable specifying the minimum time for a reliable exchange.

Proposed parameters:

- Minimum reliability of a whole trip in percents.

  or
- Minimum reliability of each exchange on the trip.
- Number of travel minutes to sacrifice for a percent in reliability of one exchange.
- Number of minutes it takes for an exchange to be considered safe.

5.2.7. Trip duration

One would think that after specifying the time of departure, the concern of the application would be to find the trip that will take the least time to get there. However, duration of the trip itself is not the whole time variable that needs to be considered. The other part is the time between the user specified time, to the actual departure/arrival. For example, user wants to find the best connection after the time 6:00am.

The first connection to consider departs at 7:00am, and arrives at 10:00am. The other connection departs at 9:00 and arrives at 11:00am.

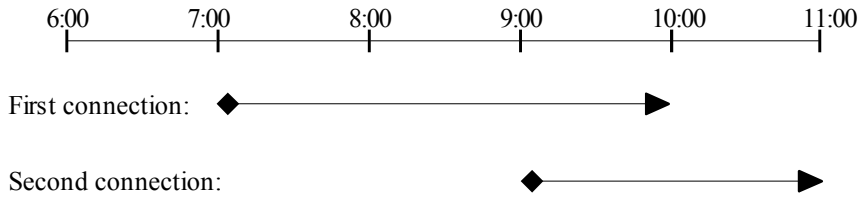


Diagram 1: Trip duration conflict

The shortest duration is the second one, taking only two hours. But the first connection, despite taking an hour longer, arrives to the destination one hour sooner. Since the user wanted to get to his destination as soon as possible since the time 6:00am, it makes sense to choose the longer one. I this case what the application is really trying to minimize is the time it takes from the specified time to arrival. So there is a question whether the duration of the journey is to be minimized alone, or is the delay between the specified time and the duration of the journey also the subject of minimization.

One way to avoid this is to have the departure or arrival time bound between two times, minimizing the other parameters like before, but the duration of the trip would be no longer dependent on specified times other than being between them. This could be beneficial when the user is busy and has to get somewhere, so he needs to waste as little time as possible by travelling.

Proposed parameters:

- Second time parameter, with the intent to find the shortest trip between the earliest departure an the latest arrival time.

5.2.8. Transit specific properties

Both stations and vehicles can have some properties. It can be their quality, safety, services and more. Those properties can be different for different transit companies and countries, so when making an universal travel planner, they have to be supported but cannot be specifically defined. Working with such properties can be difficult, so let's see what influence they can have.

When the property is a service, or a type of quality, user might want to restrict the trips to provide those services. So the property can be a restriction on what vehicles or stations the trip can use.

Properties do not have to influence the search at all. Some may be there just for informational purposes, that could be displayed with the result. That could be the presence of a dining car in the train and a disability support. These properties can be specified as restrictions as well, but do not have to be.

On the other hand some properties may be limited to informational purposes only. That could be those that describe something about the trip as a whole, but cannot be described for parts of the trip separately. The search algorithm would have a hard time to optimize for those, for it would learn their value only after the trip is computed. An example may be a cost of a trip. Some transport companies do not charge money for the length of the trip directly, but rather use their own complex system to calculate the cost, with special fees for quality or for crossing from one region to another. This can be relatively easy to implement for showing the price tag of a finished trip, but it would be impossible to tell, how much only a half of the trip costs, since the sum of prizes of the two halves can be different from the price of the whole trip. Only when the cost of the trip is clearly defined by the operators, it is possible to optimize for it.

The issue of properties and their role and implementation will be covered later in discussion about the search algorithm itself and how the optimizations could be implemented.

5.3. Comparison with existing planners

Part of evaluating the search parameters is to compare them with the ones being commonly used. Compared journey planners will be the ones listed in the chapter 4. Existing journey planners.

5.3.1. Parameters available on the mobile device

Some parameters are common to all of the listed planners:

- Origin and destination.
- Time.
- Specification if the supplied time is a time of arrival, or time of departure.

These are all the parameters Google Transit and Transport Direct provide in their mobile version. Idos, being a standalone application provides additionaly:

- Maximum number of exchanges.
- Minimum exchange duration.
- Maximum exchange duration.
- Stations where only at them can the trip make an exchange – in this case just one station.
- Stations the trip must pass, independent on if there is an exchange or not – again only one station can be specified. This parameter is mutually exclusive with setting exchange location in the previous parameter.
- Number of travel minutes to sacrifice for one less exchange.
- Number of minutes it takes for an exchange to be considered safe.

5.3.2. Parameters available on the web page

On the web page it is possible to fit much more search parameters than on the mobile device. And since many new mobile devices can view web pages in the same way normal computers can, web page search parameters will be considered as well.

Google Transit offers the least search parameters, the search on a mobile device and on a web page offers the same features.

Transport Direct offers much more additional parameters:

- Transit specific property – type of transport, train, bus, underground.
- Maximum number of exchanges.
- Minimum exchange duration.
- Stations the trip must pass, independent on if there is an exchange or not – only one station can be specified.
- Average walking speed.

Idos additionally offers:

- Stations where only at them can the trip make an exchange – on the web page three stations can be specified instead of one.
- Stations the trip must pass, independent on if there is an exchange or not – again three station can be specified. As on the mobile device, user cannot use this parameter at the same time as the previous one specifying exchange places.
- Maximum walking distance user is willing to traverse.
- Transit specific property:
  - Type of transport, train, bus, city public transport.
  - Train quality restriction.
  - Presence of a sleeping car on a train.
  - Option to preffer frequented connections.

It is interesting that the Idos web page lacks some features of the mobile application, suggesting that they are much different from each other:

- Number of travel minutes to sacrifice for one less exchange.
- Number of minutes it takes for an exchange to be considered safe.

5.3.3. Comparison

Abilities of existent journey planners showed that only a few basic search parameters are necessary. When a journey planner provides additional criteria, it is mainly because it has resources to do so, which can be seen from abilities of mobile versions in contrast to their web page alternatives. The fact that additional parameters are optional is clear form the abilities of Google Transit, which is able to gain populatity without them. One may even say that its simplicity is an advantage.

The search parameter importace is not the main reason why some planners include them or not. Even unimportant parameters can be just ignored or hidden from the user, but still be available. The main reason is that some search parameters are actually impossible to implement in certain contex. That will be covered later.

6. Public transit information

In order to find a journey between stations, apart from search parameters the planner needs actual data to search through. Such information includes list of stations, connections between them and traffic schedules. This chapter will cover where the data comes from, in what form and how hard is it to get it.

6.1. Data sources

Each transit company keeps records about its traffic schedules and much more. Traffic schedules are just a top of the iceberg. Basic records include vehicle inventory, employee records, list of stations, service depos, tracks and all the equipment. With this data there are task to do like the design of transport routes, design of public and service schedules, assignment of vehicles to individual connections or assignment of personnel to shifts and vehicles. The final schedules available to the public are result of a long and careful planning. Even though such data are generally available in forms like printed schedules, application ready data are often withheld from the public. Transit company provides them to selected sources carefully, since they are a valuable comodity for often commertical journey planners. Even if a journey planner is free of charge, it might generate profit through advertisements or by other finantial support, for example by national transport department funding. On the other hand, making the information available to successful journey planners promotes services of the transport company, encouraging it to share the data. Licencing issues often accompany publishing of the data.

Google is approaching the data gathering by providing its data format GTFS and letting transport companies provide the data themselves through Google Transit Partners Program. This approach seems to be working, for as of 16 April 2010, the program reports an overwhelming interest, having to queue the requests from transit companies.

Transport Direct, even while covering only the United Kingdom excluding Northern Ireland, also acquires its information from many differerent sources depending on different transport modes. An example may be Traveline – for buses, tram, light rail and ferries, TheTrainline – for train information, East Coast and others.

In the Czech Republic, one company – CHAPS, was chosen by the Department of transport to manage national information system of traffic information. All transit companies are required by the law to provide their traffic schedules to this company, aside from a few exceptions. CHAPS provides many services like the Idos journey planner, some transport coordination and management. The traffic information gathered is available for a certain fee.

6.2. Data format

6.2.1. Transmodel

There are many transit operators all around the world. In the past nearly every company had its own conventions on how to store the traffic data, making cooperation between them difficult. The first serious attempt to standartize the format was Transmodel [14]. It was established in 1992 between several European coutries and has been evolving eversince. It is a standard from the European Committee for Standardization (CEN), that provides reference data model for all public transport information. It covers a lot of public transport concepts like scheduling, fares, driver rosters, vehicle planning, and most importantly, journey planning. Concepts are described by Entity Relationship Model and UML and described in detail, providing an unified way to represent the public transport data. One other feature is establishing a clear terminology, since many operators have a different view on what many terms mean, like a trip, journey, service journey and a route. Misunderstanding these terms can lead to compatibility issues not just on code level, but also in research papers and publications.

Transmodel is very generic in describing concepts, leaving specific implementation on readers. This provides sufficient freedom while ensuring compatibility between individual Transmodel inspired systems. This freedom led to more specific standards, like TransXChange, which is a XML standard used in United Kingdom for sharing bus timetables.

6.2.2. GTFS

Originally called Google Transit Feed Specification, it was created as a common data format for information supplied to Google Transit. However over time more applications started using this format, with many transit companies sharing their data between each other in it. Its widespread use led to the replacement of the word Google, making it General Transit Feed Specification.
Unlike Transmodel and its standards, GTFS defines very simple and specific data format for schedule information and no more, only what is necesary for journey planning, without concern about internal company data management.

6.2.3. JDF

As was mentioned before, transport companies in Czech Republic have to supply traffic information to a national information system of traffic information. JDF is a data format mandatory for this data. In scope it is similar to the GTFS, it includes only information relevant for the journey planning. The main difference is that the format closely resembles that of the printed schedules, containing information about train and bus properties like disability support, additional space for luggage and bicycles, presence of a dining car and seat reservations.

6.3. Real time information

In addition to static information there are some dynamic factors to consider.
First are changes in traffic schedules. Some are planned, like track repairs, other are unexpected, like accidents. Both can be handled the same way static information is, by updating current data.
More interesting is real time information influencing traffic. This includes estimation of current vehicle position, current delays and traffic congestions for bus routes. For this purpose there are data standards, enabling journey planners to retrieve such information as well. One such standard is SIRI - Service Interface for Real Time Information, for sharing real time

delays and timetable updates among other things. It is based on Transmodel, which in its extensivity covers even this issue. Sources of this data are usually centers of traffic operations and control.

Integrating dynamic information with static information is the easy part. Before feeding data to the search algorithm, static and dynamic information is combined and the result is passed on.

6.4. Connecting multiple data sources

A data set of schedules often cover just one or a few modes of transport. An issue arises when several data sets have to be combined, for example when a journey planner supports exchanges between trains and buses. A journey planner supporting more methods of transport is called intermodal. The issue itself is generally not solved by the journey planner, but by traffic companies supplying traffic information. When a transport network is connected to a different one, part of the traffic information is connections between their stations. It includes obvious links like a connection of a tram station with a bus station located in front of it.

In case this information is not available, it can be sometimes generated from location of stations, or connected by walking distance.

Similar problem arises when incorporating different versions of schedules, when they change. The journey planner has to have correct information even at the exact moment of the change, when one part of the journey follows the schedules before the change and other after the change. Such differences are handeled in details of the data representation. Transmodel has quite complex schema with interconnected information about routes, vehicles and lines, so it solves this problem by adding versioning of schedules. GTFS specifies transit connection more individually with little connection to other data, making it simple just to have different information on schedules before and after the change.

More issues in this topic will be covered in the algorithm chapter, like how to process transit networks too different from each other, or when a combination of networks is too large to process.

When multiple data sources need to be combined, it is not just the issue of how to connect them. Basic information has to be connected, but optional information can be missing completely in some data sources, so there is nothing to connect. The more sources are

included, the more it is probable that an optional data category will not be complete. This explains why universal journey planners like Google Transit provide so little search parameters and why local journey planners can provide plenty of them. When developping a planner for just one network, it can be custom-tailored to it, including optimizations like the cost of the trip. Idos is an example of such custom-tailoring, providing the search parameters for quality of trains.

6.5. Data destination

When the input data are gathered, they are stored by the journey planner for future usage. Interesting point is that the data are not just used by the search algorithm. Some data are stored for also for their informational purpose only, providing interesting details about already found journeys. It includes detailed train information, disability suport, quality of services, fares and CO_2 cost of the trip. Even though such information can be used for optimization as well, generally it is just stored to provide more information about the trip to the user. How this information is presented will be covered in the user interface chapter.

7. Search algorithm

The search algorithm is the core of the journey planner. It is a separate component, recieving two inputs, traffic data and search parameters, providing the optimal result found. In this chapter possible implementation will be discussed, along with other issues that the algorithm has to deal with.

7.1. General design

The task at hand inevitably contains the problem of finding the shortest path in a graph. Some parameters will require some customization to this widely used problem, but the concept is still the same.

Solving the shortest path in a graph problem will be illustrated on the Dijkstra's algorithm [15]. It works for both directed and undirected graph, in this case the traffic is naturally directed. In a graph with weighted edges with non-negative values, the goal is to find a shortest path between an initial node and a target node.

The algorithm runs as follows:

1. Each node has a value distance, that is currently the best known distance of this node from the initial node. At the beginning only the initial node has the distance set to zero and all others infinite.
2. Two sets of nodes are maintained, those with the guaranteed best distance from the initial node, named visited, and others, that are unvisited. At the beginning only the initial node is visited, the rest are unvisited. The initial node is chosen as a current node for the first repeatable step of the algorithm.
3. All neighbors of a current node on the outgoing edges have their distance recalculated by comparing its current value to the value reachable by path through the current node and the edge connecting the current node to the neighbor. If the new path is shorter, the resulting better distance value replaces the old one in the neighbor, otherwise the neighbor keeps its previous value.
4. After all the neighbors are recalculated, mark the current node as visited.
5. Set the unvisited node with the best distance as the next current node and repeat from the step 3, unless the current node would be the target node, then the algorithm ends.

7.1.1. Basic algorithm implementation

Next we will customize the algorithm for the journey planning. For now we will ignore the more advanced parameters like exchange restrictions and concentrate on finding a way to satisfy the basic ones.

The transport network is naturally representable by a graph. The simplest way is for the stations to be nodes and the connections between them to be edges. The distance of a node would be the time it takes to get there from leaving the initial node. Having many connections between the same two stations at different times could be solved during execution by including only the outgoing connections available after the best time value of a current node. There would be however a problem with the optimization of the trip duration.

Let's illustrate it on an example. User wants to travel from an initial node A to a target node C. There are connections between A and B every hour, and between B and C every day at 8:00 pm. When searching for a connection after 8:00 am, the algorithm would find the first morning connection to B, then a whole day wait at B and finally a connection to C in the evening. When the optimal way would be to get to B by an evening connection arriving just before the one from B to C.

The problem with this implementation is, that the algorithm finds not only the best connection between the initial node and the target, but also the best connection between the initial node and all the other nodes on the way. Which is not the thing we are searching for. As shown on the example, we also want the trip to be as short as possible.

To try to correct this we could use this approach as a first phase, finding the best time it takes to get there. Then the second phase would be to run the algorithm a second time from the target node backwards, finding also the latest time to depart.

To do the task in one phase only, there has to be more information kept about the nodes.

Visited nodes are guaranteed to have the best distance possible. We cannot guarantee the best route in the middle of the algorithm, since there is no way to know yet when the next connection will depart. That hasn't been computed yet. So instead of just one best connection from the initial node to the others, let's keep more of them. To determine if the route to the current node is the best, we have to know the unknown variable of when the best route will continue. So let's keep the best route for all the possibilities. The result will be different best

routes for a node for consecutive time intervals, as shown on the following diagram.
In the example, until 8:00 am the best way to arrive to one station A is by a train 1 arriving at 7:50 am. But there is another train 2 arriving at 8:00 am that allows the user to depart later. So at this node in the interval 7:50 to 8:00 am the best way is to arrive by the first train. In the next interval starting at 8:01 am the best way to arrive by the second train. Duration of exchanges is left out for the sake of simplicity.

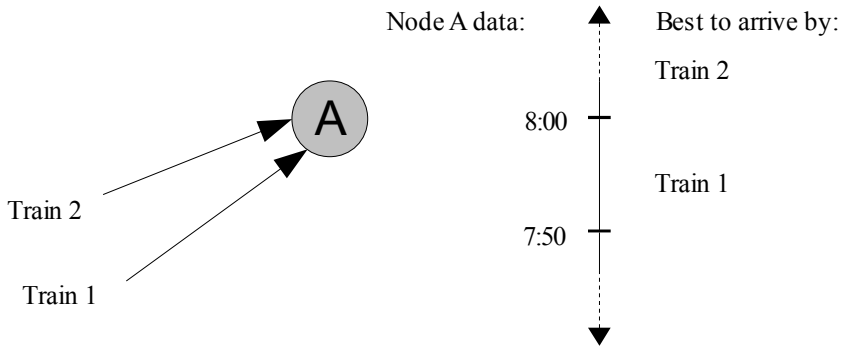


Diagram 2: Time interval data stored on a station

To make it more intuitive, it is possible to represent the time interval as a series of nodes. A station would be represented as a set of nodes, each node would have its designated station and a departure or arrival time. When a train travels from A to B, it would travel from a node from the set of station A, this node would be identified by station A and the time of departure. Arrival would be represented by a node identified by station B and the time of arrival. Each node at a station would be connected by and edge to the closest future node of the same station.

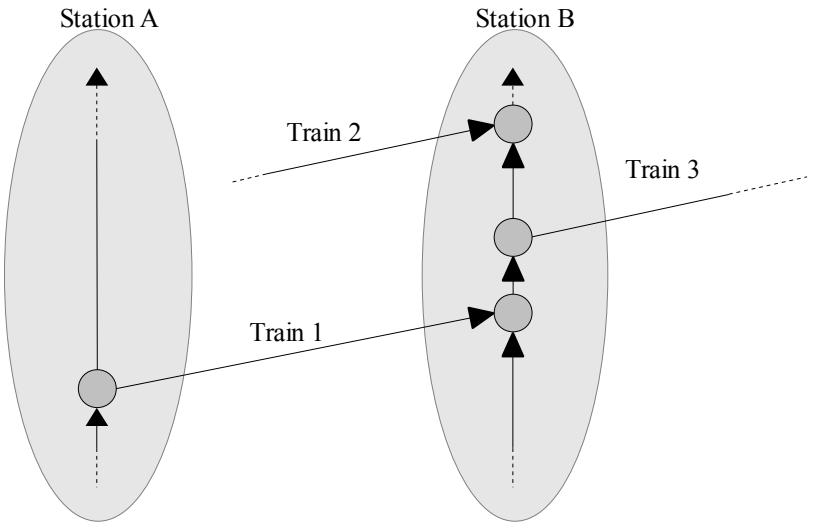


Diagram 3: Time interval data represented by nodes

The visited nodes will be those older that the current time. So current time will play one role of the distance of the current node in a classic Dijkstra's algorithm, the difference between visited and unvisited part. The other responsibility of the distance is to be the sum of weights on edges on the best route from the initial node to the current node. This functionality will be separate, weights will reflect the time of the trip and later other properties as well. This is a major modification of Dijkstra's algorithm, and in an ordinary graph it would not work anymore. But the graph we are working with has an additional property that keeps the algorithm correct. There are no edges from the future to the past, you cannot arrive before you departed. This makes the graph acyclic.

To check if the separation of visited and unvisited node sets is correct in respect to distances, let's look at how it could be broken. There would have to be a route from an unvisited node to a visited node, that would have better distance value than the value of the visited node. But this is not possible simply because such edge cannot exist, it would lead from the future to the past.

In the previous classic algorithm such edge can exist, what is guaranteed is that the resulting distance cannot be better. When selecting the next current node, it is chosen as a minimum of unvisited node distances. Since edges cannot be negative, the unvisited node that would be the source of the conflicting edge would already be part of the visited ones. The principle of separation of visited and unvisited parts then holds for both the classic algorithm and the journey planner modification.

Choice of the next current node will be any unvisited departing node with the lowest time stamp. That is the only way to keep the current time as a separator of visited and unvisited parts. The arrival nodes do not concerns us in this choice, they are only used in computing the best available route's time interval.

Exchange time issue remains to be solved. It involves the step when neighbours of the current node have their values recalculated. What value to assign to the neighbor? It should be a sum of the best value of the current node and a weight of the connecting edge between the current node and the neighbor. What we have available is a set of time intervals (or a set of arrival nodes of one station) showing the best routes to get here with distances on those nodes. It is not always possible just to add the weight of an egde to the neighbour to the distance of the last arrival node. When there would be an exchange, the exchange duration would have to be considered. To solve this, we would not be adding to the distance of the last arrival node, but to the distance of the last arrival node that can support the exchange time.

When the algorithm finishes, it is possible to find the best path just like in the classic Dijkstra's algorithm, where each node remembers the previous node in the route. Here the previous station and connection are remembered too, just have to be determined by the departing time of the next step in the route. And as said before, by possible time for an exchange.

7.1.2. Including advanced parameters

There are two main types of parameters that can be applied. The first one is a global parameter. A restriction on a path as a whole, that is impossible to determine in the middle of the path. For example a fixed maximum number of exchanges a trip can have. This is rather hard to implement in a graph algorithm, since in the middle of the graph one cannot predict now much exchanges will be required in the rest of the path, so it is impossible to decide which path is best on the spot. This is true for all the other optimization parameters that places a fixed limitation on the entire path.

Another type is a flexible restriction that can be determined in the middle of the path. Like limitation per station or an edge. This type is much easier to implement, simply by modifying the distance value.

In the one phased variant, global restriction can be implemented by adding another dimension to the time intervals. Instead of remembering just one best route for certain time interval, it is possible to remember the best one for each value of the uncertain variable. For example with number of exchanges, the algorithm would remember what is the best route to get there with one exchange, two exchanges, three and so forth for every possible value of the global restriction. This way, when the algorithm finishes, the fixed limitation can be determined just by looking at the corresponding best solution at the target station and backtrack. When there is an exchange during the backtracking, the correct solution is with one less exchange from then on. More global parameters can be combined by adding more dimensions.
In the simple two phased variant this is possible as well, in both phases in each station remembering not just one best route, but more dimensions of solutions for each global parameter.
As for the flexible parameters, they do not need another dimension, they can be incorporated in the edge weights. So far only the duration of the trip was included in the weight. When trying to optimize other parameters, one has to combine them with each other. One way to link them is to specify how much the user is willing to sacrifice in one property in favor of another one. This possibly makes a big number of combinations that rise quickly with additional properties. The duration of a trip is the main property to minimize. That makes it ideal to serve as a common connection between properties. In other words each property would have to specify how much of the time of the trip to sacrifice for a given property unit.
As with global restrictions, it is usable in both two phase and one phase algorithm variants.

7.1.3. Including complex parameters

So far we talked only about clearly defined parameters. But some change from place to place or provide other complications. Then the algorithm has to be customized to the specific requirements.
An example would be a price of the trip. Many operators apply complex system of fares, that differ between each other. If fares are not simple, their usefullness in the planner is at most informational, computed for the journey after it has been found. If it is supposed to be optimized, the algorithms would have to be customized to specific fare systems.

7.1.4. Including transit specific properties

Most of transit specific properties are restrictions, like what type of vehicle the journey can use or disability support. An advantage of these properties is that thay are absolute, a journey can either use the resource or not, they do not have to be optimized. Unusable resources can be filtered out before they are even presented to the search algorithm.

7.2. Implementation of specific parameters

7.2.1. Time of arrival vs. time of departure

This is a basic feature. When departure time is selected, the Dijkstra's algorithm will start from the departure time and progress forward. When arrival time is specified, the algorithm will go backwards instead. Only concern could be the difference in how the specific traffic data are retrieved in respect to the internal data representation.

7.2.2. Minimization of number of exchanges

In the algorithm, there are two main ways for the user to control the resulting number of exchanges. The first one is to specify the maximum number of exchanges the journey can use. The absolute maximum is a global restriction with the need of another dimension when remembering node results. The form of this dimension is, as was said before in an example, to maintain different sets of best paths according to the number of exchanges the paths already spent on the way. When the algorithm reaches the destination, data will be available on how to get there in one exchange, two, three, etc. Only then the result can be chosen, but not sooner.

An alternative is to use flexible restriction, to limit the number by tying it to the trip duration time. User would then specify how much more time of the trip he would endure for one less exchange. This can be implemented rather easily, by adding the time to the route weight in the graph when handling exchanges.

7.2.3. Exchange duration

Minimum exchange duration is always incorporated in the basic algorithm, even if it is not specified. When minimum exchange duration is set, it simply replaces the default value used. During graph traversing, minimum exchange time is taken into account when updating the distance of nodes. When the best incoming connection arrives by the same vehicle as the outgoing one computed, exchange duration is ignored. When the vehicle is different, the incoming connections that arrive after the departure time subtracted by the minimum exchange time are ignored and the best incoming connection is selected from the remaining ones.
Maximum exchange duration is similar. When searching for the best incoming connection and there are none that arrived in that duration, the outgoing connection is ignored as not reachable at that time.

7.2.4. Exchange location restrictions

When a trip can make an exchange only at specified locations, implementation is also easy. During the graph traversing, the computation at specified locations will remain the same, with a possibility of an exchange. When elsewhere, in those stations a trip can continue only by an edge that has the same vehicle specified as an arriving edge. Technically, when updating distances of neighboring node through an edge, there either is a corresponding arrival edge to the current node or there isn't. If it is, neighbours distance will be calculated using the arrival egde distance. If it isn't, the outgoing edge will be ignored as unreachable at that point.
Impementing the option where not to make an exchange is the very same thing, just by reverting the selection of allowed pass locations.

7.2.5. Mandatory pass locations

Approach to this problem differs depending on whether the locations have to be passed in certain order.
In case that the order is fixed or there is only one location to pass, the solution is to treat the first location as a destination. After it is reached, continue to traverse the graph from the first

location in a different, separate data set. It can be viewed as another dimension of information like the ones used for different number of exchanges already spent on the way, when the maximum number of exchanges is limited. Continue to traverse both the graph between the origin and the first location as well as between the first location and a second one. Only point where these data sets connect to each other is in the first pass location. Graph traversing continues until another pass location is reached, when a third data set is created connected with the second one in the second pass location. After the destination is reached, the search can end. It is important to traverse all the data sets at once. If the search just found separate trips between pass locations, all the global optimization information would be lost, including the basic trip duration checking descibed in chapter 7.1.1.

When the order of pass location is not fixed, there has to be a separate data set/dimension for every combination of pass locations already visited. That is the reason why journey planners often limit the number of pass location. If for example three pass locations were specified, there would be data sets like „first visited, second not visited, third visited“ and „first not visited, second not visited, third visited“. These two data sets would be connected by the first pass location and only in one way, since that is where one condition is satisfied. Data set connections are shown on the following diagram.

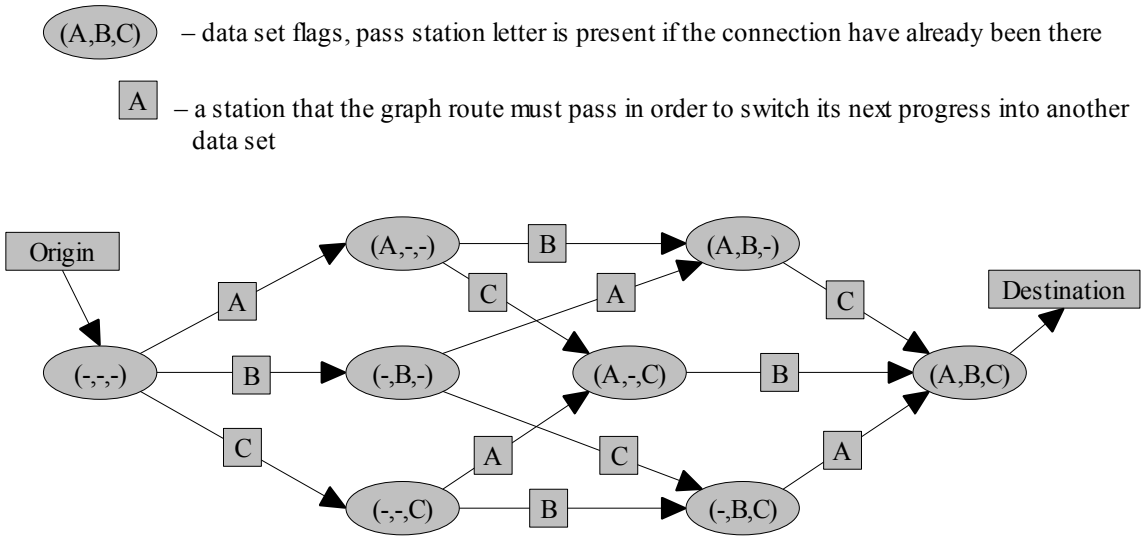


Diagram 4: Data set crossover

The diagram shows that with just three pass stations, eight datasets are needed already.
On the other hand, implementing the option of stations the trip must not pass is easy, just by excluding the station and its incomming and outgoing edges in the graph.

7.2.6. Walking distance

Possibility of using walking distance is already present in the traffic data and is part of edges presented to the search algorithm. It is unlikely that the walking distance would be calculated just by location of stations, because such calculation lacks information about obstacles or roads.
The maximum walking distance can be applied in the data selection, excluding edges that are too long. The average walking distance can be applied during calculating of the time weight of a graph edge representing the walk connection.
In the source data walk connection can be specified by its length or already by a time duration it takes. When average walking speed is not specified, there is a default value.
When walk connection is represented by distance, there is no problem in applying both parameters. But when represented by time already, it has to be converted back using the default walking speed before the parameters can be applied.

7.2.7. Reliability implementation

Optimizing reliability requires some additional data about traffic changes. Specific data about real time delays and closures are the best to work with, but they do not have to be always available. In that case it would be good to have at least data about previous problems, that would allow to predict the traffic situation to some extent.
Relevant real time reliability information was covered in the transit information chapter, like current delays, where it was incorporated into the rest of the data before it reached the search algorithm, making it transparent. That data along with their history has to be accessed through additional input channel between traffic data and search algorithm components. This data will be fed to a reliability estimation algorithm, that will provide reliability in form usable directly in the search algorithm. It would take the history of breakdowns and delays and using some

statistic methods it would produce the two values proposed in the 5.2.6., a probability of an exchange due to delay of a previous connection and a probability of failure of a connection on the way.

When no reliability data are available, it is still possible to work without them. When a reliability parameter depends completely on existing parameters, no more data is required. The number of minutes it takes for an exchange to be considered safe is based directly in the minimum exchange time. It establishes two categories for an exchange, if it is considered safe, or if it is possibly unsafe, when the exchange time is less than the specified value, but more than the minimum exchange time. This provides only one of the proposed probability values in 5.2.6. and that is a probability of an exchange due to delay of a previous connection. It may not seem so, but when the exchange would take more than a duration considered safe, it would be like if the estimation algorithm returned 100% reliability. If the the exchange would take less than a duration considered safe, but still more than the minimum duration, it would be like if the estimation algorithm returned some fixed value like 60% reliability. It is interesting to note that parameters like average walking speed and minimum exchange time are in a way reliability parameters. An exchange taking less than the minimum exchange duration is in a way automaticaly considered unsafe with 0% probability and discarded completely.

If from real sources or not, the algorithm now has the two values from 5.2.6. available. If they are not present, 100% can just be used. When the search updates the node distance, it can combine the probability of a missed exchange from the arriving connection and the probability of breakdown of the departing connection. The resulting one number will be considered for this specific exchange. Now for the implementation.

In the algorithm, the reliability faces the same issue as the number of exchanges. Will the parameter be global, with greater memory requirement, like a total chance that the trip will fail? This includes the minimum reliability of a whole trip in percents. Or will it include local limitation, like minimum reliability of each exchange on the trip or the number of travel minutes to sacrifice for a percent in reliability of one exchange.

Global restriction on reliability is much harder that the number of exchanges. With the minimum reliability of a whole route in percents, there are no fixed values in the added dimension, the memory requirements would increase with every new reliability value. Instead of remembering results for specific values, there may be intervals just like for time in one

phased algorithm.
Local limitations per station are easily implemented. The minimum reliability of each exchange is like minimum time for an exchange, algorithm decides on the spot if the exchange is possible or not, no optimization. Number of travel minutes to sacrifice for a percent in reliability of one exchange can be computed between edges locally too, incorporating it into the sum of weights.

7.2.8. Trip duration

Implementing trip duration minimization using a second time parameter, binding the trip between arrival and departure time, does not involve changing the algorithm during its graph traversing phase. It will start traversing from the specified time of departure, the difference will be that it will not stop when first suitable path is found, but it will continue until it reaches the specified arrival time. Now at the destination station it has several time intervals with several best incomming connections during them. The solution is to select the one taking minimum distance (which means the trip duration, with possible other parameters mixed in, if there were other optimizations). Normally the algorithm finishes when a first suitable connection is found, so there is nothing to choose form.
Another use for not finishing at the first solution is when more than one solution is required. Journey planners often list not just one, but several consecusive results. The longer the algorithm keeps running, the more solutions it will provide. No solution will be less than optimal, each will be optimal for its own arrival time interval.

7.2.9. Conclusion

Usefullness of basic parameters cannot be debated, a journey planner cannot work without them. Other parameters are optional, providing results for more specific user needs. When looking at existing journey planners, there is a corellation between how many transport networks and areas it covers, and how many search parameters it provides. Each network is a little different and these differences accomulate, restricting use of more and more features. That is the reason that the most universal journey planner – Google Trasit – enables only the

most basic searches, while very specific ones like Idos are free to implement parameters suited to local transports.

So search parameter usefullness or popularity does not influence much if those services will be available, only the possibility and ease of their implementation is relevant. Only when a parameter is completely useless or unlikely to be used even rarely it is then a waste of time to implement. Most important factor is whether the traffic data necessary for applying the parameter are available.

A parameter that suffers the most from the lack of data is reliability, with its usefullness tied directly with how much and how precise real time information can be used.

The implementation of advanced parameters distinguished two kinds of them, global and local. Local parameter was a restriction or an optimalization specified for part of the journey, at an exchange. Those were easy to implement with little more resources necessary. On the other hand global parameters placing restrictions on the journey as a whole led to much more resource usage. Restricting parameters could be combined as necessary, but only one optimization at a time could be performed. To combine optimizations it was necessary to define the relationship between them through travel time, by specifying how many minutes to sacrifice for a unit of that optimization.

7.3. Handling large traffic networks

When transport networks are too different from each other, it may be beneficial to allow different parameters for different modes of transport on the trip. Like different minimum exchange times. In most cases the algorithm can be customized by using already mentioned techniques in different parts of the journey.

The main problem arises with significant sizes several networks can achieve. A combination of a backbone transport like airplane travel with local bus and train networks would lead to gigantic graphs. Graph traversing would then include every little town in a continent before reaching a distant destination. Solution is to separate the search to different parts, first to find a local connection to several nearest backbone stations. Then to estimate the backbone stations nearest to the destination and find the connection on the backbone network. Final step would be find a connection from the backbone stations to the destination. Resulting trip is not guaranteed to be optimal, restriction on pass point might exclude the optimal trip, but

considering the computing resources reduction, it could be a good tradeoff.
When trying to search through different closely connected traffic networks, one solution is to use distributed computing, allowing several journey planners to work together. This was studied as part of the JourneyWeb [16] project which produced an XML protocol for this purpose, now used in Transport Direct Portal in the United Kingdom.

7.4. Components

Apart from search algorithm itself, core of the journey planner can provide additional features.
Before going into them let's specify the aplication structure. A journey planner has several layers, that can be viewed as individual components. Component design can be separated into data retrieval, data storage and preparation, search algorithm and user interface. Having each component clearly defined and separated has several advantages.
One of them is using multiple search algorithms. Many search parameters require customisations to the algorithm, that lead to significant increase in resources used like memory and computing power. When only some optional parameters are used it would be beneficial to use a search algorithm optimized for that subset of features.
Clear separation of user interface also makes running several different query methods easier, as was proposed in chapter 3.1.4.
Additionally when running a very popular journey planner, one server on the Internet may not be enough. Multiple instances of the same components on different servers can work in unison to split the load, creating a flexible pool of resources, adaptible to user traffic. Such design could even incorporate result caching, if the traffic was significant enough for many queries to repeat.

7.5. Conclusion

Many customizations require sufficient additional data and more computing resources, mainly significant amount of additional memory. For a standalone journey planner located on the mobile device it presents a tradeoff between the number of features available and the number

of devices that could support such a planner due to memory constraints.

Commercial journey planner algorithms are far more efficient than the example that was given here. Such algorithms are projects with a lot of resources given in their development and efficiency, being an important trade secrets. That is why this work does not try to compete with them by researching the algorithm in great detail, but provides instead an example on which the issues can be illustrated.

8. User interface

User interface is the part of the journey planner used to interact with the user. It handles user input and presentation of results.

If the journey planner is connected to other services, it is done in the user interface as well. The service provides part of search parameter input to the planner, while it can also share part of the whole user interface. Notable example of strong integration is the relationship of Google Transit and Google Maps.

In this chapter we will cover what features the user interface can provide along with how to design it.

8.1. Features

Apart from just specifying search parameters, there are other features user interface can provide.

Keeping a history of entered parameters can reduce the amount of input user has to type repeatedly. It can be done by providing several last used values in individual parameters or enabling specification of favourite values, much like an address book for an email. History values should not be separated for every search parameter, but for types of search parameters. A location parameter type is common to the origin, destination, exchange place and pass limitation.

A result contains information supplied from the search algorithm. Informational properties mentioned in 6.5. like a bus line number, train number and station names are retrieved separately and combined with the result provided by the search algorithm before it is presented to the user. Additionaly can they spark an interest in the user. Therefore this information may provide a link to even more detailed information, like the whole route of a specific train, detailed station properties and even realtime information about delays. This is a point where journey planners tend to offer a journey booking service, making the planner an effective advertisement tool.

After the search is completed, user might want to enter another search connected to the previous journeys found. Relationship may be journeys on the same route after the last ones,

before the last ones and in the opposite direction. For this purpose there could be a button for each of those possibilities, or more universal button for going back to the input page, while keeping the previously entered vales in the input fields, enabling the user to quickly change only a portion of parameters before searching again.
Whether reliability is included or not, user might find himself in a situation where he missed an exchange or the supplied results are otherwise unavailable. Then there could be an option to automaticly rerun the lasts search from the current point of failure to the destination. If the mobile device can supply the current location automaticly, user wouldn't even have to supply any new information.

8.2. Display differences

Three main categories of mobile device displays are available.

- A pure text display with only several lines, common for older cell phones.
- A more advanced display on PDAs and Smartphones providing graphic display not bound purely by text.
- The most advanced providing a full featured display capable of showing standard web pages.

In the first two limited categories the only option is to provide only what is absolutely necessary for the search, that is an origin, a destination and a time of departure/arrival. If the device can provide the origin based on the current location, it is even better. For customizations there should be a button for menu selection with other features available.
In the first text only category, even under options button only a few more options should be presented. Any more will make the application confusing. In the second category more options can be provided in the additional parameters section.
In the third category, all the options available can be presented like they are on a normal web page.
Additional features covered in the 8.1. chapter should not be presented on the text only displays, but in second and third categories are recommended.
Result presentation should too be customized to the environment. One advantage of result presentation over user input is that the user does not have to precisely understand it, enabling the developer more freedom without confusing the user.

8.3. Web page design

If the journey planner is a web page, it is necessary to make use of web page customization tools provided by the language, allowing to make the web page look differently for different screen sizes and display capabilites. The same is true for mobile device application parts and their user interfaces. Developing languages provide this information through libraries and variables.

The dotMobi top-level domain on the Internet is a movement started by a cooperation of mobile device manufactures to provide web content suited for mobile devices. Together with W3C Mobile Web Initiative it developped Mobile Web Best Practices [17] as guidelines for creating mobile web content, along with an online tool to check web pages for mobile web readiness [18].

While dotMobi domain provides mobile content directly, there are some services that provide the content indirectly. One example is Opera Mini [19] mobile web browser. The viewed web page is first loaded through a proxy server on the Internet as a whole web page. In there it is converted and compressed into a mobile format, in this case OBML (Opera Binary Markup Language). From the proxy server the data reaches the mobile device already transformed, suitable for limited transfer speeds and more compatible with mobile device screens. This approach is similar to WAP gateways, that provided mostly just WAP customized web pages. This approach enables access to most web pages on the Internet.

Newer devices do not need special web page formats and enable a full web browsing. Only limitation remaining is the screens size, which is solved by scrolling.

8.4. Popularity

After the journey planner is completed, it has to find its way on the mobile devices. Before Google Transit, there was not much of a choice for users. Journey planners of each area were mainly designated by local authorities or decided by traffic companies. Google Transit provided an alternative to local applications, as well as promoting global standartization of traffic data. That openned the market not just for journey planners, but other traffic applications.

User has to first know of the journey planner's existance in order to use it. One way to promote it is by advertising, other by including it in the standard aplication set of new mobile

devices, like Google Maps in iPhone. Such move requires strong cooperation with device manufacturer, but is the most effective.

Apart from knowing about it, a factor is the trouble of installation. Web pages can at most provide a bookmark, application parts need to be installed. Installation can be handled with one click or take the user through a dialog hell, which can turn away many users.

Other factors which decides if the user will be willing to use the application are:

- Ease of use – Depends on how confusing the planner is to operate and how clear are the instructions provided by the result.
- Feature set – How many features does the planner provide and of what quality. Competes with the ease of use, since often more features makes the application less intuitive.
- Compatibility – Directly influences popularity by increasing the base of potentional users.

8.5. Summary

User interface has shown to be more than a way to hande the user input and showing results. Apart from providing additional features, it is a face of the journey planner, heavily influencing how users will percieve it. Without an appealing user interface, the best journey planners can be discarded as confusing and useless.

9. Conclusion

In this work we have looked at the aspects of journey planning on mobile devices.
First we studied the mobile device itself, showing great differences in power and capabilities. Next it we discussed what structure the tool can have, revealing the conflict between the ease of use of web pages and the universal support od SMS messages. Afterwards we analyzed the development environment providing the benefits of using Java ME, while showing that additional compatibility can be achieved by programming several frontends in other languages as they are supported by the device manufacturers. Next we examined existing journey planners along with specific search parameters revealing that their structure and capabilities depend on how many areas they are trying to cover and how much different transports they are trying to combine. Their accessibility and ease of use from the mobile device perspective however depend more on how much effort it is given in providing it.
Following we researched the source and form of input traffic data, both of static and dynamic information. We showed that the capability dependence on covering different transports and areas is based on the different traffic data provided for them. Also we divided the purpose of the data between the search algorithm input and the information for user's convenience.
To illustrate implementation difficulties we presented an example algorithm along with possible ways to incorporate individual search parameters. How hard a search parameter could be implemented did not depend on its usefullness. To reduce the complexity of the problem we recommended to distribute it or to make use of component design advantages.
In another part of the work we studied the user interface, revealing its responsibilities other than to hande the user input. It also can provide additional information and ease the user input. We proposed a proper interface design along with tools to create it.

Apart from common features we investigated a possible aspect of journey reliability, how to specify and implement it. The main problem it faced is the lack of traffic data that are necessary for reliability estimation.
Original goal of this work was to develop a journey planner providing features and universality others do not, but because of the clarification of feature limitations and rapid development of existing journey planners with much greater resources for their development, the goal became obsolete. That is why I shifted the focus of the work to provide deeper

understanding of the subject.

Growing popularity of mobile devices along with fast hardware advances will soon push advising applications more into our lives. Future comes when a journey planner will be just one of many tools ready to give us guidance at anytime, anywhere.

References

[1] Mobile phone, http://en.wikipedia.org/wiki/Mobile_phone

[2] PDA, http://en.wikipedia.org/wiki/Personal_digital_assistant

[3] SMS, http://en.wikipedia.org/wiki/Mobile_phone

[4] Clickatell, Available: http://www.clickatell.com

[5] The Kannel Group, (2009), Kannel: Open Source WAP and SMS gateway. http://www.kannel.org

[6] Oracle Corporation, (2010), Java ME Technology. http://java.sun.com/javame/technology

[7] Canalys, (2010), Worldwide smart phone market by OS vendor. http://www.canalys.com/pr/2010/r2010021.html

[8] Microsoft Windows Mobile operating systems, http://www.microsoft.com/windowsmobile

[9] Microsoft Windows Embedded Studio development tools, http://msdn2.microsoft.com/en-us/embedded

[10] Google Transit, http://www.google.com/transit

[11] Transport Direct Portal, http://www.transportdirect.info

[12] Idos, http://jizdnirady.idnes.cz

[13] Smartřády, http://www.smartrady.cz

[14] CEN TC278, Reference Data Model For Public Transport, EN12896

[15] Dijkstra's algorithm, http://en.wikipedia.org/wiki/Dijkstra's_algorithm

[16] JourneyWeb, (2004), http://www.dft.gov.uk/journeyweb

[17] W3C Mobile Web Best Practices, http://www.w3.org/TR/mobile-bp

[18] W3C mobileOK Checker, (2004), http://validator.w3.org/mobile

[19] Opera Mini, http://www.opera.com/mobile