Experience Augmented Pedestrian Wayfinding

Master Thesis, 30 credits

Håkon Arneng Holmstedt

January 8, 2007
Halden, Norway

Østfold University College
Mobile Applications Group

Abstract

This thesis explores collaborative route planning for pedestrians, where the users's experience with the system and with the suggested routes augment the route planning for other users.

I describe a technique for sharing experiences between groups of users that share some common characteristics. The technique is demonstrated with an experimental application running on Smartphones and PDAs.

The device that the application is developed on is connected to a GPS device via a Bluetooth connection, and it uses open, freely available Web Map Services (WMS) for map images. The application allows for point to point route planning, searching for a route to the closest point of interest of a particular kind (f.ex. the closest accessible parking space), user feedback to routes, GPS tracking and recording of user movements, and creation of new roads by the individual user.

Route planning is performed using the classic A* algorithm on a road network consisting both of proprietary data (from the Norwegian Mapping Authority) and open data (from the Open Street Maps project).

Keywords: Collaboration, Mobile Devices, Pedestrian Navigation, Route Planning

Acknowledgements

This thesis would be impossible to write without the help and support of a number of different people and institutions.

First and foremost, I would like to thank my advisor, Gunnar Misund, for helping me focus on the important issues and for making me understand the errors of my ways when I have strayed.

I would like to thank my family and my very dear Kate Tonkin for their steadfast support and patience with me during my long, academic isolation. Without them, I would certainly be lost a long time ago.

I would also like to extend my thanks to Harald K. Jansson and Torbjørn Halvorsen for their cooperation and friendship during our work on the Okapi Framework. They have been a steady stream of ideas and I have learned a lot by working with these fine gentlemen.

The Østfold branch of the Norwegian Labour and Welfare Organisation, Halden Ergotherapy-service, and the Halden chapter of the Norwegian Association of Disabled have been most helpful in regards to testing the system, and I would like to thank them for their enthusiasm and aid.

Table of Contents

Chapter 1

Introduction

In my thesis I will explore the problems of route planning for pedestrians, and I will focus on pedestrians that have impaired mobility such as wheelchair users.

Route planning is, in simple terms, the act of finding a way to move from one point to another. For example, if you are sitting in your car and you want to go a particular place, you first need to know how to get there. In most cases, you already know the way and simply drive the same route as you always do, but sometimes you need to cover unfamiliar territory. In such cases, you may consult a digital route planner.

The market is replete with route planning devices that one can use to find the way when one is driving. These devices uses positioning systems to discover where in the world the car is and come equipped with detailed road information that helps it calculating a route. The final route is usually represented by visual clues overlayed a map, and there is often a voice telling you when to turn left or right.

Although route planning for cars is by far the most common type of route planning in the contemporary consumer market, one can also imagine uses for route planning for so-called soft trafficants. These trafficants include pedestrians, cyclists, and similar. For example, a cyclist may want to find a scenic route through a particular area of rural landscape, or a wheelchair user in a new town may wish to find the way to the nearest hotel with wheelchair ramps.

Pedestrian route planning introduces some issues that are not seen in the more common route planning scenarios. One such issue is the map that needs to be far more detailed than the maps used for vehicular route planning (route planning for cars and other road-vehicles). With the latter form of route planning, one needs only know a few things like where there are intersections, whether a particular road is one-way or not, and possibly what the speed limits are. However, in the case

of pedestrian route planning, it becomes necessary to include such things as information about sidewalks and crosswalks.

A further complication of pedestrian route planning lies in the underlying algorithms. Regular route planning employ a number of optimization based on some common assumptions. For example, in a car one can assume that it is good to get onto a highway as early as possible in the trip and only get off the highway when necessary. The highway is a road that offers high speeds and ease of navigation. By assuming this, one can reduce the route planning problem to include small roads and by-ways only near the start and end of the proposed trip. However, there are no good analogies to the highway for the pedestrian. In essence, any road is a good road and no road offers higher speeds in and of themselves. It then becomes impossible to reduce the planning problem in the same way as with vehicular route planning.

To further complicate matters, pedestrians come with a mulitude of different needs. A dramatic example is the difference between a wheelchair user and blind person. Both want to use the sidewalks and crosswalks available in a city, but one is hindered by tall sidewalks that hamper the wheels of the wheelchair, while the other is helped by the very same tall sidewalks that represent good clues to where the sidewalk ends and the road begins.

Clearly, the issues surrounding route planning for pedestrians with disabilities are many and challenging. I want to find out:

> *Is it is possible to create a mobile application for pedestrian routeplanning that give good routes for users with different and sometimes conflicting needs and interests?*

I will try to answer this by attempting to create an application that performs such a task. To achieve this, I will need to solve a number of sub-tasks:

- Create a mobile application capable of displaying maps and routes
- Create a mechanism for generating routes
- Create a mechanism for determining the subjective quality of a route and apply this to the routeplanning mechanism

If the three subgoals are met, I believe that I have shown that the issue outlined in the problem statement is indeed solvable with contemporary technology and data. The key to the problem lies in the third subgoal where routes are assigned some subjective measure of quality. Done right, this gives rise to personalized routes as the routeplanning algorithm will take into account the user's preferences through this measure.

The problem I want to explore relates to whether or not currently available technology and data is enough to create an application as I have outlined above, rather than looking at the human-computer interface (HCI) aspects of developing mobile applications.

Methodology

This Master Thesis is an exploration into the feasability of personalized routeplanning on mobile devices. Because of this, it is natural to focus on the practical aspects of the problem, i.e. a prototype application. I will persue the problem as a practical case study. The case will be to develop a proof of concept application that solves or at least highlights the issues involved in creating a personalized mobile navigation aid for pedestrians.

Development of this prototype will occur in a number of steps:

1. *Pre-analysis.* This step includes gathering background material for the project. I will need to learn about classic route planning techniques as well as gaining an understanding of current trends and developments in both the mobile applications field and the broader field of collaborative information systems.

2. *Analysis.* In this step I will consolidate relevant knowledge for the project and decide on suitable platforms and frameworks for the project. The step will also include selecting a relevant user group for future testing and determining what specific tasks a prototype must be able to perform in order to adress the problem statement of the thesis.

3. *Design.* Here the prototype will be designed, i.e. I will develop a data-model for the domain as well as deciding on the architecture of the prototype. The design phase will focus especially on capturing the interaction between user collaboration and personalization.

4. *Implementation and testing.* The final step includes developing the final prototype and perform a number of tests to discover whether or not the concepts developed in earlier steps are sound. This step includes a meeting with actual users to further anchor the prototype in the real world.

The prototype that I will design and develop is intended as a proof of concept. That limits the amount of time I will spend on otherwise critical factors such as usability and robustness. On the other hand, I expect the prototype to perform the necessary tasks identified in the analysis step.

Document outline

This thesis is organized in five chapters, where the first is the introduction you have just read. Chapter two covers various topics that serve as a backdrop for this thesis. It will introduce the reader to two projects that have inspired my own project. It will also go over the technologies necessary for the realization of the project, and give some details regarding route planning including common algorithms. Finally, chapter two will give a brief overview of previous work done in the field of pedestrian navigation.

In the third chapter I give a detailed description of my suggested solution with a design and some notes on the final implementation. Chapter four details testing of the system, and in chapter five I reflect over the results of my work.

Chapter 2

Background

Before I go into the details of what I want to do with this project, I will discuss a numer of related topics. First, I want to look at two other projects that have a direct link to my work. Then I will review various technologies available that can contribute to realize the project. I will proceed with discussing route planning more indepth, looking at some selected algorithms and datastructures amongst other things. There is also some previous work on pedestrian navigation that I will examine before I offer a preliminary outline of the application I will attempt to create.

2.1 Project background

There are two projects that serve as a foundation for my thesis. One is the Accessiblity project which provides among other things a case for the project, i.e. a particular set of users that can provide a focus for the design. The other project is called the Okapi Framework which is an application for Windows Mobile that I will use as a platform for my own work.

2.1.1 The Accessibility Project

The Accessibility Project is a joint project headed by the Norwegian Mapping Authority. The project's initial aim was to provide maps to help physically disabled people navigate the streets of Oslo. In the future, the project is expected to branch out to several major cities in Norway.

The project is divided into three phases where the first phase consisted of gathering accessibility data and presenting this on a paper map[3] (see figure 2.1 for a small cutout of the map). The data that was gathered was on the quality of the sidewalks, the presence of traffic lights, and the presence

of street-crossings. The map also included data from a previously created Accessibility Guide which is a catalogue of accessible resturants, shopping malls, etc. in Oslo.

The second phase of the project is adopting the paper map for web publishing. This phase is being handled by the private company Norkart [34] and has resulted in an interactive map of Oslo showing all the same data as the paper map. In addition, users are able to turn on and off various features as they see fit. For example, one might not be interested in the multitude of wheelchair accessible museums if one is looking for how to get from the train station to the hotel, and in such a case one could simply turn off museums and see a map with only pertinent information.

In the third phase of the project, the Accessibility map is to be made available on hand-held devices. In this phase, the Østfold University College has participated by offering prototypes and ideas for solutions.

In addition to the Norwegian Mapping Authority and Norkart, the Norwegian Association for the Disabled is involved in the project. They supply test users and the most pertinent domain knowledge.

2.1.2 Okapi

Okapi is a general pedestrian geotagging tool created at the Østfold University College by myself and two other students during the spring of 2006 as a redesign of an earlier project by the same name [18]. The aim of Okapi was twofold: Firstly, the application would provide the user with the ability to browse maps on a mobile phone or Pocket PC. Secondly, the application would allow the user to create so-called geotags or points of interest (POIs).

The application has been used in the Accessibility project as a prototype for its third phase. There are also suggestions that is should be used in data gathering for future expansions of the project.

Figure 2.2 shows mapbrowsing in the first Okapi version where user generated POIs are represented by smilies.

Geotagging

The greatest strength of the Okapi application was in its focus on user generated content. It allowed each user to tag the map with a wide variety of information, including texts, images, sounds, and videos. Furthermore, it allowed the user to share this information with any other user of the system. This way, Okapi allowed people to share geographically positioned information in a quick and

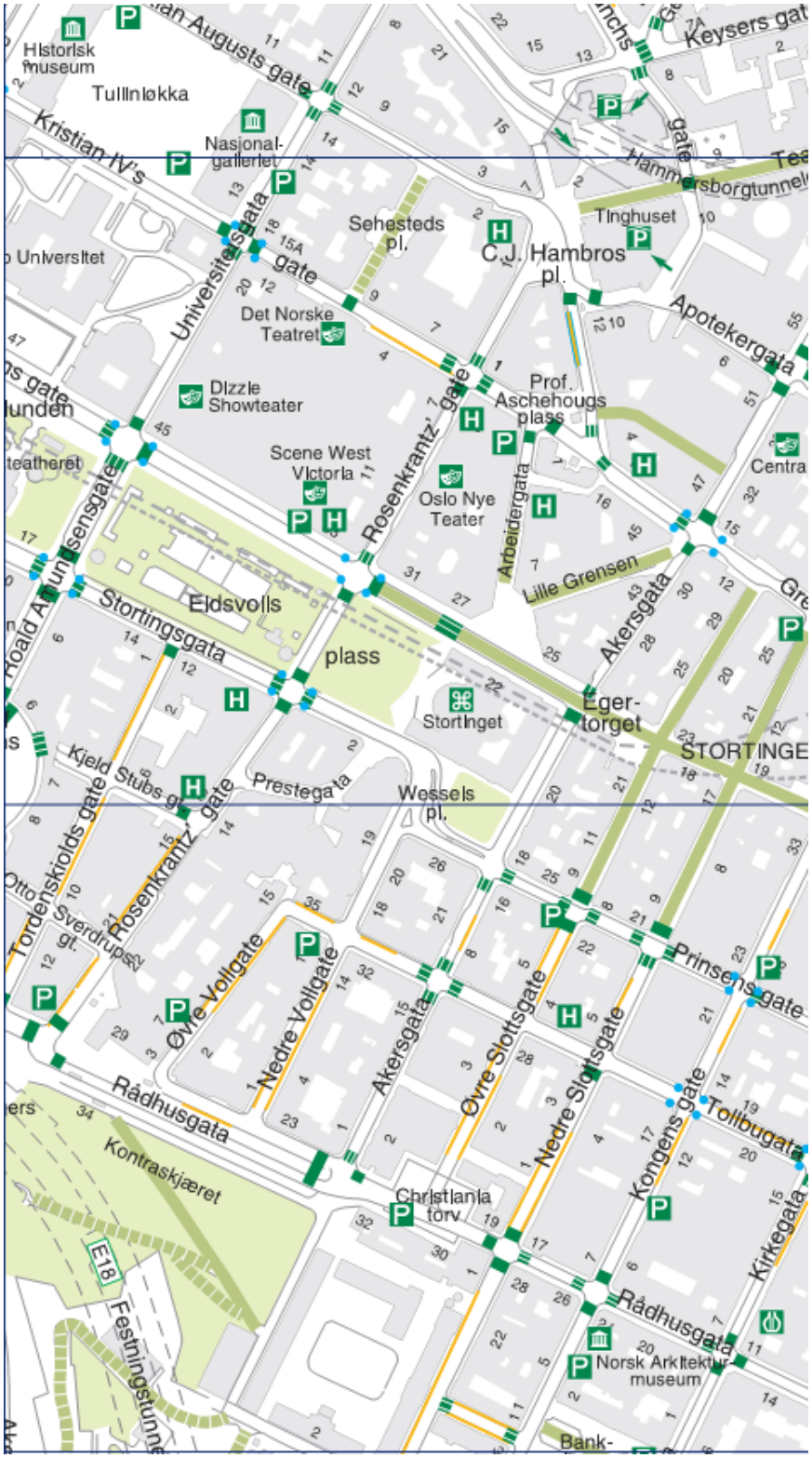


Figure 2.1: A sample of the first Accessibility map.

easy way. People could create tags showing places with a great view, restaurants with good food, or anything else they wished to share. They could complement the location and description with images of the topic at hand, for example the food served at a restaurant or a panoramic view, and they could even add spoken commentary or videos.

In the Okapi application, each tag is called a Point of Interest (POI), and these are displayed on the map as clickable icons. When an icon is clicked, the name of the POI is displayed along with a short description. By further clicking the user can see any additional media, like images or sounds that are associated with the POI. The user is also able to edit a POI by adding more text or other media.

Maps and User Positioning

Apart from POIs, the basic Okapi is a mapbrowser written in C♯ for Windows Mobile 5.0. It features map tiling for rapid displaying of maps, three tile sets, and user-positioning.

The Okapi server offers three basic tile sets: The Standard map (figure 2.3), the Satellite map (figure 2.4), and the Hybrid map (figure 2.5). The Standard map is a selection of layers offered by the Arealis Web Map Service (WMS) [24], although we are currently looking at the possibility of adopting other WMSs that give more detailed images. This scheme is intended for everyday browsing and navigation. It has relatively clean images and occasional texts to help the user orient himself.

The Satellite map is a simple map that takes its images from NASA's Global Mosaic WMS [33]. This WMS gives relatively detailed satellite imagery of the entire world, and is currently the only global map service used by Okapi.

Finally, the Hybrid map combines the GlobalMosaic and the Arealis images to show roads and buildings on top of a satellite image. Due to the low quality of the satellite images and the relatively small scale necessary to show building mass, this scheme should be considered more of a curiosity than a usefull feature. The inclusion of the scheme has been an exercise in combining multiple WMSs and shows us that this is something we can do successfully and that we should look into more at a later point.

In addition to the various maps and the ability to tag the map, Okapi offers rudimentary positioning. If the user has attached a Global Positioning System (GPS) device to the Pocket PC, Okapi can find this device and interpret the signals. This way, the system can display the user's own position on the map, and it is also able to continously update the map, so that the user is always in the center of the displayed map.

Figure 2.2: A screenshot of the first version of Okapi

Figure 2.3: Standard map showing downtown Oslo. Images courtesy of the Arealis WMS.

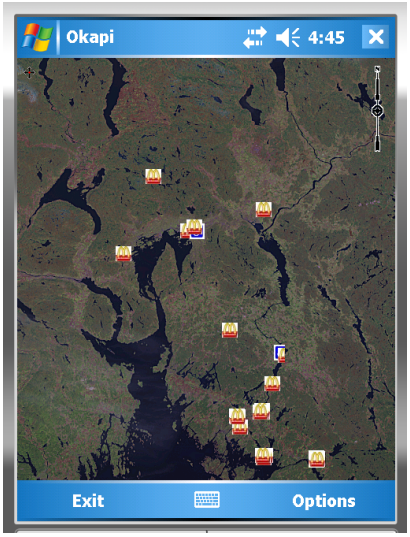


Figure 2.4: Satellite images showing Oslo and surrounding areas. Image courtesy of NASA's GlobalMosaic WMS.

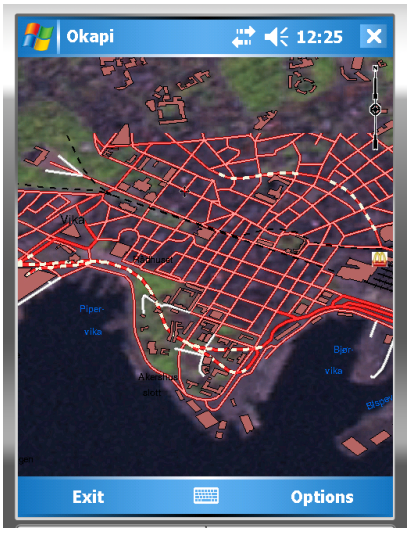


Figure 2.5: Hybrid map showing downtown Oslo. Images are a combination of NASA's GlobalMosaic and the Arealis WMS.

2.2 Technological foundation

In addition to the parent projects described above, the project will be founded on a number of different technologies that deserve some illumination.

2.2.1 Mobile devices

The application I plan on implementing will be heavily influenced by the choice of platform. Since I want the application to be used by people that are out in the real world, it becomes natural to choose a mobile device. At the time of writing Microsoft's Windows Mobile 5.0 seems the most promising in terms of both features and future compatability; It is guaranteed to have the same feature-set regardeless of the individual device, something that is sadly not currently true about Java MIDP or other platforms.

Developing for mobile devices introduce opportunities as well as challenges not seen in a desktop environment. These stem as much from the limitations and features of the devices themselves as from the context in which the devices are used.

Advantages of mobile devices

Choosing a mobile platform for developing one's application opens up a number of possibilities that one lacks on other platforms.

First and foremost, one can usually assume that the user is always around the device. That is, the user tends to carry the device around wherever they go. This leads to a major opportunity: The developer can assume that wherever the device is in physical space has some bearing on the user's location and even motivation. Long et al. described as early as 1996 a location aware family of mobile applications [29], and a number of commercial applications have been developed to take advantage of location. These applications range from the simple location based advertisement (i.e. when a user is inside a supermarket, one can push grocery-ads to his mobile phone) to location based games like Geodashing [14].

Another important difference beween the mobile platform and a desktop/laptop environment is that the mobile device is usually turned on. This allows the device to engage in continous activity that desktops are not able to. For example, one can usually assume that the device is available for network contact whenever an outside service needs or wants to push information to the client.

Finally, most mobile devices are almost always connected to some network or other. This network could be anything from a broadband WLAN to a simple GSM connection, but the result is the

same: A service provider is always able to get in touch with the device. This attribute of the mobile platform lends itself naturally to push-services. A provider can at any point send information of any kind to the mobile device. The device may then choose to either alert the user or deal with the new information in some other way. For example, a map application could be silently updating its map-set while the user is not useing the device for anything else. This way, the device always has an up-to-date map of the area it is in, and the user will have a better experience.

Special challenges with limited devices

There are a number of issues that a developer must be concious of when targeting a mobile platform.

The first problem facing developers for mobile devices is the inherent dangers in using a device that runs on batteries. Every time the device is used, it draws on its batteries for power, and once the batteries are drained, the device stops working. Unfortunatly, heavy use of the device's processor will drain the batteries and in some cases drain them fairly quickly. It is therefore necessary for the developers to make sure their application doesn't perform any unneeded actions, like keeping user interfaces running that are no longer usable. Furthermore, additional services the phone may offer, like WiFi or Bluetooth antennas use even more battery power. The developer must therefore try to minimize the time spent using these services. This can be achieved by, for example, postponing any network activity until no more actions can be taken without the network, then perform all the network activities necessary before shutting down the connection again.

Another problem with mobile devices is the unreliable nature of both the device and the networks the device uses. A running application may for example experience loss of network as the user moves out of range of any antenna or cell-tower. More dramatically, the device may decide to shut down from lack of power, or the user may abruptly turn the device off. The consequence of this is an application should never put the device in a state it cannot recover from. For example, if an application runs a database, it should have the necessary tools to recover from an aborted write action.

An issue that is more in the realm of user interface design is that of user focus. In contrast with a desktop environment where the user is assumed to be sitting down to do a piece of work, a mobile device is usually used casually. The user tends to keep the device active for only short periods of time, and is used to put the device away at any time and for any reason. In essence, the user's concentration is rarely on the device.

Finally, with a device that is easily stolen or forgotten in a multitude of locations, privacy becomes a concern. One can therefore never assume that the user is the owner of the device, and so

storing personal information like passwords becomes an unacceptable risk.

2.2.2 Web Map Services

A Web Map Service (WMS) is a server that lets users download maps according to a specific set of rules.

The WMS interface was developed by the OpenGIS Consortium (OGC) to allow clients to access maps on any server and even combine maps from any number of servers in a quick and easy fashion. The WMS specification goes into detail about how a server should treat various queries and how the response should be fashioned [6]. It allows a client to specify a large range of options that mostly boil down to what the map is to be a map of. These options are called layers and determine the kind of data that the WMS will make use of when drawing the map. For example, the Arealis WMS has a great many layers available for Norway. A client could then ask for a map with soil-types, placenames, and roads (figure 2.6(a)) or a map with bodies of water, height curves, and such (figure 2.6(b)). The user can also specify such things as the size of the resultant picture in pixels, the format of the image (i.e. PNG, JPEG, etc.), and of course the geographic bounds of the image.

However, the greatest strength of the WMS specification becomes appearant whenever two or more servers implement the interface correctly. Figures 2.6(c), 2.6(d), and 2.6(e) show a scenario where a client downloads a road network from Arealis and a background image from OnEarth. Since both these services implement the WMS interface it is trivial to combine the two images into a new map showing roads on a satellite image.

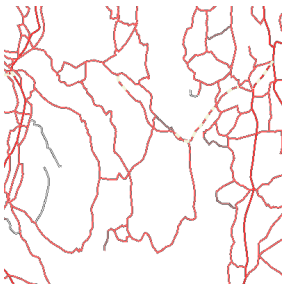
The WMS specification is actually one of several specifications developed by OGC to create a interoperable set of services for the exchange of geographic information. Another specification is the Web Feature Service (WFS) [48]. The WFS defines an interface to get access to vector data. This is the raw data used for drawing maps, and access to this allows a client to both draw their own maps or do other more advanced analysis. The data is presented in the XML-language GML (Geographic Markup Language) that is also created by OGC for the exchange of raw geographic data.

2.2.3 Positioning

As mentioned earlier, location becomes an important parameter when dealing with mobile devices. Here I will introduce some common means of positioning a mobile device and I will look especially at outdoors positioning using the Global Positioning System.

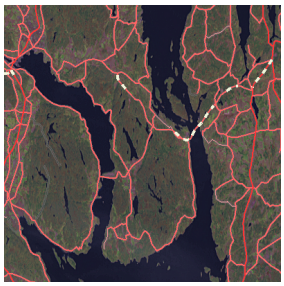
(a) The request asked for soiltype, roads, placenames, and municipality borders

(b) The request asked for bodies of water, roads, height curves, and municipality borders



(c) A request for only roads from the Arealis WMS

(d) A request for a satellite image from the OnEarth WMS



(e) Blending the two resulting images produces a nice road map

Figure 2.6: Some examples of WMS use

There are two basic approaches to mobile positioning resulting in two very different situations, the first of which is self-positioning. With this approach the mobile device uses some way of discovering its location and can modify its behaviour, or the behaviour of onboard applications, accordingly. The other approach is remote positioning, which occurs when a mobile device's position is calculated by some separate device or process. This kind of positioning is more interesting in top-down systems like billing systems, or in situations where a third-party needs to know someone's location. The latter situation covers emergency services that try to locate missing persons by discovering the location of their mobile phones such as E911 [11] and E112 [45]. Of the two approaches, the first, self-positioning, is the most interesting for my work.

Self-positioning comes in a number of flavours, each with its own strengths and weaknesses. Evennou et al. suggests a method for indoors positioning using particle filtering, knowledge of the map of the user's surroundings, and the signal strength of the local wireless network to get a fairly accurate location of the user's position in the building [8]. Another method is to use ultra wide band radio signals, and such a method can yield fairly good results [22]. Even better precision can be had by exploiting ultrasound for positioning [19]. All of these methods have two major problems, though. One is that they only work in predetermined and prepared areas, i.e. one needs to set up the infrastructure for positioning in the exact location the system is to be used. Another problem, at least in terms of usefullness in this thesis, is that they are all meant for indoors use.

Outdoors self-positioning is fortunatly far easier and allows for near universal service. There are two basic methods for ubiquitous, outdoor self-positioning: Global Positioning System (GPS) or equivalent and Mobile Terminal Positioning over Satellite UMTS [51]. Both methods exploit the global availability of satellites to calculate the location of the mobile device. The latter method offers a precision of only about 150 meters, but puts fewer demands on available satellites than GPS. Such precision can be enough in many situations, but for routeplanning where the route might not be much longer than a few hundred meters, GPS becomes the only real alternative.

The Global Positioning System

The GPS is a positioning system owned and operated by the United States armed forces and was heralded by the first launch of an experimental GPS satellite in 1978. Today a large number of such satellites orbit the earth, providing fairly accurate time and place positioning to anyone with a receiver capable of intercepting the signals.

A GPS receiver is a device that is capable of intercepting the GPS signals from the various

satellites and interpreting them. Such devices are commonly available on the market and the basic models cost less than an entry level mobile phone. As location aware services become more common, we can also expect to see an increasing number of mobile devices with integrated GPS receivers. For now, however, the most common receivers are car navigation computers and standalone receivers with bluetooth antennas for communication with other devices.

Provided one has a GPS receiver, one can ascertain a number of things. The receiver will output in any way it can a number of NMEA strings. These strings are well formatted strings encapsulating the appropriate data. Each string encapsulates a different set of data, so that an application usually only reads the string that most closely serves the application's needs.

Table 2.1 shows three selected NMEA strings that are commonly used by GPS applications. I will detail the parsing of the first string as it contains the most information and is the most pertinent to the thesis.

The common GPGGA string details the GPS fix data gathered by the GPS device. That is, this string contains information about where the GPS receiver is in time and space, in addition to some information about the quality of the data. The string consists of 16 fields as seen in table 2.2. Some of these fields take more interpretation than others, and I shall attempt to explain the ones that are not intuitive.

The second field, Time, is expressed as hours, minutes, and seconds in Zulu time. Using the example value of 151023 we get the time 15:10:23 Z which is ten past three in the afternoon in London.

Latitude and longitude are also compounded values and they are parsed in the same way. The first two digits represents degrees, and the final six digits represent minutes. So the value 5923.6843 means 59 degrees and 23.6843 minutes which in the more humanly readable form is 59' 23" and 41.058 seconds. Fields four and 6 simply state whether the preceeding value is north or south (or east or west) of the equator (or the Greenwich meridian).

Fix quality can be either 0 for no fix, 1 for GPS fix, or 2 for DGPS fix. DGPS fix means that the device is receiving signals from a special ground station that can increase the accuracy of the fix to less than a centimeter. DGPS is a service available mainly to military operatives or civilians that are willing to pay large sums of money for that kind of precision.

Horizontal Dilution of Precision (HDOP) is a measure of the quality of the fix and the number is calculated internally by the GPS receiver. The lower the HDOP number is, the better the quality of the fix. A number below 4 is considered good, while a number above 6 is not to be trusted. In between lies gray areas where the application needs to make some judgement calls as to whether

NMEA	Description	Example
GPGGA	GPS fix data	$GPGGA,151023,5283.6843,N,1109.1079,E, 1,05,1.5,150.3,M,12.0,M,,*75
GPGSA	Satellite overview data	$GPGSA,A,3,04,05,,09,12,,,24,,,,,2.5,1.3,2.1*39
GPZDA	Date and time	$GPZDA,201530.00,04,07,2002,00,00*6E

Table 2.1: A few typical NMEA strings

Field	Description	Example
1	Sentence identifier	$GPPGA
2	Time	151023
3	Latitude	5923.6843
4	Latitude hemisphere indicator	N
5	Longitude	1109.1079
6	Longitude hemisphere indicator	E
7	Fix quality	1
8	Number of satellites seen by this receiver	05
9	Horizontal Dilution of Precision	1.5
10	Altitude	150.3
11	Unit measure of altitude	M
12	Height above the WGS84 ellipsoid	12.0
13	Unit measure of height	M
14	Time since last DGPS update	blank
15	DGPS reference station id	blank
16	Checksum	*75

Table 2.2: The 16 fields of the GPGGA string

the fix should be used or not.

The fields referring to DGPS updates and station ids are not commonly used in civilian applications. They refer to the positioning aid that can be rendered from specialized ground stations in certain areas. However, due to the prohibitive price to access such station, most GPS receivers are not capable of this kind of positioning.

The checksum is a value that can be used to ensure that the NMEA string was transmitted correctly. The checksum can be calculated by performing an 8-bit exclusive OR operation on the entire string between the $ and * delimiters.

A sampling of other positioning methods

Two other means of positioning worth mentioning are positioning by cell id and positioning through a gazetteer.

Cell id positioning is a somewhat crude, but functional positioning technique where the mobile device is located by figuring out what cell tower it is currently communicating with. In the simplest form, the device is simply associated with the current radio tower it uses and assumes it is within some distance of this. Such a method of positioning gives a fairly inaccurate position, but it can be improved upon. A mobile phone does not only see its current cell-tower, but it can also see a number of other weaker signals. If the device knows the position of three towers, and can figure out the strength of each signal, it can calculate its position with a relatively high degree of accuracy.

An even less accurate positioning method is positioning by gazetteer. Using this method the user inputs his location as a text string, for example the user may say that he is in Oslo. This textual position is then submitted to a gazetteer-service that knows the location of several such places and can give this in geographic coordinates. This method gives a very poor accuracy for positioning, but for some situations, it is good enough.

2.2.4 Data

For this project I have three main sources of data. I get my road data from the VBASE database and from the Open Street Maps project, and I get information about special points of interest from the Accessibility project.

VBASE: Road Network with SOSI

VBASE is a centerline road network database that I have been given partial insight into via the Accessibility Project. This data is represented using the SOSI (Samordnet Opplegg for Stedfested Informasjon, i.e. Coordinated Methods for Locative Information)[41] standard developed and maintained by the Norwegian Mapping Authority and others.

The data is presented as a flat text file and it is defined by a Baccus-Naur grammar. First in the file is a header that describes the data. This is followed by a number of divider lines and then the actual data. The VBASE file I have been allowed to use has a header that looks like figure 2.7. The lines that will be important to my project are 13, 14, and 15. These describe the coordinate system used, the coordinate shift to be applied, and the size of the units in the file.

The coordinate systems used in SOSI files are described using a single number that correlates to the SOSI table of coordinate systems [42]. KOORDSYS 22 means that the file uses Universial Tranversal Mercator zone 32 coordinates based on the EUREF89/WGS84 datum.

The SOSI header also defines a relative origin for the file, although in this case, this origin is

```
01 .HODE
02 ..TEGNSETT ISO8859-10
03 ..OMRÅDE
04 ...MIN-NØ  6631301 582962
05 ...MAX-NØ  6667873 609395
06 ..SOSI-VERSJON 3.2
07 ..SOSI-NIVÅ 3
08 ..EIER "Statens kartverk"
09 ..VVERS
10 ...VVERSNAVN "2005 VEGNETT    "
11 ...VVERSDATO 20041011
12 ..TRANSPAR
13 ...KOORDSYS 22
14 ...ORIGO-NØ 0  0
15 ...ENHET 0.01
16 ..OVERORD_KVALITET
17 ...PROSESS_HISTORIE "20051103 - Trans (SKT2NOR1): fra 3 til 22"
```

Figure 2.7: The header for a partial VBASE export to SOSI, comments have been removed

same as that of the coordinate system. The origin simply defines a constant value pair that should be added to any coordinate pair found in the file.

Finally, the SOSI header defines a unit value. This is usually some number between zero and one that is used to find the actual coordinate. In the case of the VBASE file, the unit value is 0.01 meaning that every value found in the file is one hundred times larger than the actual coordinate.

Putting all of this together and looking at an example curve (figure 2.8) we find that the first coordinate, on line 14, is ”664330351 60310466 14840 ...KP 1”. Applying the trivial origin and the unit we get $(664330351 + 0) * 0.01$ $(60310466 + 0) * 0.01$ $14840 * 0.01$ which gives a coordinate of 6643303.51 603104.66 148.4. If we translate this to geographic coordinates while maintaining the WGS84 datum we get the coordinate 59.9143067 North, 10.8438852 East and a height of 148.4 meters above sea level. Finally, the last token of information on this line, ”...KP 1”, means that this point is belongs to more than one curve and is to be considered a node in the graph representation of the network.

Parsing and reprojecting the VBASE data gives a dataset that adequately represents Oslo for the purposes of this project.

```
01 .KURVE 15:
02 ..OBJTYPE SenterlinjeVeg
03 ..LTEMA 7001
04 ..KOMM 0301
05 ..DATO 19970602
06 ..KVALITET 22 36
07 ..GATE 20202
08 ..VLENKE
09 ...VNR K V 20202
10 ...VPA 2 1561 1756
11 ...VKJORFLT 12
12 ...VLENKEID 13734
13 ..NØH
14 664330351 60310466 14840 ...KP 1
15 ..NØH
16 664326849 60311167 14830
17 664325288 60311517 14880
18 664324136 60311847 14900
19 664320674 60312927 14970
20 664317772 60314013 15110
21 664316507 60314566 15180
22 664313177 60316123 15370
23 664311984 60316804 15420 ...KP 1
```

Figure 2.8: A single object from the VBASE Oslo data.

Open Street Maps

The Open Street Maps (OSM) project is a collaborative project aimed at providing open, free, and accurate streetmaps of every city on the planet. The original project started out by mapping London, but as the number of volunteers attached to it grew, it came to cover more and more cities.

The project relies on volunteers tracking their movements using GPS devices. These tracks are uploaded to the OSM servers where they are analyzed and merged with the data already present. The resulting data is available for perusal by anyone either through the OSM websites or through any other application that connects to the OSM servers.

The way I have accessed the OSM data is through an application called JOSM (Java Open Street Maps). This application allows me to select a rectangular portion of the earth and export this as XML data. The XML schema for this data defines nodes, segments, and ways, all of which are identified by unique integers.

A node in the OSM schema is a single point in space. This point may have additional data associated with it, such as a description or a label, but are meant as anchors for higher level structures. A segment is an ordered pair of nodes that define a traversible road while a way is an ordered list of segments describing any road, path, or otherwise that has been uploaded to the project's servers.

The schema also defines a datastructure known as 'area'. This datastructure is not used in my project and does not appear in the data export I made for Halden.

Accessibility: Points of Interest

In addition to the road map, it would be of interest to include data on points of special note. During the first phase of the Accessibility Project mentioned earlier, a lot of data was gathered that is best represented as points. The project gathered information about accessible toilets, museums, hotels, parking, and a host of other subjects. This information has been made available to me in the same format as the VBASE data, and this allows me to include it in my route planning.

Since I have the data for the roads in Oslo, it should not be very hard to associate these points of interest to locations in Oslo and allow a user to ask for routes leading to a particular feature, like a museum with wheelchair ramps.

2.3 Route Planning

The field of route planning will naturally be central in my thesis. Here I will try to describe the problem of route planning and look at some of the more common solutions including a look at the best known algorithms.

Route planning is the task of finding a path from point A to point B, preferably at the lowest possible cost. Here, cost refers to a measure of the resources one has to expend to traverse the path. The measure itself is arbitrary, but should have some relevance to both the path and agent traversing it. For example, for a pedestrian, distance is a good measure of cost, while for a car, the expected time spent traveling might be better.

The subject has its roots in the so-called Travelling Salesman Problem (TSP), although the mathematical background can be traced to a game designed by William Rowan Hamilton in 1857 [2, p569]. In this problem a salesman wants to travel through a particular set of cities, visit every city exactly once, and return to his city of origin. The salesman wants to do this in as short a time as possible. It has been shown that finding an optimal solution for this problem is NP-hard[1]. To solve the problem, one must find every Hamiltonian cycle in the graph, and test them for optimality. Given n cities, this makes for up to $n!$ cycles, which means that we have a lot of comparisons ahead of us.

Fortunately, route planning usually deals with less complicated problems, mostly variations of the Shortest Path Problem. The simplest form of route planning, which happens to be the kind of route planning I will be looking at, is single-source single-destination (One-to-One) shortest path search. In this kind of problem, we want to find a path from one specified point one the map to another specified point. Thus, we have no requirements for a cycle, nor for the shape of the path, and the problem becomes much simpler.

2.3.1 Classifying Route Planning Problems

There are three main classes of route planning problems, and they all have different applications and solutions. The classes are One-to-One, One-to-All, and All-to-All.

The One-to-One class of problems is the simplest case. Here, we have a specific target and a specific source. The problem is then simply a matter of finding a single path between the two points. In addition, one may want some way of showing that the path found is an optimal path between the two points.

[1] By NP-hard, it is meant that the problem is *at least* as hard as any NP problem.

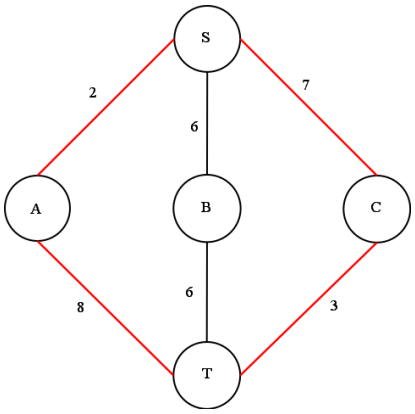


Figure 2.9: A graph with two optimal paths between S and T, each marked in red.

Note that I say *an* optimal path rather than *the* optimal path. This is because there may be several distinct paths from the source to the target that each are optimal. In the graph show in figure 2.9, for example, we have two distinct paths from S to T that are both optimal. Only the path through the point B is sub-optimal.

One-to-All problems are problems where we wish to find the shortest path from a specific point to all other points. This is intuitively a more involved task, but it is solved quite elegantly using the well known Dijkstra's Algorithm that I will look at in some detail later. A subset of this problem is One-to-Some, where we want to find the paths to all points that fulfill some criteria. For example, it could be interesting to find the shortest path to all accessible bus-ramps so that we could find the closest accessible bus-ramp. More commonly, this type of route planning occurs when we have a fleet of vehicles at a central garage and a number of destinations for our vehicles.

The third class of graph search problems are known as All-to-All or All-Pairs. In this class, we wish to find the shortest path between all the points on the map. It can be solved using the space and time efficient Floyd-Warshall algorithm [2, pp584-587]. This kind of problem is most commonly tackled in network routing and similar.

2.3.2 Graph Representation

When dealing with route planning problems, it is common to represent the map as a graph. The graph is a collection of nodes and edges, where each edge represents a road and each node represents a place where roads meet. This way, we create a road network of interconnected edges that mirror the real world as closely as possible without having to store unnecessary information.

Because of the vast amount of information usually present on a regular map, representation of the road network quickly becomes an issue. Ideally, one would simply hold in memory a number

of nodes, their connections, and the cost of using these connections. However, in real life, such an approach will lead to catastrophy on account of the size of real road networks. Instead, one must find a strategy for storage and retrieval that is efficient in terms of time and space. Commonly, one uses the graph metaphor as it lends itself intuitively to any network of roads and intersections, but the problem of storage and retrieval remains the same.

The storage of the graph should be so that it supports a number of queries [5] :

- Shortest way queries and context-specific queries (considering the amount of points of interest, the degree of slopes, user profiles, etc.);
- navigation in graphs (predecessors, successor nodes and edges);
- search in hierarchical graphs (distinction between levels of detail);
- spatial selection of sub-graphs (contains and intersects region query).

If met, the demands outlined above give a database that can very efficiently support route planning.

The first requirement includes context-specific queries and is of special interest to my project. To meet this requirement we need to associate a potentially large variation of meta-data to each road segment.

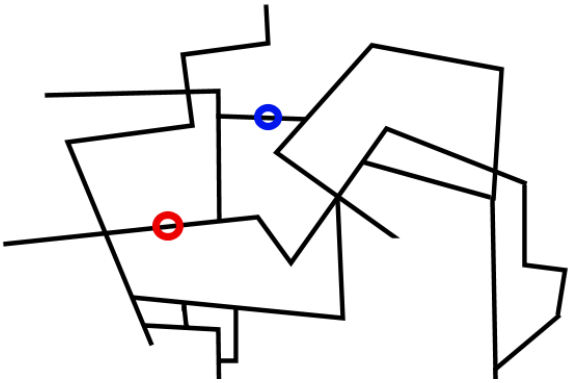
In order to navigate the graph, we need properly defined successors and predecessors. If the database maintains a relation that defines geographic points as nodes, this is easily done by allowing queries to made about what edges run out of any give node, and having every edge know about what it's start and end nodes are.

The two last requirements are more challenging to meet.

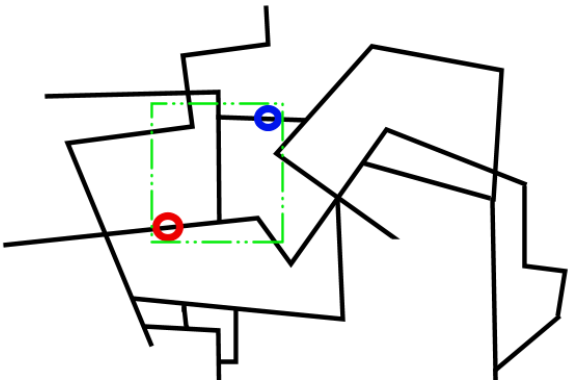
One way of efficiently selecting sub-graphs by geography is to use Minimum Bounding Boxes. This technique does demand some forethough, however, as it may break the search algorithm.

Minimum Bounding Boxes

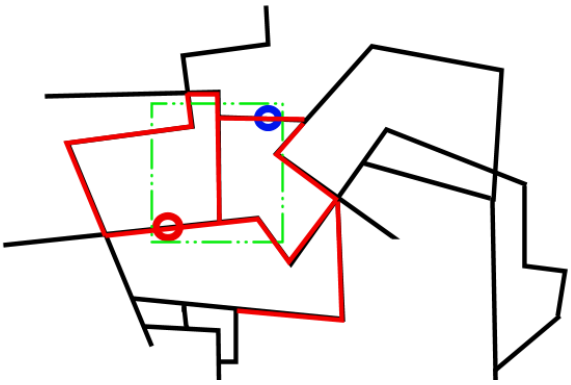
A strategy to reduce the search space is to use Minimum Bounding Boxes (MBB). An MBB is a box that exactly encapsulates a geometric shape, like a curve. If each curve in the dataset is associated with an MBB, one can limit one's search network to only those curves whose MBBs intersects or are contained within the box created by the source and target points of the search. Figure 2.10 shows a situation where most of the network is never loaded into memory, but where an optimal path can still be found.



(a) The entire map with source and target position marked



(b) The bounding box for the source and target is shown in green



(c) The lines marked red are the only lines loaded into memory

Figure 2.10: Extracting road segments based on a Minimum Bounding Box

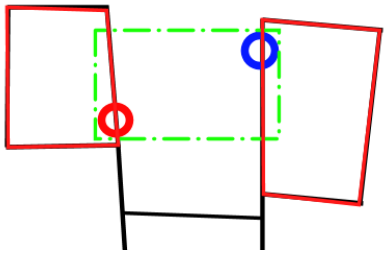


Figure 2.11: The pruning box approach removed the only viable path

Figure 2.10(a) shows the entire map. The red circle is the current source from where the path finding is to take place, and the blue circle represents the target. The black lines are roads, and they are represented as curves in the database. Each curve represents a stretch of unbroken and unintersected road, i.e. a single road with many byroads would be represented as many curves.

In figure 2.10(b) I have drawn a green square around the source and target. This square is then used to select out the edges that is to be used for route planning. Every curve in the database has an associated MBB, and if this MBB intersects, contains, or is contained by the green square, then the curve is included in the search. Figure 2.10(c) shows the curves that have been selected from the database.

The use of MBBs can radically reduce the memory needed for route planning in large networks. However, by implementing such a pruning strategy, the algorithm is no longer guaranteed to work. By selecting out parts of the network we are no longer able to say that if a route exists it will be found. There may exist routes from the source to the target that uses roads that do not intersect with the bounding box at all. In fact, in some rare cases, the optimal route or even the only route may need to use roads outsided of the pruning box (see figure 2.11).

2.3.3 Classic Algorithms for Route Planning and Graph Searches

At the core of route planning and graph search lies some small and elegant algorithms. I will present some of the best known algorithms known from graph searching, and begin with Dijkstra's Single Source Shortest Path (SSSP).

```
01. function Dijkstra(Graph, Source, Target)
02.    for each Vertex V in Vertices(Graph)
03.       Weight(V) = infinity
04.       Parent(V) = null
05.    Weight(Source) = 0
06.    ResultingSet = empty set
07.    Workingset = Vertices(Graph)
08.    while Workingset is not empty
09.       Vertex U = remove lightest vertex from Workingset
10.       if U equals Target
11.          terminate search
12.       ResultingSet = ResultingSet union U
13.       for each Edge E(U, End) in Edges(U)
14.          if Weight(End) > Weight(U) + cost of traversing E
15.             Weight(End) = Weight(U) + cost of traversing E
16.             Parent(End) = U
```

Figure 2.12: Dijksta's Single Source Shortest Path algorithm

Dijkstra's Algorithm

The dutch computer scientist Edsger Dijkstra was the first to formalize the shortest path problem and proposed a general solution in 1959. Dijkstra's algorithm relies on relaxation of individual nodes, and continues to relax nodes until a minimal spanning tree including the source and the target node, but not necessecarily all of the nodes in the original graph, is found.

When we are looking for the path from a specified source to a specified target, the algorithm looks like figure 2.12. However, if we ignore lines 10 and 11, the algorithm will allow us to find the shortest path to any node from the source node, making this a one-to-all algorithm.

The first seven lines, including the function signature, sets up the system for processing. Lines 2 and 3 sets the cost of every node to infinity and every node's parent in the resulting path to null. Further, the cost of the source node is set to zero, as it is obviously free to get to where we already are. The working set we define on line 5 starts as the complete set of nodes in the graph and is gradually shrinked until it is empty or the target node is found.

On line 9 we select the currently cheapest node to travel to in the working set. On the first iteration this will be the source node, as it is free and every other node cost infinitely much. Provided we have not yet reached our goal, we then relax each edge of the selected node. We perform this relaxation by testing to see if the cost of traveling to the node on the other side of the edge in

question is cheaper by travelling through the currently selected node (line 14), or if a shorter path has already been discovered. If it is cheaper to use the current node, it is set as the parent node of the node on the other side of the edge (line 16) and the cost of getting there is updated (line 15).

The weight of each node is the summed cost of the current best path to the node from the source node. It will be useful for later discussion to define a function $f(n)$ that gives this weight for any given node n.

When we have reached the target node, we have a set of nodes $ResultingSet$ that contains at least every node in the path from the source to the target. By beginning at the target node, the path can be recreated by following $Parent$ references all the way to the source node. This path is guaranteed to be a shortest path from the source to the target.

An important limitation of Dijkstra is that it is unable to discover an optimal route through a graph containing negative edges.

A brief analysis of Dijkstra In the worst case, the Dijkstra algorithm must scan every edge of every vertex. It will scan each vertex at most once and every edge at most twice.

Assume a graph with v vertices and e edges. In the most naïve implementation of the algorithm, we store the vertises in an unordered fashion, like an array or a linked list. Then line 9 in the algorithm becomes a linear search that needs to search the entire remaining graph each iteration. We get a run time complexity of $O(v)$, i.e. one $O(v)$ search for each of the v vertices. Clearly, a less naïve datastructure would improve the efficiency of the algorithm. For example, a min-heap would reduce the complexity of the search for smallest node to $O(\log v)$, for a $O(v \log v)$ complexity.

Furthermore, the complexity of the cost function can play a role in the efficiency analysis of the algorithm. Assuming a cost function of complexity $O(n)$ for some n, we get a total complexity of $O(n * v \log v)$. In the simpler situations, the cost function will in fact be a constant function, and will therefore not have a serious effect on the algorithm. It is conceivable, however, that the cost function could be more complex, and in such cases the run-time efficiency of the algorithm would suffer.

Another concern is the order in which alternative paths are explored. Dijkstra's algorithm will always explore the cheapest path from the source node, making it a bredth-first algorithm. This path, however, may not actually lead to the target node, so until we have discovered the target, it will explore in every direction. One can imagine this as an ever increasing circle expanding from the source node until the target node is found. A better approach might include some estimation of whether the explored path actually leads towards the goal. The well known family of Best First

```
01. function BestFirstSearch(Source, Target)
02.    OpenSet = Source
03.    while OpenSet is not empty
04.       WorkingNode = BestNode(OpenSet)
05.       if WorkingNode is Target
06.          return PathToWorkingNode
07.       OpenSet = OpenSet union WorkingNode.Successors
```

Figure 2.13: The basic Best First Search Algorithm

Search (BFS) algorithms does just that.

Best First Searches, or Heuristic Searches

Best First Search algorithms are a family of algorithms that use some way of estimating what direction the search should take [39, pp94-129]. They are a reaction to the uninformed algorithms that grew from Dijkstra's algorithm. The basic idea is to avoid searching down paths that lead away from the target node.

Figure 2.13 shows the basic Best First Search algorithm, with a lot of detail left out. An implementation needs to make some allowance for generating the path hinted at in line 6, unless the algorithm is just deciding *if* there exists a path from the source to the target. The other detail of some importance is left out of line 4. In fact, this line introduces the heuristic function that we will call $h(n)$, where n is a node in the graph.

A heuristic function is simply a function that guesses whether or not a node is likely to be on the path to the target node or not (see the next section). The function usually assigns some numeric value to a node, so that it is possible to compare two nodes to see which node should be explored first.

In a route planning context, it could be considered expedient to let the heuristic be the straight line distance to the source. However, while this makes sense in many situations, the heuristic fails in maze-like environments. Consider a situation like figure 2.14. The red dotted line represents a path that would have to be fully explored by any basic BFS algorithm, since every point on the red path is closer to the goal than the first node (and second, and third, etc.) on the blue path. Sadly, the blue path represents the only path that will take you to the goal.

Clearly, basic Best First Search algorithms have weaknesses that needs to be adressed, and this is where A* (pronounced "A star") comes into play.

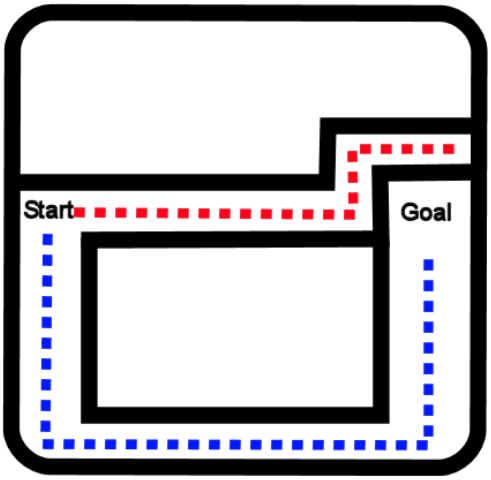


Figure 2.14: A Best First Search would follow the red path to the end before exploring the only viable path.

The Heuristic Function The heuristic function is quite simply a function that looks at the current node and guesses how far it is to the target node.

The quality of the guess impacts the search time of the algorithm. The more accurate the guess, the quicker the search. Obviously, if the heuristic guess correct each time, i.e. the heuristic function is actually an $oracle$ function, the algorithm will choose the correct path every time, and the shortest path will be found immediatly. The worse the heuristic guesses, on the other hand, the less likely it is that the algorithm picks the correct path to explore, and thus the query takes more time.

A typical use of a heuristic search is when solving the Slide Puzzle. In this puzzle, the player is presented with a tiled image where the tiles are placed incorrectly, and one tile is missing. It is the player's task to slide the tiles around the board until the image is restored. In figure 2.15 I present a slide puzzle with numbers for the sake of clarity.

A common heuristic in the situation represented by the slide puzzle, is to use the sum of the Manhattan Distances[2] needed for each piece to slide into place. That is, a piece that must be moved three times in order to be at its correct position would have a Manhattan Distance of three.

When the search algorithm has reached a particular situation (our opening position figure 2.15(a)), it evaluates the heuristic function for the four possible next moves (2.15(b), 2.15(c), 2.15(d), and

[2] A.k.a. Taxicab Geometry, Minkowski Distance etc.

2.15(e)) before it chooses what direction to explore. In this case, figure 2.15(e) is thought to be closest to the solution and therefore it is explored first. This quickly leads us to the target configuration seen in figure 2.15(f).

In the slide puzzle, we could have used any number of heuristic functions, but the Manhattan Distance works well enough and it fulfills a number of the requirements of a good heuristic. The most important of these requirements is $admissability$ [46]. A heuristic is considered admissable if it is guaranteed to always give a value at least as good as the true value for any given node. In other words, if we have an oracle function $o(n)$ that gives the actual distance from the node n to the target node, then the heuristic $h(n)$ would be admissable if and only if $h(n) \leq o(n)$ for all nodes n.

An admissable heuristic can also be either informed or uninformed. The heuristic is informed if it equals the oracle function on all input. Clearly, a fully informed heuristic is, in any real world situation, unattainable. On the other hand, a heuristic may be uninformed. In this case, the function returns zero for all nodes. An uninformed heuristic is by itself useless, but by introducing this term, one can for example discuss Dijkstra's Algorithm as a heuristic algorithm with an uninformed heuristic.

In real world route planning situations, it is common to take the Euclidian distance to the target as a heuristic. This is clearly an admissable heuristic, as there can never be a route shorter than a straight line, as long as we stay within classical physics. It may not be a particularly good heuristic, however, in the case of urban environments, where many streets are one-way, and one often has to drive north to get south.

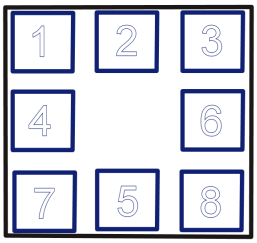
A*

The A* algorithm was designed in 1968 by Hart and others [10] to improve upon the now forgotten algorithm A. Like the Best First Algorithms, it employs a heuristic function to guess whether a node should be explored or not. However, it also takes into account the path already traversed to reach each individual node.

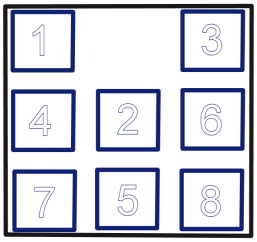
Given a source and a target node, the A* algorithm looks like figure 2.3.3. It is guaranteed to find an optimal path through the graph.

The three first lines initializes the algortithm, giving us an empty set of *closed* nodes and a set of calculated paths consisting of only the source node. The implied function MakePath (line 3 and 13) creates a directed path from the first node in the list to the last, through every node in the list in succession.

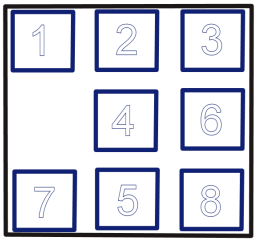
The iterative part of the algorithm is lines four to thirteen. On line five we remove the most



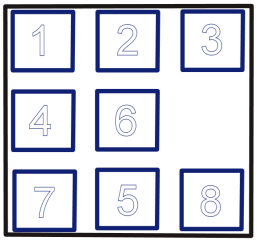
(a) Opening position, Manhattan Distance: 2



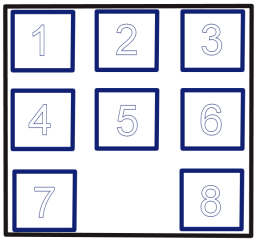
(b) Manhattan Distance: 3



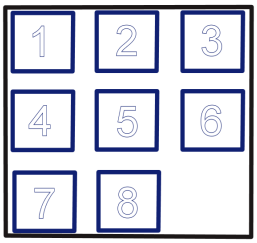
(c) Manhattan Distance: 3



(d) Manhattan Distance: 3



(e) Manhattan Distance: 1, the heuristic favors this move



(f) The puzzle is solved

Figure 2.15: Various options for the slidepuzzle

```
01. function A* (Source, Target)
02.    ClosedSet = Empty Set
03.    OpenSet = MakePath(Source)
04.    while OpenSet is not empty
05.       Path = OpenSet.RemoveBestPath
06.       Node = Path.LastNode
07.       if Node is not in ClosedSet
08.          if Node is Target
09.             terminate search and return Path
10.          ClosedSet = ClosedSet union Node
11.          for each SuccessorNode of Path
12.             if SuccessorNode is not in ClosedSet
13.                OpenSet = OpenSet union (MakePath(Path + SuccessorNode))
```

Figure 2.16: The A* algorithm

promising path from the open set. We find the most promising path by applying the heuristic function to the path, i.e. the implied function RemoveBestPath implements our chosen heuristic. Obviously, it is expedient to implement the open set as some sort of priority queue like a min-heap or a fibonacci-heap.

Line six to nine tests if the path leads to the target node, in which case we terminate the search. Otherwise, we close the last node in the current path, and create a new set of paths by extending the current path with all the successors of the last node in the current path (line eleven to thirteen).

Efficiency of A* The A* algorithm promises to be a very time-efficient algorithm, although this depends to some degree on the heuristic function. In the extreme case where $h(n) = 0$ for any n, we only consider the current length of the path from the source. In fact, in this case we have reverted to Dijkstra's Algorithm, and so the algorithm would perform likewise. In fact, A* will in the worst case perform just as bad as Dijkstra's, and so the algorithm has a complexity of $O(nlogn)$. However, given that A* performs as bad as Dijkstra only in the worst cases, it follows that the algorithm has a fairly decent average run time.

On the other hand, A* is not a very space-efficient algorithm. Since it insists on keeping in memory all the generated paths until an optimal path is found, it usually runs out of space long before it runs out of time. Some efforts have been made to remedy this situation, and Iterative Deepening A* was one of the earlier attempts.

Iterative Deepening A* The A* algorithm is one of the best known search algorithms around, and so it has spawned a number of improvements and variations. One such variation is known as Iterative Deepening A* (IDA)[35]. This algorithm is a combination of A* and Iterative Deepening Depth First Search (IDDFS) search.

During the execution of IDA, the system keeps track of the current iteration and an associated cutoff value that is increased with each iteration. If a path has a cost that equals or exceeds the cutoff value, it is ignored until the next iteration. When the the algorithm iterates, the new cutoff value is equal to the cost of the cheapest path that exceeds the old cutoff value.

Although IDDFS can be shown to be space efficient, the hybrid solution IDA does not do well in real life. While IDA executions where cost is measured in units will do the job well enough while keeping memory usage to a minimum, it gets into trouble when applied to other situations. If IDA is faced with a problem where cost is measured in real numbers, the iterations will start to grow the admissable solution space too slowly, which means that the algorithm will search through the same solutions extremely often before finally finding an optimal solution.

2.3.4 The Cost Function

Every route planning algorithm uses some concept of cost, and this is usually encapsulated by the ever-present cost function. This function determines the cost of traversing a node or an edge in the graph.

Commonly, cost functions are relatively simple approximations. For a standard route planning application where the graph consists of nodes along the real, physical road network, the cost function may be the geographic distance between to adjacent nodes. Given a network that faithfully represents the actual roads, every node would be connected by a straight stretch of road, so the function would return a value that has real, physical meaning.

Other common functions include estimated time to travel from one node to another. This time may be a function of distance and known speed-limit, or it could be an average of reported time spent traversing the edge in question.

Multivariable Cost Functions

The most interesting cost function are those that take into account several discrete parameters, such as the function described by Rippel et. al. [37]. Basically a multivariable cost function looks at several different aspects of each path and weighs these parameters. For example, a cost function

for pedestrians could at both the length and incline of the road in order to find a pedestrian-friendly road.

The challenge of designing multivariable cost functions lies in finding a balance between the variables and weighing them agains each other. In the case of using both length and incline of a road, one cannot use the absolute values themselves. Because one parameters describes a physical length and the other describes a ratio between two physical lengths, one must find another way of combining the parameters to express one non-negative cost. In our example, one could multiply the parameters like this: $length * (1 + vertical\ meters\ per\ horizontal\ meter) = cost$, while other parameters may need other formulas.

Context Awareness in Cost Functions

When we introduce context awareness into our cost functions, things start to get interesting. With context awareness we mean that the cost function takes into account the user's situation and motivation. For example, if the cost function knows that the user is a wheelchair user, it will increase the cost of paths with steep inclines and disregard routes that are inaccessible to wheelchair users.

Context awareness demands that a lot of data is available on both the user and his immediate situation.

2.4 Related work

I am far from the first to have looked at pedestrian navigation, and I will introduce a small sampling of what has already been done on this topic. I will also introduce some concepts that, while they are not directly related to pedestrian navigation as such, they lend themselves to my project. More specifically, I will look at the concept of collaboration and knowledge sharing.

2.4.1 Pedestrian Navigation

Pedestrian navigation is a subset of geographic information systems that has grown out of the ever increasing power and connectiveness of mobile devices like mobile phones and PDAs. It is concerned with the same basic problem of the more common navigational systems used in cars, but it's faced with a number of issues that vehicular navigation can ignore.

Perhaps the most immedate such issue is related to the role the navigational device plays for the user. Pedestrian navigation systems are usually developed for small, mobile devices like cell

phones or smart phones. These devices suffer from rarely having the full focus of the user, and are instead relegated to a secondary position in the user's awareness. When such a device is used for navigational purposes, this problem is exacerbated by the user's often stressfull situation. For example, a tourist may be trying to find his way to his hotel while dragging heavy luggage behind him and taking in the sights and sounds of a new city. In this situation, he has only very little time and patience to stare at the small screen of his mobile phone or wait for the device to gather the necessary data for his query. Several attempts have been made to adress this: Sester provides us with an overview of several methods for generalizing and presenting spatial information on limited devices [43]. Her work provides an excellent starting point for providing intuitive and user-oriented geographic displays. Baus et al. created a hybrid system that takes the user's position, movement, and even plans and motivation into account to present only the most pertinent information to the user in a way that suits the situation [4].

East-Asia, and especially Japan has seen an explosion in pedestrian navigation systems, in part due to the relatively low market penetration of cars, and the comparatively high reliance on public transportation. Many pioneering companies have made inroads into the field of pedestrian navigation and on of the earliest such system was DoCo-Navi. This product was released by NTT DoCoMo as early as 2000 [47] and was a subscription service that would guide users towards services of interest, like banks, hospitals, etc.. Using a patented enhanced GPS system from Snap Track, they provided a service that could make use of geographic information, even in urban canyons. At launch time, the system provided 210.000 points of interest and was released on a special "Naviewn" device shown in figure 2.17. Other service providers, like KDDI followed up with similar services in the following years [25].

The DEEP MAP project is an ambitious project to create a framework for intelligent spatial information systems [30]. It is an agent-based system that, among other things, includes a route agent. This agent is responsible for generating routes that the user may be interested in following. These routes are customized so that each individual user are given routes to their preference. The preferences are captured by the system as both hard and soft parameters. The hard parameters describe such things as distance, height, and other physical attributes of the environment, while the soft parameters attempt to +capture more subjective features like nice viewpoints, traffic noise, etc..

Another team looked at classic machine learning techniques to aid navigation and route planning for disabled pedestrians [23]. This project used a new method for learning fuzzy models that would allow the system to learn about what sort of routes would be appropriate for different users. Each user would describe himself in terms of things like disability, age, gender, etc. The users would then

Figure 2.17: The DoCoMo Naviewn device connected to a mobile phone.

fill out a form about what sort of things they would consider obstacles to their travel, and the system would extrapolate navigation rules from this.

The most prolific area of pedestrian routeplanning, however, has been tourist guides. Hujinen provides us with a comprehensive overview of european projects surrounding so-called mobile tourism [21], although she points out that the social aspect of tourism is missing in most of these projects.

Some projects, like WalkOnWeb[1], allows hikers to share routes or tracks with eachother, although editing such tracks is limited to the original author. Another project called MAPPED aims to provide complete route planning for disabled users. This includes planning routes using public transportation, wheelchairs, etc., all the while factoring in accessibility factors.

Many other projects, like Crumpet[40] and Eureauweb[27], have already conluded and they have resulted in mobile applications that let tourists plan routes, browse maps, and read information about typical tourist attractions, resturants, etc.. Crumpet delivers typical tourist information in the form of recommendations based on location and personal interests, while Eureauweb specializes in planning trips based on Europe's waterways.

2.4.2 Collaborative Data

Humans are social animals, and throughout our history one can find evidence that we like to share our knowledge with eachother. From the earliest cave-paintings to text-messaging and e-mail, people have had the need to express themselves and share their thoughts and ideas. As technology has

progressed, so has the oppurtunities for communication: the bookprint made it possible to spread ideas and knowledge to the masses, the telegraph, the telephone, and the radio made it possible to communicate over large distances, and the internet made mass-communication trivial.

The latest addition to this ever-increasing number of available media was the World Wide Web. The web offered instant, world wide publication and anyone could in principle become a web-author. However, due to the cost of running the servers needed to maintain web-pages and the relativly high technical threshold for setting up web-pages, the web remained a media for the larger organizations and the specially interested. Some web-services, like the bookstore Amazon.com offers users the oppurtunity to rate their products, and these ratings can then be used to build networks of similar preferences across users. That is to say, by critiquing a book for example, you also tell Amazon something about yourself, and this helps them tailoring advertisement on a user by user basis. However, the user himself is still a relatively passive agent, merely offering up information about himself to the system. This has all changed with the advent of the blog and the wiki.

The blog is a variation of the traditional homepage. However, rather than being hand-coded it has a robust collection of scripts that allows the user to author content without having any technical insight themselves. This lowering of the threshold has caused a massive increase in the use of the web as a place for self-expression. Rather than simply being consumer, people are becoming active participants and are shaping the web to their own preference.

While the blog is a personal creation with one author, another technology, the Wiki, is opening the web for collective authoring.

Wiki

A Wiki is a web-based Content Management System (CMS) where everyone can contribute. The concept was first explored by Ward Cunningham in 1995 [13] and made its way into the public conciousness via the Wikipedia project [50]. The basic idea of the wiki is to create an environment where multiple users can author and edit the same document through the well-known web-browser interface. Some wikis are small, closed systems where only selected users may edit pages, while other wikis, like Wikipedia, offers editing oppurtunities to anyone.

A common use of a wiki is for maintaining technical documentation for a development project. As issues become appearant, they can be documented online and this documentation becomes available to every reader of the wiki immediatly. Since every wiki maintains a history of every change made to every document, it becomes easy to track issues, their discovery, probable causes, and their eventual resolution.

The most dramatic use, however, remains the Wikipedia project. This project is

> *"an effort to create and distribute a multilingual free encyclopedia of the highest quality to every single person on the planet in their own language"*

according to its co-founder Jimmy Wales [50]. It achieves this by allowing any reader to edit almost any entry. The idea is that while no individual knows everything about a subject, when many individuals come together to describe something, the description will over time become accurate. Kolbitsch and Maurer compare this phenomenon to an ant hill [28]; Although no single ant knows how to build and maintain the ant hill, they each know just enough to do their job. Over time, the ant hill emerges as a result of the concerted labour of the many. In the same way, the entries in Wikipedia grows steadily in accuracy as more and more people read and contribute with their own insight and knowledge. Acts of vandalism, lies, or other negative contribution is quickly remedied either by subsequent corrections or by reverting the article to an earlier version.

Although wikis have brought document collaboration to a new level, they are not without their problems. Wei et al. points out that a basic wiki is usually both intimidating to novice users and often lacking in usability [49]. This point is in part refuted by an experiment performed in Canada, where usability researchers had children try to write a story in collaboration [7]. The experiment showed that non-technical users could easily use a wiki-style tool to collaborate on a story. However, the participants had problems dealing with the hyperlink aspect of the media, so the technical threshold has not quite yet been removed.

Other problems associated with wikis stem from the number of users, and the diversity that naturally occurs. Especially when dealing with controversal issues, one can experience so-called editor-wars. These wars occur when two or more groups of editors have conflicting views on a subject. Often editing is reduced to reverting an article to a point before the opposing viewpoint is entered, and as both or all parties engage in such behaviour, the history page grows with needless revisions and becomes unmanagable.

Other popular kinds of online sharing

As the idea of sharing one's knowledge and insight caught on, a new trend emerged. With new technology is has become easy for non-technical users to record sound, video, and images digitally and publish these online. Service such as Flickr and Photobucket allows users to upload photos and share them with anyone browsing the websites. Uploaders are encouraged to create tags, essentially

just descriptive words, to classify their images, and these tags become the basis for performing searches.

Some of these services operate under licences that allow anyone to download any image from their database and use these in their own work. This way, these services become a commons for intellectual property, where budding artists can find material free of charge.

Podcasting is another way people are expressing themselves on the internett. Podcasting is a kind of radio, where people create soundfiles in which they talk about subjects that interest them. These files are made available on the web and distributed like any other media file. Essentially, podcasting is to the spoken word like blogging is to the written word.

Sharing specialized knowledge: the Open Street Map initiative

One instance of collaboration is of special interest for my thesis, and this is the Open Street Map (OSM) initiative [31]. This project is organized like a Wiki for geographic data. Aimed at offering the prohibitively expensive street data in England, the Open Street Map initiative began as an effort to make a free and open street map of London. Since then, the project has grown and now covers many cities all over Europe.

The project relies on the public to generate its maps. In order to contribute, a user tracks his own movements useing a GPS. The log from the GPS showing where the user moved is then uploaded to the OSM servers where it is made available to the general public. These GPS tracks are simply an ordered series of geographic coordinates that define lines or curves. When many users upload logs that describe lines or curves that are very close to eachother, it becomes likely that the data describes an actual road.

The project has expanded to allow users to add metadata to their geographic uploads. One can for example add street names to tracks.

Data from the OSM project is published as XML files that are easily interpreted by client applications. The XML defines points as geographic coordinates without height. These points are subsequently referred to in line definitions. A line in the OSM data set is a stretch of unbroken road, i.e. a line is terminated by either a dead end or an intersection at either end, but does not have any intersections other than at its ends. Lines can also be combined to form so-called ”ways”, which represent named roads.

The OSM project is just one example of the idea of sharing knowledge for the common good spreading and becoming a modus operandi for people and organizations around the world. However, as more and more people begin sharing, an issue of trust emerges.

Trust

In recent years, a lot of research has been done on the topic of trust in electronic commerce. The work has been motivated by the inherent trust a user must show a system before he commits to use it for financial or otherwise sensitive transactions. With the steady growth of online identity theft [20], the topic remains vital to online commercial activity.

Kini and Choobineh suggested an Integrated Model Of Trust that takes into account many of the divergent trends in previous trust research [26]. This model focuses squarly on the human-computer side of the trust issue, however, and fails to adress the inter-process trust necessary to perform complex transactions. This inter-process trust describes to what degree an application can trust other applications to perform their services and not do anything to compromise either the users or the system as a whole.

In the topic of collaboration and data sharing, the problem of trust is somewhat different than in online commerce situations. We are for the most part, less concerned with the validity of identities or transactions. Instead, we become interested hard-to-quantify attributes like what level of agreement exits between ourselves and others regarding music-tastes or political inclinations. For example, in a hypothetical book-review application, we could be interested in getting suggestions for new readings based not only our own past reading, but also the past reading of other users with similar reading habits.

One way of dealing with such inter-personal trust is for the system to simply assume that everyone is similar, and differentiate based on actions. The hypothetical mentioned above is one such example where your previous actions alone determine what other users you are expected to trust. While such a model of trust is relatively easy to implement, it lacks in depth and customizability.

Another approach could be to let people decide who to trust and propagate this through the system. For example, user A may trust the judgement of user B, but not of user C. When user B makes a recommendation, this is passed on to user A, but user C's recommendations are not. Furthermore, if user B values the opinions of some fourth user D, this trust is reflected back to user A which gains some level of transitive trust for user D. Such trust networks have received some criticism [15], especially in situations where trust may be forwarded unintentionally. However, I believe that the critique is unwarranted provided the trust does not allow third parties to perform actions on our behalf. That is, provided the trust network *only* details opinions and raw data, it is safe to assume that the friends of our friends are also friendly.

2.4.3 Collaboration in Route Planning

In situations where you have many agents[3] searching the same space for routes, it can be useful to let the agents share the routes they find.

In real time strategy games, one often wants many agents to move from one place to the other. In such a case, it can be useful to let one calculate a path, and then share this path with the other units on the gameboard [36]. The other units may then be able to move in a constructive direction while recalculating a better path for themselves. Real world path finding may not be quite as clear cut, but we still find it beneficial to exchange information.

McGinty and Smyth [17][32] identify two types of route planning scenarios that are relevant when sharing experiences. The first type refers to solving an unfamiliar problem in a familiar territory, while the other type involves solving an unfamiliar problem in unfamiliar territory. In the case of familiar problems, we already have a solution that we can reuse. If do not have any familiar problems, we must instead explore the unknown territory.

However, other work shows that purely Case Based Reasoning (CBR) will reach a saturation point where additional cases will not improve the routes to any reasonable degree. In fact, too many cases will slow down the search and thus be detrimental to the planning process [44]. This problem is known as the Utility Problem [9] and is a well known problem in CBR. What happens is simply that the number of cases grows to such a size that simple case-retrieval becomes such an involved task that the overhead for memory-access outweigh the time saved by not exploring bad solutions. That is to say, the baby is thrown out with the bathwater.

Haigh et al. [16] describes a system where users may give a simple critique of the provided route. The critique consisted in that case of a single positive number. This number would be integrated into the route's current rating β, and this combination of users's experiences would be used when determining whether a particular case would be used in a solution. This way β becomes not only a measure of the quality of a particular case, but it also becomes a factor determining whether the planner should use old cases or explore unknown territory.

The Haigh approach to learning user preferences and critiquing paths solves two problems in one go. Firstly, the Utility Problem is dealt with by simply reducing every experienced case to a number and then incorporating this into an already existing feature. This way the volume of experience does not affect the search time in any way; the algorithm will only look at one single value regardless of the number of cases that have been entered into the system.

[3] I define an agent to be an entity with goals, this includes both human and software agents.

Secondly, the single value evaluation captures the user's preferences implicitly. As noted by Rogers and Langley [38], it is often futile to attempt to accurately model the user's preferences in a route planning environment. The user may not know his or her true preferences, it is costly and impractical to perform in depth interviews with all users, and many features that would weigh in on a route may be of value only to some or even one individual user. Rogers and Langley present a neural network that learns the preferences of the individual user after a long series of user feedbacks. The solution provided by Haigh, however, assumes that most users will have many similar preferences and collects every user's feedback in one universial quality measure. This way the system quickly learns what roads to avoid and what roads to emphasize during a search.

The main drawback of the Haigh solution lies in its universial application. In using only one value for each road, the individual's preferences are ignored. Given that different users may have different preferences, this could lead to a situation where generally good routes are ignored. In the case of regular route planning this may not be very problematic, but in the case of personalized route planning, where the users have very different preferences, it becomes a major obstacle. For example, if there's a tall step from the sidewalk and out to the road at a zebra crossing, a wheelchair user would find this detrimental. On the other hand, a blind person can use such a step to identify the change from sidewalk to road. A blind person is therefore advantaged by the feature that disadvantages the wheelchair user.

Returning to McGinty and Smyth we find that one way of adressing the problem of overly general quality quantifiers could lie in an agent based approach. Instead of assigning each stretch of road one universial value, the individual users, or agents assigned to users, maintain a value for how they percieve the quality of the road. When an agent wants to find a route someplace, it asks agents that it knows have similar attitudes for advice on what roads are good or bad. This would imply some basic implementation of a trust network [12], but given an efficient such network, the drawbacks of CBR could be avoided.

2.5 My contribution

For my thesis work, I wish to create a proof-of-concept application capable of generating and displaying personalized pedestrian routes on a mobile device. I propose to create *the Ranger*, which will be an expansion to the Okapi application I described earlier. The Ranger will consist of a server side application that is responsible for maintaining all the necessary data and for performing any calculation that is needed. It will also have a client side component that will be integrated into

the Okapi mobile application. This component will gather input from the user and present results generated by the server.

I will attempt to personalize routes by inferring user preferences through user feedback, rather than by forcing the user to conform to static parameters. User feedback will be disseminated over usergroups determined by self-identification, i.e. everyone that self-identifies as a jogger will share feedback with everyone else that identifies as a jogger.

I will use open and freely available road data for the majority of the route planning, although during the inital stages of development I shall use the closed VBASE dataset as it is known to be good and this helps me focus on my own implementation. However, I believe that the open, free road data available is good enough for route planning of the kind I propose, and when the application is implemented, the proprietary data should be replaced by open data.

The Ranger will allow users to modify the network of roads to reflect their experience of the real world, and I believe that over time, this will lead to a more accurate network of traversible paths. In turn, this will lead to a situation where I can generate good routes for a wide variety of users with different and even conflicting needs.

The application will use GPS positioning to position the user on the map and for tracking user movements. This should allow the Ranger to infer roads in places where it lack road data. It will also draw upon the considerable work performed by various institutions and individuals to make good, detailed map-images freely available through open standards like WMS.

Chapter 3

Design of the Ranger

This chapter contains an examination of the design of the Ranger. I will approach the design by first looking at some scenarios for the typical use of the system. This will give the design some focus, as I will know how I expect the user to interact with it. The user himself will also be examined and I will explain how I intend to model him. The datamodel for the project will also be examined and I will offer a detailed look at both the database design and the final design of the Ranger client and server. The actual implementation details can be found in appendix A.

3.1 Scenarios

A good place to start designing any system is looking at various scenarios in which the future system is used. I describe three typical uses of the Ranger module of Okapi. These cover the basic usages I envision for the module.

3.1.1 Finding your way

In this scenario, a person is located somewhere in Oslo and needs to visit a toilet. He is a wheel-chair user, so he may not be able to use just any toilet, but is in need of an accessible toilet. The user consults his SmartPhone and asks for the nearest accessible toilet. The mobile then passes the request off to the sentral server which answers with a location and a path to that location from where the user is. The user is presented with a path from his current location to the nearest accessible toilet, and he is even given a short description of the location he is being guided towards. The path appears on his SmartPhone as a map with a curve drawn from the location of the user to the toilet.

3.1.2 Giving back to society

Here, an experienced user has discovered a short cut through a park that is very well suited to his particular preferences. He decides that it would be nice if more people knew about this shortcut, as the alternatives are long winded and fairly difficult to traverse. The user has access to a GPS, so he simply tells the Ranger to record his trip the next time he uses the path. Once he has reached his destination, he asks the system to upload the usertrack he has created, and it becomes available to all other users of the system.

As an alternative, a user has discovered that the Ranger never gives solutions crossing through a park he knows has excellent paved paths. To share his knowledge with other users, he proceeds to draw a line across the park on the map where he knows the paths are. These tracks are uploaded to the Ranger and integrated into the network of roads and paths.

3.1.3 Making yourself heard

A user has asked for a route somewhere, but finds that the Ranger gives a poor solution. The route takes the user down a road that lacks a sidewalk and it is therefore unacceptable. He selects the route given to him on the map and selects the option for feedback. When he gives the track a bad grade, the Ranger is able to take this into account whenever someone asks for a route in the same area.

Similarly, the user could have given the thumbs up for a route that was especially well suited for his needs. In such a case, the selected route would be preferred by the system when calculating routes in the area.

3.2 Assumptions and Scope

I will have to limit the scope of the project and make some assumptions about both the available data and the usage situation that one might not have made in a production quality project. Instead, I will focus on creating a proof-of-concept application that demonstrates the viability of my thesis. The project will still be functional as a route planning tool, and I believe it will be able to answer the problem statement about whether it is possible to create a route planning application that can give good routes for diverse usergroups.

The base road network I will use for the route planning comes from the VBASE data. This data is actually centerline road data for Oslo. That means that my road data does not include things like

sidewalks, so I will assume that every road is traversible by anyone. I believe that with use, tracks using roads without sidewalks, or where the sidewalk is inaccessible will be selected against as the users give such tracks bad reviews.

Furthermore, I have assumed a planar road network. That means that I disregard bridges, underpasses, and other similar features. The result is that the system may believe that an intersection exists in places where they don't, for exampel where a footpath crosses a road by brigde.

3.3 Modelling the User

In designing this project, I need to give the user considerable thought. The system is supposed to be able to give the user a route that fits the user's particular needs: Two users standing at the same spot, asking for a route to the same place, should not get the same route if they have different needs and preferences. To achieve this, I need a model of the user that is accurate enough for me to be able to plan the routes properly.

There are two basic approaches to modelling the user. These method differ both in the way data about the user is gathered, and how this data is being applied.

The most obvious way is to perform an in-depth analysis of the intended user groups. By performing several interviews and usertests as the system is developed, the user model is gradually refined until a satisfactory model is achieved.

This approach has many benefits, not the least that the user model will be well documented. However, it is an expensive process and the end result is a somewhat static model. If the model fails in some way, it may be difficult to address the problem.

The other approach is the implicit user model. In this case, the user is never actually modelled. Instead, the system starts out by making a number of assumptions as to what is good and not, and as the users utilize the system and offer feedback, the system adjusts its parameters. With use, such a system will eventually offer a very accurate image of the user's preferences[38].

The main drawback of the latter method is that it takes time for the system to properly adjust to the all the user groups. During this time the system will likely generate less than optimal solutions, and this may cause discontent amongst the users. However, given the difficulties of designing an accurate user model, I believe that the implicit model is the better alternative.

For the Ranger project, I have decided on an implicit usermodel. Instead of trying to describe the individual user or even the various groups of users, I define a number of labels. Users will be free to select any one of these labels to describe themselves. When a user critiques a track, his impressions

are shared with everyone else that has chosen the same label. In other words, if a user labels himself a powered wheelchair user, everyone else that shares that description will benefit from his attitudes towards the various tracks.

The labels I have defined for the system are

- Powered Wheelchair
- Wheelchair
- Crutches
- No special needs

These labels define a preliminary look at various categories of physical disability. The labels describe only the means of transport a person uses, as I believe this to be the most important for the project. These labels also define an important point to be explored during user-testing, i.e. will users find these labels adequate or do they have other thoughts surrounding grouping.

3.4 Collaboration in the Ranger

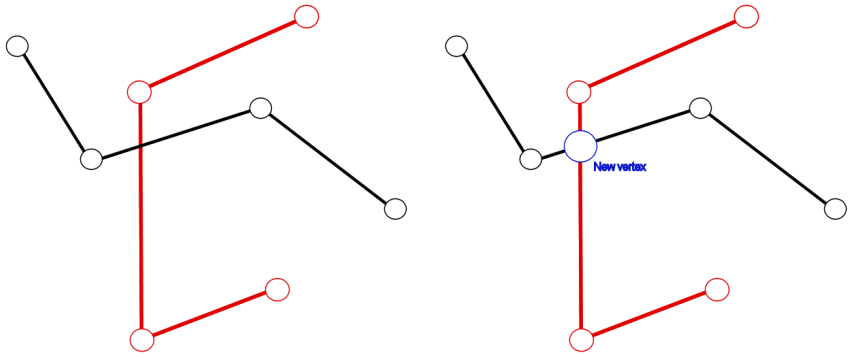
The project hinges on collaboration between the users. This collaboration takes form in two particular ways: The creation of tracks and the assesment of paths.

To share a track, the user must record his or her movements on the device and then upload this to the Ranger database. The recording can be automated, i.e. one can use a GPS to mark one's moves, or it can be manual. In the case of manual recording, the user draws a track by placing points on the screen. When a track is uploaded, it must be combined with the road network already in place. This can lead to some situation that needs special consideration.

3.4.1 Combining Tracks

When users record new tracks, it is highly likely that some of their recorded positions will coincide closely, but not completely, with points already existing in the database. Furthermore, it is not unlikely that tracks will cross either other tracks or the base road network. Whenever anything like this happens, the system needs to be able to take appropriate action.

In the case of a track crossing an already existing edge (see figure 3.1 for an example), it becomes necessary to insert a new node into the network. This node represents the place where the



(a) The black track is the road network and the red track is a newly recorded user track.

(b) The blue circle represents a newly created node that binds the tracks together in a network.

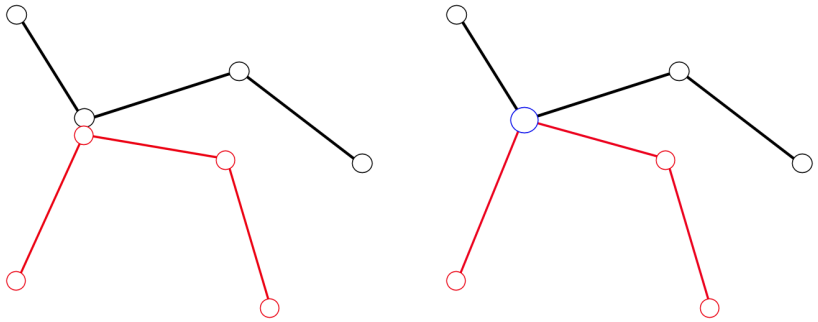
Figure 3.1: Crossing tracks

track and the edge meet. There are two tasks that the system must perform here. First, the system must discover the intersection, and then it must create the node to represent it and alter the affected edges appropriatly.

The first task is easy to solve using the JST Topology Suite, an open source Java library for representing graphs and geometry of any kind. It provides efficient methods for discovering intersections between geometric objects which vastly simplifies the process of solving the first task in integrating a crossing track. Using this library, I can simply ask for the point of intersection between two lines, the JST equivalient of a curve, and the suite returns a point or an error-value if the lines does not intersect.

Once an intersection has been found, the second problem of altering the geometry to fit the intersection begins. The result of the merging should be a set of edges such that it is possible to traverse both the original edges, and to go from one of the original edges to the other via the intersection.

In order to create the intersection between the edge and the track, a number of tasks must be performed. Specifically, these tasks are creating a reference-point at the point of intersection, splitting the two intersecting geometries into four geometries that meet at the intersection point, removing the old edges from the network, and inserting the new edges into the network. Furthermore, the old edge that was present in the network before the user's track was inserted may have had various feedback associated with it from earlier use of the system. This feedback must remain intact after the merge and be applied to the two new edges that cover the same ground as the old edge.



(a) The black track is the road network and the red track is a newly recorded user track.

(b) The blue circle represents the merging of two nodes.

Figure 3.2: Incident tracks

The other possibly problematic case is that of incident tracks and edges, or more specifically incident points (see figure 3.2). In this situation there exists a point on a track that lies very close to a point on the already established network. The goal of dealing with this situation is to alter the user's track just enough that it uses a point on the established network rather than its own point, provided this does not change the user's track too much.

Fortunately, this is a relatively easy job compared to insertecting edges. For each point of the user's track, we use the JST Topology Suite to locate the closest edge. This is done by applying a small buffer to the point, turning it into a circle. This circle is then intersected with the selected edge. If this circle intersects with the network edge, we find the closest point on the edge to the original point. This closest point then replaces the original point in the user's track.

Figure 3.3 outlines the pseudocode for inserting a new track into the network. Lines two to four deals with incident tracks by altering the user's track to fit the network in those places the two are very similar. In line five, I use the JST Topology Suite to discover intersections between the network and the user's track. Provided there are intersections, these are then used to split both the network and the user's track in lines nine through 16.

The first thing that is done is to create a local copy of any feedback that may have been given to the part of the network that has been intersected. This part is referred to as *oldEdge* in the pseudocode. The old edge is then removed from the network and new edges created on line 12 and 13. The edges that came from the old edge are given the same feedback as the old edge had, and

```
01. insertNewRoad(network, newroad)
02.   for each point P on newroad
03.     if P is closer than some cutoff value V to a point Q on network
04.        replace P with Q in newroad
05.   intersectionSet Is := JST->findIntersections(network, newroad)
06.   if Is is empty
07.     network := network union newroad
08.   else
09.     for each intersection I{oldEdge, newEdge, Point} in Is
10.       feedbackSet Fs := any feedback related to oldEdge
11.       network := network \ oldroad
12.       oldRoadSet := split(oldEdge, Point)
13.       newRoadSet := split(newEdge, Point)
14.       apply Fs to oldRoadSet
15.       network := network union oldRoadSet
16.       network := network union newRoadSet
```

Figure 3.3: The algorithm for inserting a new track

finally both sets of edges are inserted into the network.

The other way of collaborating is for users to critique the routes they are given by the Ranger. When they do this, they essentially share their experiences with the system, so that others may learn from them.

3.4.2 Sharing Experiences

In addition to sharing tracks, the users may share their opinion on how well suited a particular path was to their needs. This serves to maintain a model of how the cityscape is percieved by the users.

Each usergroup maintains a running average of the points each segment has scored. This number gives an approximation for how well suited a particular segment is for the usergroup in question[1]. This way I use Haigh's idea of the β-value [16] to capture the general value of a stretch of road. The β-value is a single real number that is factored in with the cost-function like this: $cost = \beta * realCost$. However, I do not fall into the trap of using a single value for all users [17][32], creating a universially inaccurate measure of quality. Instead, the users define themselves as part of a particular group, and these groups share experiences. Provided the users are honest in choosing a group and the groups themselves are well enough defined to capture the general needs of different

[1] One could also use the weighted averages of other usergroups, provided one had some way of establishing realtionships between the usergroups. This is for future considetion, though.

users, this method should alleviate the problems of universial application while avoiding the need to know each user's individual needs and wants.

An Example of Sharing

Figure 3.4 shows a scenario where one user's impressions of a path affects the subsequent requests for other users. Assume we have two users, named A and B. These users are positioned in different locations in the city, but they want to go the same place, see 3.4(a). Person A makes the first request for a path, and is given a straight line east. This path is given a score of 90 and is the cheapest possible route to the Goal. Note that the score is in an arbitrary unit.

However, A is not happy with the path he was given. For some reason, A feels that the path does not provide for his particular needs. Perhaps it was too steep at times, or maybe it lacked proper sidewalks. No matter the reason, A gives the path a scorching critique, and the system re-assigns cost to the edges involved, see 3.4(b).

At some later point, person B asks the Ranger for directions to the Goal. Although a straight line to the east would have been the best path according to the basic network, the system gives B an entirely different path, see 3.4(c). This alternative route takes into account A's critique of his path and takes a detour that may be better suited to A and B's user group.

3.5 Datamodel

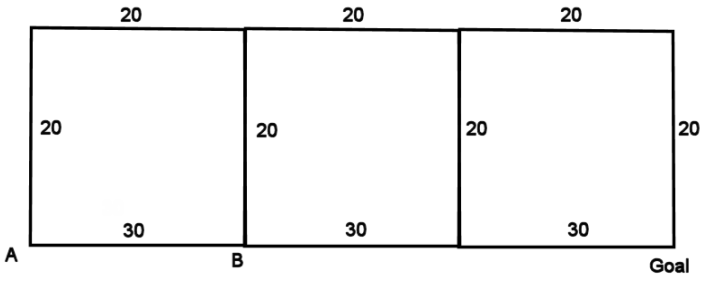
Here I will outline the logic to the data used in the project. I will describe the concepts that I need to design the Ranger and how they are related to eachother.

The Ranger data is basically geographic data and associated metadata. The geographic data represents roads or other traversible ground and intersections that allow one to move from one road to another. The metadata associated with the geography describes the quality of the roads relative to the various user groups. Figure 3.5 shows the relationship between the various entities in the model, and I will describe the various entities in some detail.

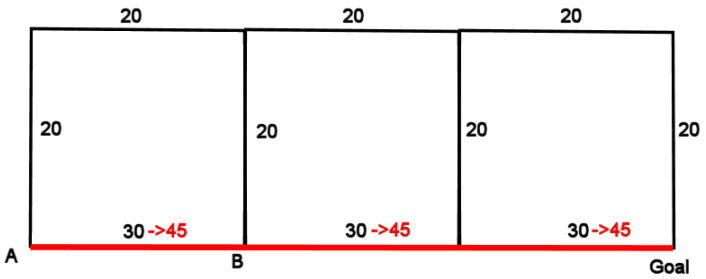
The Point

A point is a geographic position identified by latitude and longitude. This represents the smallest geographic entity in the Ranger, and serves as a building block for many other constructs.

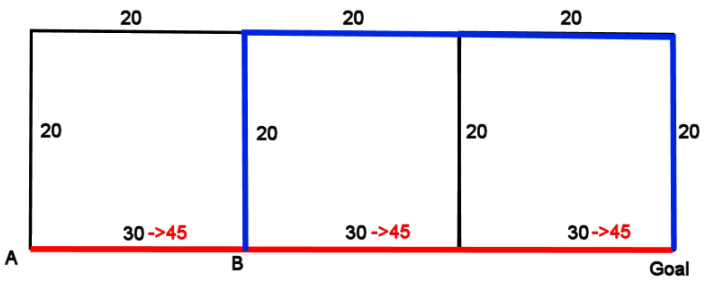
A point is created when the system needs to represent geography and it wants to use a latitude/-longitude pair that does not already exist. A point is never destroyed as it might always have some



(a) A and B both want to find a route to Goal



(b) A get a bad route and gives it a poor review



(c) B is given a route augmented by A's review

Figure 3.4: Changing the cost of travelling depending on user's experiences

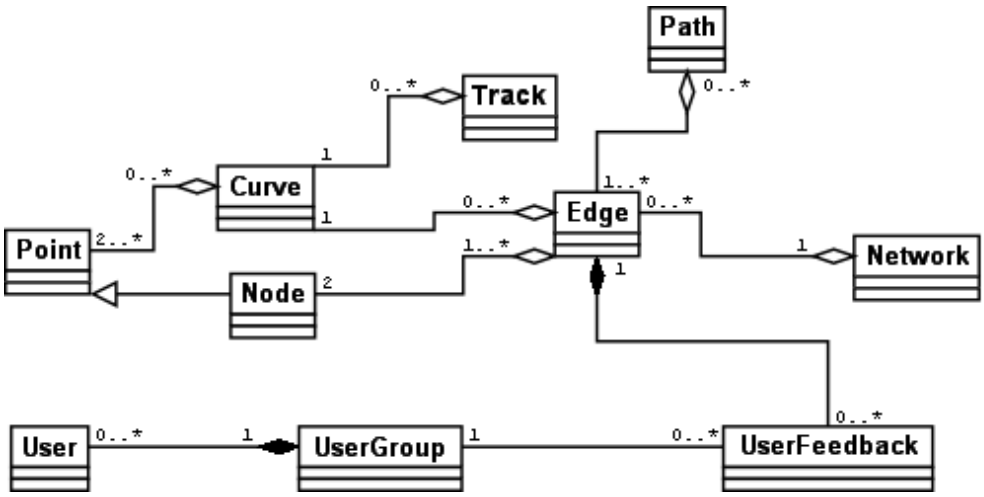


Figure 3.5: The datamodel

use in future geometries.

The Node

A node is a point that sits at the end or beginning of an edge. In addition to having a location inherited from the Point, it also knows about its neighbouring nodes. A node's neighbourhood consists of all the nodes belonging to the edges that this node belongs to. In figure 3.6, node A has three neighbours coloured blue.

A node exists as long as it belongs to at least one edge. When the last edge it belongs to is removed, the node is demoted to a regular point.

The Curve

A curve is an ordering of points that define a connected set of lines. Figure 3.7 show a number of such curves. The points labeled P0, P1, and P2 form one curve, while the points P3 and P4 form another curve. The point P5 does not participate in a curve.

Curves describe the geometry of tracks and edges. They are created when the system wishes to describe the actual geography of an edge or if the user draws a track.

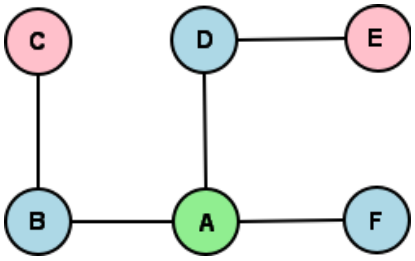


Figure 3.6: Node A's neighbourhood includes nodes B, D, and F, but not nodes C or E

The Edge

An edge is a central object in the model. It represents a single, unbroken stretch of traversible ground, and a collection of edges makes up a searchable network. An edge is created when a stretch of road from one intersection to another is being described in the system. It lives as long as no changes are made to the network it belongs to, but may be destroyed if a new edge is created that intersects or runs incidental with it. In such a case, new edges will be created to take the place of the destroyed edge.

An edge is defined by a single curve and two nodes. The curve represents the geometry of the edge, i.e. its geographic appearance. The nodes represent the two ends of the edge.

An edge may also have user feedback associated with it. This user feedback will affect the value of the edge, which is calculated from the geographic length of the edge's curve.

The Network

A network is a collection of edges and nodes. It is the umbrella object that is used for searching for paths and for inserting tracks. Every time a route planning or track insertion operation is to be performed, a network is created from edges located around the geographic location of the operation. For example, for a route planning session from the hotel Oslo Plaza to Oslo City Hall, only edges in Oslo Downtown is included.

The network allows changes to be made to it's members. Specifically, it allows edges to be created and destroyed in response to the integration of tracks, and it allows edges to change their user-feedback members in response to actual user feedback.

The Path

A path is a continous ordering of edges through a network. They are created when the Ranger performs route planning tasks, and live throughout the route planning session. The first path that is

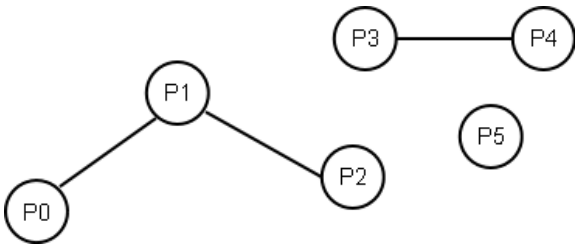


Figure 3.7: Two curves and a loose point.

discovered during route planning that connects the start and end point of the search is considered the solution to the query.

The Track

A track is a recording of a user's movement in the real world, either created automatically by a GPS or manually by a human. The track is created as a user-action to record his or her movements. It lives until either discarded by another user-action, or replaced by a path representing the same geometry.

The UserFeedback

UserFeedback represents the user's critiques of edges. Each instance of UserFeedback is a combination of all the feedback received for a particular edge and a particular usergroup. User feedback is created the first time a particular edge is given critique by a new usergroup, and is then mutated with every subsequent critique offered by members of the same usergroup. User feedback can also be created in the event of an edge being created. If the edge is created to replace an earlier edge with user feedback, a new UserFeedback instance is created to be associated with the new edge.

The user feedback is discarded when the edge is belongs to dies.

The UserGroup

A usergroup is a collection of users that share some characteristics. These characteristics are not spelled out, and each individual user must join a usergroup that seems to fit their own characteristics the best. A usergroup is identified by its name which should be enough to describe it to human users.

Usergroups are permanent objects in the Ranger, although they may be created or destroyed by a database administrator. I future iterations, users should be allowed to create new usergroups to

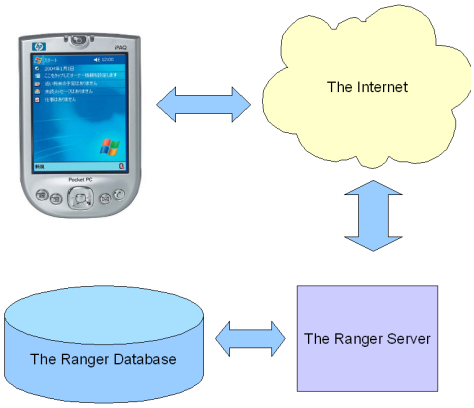


Figure 3.8: The Rangerarchitecture in brief

better meet their individual needs.

The User

A user represents an actual human user of the system. The user is created when someone registers with the Ranger, and lives, in theory, for ever. Users must belong to a single usergroup, but can change usergroup at will.

3.6 System Architecture

The Ranger is designed around a classic client-server architecture (see figure 3.8). The server sits on the data and creates paths on demand from a client, and the client presents these paths to the user. The user may then change the paths or create new tracks and store these on the server.

The client resides on a handheld Windows Mobile device and communicates with the server using the device's network connection. This can range from Wireless Broadband to a GSM modem connection. The server resides on a stationary computer and is accessed through a regular web-server, such as Apache 2.0. The database resides on the same computer as the server code. It is contacted through Java Database Connectivity (JDBC) calls via Apache as well. JDBC is a well known architecture for Java applications to connect to databases and execute SQL statements. The

client-server communication is purely synchronous http-requests, where the request takes the form of a parameterized URL and the response is an XML document.

A more detailed description of the client, server, and database can be found in Appendix A.

3.6.1 The Client

The client is the user's window into the Ranger system. It is responsible for presenting the user with the various routeplanning services offered, and for displaying the tracks the user has asked for. It must also keep the user informed about any errors that may occur or if the system needs time to think.

The client adds two menu-items to Okapi's already existing menu. These menu-items are divided by functionality: One provides the user with an interface to route planning. The user may place starting points and end points, or he may ask for a route to the closest point of interest. He may also use this part of the interface to offer feedback to the routes available on the client.

The other menu-item allows the user to draw his own tracks. Here the user may choose to create a user track by adding points at the current position of the cursor, or he may opt to start GPS tracking. GPS tracking is performed by the client polling any GPS connected to the mobile device. This GPS polling is handled by the Windows Mobile GPS Managed API.

Tracks can be edited by the user. As tracks are essentially just ordered points, they can be edited by simply relocating the points. The clients allows the user to do this in a drag-and-drop fashion. The user can click on a point and then drag it to a new position. When the user lets go of the point, it is given it's new position, and the track is updated.

The client communicates with the server using http-calls; simple URLs with field-value pairs to pass parameters. One such call could be `http://platypus.hiof.no/okapi/server/xsd/trackster.php?userID=123&act=feedback&route=234&score=0.5`. Here, the user has offered feedback to a particular track.

The response from the server is always an XML document that contains a reference to an XML-schema for validation purposes. By validating the server XML, the client can always be certain is has been given a proper, well-formed response to its request and can treat the document accordingly.

When the client receives routes from the server, it displays these as a yellow line on the map. These routes can be selected by the user, and when selected they are outlined in red to make them more visible. In order to offer critique of a route, the route must first be selected. The user then has the option of different fuzzy terms to describe his feelings about the route, ranging from "Very bad" to "Very good". These terms are translated to numberical values and transmitted to the server.

3.6.2 The Server

The Ranger server is responsible for managing tracks and paths, as well as user feedback and geometry-merging. It's responsibility can be broken into three main areas. First, it must perform route planning tasks. Second, it must handle user feedback and organize this is a sensible way, and finally, it must integrate the users's tracks into the already existing geometry.

The route planning task is in many ways the least innovative part of this project. The Ranger server handles route planning by performing the A* algorithm, using a MinHeap to store intermediary solutions. The heuristic used is the Haversine distance from the end-node of an edge to the goal position.

The Haversine formula is a formula for calculating the distance between two points on a sphere given the radius of the sphere. Since the earth is not spherical, this leads to some inaccuracies, but these are negligible considering the relatively small area the project is concerned with. This heuristic is guaranteed to be admissable, as there is no shorter distance between two points on a sphere than that which is described by the Haversine formula.

The route planning is influenced by user feedback as well as the heuristic, and the server needs to maintain control of all the users's input. The server assigns each edge in the network a β-value for each separate user group. This value represents the average score each user has given the edge during its life. This β-value is multiplied with the actual cost of an edge to produce a weighted cost. The cost of an edge is taken to be the length of the curve that represents the edge's geometry.

The final responsibility of the server is to merge the original geometry with any new tracks the users may produce. To perform this, it uses the algorithm I outlined in Figure 3.3. This algorithm alters the user's track slightly to avoid situations where many similar tracks generate multiple edges covering nearly the same real-world paths. Once this alteration is done, the algorithm attempts to create new nodes where the user's track crosses or touches already present geometry. This implies that some old edges may need to be replaced by new edges, and that old feedback must be copied to the new edges. When the algorithm terminates, the network should have been altered so that the new track is traversible and searchable like any other part of the network. The original network should retain enough edges, or have old edges replaced so that any path that was discoverable before the geometry merge is still discoverable after the merge.

- Point to point route planning
- Route to nearest point of interest
- User group based route ranking
- User created tracks

Figure 3.9: Rangerservices

3.7 Services

This section will outline the various services that the Ranger is to offer the user. Figure 3.9 shows an overview of the these services.

Point to point route planning

Point to point route planning involves the user selecting two points on the client's screen and then having a route between the points generated by the system. The user will designate a start point from where the route planning begins. He will also designate a target point that the system will search for during the route planning.

When the user asks for a route, the client will indicate that it is searching for a route, and the user is free to persue any other activity during this time. When the system has found a route, the indicator disappears, and a route is drawn on the map.

Route to nearest point of interest

This services resembles the point to point route planning, except that the system will discover the target point based on criteria given by the user. With route to nearest point of interest, the user specifies a starting point, and then asks for a route to the nearest point of interest that matches what the user is looking for. For example, the user may ask for the closest accessible parking to his current location.

User group based route ranking

Once a route is displayed on the client screen, the user may offer feedback on the route's quality. The user can click on the track and enter a feedback/edit screen. This screen allows the user to make a statement of quality of the track and have this uploaded to the server.

By collecting feedback this way, the Ranger aims to generate routes that fit the user's needs better than straight up route planning.

Usertracks

The final service the Ranger offers the user is usertracks. This service is actually split in two: Manual drawing and GPS tracking. With manual drawing, the user places points on the map using the stylus or other direct input device. These points are linked together to create a continous curve, or track through space. As an alternative, the user may connect a GPS device to the client and have the GPS draw points on the map.

In either case, when the user has created a usertrack he is satisfied with, he can give the usertrack a name and upload it to the server. The server will then attempt to integrate the usertrack with the road network already present in the database. When the integration is complete, the client presents the user with the resulting track, which may be slightly altered to better suit the underlying model.

Chapter 4

Testing

Testing is the bread and butter of a system developer. It includes everything from ensuring that single methods or code-snippets do as they are intended, to having end-users try out the system and offer feedback as to what is good and what needs improvement.

Although the Testing chapter comes near the end of the thesis, be advised that testing has been an integral part of the development from the very beginning of this project. In addition, at several points during the development, I have had fellow students and staff at the college try out the system and offer me feedback.

In this chapter I will describe two major tests and their results. These tests include a final, in-house system test that ensures that the system performs reasonaby well and that is adresses the issues outlined in the introduction, and an informal user test to explore the application's ability to interact with real users and garner experiences regarding the user interface.

4.1 System testing

For the sake of this project, system testing takes precedence over user testing. This test will result in an overview of what functionality is ready for user-testing, and what functionality needs more work. However, due to the lack of time and resources, this will not be a comprehensive test covering every contingency. It is intended to demonstrate functionality rather than ensure robustness.

The test is divided into four parts, each exploring different aspects of the Ranger system. Many parts are partly dependent on one or more of the former tests to have ended successfully in order to be run. Most parts have several sub-parts that all must be tested. In some cases it is possible for a test to succeed in several sub-tests and still fail.

Every test is performed on a Windows Mobile 5.0 Pocket PC VGA emulator for best screen shot opportunities. The emulator is cradled using the .NET Device Emulator Manager and connected to the internet through ActiveSync 4.0. These tests will not depend on bandwidth, so using a high-speed connection shouldn't be a problem.

The server and the code-base are both final before these tests. That means that as long as the tests don't uncover major issues, there won't be any fixes made until after the user test.

This section includes a lot of screenshots. On many of these screenshots, one can find a blue circle and/or a red cross. The blue circle represents a point selected by the user as the starting point for a route planning session. A red cross on the screen is a point selected by the user to be the goal position of a route planning session.

4.1.1 Calculate Route from A to B

This is the most basic functionality of the Ranger. Here a route is requested between two user-defined points on the map. There are a number of various cases that must be tested:

1. Source and target point are positioned at or near nodes in the network
2. Source and/or target point is/are positioned at or near an edge in the network, but not near a node
3. Source and/or target point is/are positioned inside the network, but not close to any edge or node
4. Source and/or target point is/are positioned outside the network and not close to any edge or node

Criteria

The four variant tests have slightly different criteria for success.

In the case where both source and target point are positioned at or near nodes in the network, success means that the client displays a yellow line running from the source point to the target point. This yellow line must follow roads or other traversible paths on the network at all times. Minor success is acheived if this yellow line connects the two points. Major success implies that this route is in fact an optimal route between the two points. Testing for major success may be difficult in a real world application, however, and minor success is sufficient.

The second case deals with situations where one or both of the user's points lie on the network, but not near any node. The critera for success are the same as with the above.

In the third and fourth case, one or both of the user placed points are positioned so that they don't have a reasonable approximation to the network. In both cases, the system should return an error message. If these two tests succeed, the client will indicate that it is done loading tracks, but no tracks will be displayed. If the user opens the track overview, it will show an error message for the last requested track.

Tests

Setup The setups for all four tests are identical. I start the Okapi application on the emulator by clicking on the Okapi executable. In tests one through three, I move the map to Oslo downtown where I know I have a road network. In the last test, I move the map to Halden, where I don't currently have a network.

Result

Variant one For the first variant test, I started the Okapi application from the emulator. Figure 4.1 shows two typical runs. In sub-figures 4.1(a) and 4.1(c), I have positioned target and source near intersections visible on the map. In the latter figure, the intersections become visible as you zoom in. I know that intersections that are visible on the map are represented as nodes in the road-graph, so these setups fullfills the demand of this variant test.

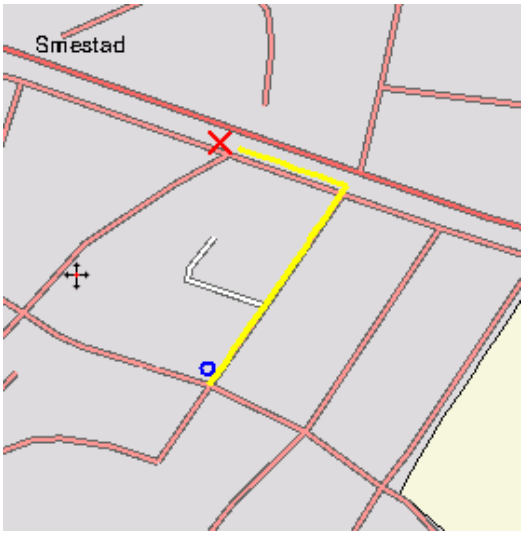
Once the start and end points are set, I double click on the screen to get the context menu, and select "Find route" from this. Sub-figures 4.1(b) and 4.1(d) show the reaction from the client once the server is done calculating the routes.

This test can be considered a success, as routes were found between the two points specified by the user, and these routes were successfully displayed on the screen.

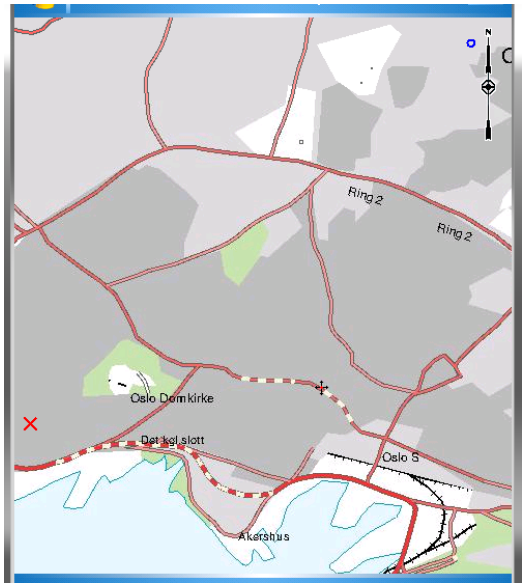
Variant two The second variant test started just like the first. I positioned target and source points on roads on the map, but away from visible intersections (see sub-figure 4.2(a) for a typical placement).

The resulting paths from these tests, like the one in sub-figure 4.2(b), lack precise start and end points. Instead, it looks as if the system selects points from the edges the user placed his points on, to use as source and target.

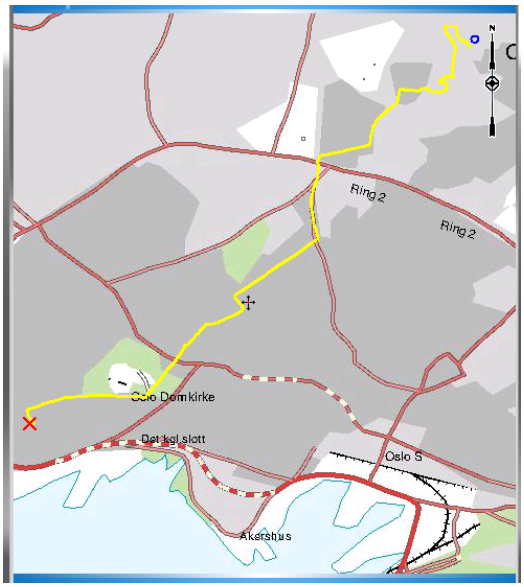
(a) The user has set up a start and end point



(b) The resulting path

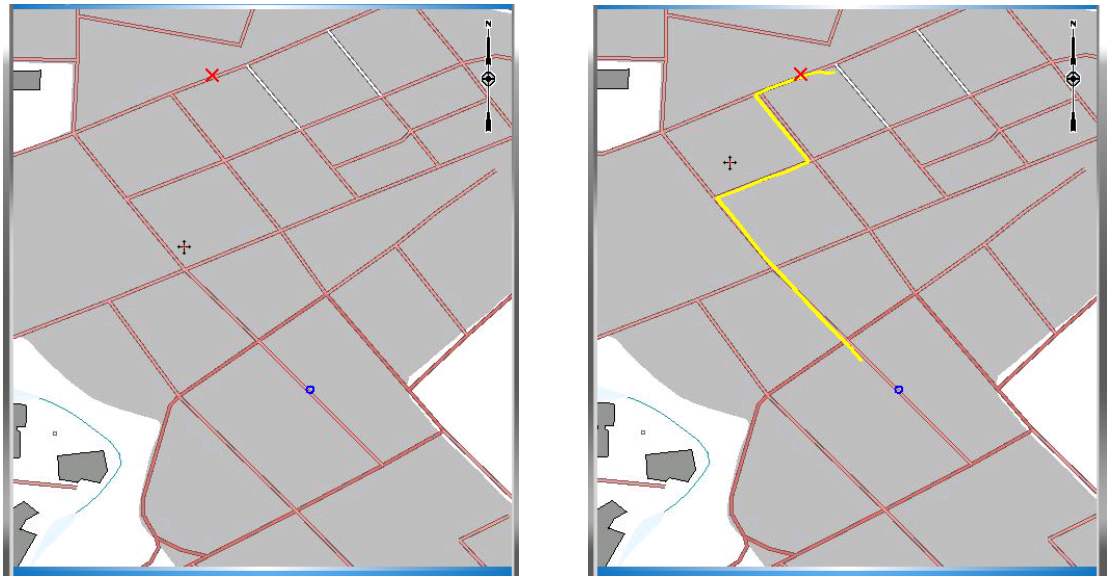


(c) The user wants a significantly longer route



(d) The resulting path is more detailed than the shown map

Figure 4.1: Screenshots from Test 1 Variant 1



(a) The user has set up a start and end point

(b) The resulting path is not as good as in variant 1

Figure 4.2: Screenshots from Test 1 Variant 2

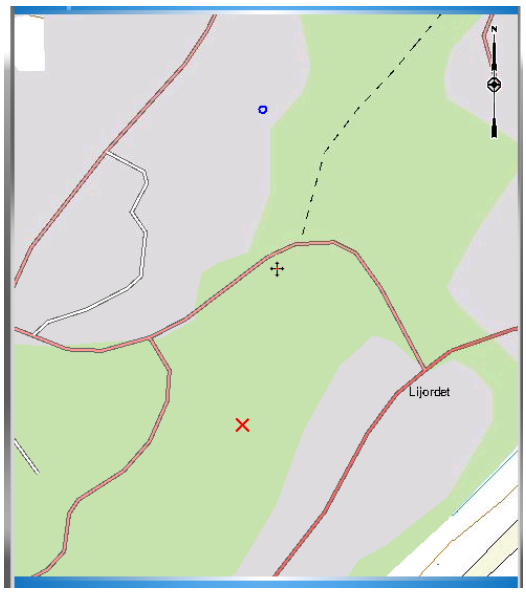
This test is a minor success. Paths are discovered that would help a user find his way between the points. However, the paths only approximate the user's selected points, and can therefore be confusing in certain situations.

This is a point where improvements should be made, were this system to be released.

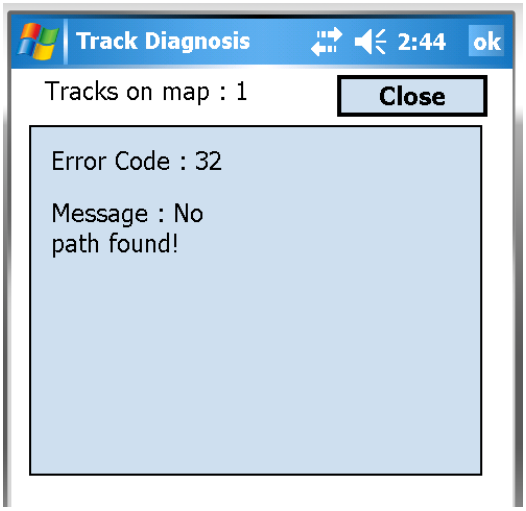
Variant three For this variant test, I set up points inside of Oslo, but in parks or small forests, where I knew the network didn't cross. See figure 4.3(a) for an example. In most cases, the client would show a "Track Loading" icon briefly. When the status icon disappeared, the client showed the same state as before the path finding request. However, when the user opens the feedback menu, he is met with an error message saying that a track could not be found (see figure 4.3(b)).

Every now and then, however, the system would give tracks that only approximated the user's points very crudely. Figure 4.4 shows one such situation, where the user positions a source point in the middle of nowhere. Here, the system, rather than admitting failure, selects a point a rather long way away from the user point and start.

Variant four This variant involved me asking for tracks far away from Oslo. The setup and results for this test were identical with variant three, but I noticed that the system reacted far more quickly.



(a) The user has set up a start and end point inside the extremes of the network, but away from any nodes or edges



(b) The resulting error message

Figure 4.3: Screenshots from Test 1 Variant 3

This is likely due to the network not being loaded into memory before the request is denied.

Both variant three and four can be deemed successes, with the proviso that the crude approximations when the user's points are not on the network consist a good enough solution.

4.1.2 Calculate Route from A to a Point of Interest

This test is in many ways similar to test 4.1.1. However, instead of the user defining the target point, the system discovers a suitable target point based on the user's criteria.

The various cases that must be tested are identical to those of test 4.1.1, but does not include the target point. However, these cases must be performed for each possible set of criteria the user may set.

Criteria

For each set of target POI (these are ”Toilet”, ”Subway Station”, ”Parking”, ”Attraction”, and ”Hotel”), the client must display a route from the user's selected starting point to a POI of the selected sort.

Some POIs may suit several target types. For example the POI category ”Hotel” will often have access to toilets, so a search for a POI of type ”Toilet” will succeed even if it points to a ”Hotel”.

These tests succeed if the client shows a route from the source point to a POI that meets the criteria.

Test

To perform these tests, I go through most of the same steps as in test 1. However, in each case I refrain from placing a target point. Instead, I double click the screen and from the ”Route Planning” menu I choose ”Find nearest...”. Then I select one of the displayed categories and await the result. Every available category is tested for each of the four variants of the test.

Result

Each test gave more or less the same result as their test one counterparts.

Variant one and two both succeeded, with variant two lacking in accuracy.

Variant three suffered from the same crude approximation that was discovered in test one, while variant four succeeded with flying colours.

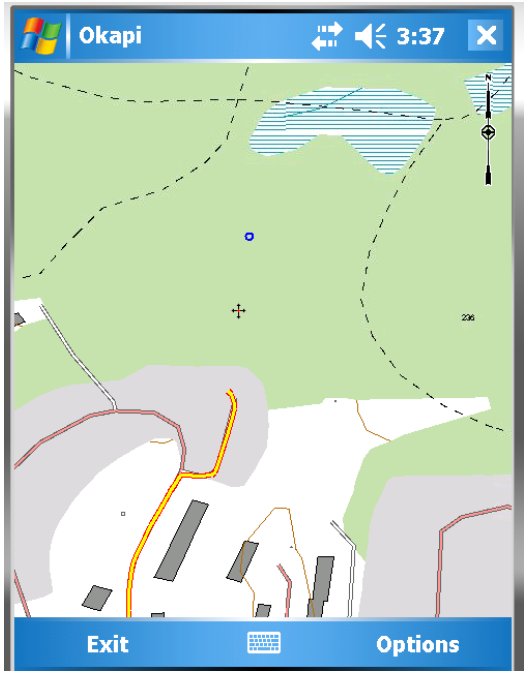


Figure 4.4: Sometimes the approximation is over-eager

4.1.3 Calculate Route after network has been reviewed

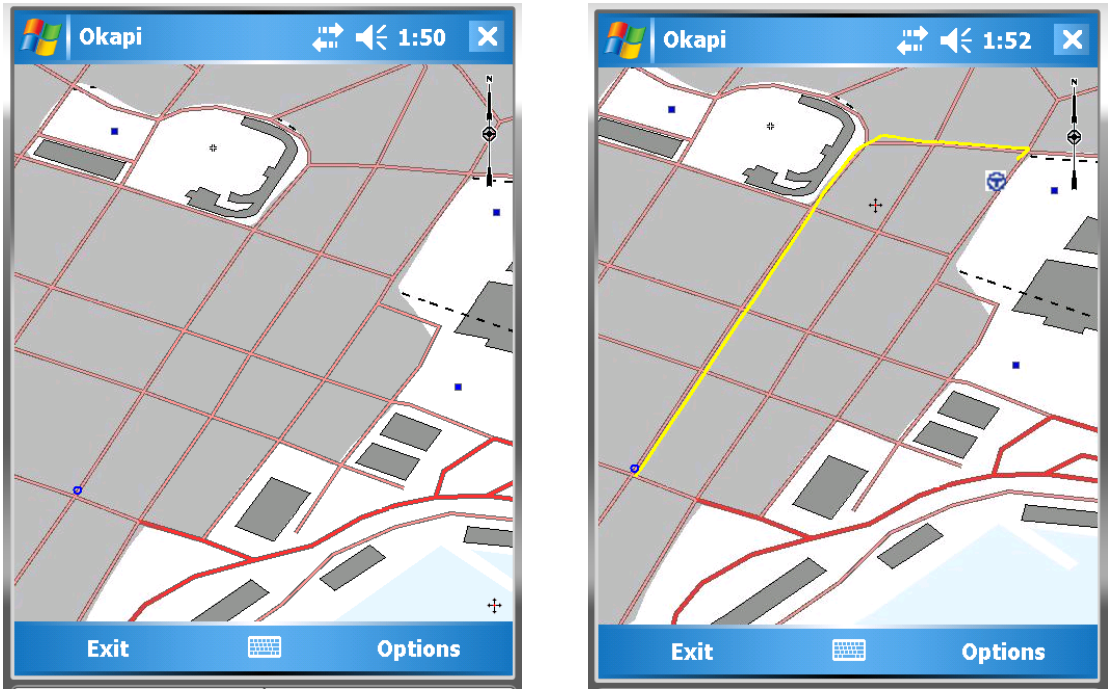
In this case, the system calculates a route between two point and takes into account feedback offered by users about the edges that take part in the search.

There are two basic variations of this test: Avoid edges that have been deemed bad, and prefer edges that have been deemed good.

Criteria

For the first variant test to be successfull, the system must first give an adequate route from A to B. Then the edges in this route must be marked as bad, and the route recalculated. The new route should avoid the edges used in the first route.

In the second case we must offer positive feedback to some edges and then see that these edges are preferred in a route planning scenario.



(a) The user has set up a start point

(b) The resulting path and point of interest

Figure 4.5: Screenshots from Test 2

Test

Variant one For this variant, I need to find a place on the network where I can easily create situations where feedback alters the system's preferred routes. An idealized network can be seen in figure 3.4, and figure 4.6(a) shows a location in Oslo that can be used for similar tests.

The last figure also shows the positioning of start and goal points.

For this variant test, I first ask for a route. When the route is received, I select it by clicking on the yellow line (see figure 4.6(b)). I then proceed to offer my feedback. I give the track the worst possible critique. When I am returned to the map, I ensure that my critique is received by checking the database.

Finally, I ask for another track between the same two points. This second route should go along different edges than the original.

Variant two Using the same location as in variant one, I can explore this test. It involves a couple more steps than the first test, though. Since the lower route is the preferred route if we discount the previous feedback, i.e. we refresh the database, we need to get access to the edges involved in the upper route in a different way. To do this, I repeatedly ask for routes along shorter stretches of the upper route. This way, I can guarantee that I get the edges I want. Once I have access to the designated edges, I give them the most positive feedback available to me. With this is set up, I can ask for the route between my original two points.

Result

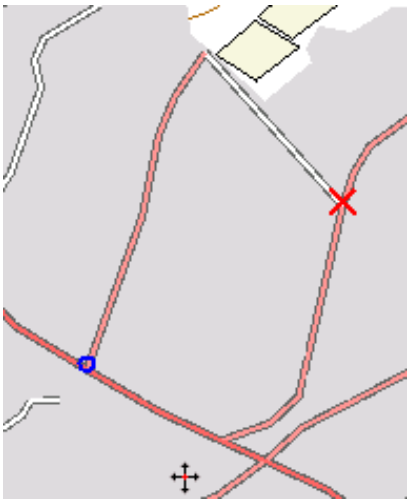
As figure 4.6(c) shows, the second path gives an alternative to the path originally suggested. This was the result in both variants of this test. In conclusion, test two was a success.

4.1.4 Adding edges to the network

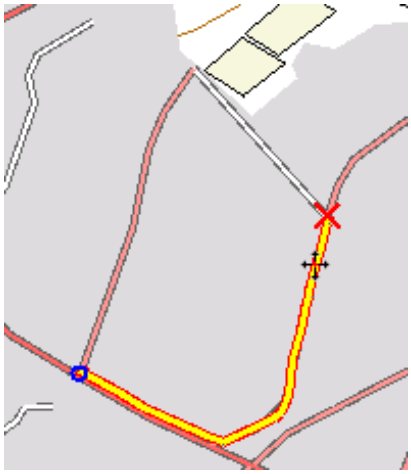
In this test-segment we add new edges to the network. There are two ways of adding edges: by drawing them on the screen or by recording one's movement with a GPS.

Once the method for adding edges are decided, there are a small number of variant cases in this test:

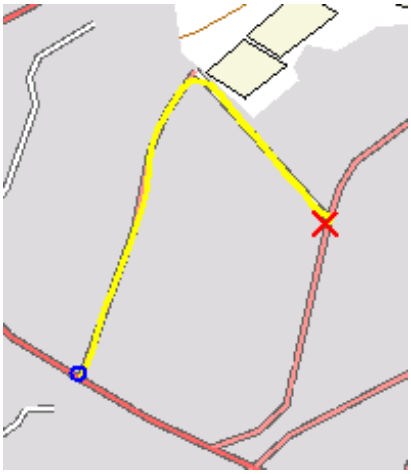
1. No part of the user track crosses or touches the original network
2. The user track crosses the original network



(a) A suitable location for the test



(b) The first track



(c) The second track

Figure 4.6: Screenshots from Test 3, Variant 1

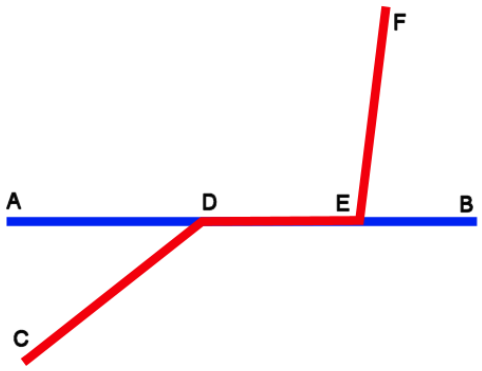


Figure 4.7: An idealization of the fourth test

3. Part of the user track overlays the original network

Criteria

For the first variant test, the criteria for success are few and simple. After the upload, the client should display the uploaded track as identical to the drawn track. It should also be possible to perform regular route planning from either end to the other.

In the second variant, we should get a new node at the point of intersection. A successfull test would result in the client displaying a track near identical to the track drawn by the user. It should also be possible to generate routes travelling between the end points of the user's track, between the end points of the edge that was intersected, and between any of the four ends to any of the others.

For the last variant, we expect a number of things. Please consult figure 4.7 for the following description. We start with the original network represented by the blue line running from A to B. The user then adds a second path starting at C, running through D and E, and terminating at F. After uploading this path, is should be possible to get a number of routes. In fact, it should be possible to get from any of the labelled points to any of the other points. This makes for fifteen possible routes, if we assume that a route can be traversed both ways.

Test

For each of these tests, I start out with a refreshed database, so that I do not have to consider any previous user created content. For variant one, I use a location on the map that I am sure does

not already contain a network. For the two other variants, I use the downtown area of Oslo, as it contains several straight edges that make it easy to plot test lines.

Result

Variant one In this variant, I simply find someplace on the map where I know there isn't a pre-existing network. This is easy, as only Oslo currently has a network.

Once I have found a suitable location, I can draw a track using the manual function of the system. The number of points I use to create the tracks differ from only two to more than twenty.

Figure 4.8 shows a user plotting close to 20 points along a known road (4.8(a)). He uploads the track to the database, and the client displays the track as it was stored in figure 4.8(b). I have selected the track to make it easier to see.

To ensure that the track is searchable, figure 4.8(c) shows a route planning request using the ends of the submitted track as start and end points.

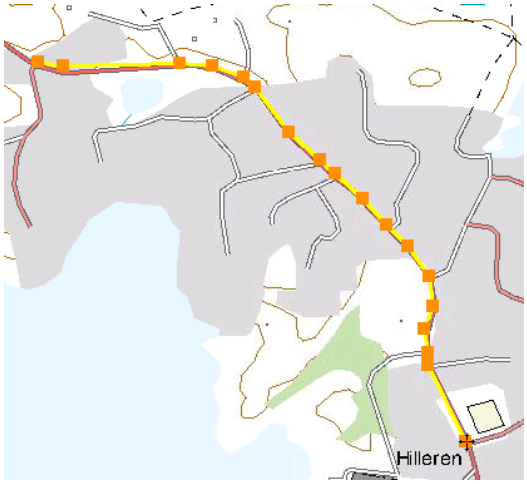
Given that the route is returned as expected after a track submission, and that the user track can be used for subsequent route planning, this test can be considered a success.

Variant two In this test, I first find a location where route planning gives a somewhat long winded route, as show in figure 4.9(a). Once such a place is found, I create shortcut by laying a user track across the already existing network like in figure 4.9(b). Then, by asking for another route between similar points as for the first request, I demonstrate that the user track has been integrated into the network of roads. This is shown in figure 4.9(c).

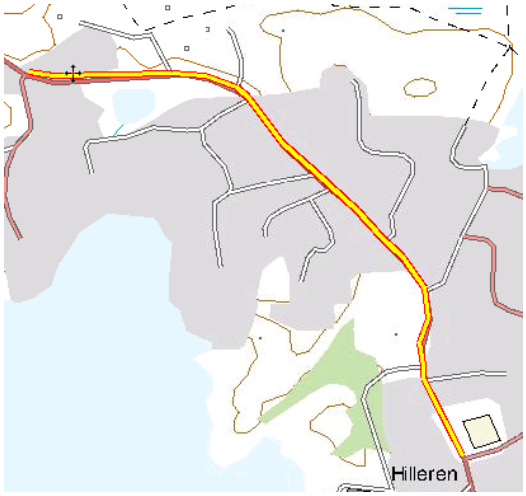
Variant two of this test is a success.

Variant three For this test, I find a location that includes a fairly long stretch of unbroken road. One such road, with the edge in question already beeing searched for and found as a route, can be seen in figure 4.10(a). I then proceed to create a user track that is partially incidental to the original track, as seen in figure 4.10(b). This new track can then be used in route planning, like in figure 4.10(c).

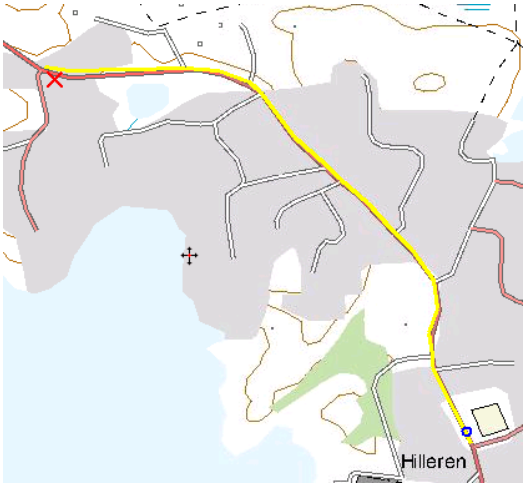
However, not all route planning using the new track works as planned. As one can see from figure 4.11(a), an extra bend has been introduced. It is as if the first point that hit the original network is disregarded with respect to linking edges together. Figure 4.11(b) shows an ever worse situation, where the system gives the user a long detour for what should have been a straight-forward route. In this case, it seems as if parts of the original network is lost. Subsequent testing shows that



(a) The user has drawn a track along a known road

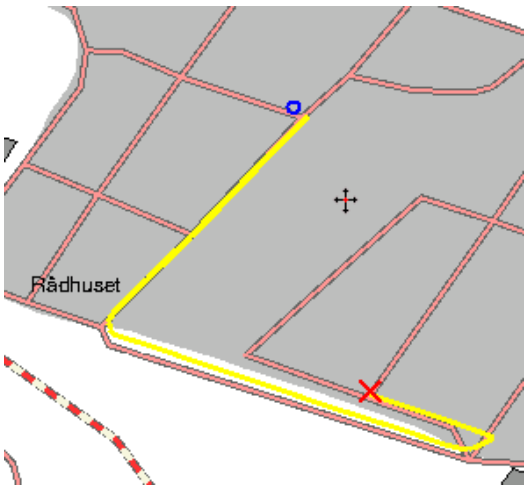


(b) The result of uploading the track

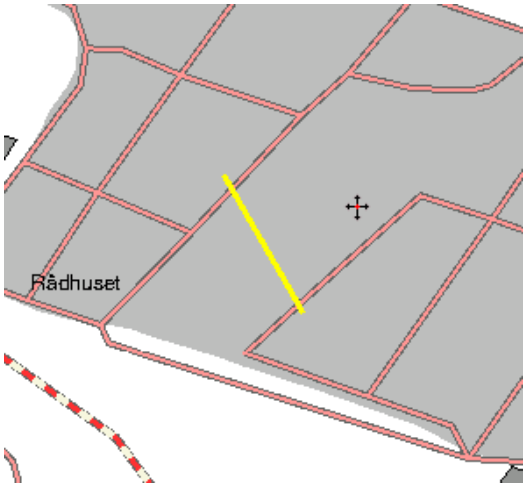


(c) The result of a route planning request close to the user track

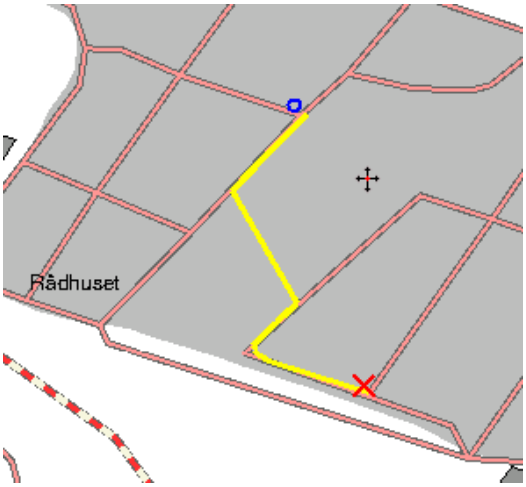
Figure 4.8: Screenshots from Test 4 Variant 1



(a) The user has been given a long winded road

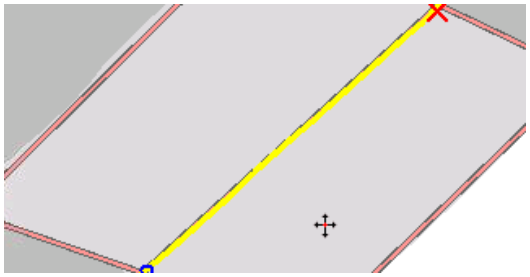


(b) The user has created a new track along a shortcut

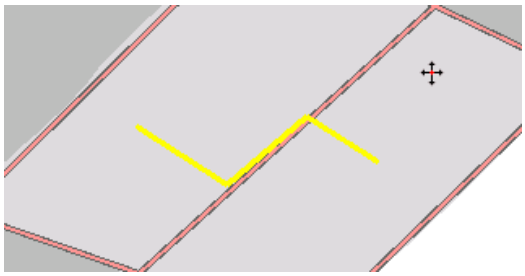


(c) The shortcut is used in subsequent route planning

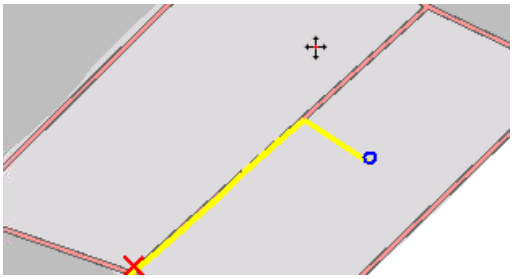
Figure 4.9: Screenshots from Test 4 Variant 2



(a) The original edge is easily traversible



(b) The user draws a track that runs incidental to the original network



(c) The mixed network is searchable

Figure 4.10: Screenshots from Test 4 Variant 3

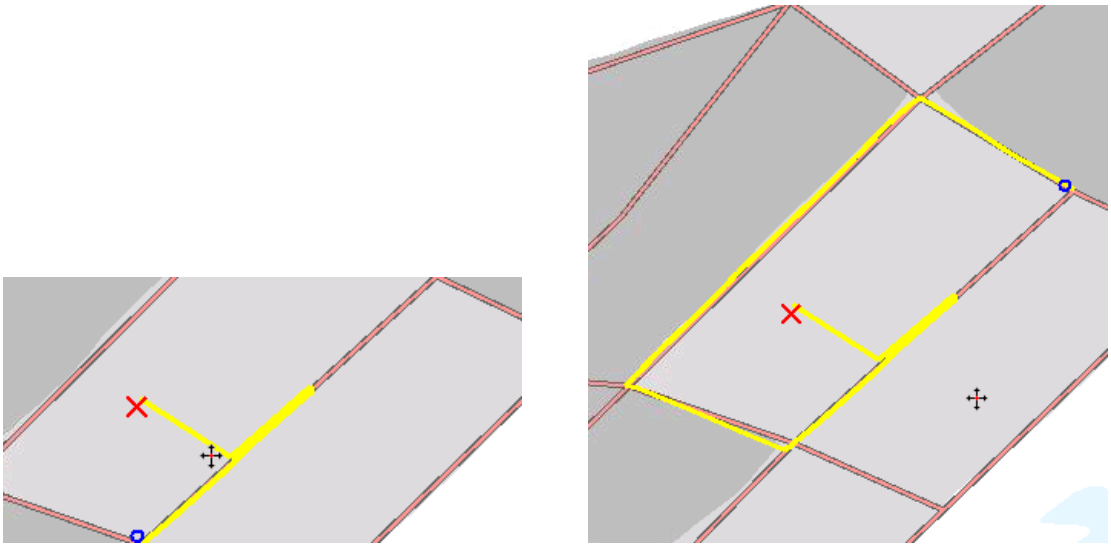
this is not the case, however, as the original edge is still traversible through the original query for this variant test.

In conclusion, this variant test must be considered a failure. It has uncovered a serious flaw that must be rectified before user testing commences.

4.1.5 Conclusions from the system testing

Four functional tests have been performed, each with a number of variants to cover the more common uses of the system. The tests ran from simple route planning to integrating user tracks into the already existing network. The results of the tests are gather in table 4.1.

All of the tests, except for one, succeeded to such a degree that no further work needs to be done. Some minor issues were uncovered, but by and large, these tests showed that the Ranger is functioning as intended. The tests that revealed weaknesses are marked with a '*'. Note that this was not a formal or thorough test to ensure robustness, rather it showed that the services offered by the system does indeed work.



(a) A strange hiccup in the track

(b) An unnecessary detour

Figure 4.11: Test 4 Variant 3 gave some unfortunate results

One single test failed in such a way as to warrant more development work before user testing. This test was a variant of adding new edges to the network. Although adding new edges for the most part gave sane results, something would go wrong when the added edge ran incidental with the original track for some distance. The testing revealed that there is a weakness in the integration where several points after eachother lie on the already established network.

This weakness was adressed by refining the track integration algorithm to include several types of intersections. An intersection between a track and the network could appear not only as a point, but as a line or even a collection of points. Regression testing after the bugfix ensured that no other functionality had been affected by the alterations, and that the repair was successfull.

After concluding the system test, I met with some potential users for an informal usertest and interview session.

4.2 User Meeting

Rather than performing a full blown user test, I organized an informal meeting with some potential users. The purpose of this meeting was to see real users experiment with the application in a natural

Testname	Variant	Result
Route planning		
	Points on nodes	Success
	Points on edges	Success*
	Points inside network	Success
	Points outside network	Success
Route to POI		
	Starting point on node	Success
	Starting point on edge	Success*
	Starting point inside network	Success
	Starting point outside network	Success
Review of track		
	Negative review	Success
	Positive review	Success
Adding edges to network		
	New edges outside network	Success
	New edges cross network	Success
	New edges lie incidental with network	Failure

Table 4.1: The results from the system testing

Figure 4.12: An informal demonstration of the project

setting. This way, I would hopefull learn something about the system's strengths and weaknesses as well as gathering some user experiences. This meeting will server as a source of ideas for future refinements and additions to the Ranger project.

The test users experimented with the system in an informal setting. Figures 4.12 and 4.13 shows photographs taken at the meeting which was held at a café in Halden.

The users raised a number of interesting points while interacting with the system, some of which pertain to my thesis on collaborative routeplanning in addition to a few regarding the usability of the system. I will include the usability issues raised for the sake of completeness, and because usability always remains important for any system. However, it is the issues relating to route planning and feedback that is the most interesting for this thesis.

Main routes and general feedback

One very interesting point raised by a user is that wheelchair users tend to use main-roads when they move about in a city. In Halden there is a pedestrian street that runs through the center of the city. This street does not have sidewalks and therefore lacks curbs which makes this street especially well suited for wheelchair users. Furthermore, Halden has a large number of difficult to navigate sidewalks lining the streets around the pedestrian street. This has the effect that the pedestrian street becomes the equivalent of a high-way for the wheelchair users.

One of the assumptions of this thesis was that pedestrian navigation cannot rely on optimalizations along main roads because pedestrians are unaffected by such things as speed-limits or traffic. With the above observation, it seems that this assumption is flawed with regards to at least one user group. It would be interesting to see if it is possible to discover more such routes by monitoring the movement of users over time and then use popular streets or roadsegments more actively in optimization.

Another point offered related to the way routes are critiqued. In the current incarnation of the Ranger, critique is offered for an entire route at a time. This feedback is then disseminated down through the system until it lands on the individual edges in the road network. This lead users to find it difficult to point out individual problems with otherwise good routes. For example, one user found a route that was very similar to one he would have chosen himself, except for one particular road that was practically impossible for him to traverse. In reaction to this, the user wished to tell the system that this particular road in extremely bad, but that the rest of the route was good.

The user suggested a method to solve the above problem. If the user could draw on screen a possible route and tell the system that this is good, this could give the user some added alternatives

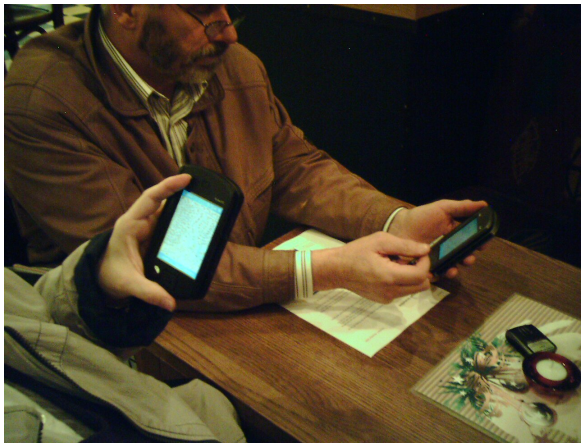


Figure 4.13: The users experimented with the application

for suggesting routes for his user group. An alternative solution could involve the user being allowed to edit the route given to him by dragging points or lines to better locations on the map. Such alterations would demand non-trivial amounts of reimplementation but remains interesting venues for further work.

Tight Geometry

As the users drew tracks on the screen, the geometry representing Halden started to become somewhat cluttered, and after a while errors crept in. Certain small areas of Halden would suddenly become impossible to search, and if the user engaged in some action that would cause the geometry in such areas to be loaded, the server code would return a fatal error. This error is traced back to loading degenerate edges which is probably caused by bugs in the track-integration code. This discovery highlighted the problem of discovering geometry-related bugs in clinical trials like the one performed prior to this meeting.

Specific critique and point-based feedback

The users also requested more specific methods for critiquing routes. As it stands, the users were only given the opportunity to assign a route a grading ranging from very bad to very good. The users expressed a desire to explain why a route was good or bad in addition to this grading. For example, one user wished to point out that a particular stretch of road was very steep which prevented him from traversing it.

The ability to comment on particular obstacles emerged as a need after some discussion around specifying the reasoning behind feedback. This ties into the original Okapi implementation that offered users the oppurtunity to create Points of Interest (POIs). While the original implementation defined a POI as simply a geographic point with associated media like images, text, or videos, it could be expanded to include relevant accessibility-data.

Finally, the users were very interested in factoring in height in the route planning. In the early stages of implementation, height was factored in, but as the data source turned out to be inconsistent in terms of whether or not height was included, it became too difficult to use in this prototype. However, given the possibility for a GPS to report its current height above the sea level, one might consider slowly growing a perception of height-differences based on tracking users as they move around.

Usability Issues

In addition to issues pertaining to route planning, the test uncovered a number of usability issues. I have included a short discussion of these, since usability remains an important topic of any development project, even though it is not something this thesis focuses on.

The primary usability issue the users talked about was the lack of a search-function. One user wondered about the usefulness of a system that planned a route between two point he already knew about. Instead of this function, she would have liked to see a method to search for particular adresses or businesses. Such functionality already exist to a small degree, but was turned off for this test, as it relies on a POI database. Such a database is available for Oslo, but not for Halden.

There were also concerns about the legibility of the map and particularly the markers that the Ranger place on the map to denote tracks, routes, starting points, and goal points. These things should be adressed before any future user tests, as they are relatively simple to fix, but very important to get right.

Finally, the action of drawing tracks on the screen seemed to confuse the users, both in terms of how to draw, but also in terms of why. Users seemed to think that drawing the track on the screen meant that you suggested a solution to the system, rather than suggesting entirely new roads. This confusion led to the suggestion for a new feature that I outlined in the previous section.

Other issues

During the conversation with the users, we discussed a number of issues surrounding the use of this system and similar systems. Of particular interest was the issue of privacy. According to Norwegian law, a person's health is considered sensitive information. For example, the member lists for the Norwegian Handicap Society is considered classified information and can not legally be published.

We also talked about the premisses of accessibility. The users came from a very active group of wheelchair users and had some fairly well grounded ideas about the issue, but they conceeded that other people had entirely different ideas. In addition, what may be considered accessible for one wheelchair user may be completely unusable for another. This means that the initial division of users into broad usergroups like ”manual wheelchair” and ”powered wheelchair” could end up being too broad a division.

I believe that the meeting with users was extremely fruitful. The users were given the opportunity to try the Ranger application, and although they never actually used the routes offered to them, they drew upon their knowledge of Halden to produce accurate feedback. They uncovered usability weaknesses that the project should adress in future iterations, and more importantly for this thesis, they pointed out important facts about how they move about in the city streets and how they would use an application such as the Ranger.

Once alterations have been made to the project codebase to address the issues raised in this meeting, it would be prudent to hold a more formal test where the users actually moved about using the routes. It would be interesting to first have people with knowledge of a city use the application for a while, and then introduce an outsider. This outsider will not know any of the city's good or bad streets, and would be entirely reliant on the system to offer good routes. In such a test, the metric would be the percieved quality of the routes from the perspective of the outsider. If the system does what it promises, the outsider should immediatly get fairly good routes and be relatively pleased.

4.3 Test Conclusions

The two separate tests performed on the Ranger, the system test and the informal user test, both showed that the system meets the requirements set out for it. The system successfully calculated and displayed routes on a mobile device, and it could be persuaded to prioritize certain stretches of road over other based on the user's professed user group and previous feedback offered by users in that group. This way, routes were given a subjective measure of quality that played into whether or not the route's particular road segments would be used in future route planning.

The system also allows for adding new roads to the network, thus giving users a truly collaborative way to build a road map for their own neighbourhoods or towns.

User testing uncovered a number of usability issues. Although these are not directly linked to this particular project, they are important issues to consider when looking at the Ranger from a wider perspective. In the future, more formal user testing should be performed to gain a more accurate impression of the system's actual usage.

Chapter 5

Findings and Future Work

The Ranger project aimed at creating a route planning application that offered good routes to different users with widely differing and often conflicting route planning needs. Furthermore, the application were to be available on mobile devices. Particularly, the project aimed to create an application to display maps and routes, a mechanism for generating routes, and a mechanism for measuring the quality of these routes in a way that related to the individual user.

The project sought to solve the last of these sub-problems by allowing users to partake in user-groups defined by some shared characteristic. The users would then be able to assign grades to routes. Routes that users found good would be prioritised in subsequent route planning activities initiated by the user or any member of the user's group, and this would guarantee that routes would be considered good by any user of the system.

The project drew inspiration from earlier work in collaboration, such as Wikipedia and Flickr, that has shown that user-contributions can lead to robust and useful content. It also owes a lot to the vast amount of previous work in route planning and graph search, especially to the research into user-feedback and collaboration in route planning that has preceeded this thesis. Of special note is the work done by Haigh et al.[16] on weighing roads according to user feedback, and that of McGinty and Smyth[32] on exchanging route planning experience between similar user agents.

Pedestrian route planning, like this application, has had fairly little attention in the Western world, with the honorable exception of a few German researchers. However, in East Asia, pedestrian route planning has matured into an industry, and much research on the subject has been done there. Especially in Tokyo, where many streets are unnamed and people use public transportation to a much higher degree than in the west, has pedestrian navigation tools found a market. This project has attempted to tap into some of the accumulated knowledge to adress issues relevant to the general

problem of personalizing routes.

Ultimately, I believe the project has been a success, although it does not represent a finished product. Testing showed that the concept was sound, although the implementation needed some refinement.

5.1 Project Results

The Ranger project sought to answer the question of whether it was technologically possible to create a mobile wayfinding application that delivered good solutions to a wide variety of users. These users would have differing motivations and needs and sometimes two users would have conflicting ideas about what constituded a good route. To answer this question, the project needed to solve three subproblems. It needed to:

- create a mobile application that could display maps and routes
- create a mechanism for generating routes
- create a mechanism for determining the subjective quality of a route

These three subgoals were met and to some degree exceeded in the Ranger application. In addition to creating a way to rate routes relative to the user, it also gave users the opportunity to create new routes where the system didn't already have a network.

The Ranger project resulted in a heavily modified Okapi application that added route planning capabilities to the original application and a server that kept track of the roadnetwork and whatever modifications users would make to it. The route planning capabilities came in two flavors: the user could either ask for a route between two arbitrary points, or from one arbitrary point to the nearest point of interest of some particular type to that point. For example, the user could ask for the nearest accessible toilet to his own position. This modified Okapi client met the first of the sub-goals, as it displayed routes and maps, and the server, through a standard A* implementation, met the second sub-goal.

More importantly, the application allowed users to grade these routes after their own experience with them. If a user found a route to be especially well suited, he could give the route a good grade. These grades were shared across users belonging to the same user-group and route planning efforts of the members of said group would be affected by the grading. Roads participating in routes that had predominantly been deemed good by users would be prioritised during route planning requested

by users in the relevant user group, while roads in bad tracks would be avoided. This way, the Ranger introduced a mechanism for subjectively grading routes to help in creating personalized routes.

The modification also added tracking options for the application. Users could track their own movements using a GPS, or they could manually enter new tracks by drawing on the screen. These tracks could be uploaded to the Ranger server and integrated into the already existing road network. This way, users of the Ranger could share tracks and collaboratively create a road-network that represented actual roads or paths that the users used in their day-to-day life. Coupled with the feedback to routes, users were given the ability to change the road network to reflect their personal perception of the world, which in return would result in route planning activities that gave personalised routes for each user.

5.1.1 Testing

To show that the application actually acheived the goals outlined above, I organized a meeting with some potential users. During the meeting, the users experimented with the application and discuss the results it gave them. The users were wheel-chair users which meant that they were benefitted by low curbs, plane and paved roads, and similiar features. On the other hand, they are disadvantaged by sidewalks with many pot-holes, gravel, dirt, grass, etc., or high steps from the sidewalk at places where they wish to cross the road. On account of their intimate knowledge of Halden's streets, they were able to use the feedback system of the application to try to affect the routes given them.

The informal test showed that the system adapted to the users's feedback, and tried to find alternative routes when the users were unsatisfied. However, the testing also uncovered some issues that needs to be addressed in any future work on the Ranger application. I have discussed these issues in some detail in section 4.2. In short, the meeting showed that the concept is sound and can be used to allow people to exchange experiences that makes route planning better. However, the feedback may need to be more nuanced and specified, as the users wanted to know more about why the system chose a particular route over others.

The technical tests of the system showed that there are some problems with the geometric analysis in integration of usertracks that hinder a smooth track-uploading experience. The problems are restricted to situations where the user uploads tracks in areas where there is already much geometry present. This leads me to conclude that the problem lies with the algorithm that simplifies and generalizes the user track prior to the actual integration. Until this problem is solved, the application can not be said to fully support user tracks. However, this problem does not appear in most cases of track uploading, so I expect to be able to fix the problem in future iterations of the project.

5.1.2 Open Data

As a side-effect of the system testing and the informal user testing, I had the oppurtunity to find out if the open data I got from the Open Street Maps (OSM) initiative was sufficient for routeplanning. As I expected, the data was sufficiently robust and detailed for me to perform block-by-block route planning even in a small town like Halden. The data was in such a shape that I only needed to translate it into the Ranger specific format to be able to use it. That is to say, no massaging of the data was necessesary and it could be used without even reprojecting the geographic coordinates, as the OSM project uses raw GPS coordinates for their tracks.

5.2 Future work

The Ranger project will be continued after the end of this thesis, and there are several directions it could take. First off, there are a number of usability issues and bugs to address before the application is release-worthy. Furthermore, discussions with users has resulted in a number of ideas for features that would be interesting to implement.

5.2.1 User-drawn Tracks

Naturally, the bug relating to tight geometry that lead to possibly corrupted or degenerate edges needs to be addressed. However, since the users didn't find the offending feature very interesting, it may be worthwhile to look at alternatives to the feature, rather than spending resources on fixing it.

Users that tried the user-track feature often misunderstood the purpose of it. They assumed that the user-track feature meant that the user would point out roads that they thought were of high quality. This way, the users thought they could suggest routes to the system and to their fellow users. Such a feature is in fact quite interesting and adds to the collaborative aspects of the Ranger application.

One way to implement this feature would be to allow the user to draw a track on the screen and then upload this to the server. The server could look at the uploaded track and try to find edges in the network that cover the same ground as closely as possible. The user-track could be fitted to a collection of edges and returned to the user for review and clearing. If the returned user-track follows the roads that the user intended, then he gives it a grading and concludes the track-upload. At this point the user-feedback would be spread along the relevant edges.

In the case that the returned user-track did not follow the intended road, the user could edit the

track by moving points or edges to the correct position. During the second upload, the system would create new edges for the parts of the track that it got wrong. This way the users could still create new edges to map out areas that lack a road network, but tracking in already mapped areas would still have a good use. The drawback to this solution is the added complexity of the user-operation, but this feature would be for experienced users.

Points of Interest

Another feature that is interesting to implement is the integration of points of interest (POIs) into the route planning. The current Ranger application allows the user to search for the closest POI of a particular kind. In the future, the user should be able to do free-text searches for POIs and then perform route planning to a POI of their choice from a search result-list.

However, POIs could be integrated even more tightly into the route planning process. If the users are given the opportunity to create special POIs designating location or areas as obstacles or features, such as access-ramps or areas with many potholes, this could be used to affect route planning. Certain POIs could give edges nearby bonuses while others could server to diminish an edge's value. The POIs would only affect certain usergroups and to varying degrees. For example, a POI denoting a tall step from the sidewalk into the street would be used to downplay a street whenever a wheelchair user performed route planning. On the other hand, the same POI would emphasize the street when a blind person asked for a route.

POIs need not be permanent fixtures in the system. For example, a user may want to document a road or a sidewalk that is currently undergoing work. This could make the road temporarily unusable and this should be a part of the route planning. However, the POI should not live on forever, as the road work eventually will end and the road will again become usable. In this case, users should be able to create POIs with expiry dates that only augments route planning during its lifetime. Such POIs could have their lifetimes extended if users find that the obstacle or path-feature doesn't go away when expected, or they could be removed if the obstacle disappears quickly.

Finally, POIs could become a part of route planning by assigning the POIs that people are interested in visiting, such as resturant or toilets, ratings that affect route planning efforts to reach it. This would mean that the A* algorithm that currently controls all route planning in the Ranger would have to be swapped out with some one-to-all algorithm that could take into account the relative values of the endpoints of the edges. With such an algorithm in place, the system could present the user with routes to locations where the quality of the end point would dictate the length of the route. That is, if the user had to walk a little bit further to get to a location with a much higher

rating, the system could suggest the longer route.

5.2.2 Miscellaneous Future Features

In addition to the suggestions above, the Ranger should look at a number of smaller features that should be added in the future. First off, the users should be allowed to create new usergroups if they cannot find one that suits them. This could be expanded to let a user create a profile with a number of preferred user groups that he could choose to use for route planning and feedback.

There is also the option of storing feedback on a person-to-person basis. This would mean that users could undo previous grading if they change their mind. It would also open for analysis on the feedback. For example, one could find people that agree a lot despite being in different usergroups, and this could in turn lead to more sophisticated exchange of experiences. Basically, a trust network could be created based on the feedback offered. However, such a change in the way to handle feedback might lead the project down the path to the Utility Problem, where the database is so full of feedback that the improvement in route planning is undone by the vast inefficiencies in applying the feedback.

Users should also be allowed to give on-the-spot feedback. There could for example be a ”Good” and a ”Bad” button on the map that lets the users say that ”where I am now is a bad place for my usergroup”. The system would look at the user’s location and apply the ad-hoc grade to the closest edge to the user’s position without giving any thought to any route that may have been calculated. Such a feature would allow more detailed feedback and avoid problems of the kind where a single bad decision causes a long and otherwise fine route to be labeled bad.

Another direction of development is looking at reusing the routes the Ranger creates. Currently, calculated paths are not kept in the system as anything other than reference for feedback. However, it is possible to use already calculated routes as elite-solutions for future searches. If a route from A to B has been found to be good, and a user asks for a route from somewhere close to A to somewhere close to B, it could be expedient of the system to first look at the old route between A and B. If this route was a good route, then it is likely that getting the user to point A, and then simply offering the old route rather than calculating a new route, is just as sensible as creating an entirely new route.

References

[1] Koen Aerts, Karel Maesen, and Anton Van Rompaey. A practical example of semantic interoperability of large-scale topographic databases using semantic web technologies. In *9th AGILE Conference on Geographic Information Science*, January 2006.

[2] James A. Anderson. *Discrete Mathematics with Combinatorics*. Prentice-Hall, Upper Saddle River, New Jersey, 2001.

[3] The Norwegian Mapping Authority. `http://www.universiell-utforming.miljo.no/file_upload/helekartet2s.pdf`. online.

[4] Jorg Baus, Antonio Kruger, and Wolfgang Wahlster. A resource-adaptive mobile navigation system. In *7th International Conference on Intelligent User Interfaces*, pages 15–22, May 2002.

[5] Martin Breunig and Wolfgang Baer. Database support for mobile route planning. *Computers, Environment and Urban Systems*, 28(6), 11 2004.

[6] Jeff de la Beaujardiere. Opengis web map server implementation specification. Technical report, Open Geospatial Consortium Inc, March 2006.

[7] A. Desilets, S. Paquet, and N. G. Vinson. Are wikis usable? In *2005 International Symposium on Wikis*. National Research Council of Canada, October 2005.

[8] Frederic Evennou, Francois Marx, and Emil Novakov. Map-aided indoor mobile positioning system using particle filter. *Wireless Communications and Networking Conference*, 4:2490–2494, March 2005.

[9] Ashwin Ram Jr. Anthony G. Francis. The utility problem in case-based reasoning. Technical report, College of Computing, Georgia Institute of Technology, 1993.

[10] L. Fu, D. Sun, and L. R. Rilett. Heuristic shortest path algorithms for transportation applications: State of the art. *Computers and Operations Research*, 33:3324–3343, November 2006.

[11] David Geer. The e911 dilemma. *Wireless Business and Tech*, pages 40–43, November 2001.

[12] Jennifer Golbeck and Bijan Parsia. Trusting claims from trusted sources:trust network based filtering of aggregated claims. *Int. J. Metadata, Semantics and Ontologies*, 1(1), 2006.

[13] Jennifer Gonzalez-Reinhart. Wiki and the wiki way: Beyond a knowledge management solution. *Information Systems Research Center*, pages 1–22, February 2005.

[14] GPSGames. Geodashing. online: http://geodashing.gpsgames.org/.

[15] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *IEEE Communications Sysrveys and Tutorials*, 3(4), September 2000.

[16] Karen Zita Haigh, Jonathan Richard Shewchuk, and Manuela M. Veloso. Exploiting domain geometry in analogical route planning. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:509–541, 1997. `http://www.cs.cmu.edu/~khaigh/papers/khaigh97d.abstact.html`.

[17] Susan M. Haller and Gene Simmons, editors. *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference, May 14-16, 2002, Pensacola Beach, Florida, USA*. AAAI Press, 2002.

[18] Torbjørn Halvorsen and Harald K. Jansson. Geographitti using mobile devices. Technical report, Østfold University College, May 2005.

[19] Mike Hazas and Andy Hopper. Broadband ultrasonic location systems for improved indoor positioning. *IEEE Transactions on Mobile Computing*, 05(5):536–547, 2006.

[20] Sean B. Hoar. Trends in cybercrime. *Criminal Justice*, 20(3):4–13, 2005.

[21] Claire Hujinen. Mobile tourism and mobile government. Technical report, EC/DC Infonomics, April 2006.

[22] S. J. Ingram, D. Harmer, and M. Quinlan. Ultrawideband indoor positioning systems and their use in emergencies. *PLANS 2004*, pages 706–715, April 2004.

[23] D. Karimanzira, P. Otto, and J. Wernstedt. Application of machine learning methods to route planning and navigation for disabled people. In *MIC'06: Proceedings of the 25th IASTED international conference on Modeling, indentification, and control*, pages 366–371, Anaheim, CA, USA, January 2006. ACTA Press.

[24] The Norwegian Mapping Authority (Statens Kartverk). Arealis wms : `http://www.statkart.no/arealis/`.

[25] KDDI. Kddi to launch ez navi walk, a full-scale navigation service for pedestrians. online, June 2003.

[26] Anil Kini and Joobin Choobineh. Trust in electronic commerce: Definition and theoretical considerations. In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, pages 51–61 volume 4, January 1998.

[27] Holger Kirchner, Bendick Mahleko, Mike Kelly, Reto Krummenacher, and Zhou Wang. eureauweb - an architecture for a european waterways networked information system. In *Conf. on Information and Communication Technologies in Tourism*, Wien, New York, January 2004. Springer.

[28] Josef Kolbitsch and Hermann Maurer. The transformation of the web: How emerging communities shape the information we consume. *Journal of Universial Computer Science*, 12(2):187–213, February 2006.

[29] Sue Long, Rob Kooper, Gregory D. Abowd, and Christopher G. Atkeson. Rapid prototyping of mobile context-aware applications: The cyberguide case study. In *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 97–107, New York, NY, USA, November 1996. ACM Press.

[30] Rainer Malaka and Alexander Zipf. Deep map - challenging it research in the framework of a tourist information system. In *Information and Communication Technologies in Tourism*, pages 15–27, January 2000.

[31] Open Street Map. Open street map wiki.

[32] Lorraine McGinty and Barry Smyth. Shared experiences in personalized route planning. In Haller and Simmons [17], pages 111–115.

[33] NASA. Onearth wms : http://onearth.jpl.nasa.gov/.

[34] Norkart. Tilgjengelighetsportalen kart.

[35] Dan W. Patterson. *Introduction to artificial intelligence and expert systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[36] Steve Rabin, editor. *AI Game Programming Wisdom 2*, volume Two. Charles River Media, first edition, December 2003.

[37] Eran Rippel, Aharon Bar-Gill, and Nahum Shimkin. Fast graph-search algorithms for general aviation flight trajectory generation. online: http://www.ee.technion.ac.il/people/shimkin/PREPRINTS/Rippel04final.pdf, May 2004.

[38] Seth Rogers and Pat Langley. Personalized driving route recommendations. In *Proceedings of the AAAI Workshop on Recommender Systems*, January 1998.

[39] Stuart Russel and Peter Norvig. *Artificial Intelligence, A Modern Approach*. Prentice-Hall, 2 edition, 2003.

[40] Barbara Schmidt-Belz, Achim Nick, Stefan Poslad, and Alex Zipf. Personalized and location-based mobile tourism service. online.

[41] The SOSI Secretariat. Sosi. Online: http://www.statkart.no/standard/sosi/html/welcome.htm.

[42] The SOSI Secretariate. The sosi standard, 1987.

[43] Monika Sester. Application dependent generalization - the case of pedestrian navigation. In *Joint International Symposium on "GeoSpatial Theory, Processing and Applications"*, Ottawa, Canada, July 2002. ISPRS/Commision IV, SDH2002.

[44] Barry Smyth and Padraig Cunningham. The utility problem analysed: A case-based reasoning perspective. In *EWCBR*, pages 392–399, November 1996.

[45] European Commision Information Society and Media. ecall factsheet. online, May 2006.

[46] Steven L. Tanimoto. *The elements of artificial intelligence: an introduction using LISP*. Computer Science Press, Inc., New York, NY, USA, 1987.

[47] Telecom Tribune. Ntt docomo to launch navigation service for pedestrians using enhanced gps technology. *Telecom Tribune*, 14(10):2 – 2, January 2000.

[48] Panagiotis A. Vretanos. Web featture service implementation specification. Technical report, Open Geospatial Consortium Inc, May 2005.

[49] Caroly Wei, Brandon Maust, Jennifer Barrick, Elisabeth Cuddihy, and Jan H. Spyridakis. Wikis for supporting distributed collaborative writing. In *Proceedings of the Society for Technical Communcation 52nd Annual Conference*. Society for Technical Communcation, May 2005.

[50] Wikipedia. Wikipedia -the free encyclopedia. online, 2006.

[51] V. Zeimpekis, R. Alvarez, R. Tafazolli, and B. G. Evans. Impact of constellation design on doppler-rate based mt positioning for satellite-umts. In *20th AIAA International Communication Satellite Systems Conferenec*. AIAA 2002-2010, January 2002.

List of Figures

List of Tables

Appendix A

Implementation Details

A.1 Client Architecture

The Ranger client is charged with a number of tasks. First, it must provide a user interface for requesting routes from the server. It must be able to display these routes to the user, and it must provide a way for the user to critique these routes. Finally, it must provide two ways for the user to draw and submit new usertracks.

The module will appear as an extention to the Okapi code, and is contained in a single namespace. The Okapi Framework will access the Ranger through a single class called Ranger that provides two menu items that the framework adds to its own menu as shown in figure A.1. These menus will give the user access to all the functionality of the project.

The full class diagram for the client side code is seen in figure A.2. Please refer to this figure for the following description of the individual classes.

Ranger

The Ranger is the entry class for the project's client side code. It provides a menu-item that serves as access to deeper menues with functionality that the user needs.

The object will draw source and target markers on the screen if the user has placed these.

When the user chooses to search for a route, it is the Ranger that builds a request and passes this off to the server. The response is validated using .NET's prebuilt XML verifiers and the Ranger creates a new instance of the Track class provided the XML is correct.

The Ranger object is also responsible for maintaining control of all the instances of the Track

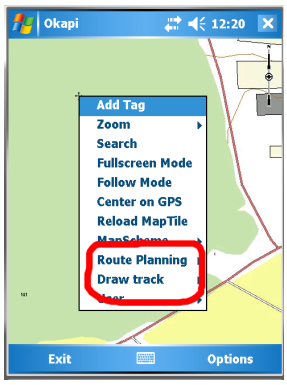


Figure A.1: The Ranger provides two new menu items in the Okapi context menu.

class that are currently alive in the system and for passing draw-commands to these when needed.

Track

A Track instance is a route from one position to another. It consists of an ordered list of SimplePoint instances in addition to a number of meta-data attributes like name and score. The score of a track is an approximate measure of the length and quality of the road, where a low score is better than a high score.

Each instance uses a reference to the OkapiCore to draw itself on the screen. The reference to OkapiCore is needed to translate the track's points which are in geographic coordinates to screen coordinates.

A Track can be selected by clicking on the track's image on the screen. If another Track is selected already, that track will be deselected while the track clicked on will be selected. If the Track that is clicked is already selected, then it becomes unselected and no track will be selected.

EditableTrack

The EditableTrack inherits from the Track and uses MovablePoint instances rather than SimplePoint instances. The EditableTrack represents a track the user has drawn either using a GPS or by placing

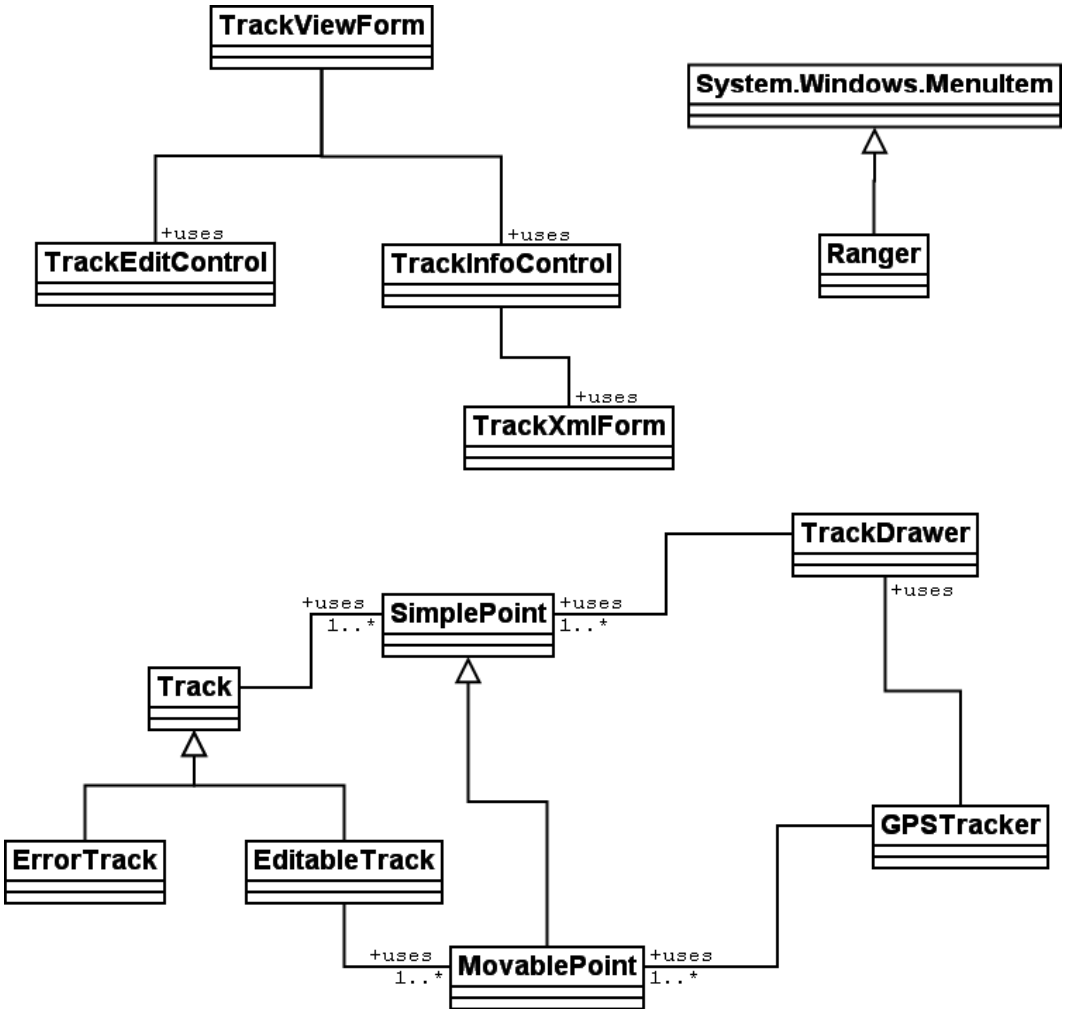


Figure A.2: The client side architecture. Some connections are left out for readability.

points on the screen directly. The EditableTrack class is the application's equivalent of the usertrack mentioned throughout this thesis.

ErrorTrack

The ErrorTrack is a special subtype of Track. It does not draw itself to the screen and instead of having a name or a score, it contains an error message. The ErrorTrack is instantiated when the server returns an error when the client expected a track. Each ErrorTrack instance is accessible through the Ranger and one can read the message using the TrackXMLForm through the TrackViewForm.

SimplePoint

The SimplePoint is a point in two dimensional geographic space. It has a latitude and a longitude and accessors to read these.

MovablePoint

The MovablePoint inherits from the SimplePoint and introduces methods for changing the position of the point. This allows the user to edit tracks by both placing points on the screen and subsequently moving them to improve the precision of the track.

TrackDrawer

This class runs parallell with the Ranger class and is also presented to the user as a MenuItem. It opens alternatives for the user to draw tracks on the screen, either by adding a single point at the location of the pointer, or by turning on a GPS attached to the device.

GPSTracker

The GPSTracker is a wrapper class that, when activated, polls Okapi's GPS namespace for a location once every 20th second. These locations are added to any existing EditableTrack instance through the TrackDrawer services. This class offers no services beyond starting and stopping the polling.

TrackViewForm

The TrackViewForm is a window that contains information about the tracks currently alive on the client. If a Track is selected when this form is created, it will contain a single TrackEditControl for

the selected Track. Otherwise, the form will contain one TrackInfoControl for each track including ErrorTracks.

TrackXmlForm

The TrackXMLForm is a simple form that displays the XML source of a track. If the track did not come with an XML source, f.ex. it has been drawn by the user on the client, this must be controlled for prior to creating this form.

TrackEditControl

The TrackEditControl inherits from the Control class in the .NET framework. It presents the user with the option of renaming a track, and giving it a score. The scores are presented as fuzzy terms ('Very Good', 'Good', 'Ok', 'Bad', and 'Terrible'). These terms are translated to a floating point score and uploaded to the server when the user chooses to do so.

TrackInfoControl

The TrackInfoControl presents information about a single Track. The information presented is the name and identification of the Track, along with its relative score and the number of points needed to draw the track.

A.2 Server Architecture

The server side of the Ranger should have been a Java Servlet, but due to restrictions on the operating system available, it had to be a standalone Java application. This causes some concerns over efficiency and resource use and in a production version of the project the Ranger would have appeared as a Servlet on a dedicated server. As it stands, the server side of the Ranger project exposes only-PHP scripts to the World Wide Web, so the application is launched through one such. The application accesses the same MySQL database that the Okapi Framework uses, although some additional tables have been inserted (see section A.3). Figure A.3 shows the class diagram for the server.

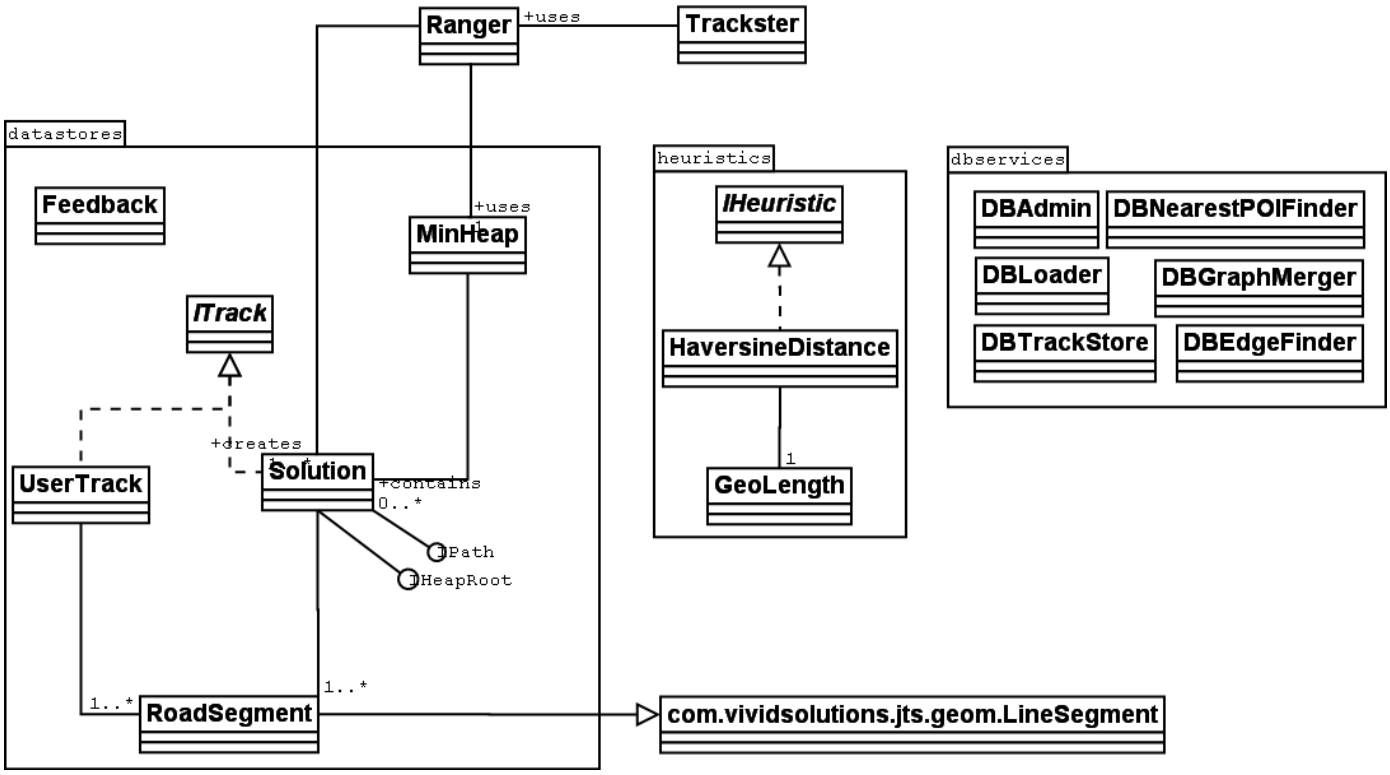


Figure A.3: The server side architecture. Many connections are left out for readability.

Trackster

The Trackster is the entry class for the server side of the Ranger. This object receives arguments from a PHP-script that intercepts the client's requests.

It is responsible for parsing and validating the client's arguments and responding with appropriate messages. If some of the arguments are invalid, the Trackster will respond with an error message. Otherwise, it will start a process to answer the client's request.

If the client has asked for a route between two points, the Trackster will use the Ranger directly to find this route. The route will be in terms of an instance of the Solution class which contains methods to be represented as XML according to the appropriate schema in appendix B.

In the case where the client asked for the closest point of interest, the Trackster will make use of the DBNearestPOIFinder to find the target location. Once this is found, the Ranger is used in the same way as before.

The client may also submit a new track. When this happens, an instance of DBGraphMerger is created and passed the relevant data. When the DBGraphMerger returns, the Trackster outputs the resulting track to the client.

Finally, the Trackster accepts feedback requests. These requests use the DBAdmin directly to update the feedback values of the edges of the specified track.

Regardless of the request, as long as the Oslo graph needs to be loaded, the Trackster makes sure this is done by calling the DBLoader.

Ranger

The Ranger class is the serverside counterpart to the client's class with the same name. It is solely responsible for finding an optimal path from a provided start-node to a provided end-node. It searches through the graph representing the road network by discovering each node's children and traversing these.

The Ranger implements the A* algorithm and uses an instance of the MinHeap to store Solution instances in a priority queue. Once the search has completed, the Ranger returns an instance of the Solution class.

Feedback

Feedback is an intermediary data-storage class. It is used by the DBGraphMerger to keep temporary copies of feedback given a particular edge. Each instance of the Feedback class represents a single

row in the database's table wf_feedback. The object provides the necessary methods for recreating this feedback for new edges.

ITrack

This is an interface that encapsulates the functionality of a track on the server. The ITrack proscribes only two methods for generating XML from a track. This allows the system to treat error messages from within the route planning mechanisms as tracks all the way to publishing to the client.

IPath

An object that implements the IPath interface can be considered a track from some start point to the current last point in the path. The services an object must render in order to fulfill this contract involves enqueueing nodes and retrieving the nodes in the order of entry. It is also necessary for an IPath to be able to create a clone of itself with an additional node at the end of the queue.

IHeapRoot

The IHeapRoot is an interface that an object needs to implement if it is to be used by the MinHeap. In order to implement this interface, an object must maintain a concept of cost and have a method to access this cost.

MinHeap

The MinHeap is a central class for the graph searching part of the route planning algorithm. It is created by a Ranger instance and it maintains a collection of IHeapRoots in an array organized as a minheap. These IHeapRoots are then accessed by the Ranger instance during a search for an optimal route.

The MinHeap contains methods for inserting new values, removing roots, and peeking at roots for non-destructive inspection of the heap.

By using an array as basic storage, the MinHeap ensures an as efficiently as possible insertion, although in the cases where the array has filled up, there will be some overhead in creating a new, larger array.

UserTrack

A UserTrack instance is represents the results of a user uploading a new track from the client. This class implements the ITrack interface, but does not contain any other functionality. It is instantiated when the DBGraphMerger has successfully inserted a user's track into the database and is meant to replace the drawn track on the client.

Solution

The Solution class defines a path that can be used by the Ranger class to search for an optimal path between two coordinates. The class implements a number of interfaces to achieve this.

It implements ITrack because it needs to be able to be published to the client as XML. The ITrack interface represents the only way the client and server can communicate geographic information between eachother.

The class also implements IHeapRoot because it is meant to be stored in a MinHeap instance during the search. During searching, several Solution instances are created as the Ranger probes the network for an optimal path.

Finally, the Solution implements the IPath interface. This interface allows the Solution object to have nodes added to it to modify the track. It also gives the system the oppertunity to clone the solution when it needs to explore a branching of the road-graph.

RoadSegment

The RoadSegment represents a single, unbroken stretch of road or other traversible ground, like a footpath. It maintains a cost variable that desribes the cost of traversing this particular stretch of road.

It also has a β value that depends on the current user. This β value is the relative quality of the road as determined by the collective input from the user group of the user that created this particular search. If the road is loaded from the database without a user present, or if the user's user group has never given any feedback to this particular road, the β value is set to the default of one.

The RoadSegment has methods and variables for keeping and maintaining a name, but this is for future expansion, when the roadnames are known.

The RoadSegment extends the JTS Topology Suite class LineSegment and inherits many useful geoemtric methods from there. Another method related to geometry is the splitting of a RoadSegment. When the DBGraphMerger inserts a new track, it may be necessary to split a RoadSegment

when a track crosses an already existing road. The RoadSegment offers methods to perform such a split, given a coordinate to perform the split on.

IHeuristic

The IHeuristic interface proscribes a means to guess the cost of travelling between two nodes or coordinates. The interface is provided so that it is easy to change one heuristic for another for testing purposes.

HaversineDistance

The only heuristic implemented in the Ranger is the HaversineDistance. By making use of the convenience class GeoLength, it calculates the line-of-flight distance between two coordinates or nodes. The line-of-flight distance is not a particularly accurate heuristic, but it is guaranteed to be admissable.

GeoLength

The GeoLength class is a very simple convenience class. It calculates the length in meters between two geographic decimal coordinates using the Haversine formula for a distance between two points on a sphere. It relies on the radius of the earth at a particular latitude, and is therefore not completely accurate. However, it is more than accurate enough for measuring distances within a single degree latitude and longitude, which is what this project is concerned with.

DBAdmin

The DBAdmin is the entry class to the Okapi database. This class handles all connections to the database and executes any query or update to this database. Each instance of this class represents a single query or batch job, and maintains any error messages that may have been generated.

DBNearestPOIFinder

By instantiating the DBAdmin, this class searches the database for the closest Point of Interest with the appropriate attributes to a given position. This class is used by the Trackster when the client asks for a route to the nearest accessible toilet, nearest accessible parking, etc.

DBLoader

The DBLoader fetches all the relevant edges from the database, and applies the feedback that is relevant for the current user to each edge. It then uses the JTS Topology Suite to turn the collection of edges into a traversible graph. This class is used by the Trackster prior to any operation that needs to search or manipulate the road network.

DBGraphMerger

The DBGraphMerger is one of the most involved classes on the server. It is responsible for updating the road network to include tracks created by users. These tracks may coincide with already existing roads, or they may cross such roads. The DBGraphMerger must alter both the user's new track and the system's old network to create a graph representation of the road network that allows subsequent route planning sessions to use the new track in searching for a good route.

DBTrackStore

The DBTrackStore is a simple class that is responsible for storing a track that has been generated by a Ranger instance. This track get stored in the database together with a few pieces of meta-data, like the time and date of creation and a track id.

DBEdgeFinder

DBEdgeFinder is another convenience class that searches the database for the edge that has the closest point to a particular coordinate. It is used prior to route planning in order to find a suitable starting and ending point. Given that the user may select both start and end at positions not represented in the network, this class tries to approximate the user's request by finding a place on the network that is as close as possible to the user's request.

A.3 Server Database

The data for the route planning will be stored entirely on a central server. This will ensure that every client will have access to the latest version of the network at all times. The database will only be accessed by the Ranger server code.

The Ranger introduces a number of new tables to the original Okapi database. It does not alter the tables that are already there, but it does reference the old database.

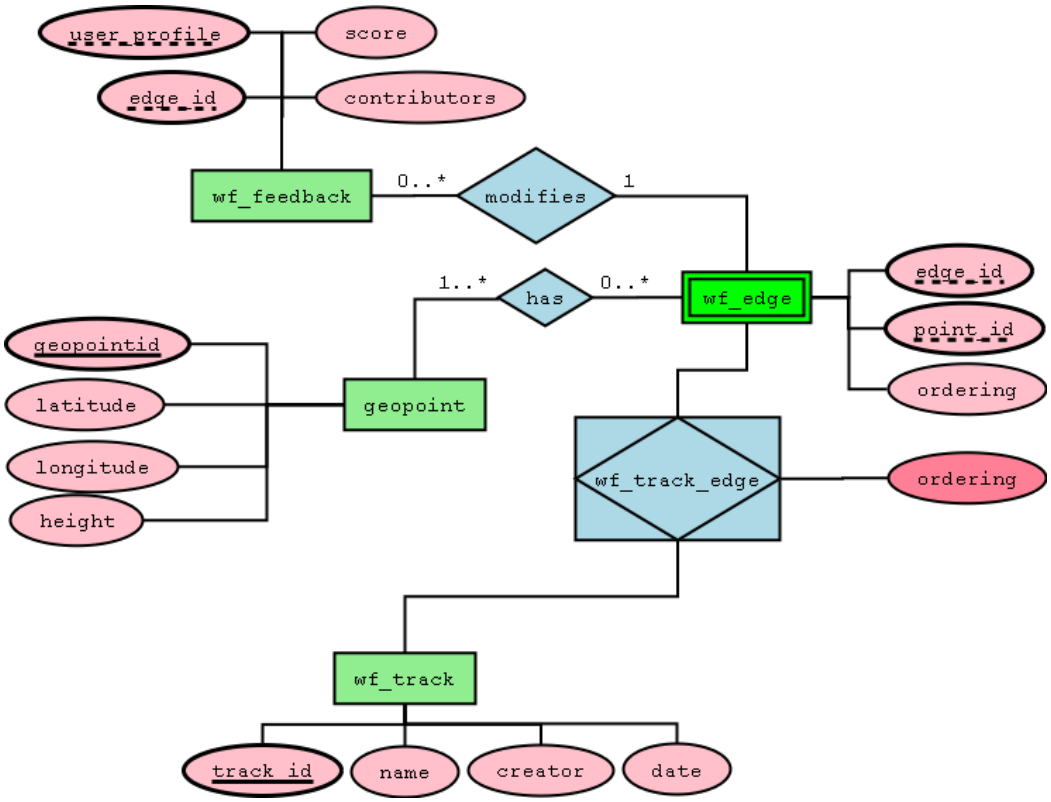


Figure A.4: An Entity Relation Diagram for the Rangerpart of the Okapi database.

The Ranger database additions are designed to keep records of the tracks calculated for users. Furthermore, the database must maintain any and all feedback users may give to the individual tracks. Finally, the database keeps the entire road network in store, so that the Ranger may search for routes.

See figure A.4 for an overview of the additions the Ranger has made to the Okapi database.

Geopoint

The geopoint is the smallest geometric unit in the Ranger road network. It represents a single point in three dimensional space, and has latitude, longitude, and height, in addition to its primary key. In the current iteration of the project, only the geopoint's latitude and longitude are used. The height

attribute is kept for future use.

Wf_edge

The wf_edge is a single, unbroken stretch or road or other traversible ground. It is uniquely identified by its edge_id, but the entity is weakly defined. The table specifies a unique constraint over the edge_id and the point_id combined. That is, edge_id may be repeated in the table, but it must have a different point_id each time. Each edge/point pair is also associated with an ordering which describes where on the edge a particular point is placed.

Wf_track

This entity describes a single track. This track may have been uploaded by a user, or it was created by the system in response to a request for a route.

The track_id is an autogenerated primary key.

The name of the track is given by a user or generated randomly by the system. This name is not currently used for anything, but as a suggestion to future functionality, tracks could be published to the user with name and date if the user asks for a route that someone else has already found.

The creator is the unique identity of the user that either uploaded the track or asked for it to be created. This identity is found in the original Okapi database.

The date is a simple SQL datestamp for the moment in time that the track was first created.

Wf_track_edge

This associative entity binds the tracks and edges together. Each track consists of at least one edge, and there is no upper limit to the number of edges to each track. Each edge may also participate in any number of tracks.

The only attribute to this entity that is not inherited from either wf_track or wf_edge is ordering. This attribute describes in which order the edges are to be placed on the track. Since tracks are bi-directional, one can traverse the track in both ascending and descending order.

Wf_feedback

The feedback entity represents an average of feedback given to a particular edge by a particular user group. The entity is uniquely identified by the key-pair user_profile and edge_id. The user_profile

is a foreign key from the original Okapi database, while the edge_id comes from the local wf_edge entity.

Each entity has a score and a contributors count. The contributors count helps calculating the change that is to be made the score attribute whenever a user offers some feedback.

An alternative way of representing feedback would be to create a new entry every time a user offered feedback, and then calculating the score run time. This would offer opportunities for more detailed analysis of user feedback.

A note on normalization

The Okapi database and the Ranger additions reside on a MySQL 5.0 server. Due to the original setup of this server, I have found that the job of enforcing referential integrity is placed with the programmer rather than the database management system where it should be. This has made the task of designing the database somewhat less formal than I would have liked, and my design reflects this. The Rangerdatabase additions have not gone through any normalization steps and they contain some design choices that deviate from good practice.

The most glaring of these short-cuts is the wf_edge entity. This entity is defined not by a unique key present on only one row in the table. Instead, it is implicitly defined by an edge identifier that is repeated in the wf_edge table once for every unique geopoint that belongs to the edge. This way, the wf_edge entity becomes its own associative entity. A better solution would be to split the wf_edge table in two where one table uniquely defined each edge and the other defined the relationship between the edges and the geopoints.

Appendix B

Ranger messages and XML Schemas

The messaging from server to client in the Ranger module is purely based on XML. Each message must conform to one of several XML-Schemas. These schemas are available on the Okapi server.

B.1 Error Message

The error message is a simple, yet pervasive message in the Ranger system. It is used in every situation where the server must tell the client that something has gone wrong.

Refer to figure B.1 for the XML Schema and figure B.2 for an example of an error message.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="error_message">
    <xs:all>
      <xs:element name="code" type="xs:integer" />
      <xs:element name="message" type="xs:string" />
    </xs:all>
  </xs:complexType>
</xs:schema>
```

Figure B.1: The Error Message Schema

```
<?xml version="1.0"? encoding="UTF-8">
<wayfinder:error_message xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"
 xsi:schemaLocation="http://platypus.hiof.no/okapi/
server/xsd error_message.xsd"
   xmlns:wayfinder="http://platypus.hiof.no/okapi/server/xsd">
  <code>
    0
  </code>
  <message>
   The operation succeeded.
  </message>
</wayfinder:error_message>
```

Figure B.2: A typical error message

B.2 Track

The Track message is a description of a single track from the database. The message should contain enough information for the track to be drawn correctly, and for the client to be able to refer to the track when talking to the server. This means that it needs to contain a unique identifier, and an ordered list of points. The ordering is done by element $pointid$ and element $parent$ that refers to the id. If the $parent$ element is empty or not present, the point is considered the first point on the track. If several points lack parents, the xml is erroneous and the client should display an error message. The $pointid$ is generated on the spot by the server.

The track is presented with a score, which is the relative score of the track.

Refer to B.3 for the XML Schema. Figure B.4 shows what a typical track might look like.

B.3 Track with POIs

Sometimes it is usefull to provide a list of POIs together with a track. When this is done, a Track/POI message is sent to the client. This message is a combination of the Track message and the POI message[1].

Refer to figures B.5 and B.6 for the XML Schema and figure **??** for an example of a track with associated POIs.

[1]The POI message is defined in the Okapi framework.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://platypus.hiof.no/okapi/server/xsd">
<xs:element name="track">
  <xs:complexType>
    <xs:all>
      <xs:element name="trackname" type="xs:string" />
      <xs:element name="trackid" type="xs:integer" />
      <xs:element name="score" type="xs:decimal" />
      <xs:element name="pointlist">
        <xs:complexType>
          <xs:choice>
            <xs:element name="point" maxOccurs="unbounded">
              <xs:complexType>
                <xs:all>
                  <xs:element name="pointid" type="xs:integer" />
                  <xs:element name="latitude" type="xs:decimal" />
                  <xs:element name="longitude" type="xs:decimal" />
                  <xs:element name="parent" type="xs:integer" minOccurs="0'
                </xs:all>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Figure B.3: The Track Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<track
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://platypus.hiof.no/okapi/server/xsd track.xsd">
  <trackname>Road to School</trackname>
  <trackid>2</trackid>
  <score>0.27</score>
  <pointlist>
    <point>
      <pointid>1</pointid>
      <latitude>59.12</latitude>
      <longitude>11.12</longitude>
    </point>
    <point>
      <pointid>3</pointid>
      <latitude>59.13</latitude>
      <longitude>11.12</longitude>
      <parent>1</parent>
    </point>
    <point>
      <pointid>54</pointid>
      <latitude>59.13</latitude>
      <longitude>11.13</longitude>
      <parent>3</parent>
    </point>
  </pointlist>
</track>
```

Figure B.4: A typical track

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://platypus.hiof.no/okapi/server/xsd">

<xs:element name="track_poi">
  <xs:complexType>
    <xs:all>
      <xs:element name="track">
        <xs:complexType>
          <xs:all>
            <xs:element name="trackname" type="xs:string" />
            <xs:element name="trackid" type="xs:integer" />
            <xs:element name="score" type="xs:decimal" />
            <xs:element name="pointlist">
            <xs:complexType>
            <xs:choice>
              <xs:element name="point" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:all>
                    <xs:element name="pointid" type="xs:integer" />
                    <xs:element name="latitude" type="xs:decimal" />
                    <xs:element name="longitude" type="xs:decimal" />
         <xs:element name="parent" type="xs:integer" minOccurs="0" />
                  </xs:all>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
      </xs:all>
    </xs:complexType>
  </xs:element>
```

Figure B.5: The Track Schema, part one

```
<xs:element name="poi_list">
    <xs:complexType>
      <xs:choice>
        <xs:element name="poi" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:all>
              <xs:element name="id" type="xs:integer" />
              <xs:element name="title" type="xs:string" />
              <xs:element name="description" type="xs:string" />
              <xs:element name="category_id" type="xs:integer" />
              <xs:element name="latitude" type="xs:decimal" />
              <xs:element name="longitude" type="xs:decimal" />
              <xs:element name="height" type="xs:decimal" />
              <xs:element name="owner"/>
              <xs:element name="datetime" type="xs:dateTime" />
              <xs:element name="expire_date" type="xs:dateTime" />
              <xs:element name="icon" type="xs:string" />
              <xs:element name="media_list" minOccurs="0">
              <xs:complexType>
                  <xs:choice>
       <xs:element name="media_item" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:all>
                        <xs:element name="poiid" type="xs:integer" />
                        <xs:element name="link" type="xs:string" />
                        <xs:element name="small_file_link" type="xs:string" />
                        <xs:element name="type" type="xs:string" />
                      </xs:all>
                    </xs:complexType>
                  </xs:element>
                </xs:choice>
              </xs:complexType>
            </xs:element>
          </xs:all>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
 </xs:element>
</xs:all>
</xs:complexType>
</xs:element>
</xs:schema>
```

Figure B.6: The Track Schema, part two