Pathfinding for Persons with Reduced Mobility - Comparing the performance of different routing-algorithms given various physical and environmental restrictions

Final Report for CSM6960 Major Project

Author: Jostein Kristiansen (jok13@aber.ac.uk)
Supervisor: Dr. Myra Scott Wilson (mxw@aber.ac.uk)

September 24, 2016
Version: 1.1 (draft)

This report was submitted as partial fulfilment of a MSc degree in Intelligent Systems (G496)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that this work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signature ..... (candidate)

Date .....

Statement

This work is the result of my own investigations, except where otherwise stated. **Where *correction services** have been used, the extent and nature of the correction is marked in a footnote(s).
Other sources are acknowledged(e.g. by footnotes and explicit references).
A bibliography is appended.

Signature ..... (candidate)

Date .....

[*this refers to the extent to which the text has been corrected by others]

Consent to share this work

In signing below, I hereby give consent for my work, if accepted, to be available for photocopying and for interlibrary loan, and for the title and summary to be made available to outside organisations.

Signature ..... (candidate)

Date .....

Acknowledgements

I would like to thank my supervisor, Dr. Myra Wilson, for her invaluable technical and moral support during the duration of this project.

I would also like to thank Dr. Edel Sherratt for her continuous support during the entirety of my MSc degree. She has helped me resolve a wide array of issues – many of which were not insignificant.

I would also like to thank Stefan Klaus MSc for helping me structure this document and make sure that it complies with the requirements set for MSc dissertations.

And finally: I would like to thank Dr. Angharad Shaw for visiting my high school back in Norway and recruiting me to Aberystwyth University. She has not been directly involved with this project, but as she is the reason why I applied for my BSc, I recognise that I would not be where I am today without her.

Abstract

Route-planning applications are designed to help their users find good paths between two or more locations on a map. These systems are useful to a wide variety of people – like tourists travelling to unfamiliar areas, or companies wanting to minimise fuel-costs and/or travel-time for their delivery vehicles. Route-planning can even be a useful tool on planes and boats, as it can help guide vessels into areas where the forces of nature give less resistance, or away from restricted/dangerous areas, but that is outside the scope of this project.

Conventional route-planning software is often aimed at one or more large groups of specific users – like motorists (commercial and/or private), pedestrians, cyclists, etc. But very few of these systems are able to plan good – or even practical – routes for Persons with Reduced Mobility (*PRM*), as these users are often simply grouped together with pedestrians or cyclists; with no special consideration taken with respect to their physical limitations.

Built-up areas pose particularly difficult environments to navigate for PRMs, as many commonly encountered obstacles like stairs, non-automatic doors, curbs, and steep slopes are effectively impassable for people with certain physical limitations. Without the aid of a route-planner, PRMs would need to rethink their planned route to a location on the fly whenever they encounter an obstacle, which can often prove quite frustrating, as an accessible path to where they want to go may not even exist.

The route-planning application described in this paper tries to address the aforementioned issues by identifying and avoiding inaccessible areas, and using accessible buildings as shortcuts in an attempt to shorten the route returned to the user. A number of different pathfinding algorithms have been tested, each of which has its own advantages and disadvantages.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction, Background & Objectives

1.1 Introduction

NOTES: - DELETE THIS -

- Route-planning applications are widely used.
- PRMs not considered by most route-planners.
- PRMs often find navigation hard (reference).
- More PRMs in university (reference).
- Aberystwyth University wants to make its campuses more accessible - ramps, automatic doors, etc.
- Project builds upon BSc project (2015) and is inspired by Access Aber (summer 2014).

Route planning has been a useful tool for thousands of years; even before we started drawing maps, and routes were planned based on landmarks and star-constellations. In our modern society, route-planning systems are becoming increasingly more accurate and commonplace in our daily lives. These systems can be used by anyone: from tourists wanting to find the nearest toilet, to the Mars rover **(reference)** boldly going where no one has gone before.

A significant amount of money and effort has gone into accurately mapping our world **(reference: OSM+Google/keyhole?)**, which in turn provides route-planning systems even more data to work with – making them more accurate and useful than ever before.

Route-planning systems are of course only practical to use if they are able to plan routes that their users are able to follow, which is unfortunately often not the case for persons with reduced mobility (PRM). Most public route-planning systems focus on providing routing-services to very large, generalised groups of people - like pedestrians, cyclists and cars, but very rarely consider PRMs as a separate group.

PRMs are often categorised as pedestrians, and are therefore given routes intended for people who are able to walk up stairs, cross streets with elevated curbs, climb steep hills, etc. Many PRMs are able to pass these obstacles with some assistance from someone else, but this does not apply

Figure 1.1: This image shows which path some PRMs would need to take to get from the computer-science department (*IMPACS*) at Penglais campus, to the Students' Union. As there are many stairs in the way, many PRMs are forced to follow a far longer route than an able-bodied person would.

to everyone; their wheelchair might be too heavy to move manually, or there might not be anyone around to give them the help they need.

A PRM could try following routes intended for cyclists, as these usually avoid stairs, but they might still face tall curbs, paths with bad surfaces, steep slopes, etc. Bike-routes also tend to be noticeably longer than pedestrian routes, so these may not always be suitable for many PRMs either.

As built-up areas are particularly challenging environments to navigate for PRMs, the project developed as part of this dissertation has put most of its focus on areas like this. Additionally: as Aberystwyth University has made a notable effort to make its campuses more accessible for everyone in recent years, and because there are more PRMs in university now than ever before **(reference)**, this project has put its main focus on the Aberystwyth University campuses.

This project builds upon the author's Major project for their BSc degree at Aberystwyth University, completed in 2015 ("Access Aber - Pathfinding"), which in turn was inspired by a separate system developed by Aberystwyth University in the summer of 2014: "Access Aber".

1.2 Project aims

NOTES: - DELETE THIS -

- Make it clear that PRMs need to be considered by more route-planners.

- Find optimal and complete algorithm(s).

- Algorithm(s) have to plan routes in near-real-time.
- Algorithm(s) have to use as little memory as possible.
- Algorithm(s) needs to be able to distinguish between accessible and inaccessible paths.
- Make a process for automatically filtering out Nodes and Ways deemed unsuitable for navigation.

- Investigate how to store, represent and index the OpenStreetMap database containing all of the Nodes and Ways to be used for pathfinding.
- Display the routes on a map. (is this really a project aim?).

The goal of this project has been to shine a light on how challenging it can be for PRMs to navigate built-up areas compared to other groups of people – with a focus on the Aberystwyth University campuses. PRMs are often grouped together with able-bodied pedestrians or cyclists in many route-planning systems, which means that the routes returned to them are often sub-optimal or sometimes even impossible to physically follow.

In order to properly show the difference between traditional route-planners for pedestrians and a route-planner made for PRMs, there has to be a programming-aspect to this project. The system developed as part of this dissertation has to identify the most important aspects of route-planning, and replicate this in a route-planning system made specifically for PRMs. These aspects include, but are not limited to: calculating routes in near-real-time, having a low memory-footprint, employing optimal and complete routing-algorithms, and of course finding appropriate routes for the target users.

The process of extensive and accurate mapping of Nodes (or individual points of navigational data) is a task beyond the scope of this project, so this data will have to be retrieved from a third party. This will be discussed in more detail later in this report.

1.3 Objectives

Chapter 2

Design

2.1 Overall Architecture

NOTES: - DELETE THIS -

- Tried to make it easy to add new Algorithms or data from different sources. Map tiles and OSM data stored locally - this is not ideal; compromise to ensure stability. 3rd parties online may take their servers down or make changes to their service. Local copy is always the same - no matter where the system is run.
- create more sub-sections here? That might look better.

The system reads a provided *.osm* file [22], and separates its contents into two Arrays: Nodes and Ways. This data is then put through a filter to remove any islands (or Nodes/Ways without any connections), in addition to all Nodes/Ways deemed unsuitable for navigation for PRMs.

The *.osm* file also contains information about "Relations", but as these represent abstract areas like the outline of a forest or housing-block, they are not useful for pathfinding, and can therefore safely be ignored by the system.[1]

A Node is a single point on the map – defined by its Latitude and Longitude coordinates. Nodes can be isolated points of interest like statues or monuments, or they can be part of one or more Ways representing larger structures.

Ways contain an ordered list of Nodes, each of which define its shape, and direction. The ordering of Nodes inside the Way is important in cases where a degree of inclination is indicated, or whenever a route passes through a one-way street. If the ordering of Nodes is not respected, then a Way's degrees of inclination may get flipped from positive to negative, or a route may become illegal to follow if it goes the wrong way on a one-way street. See Figure 2.2 for an illustration of how Nodes and ways may be represented, and Figure 2.1 for an illustration of how Ways may be expanded.

Nodes shared between two or more Ways can be thought of as intersections between those Ways. Nodes that are shared between multiple Ways are called "Tower Nodes", while unshared Nodes are called "Pillar Nodes".

[1] "Relations" also represent things like bus-routes, which could make a route-planner aimed at PRMs even more useful if considered – but this will be touched on later in this report.

Only Tower Nodes are expanded when planning routes, as this speeds up searches, and reduces memory-requirements significantly. Some Ways can contain more than 100 Nodes, but may only have 2-3 Tower Nodes. By ignoring the large majority of a Way's Nodes when planning routes, searches are sped up significantly, and less memory has to be used to keep track of which Nodes were expanded to get to another.

Routes are found by continuously expanding the Nodes connected to other Nodes; starting at a "Start Node", and ending at a "Goal Node". Some search-algorithms find the shortest path between the two Nodes by using certain heuristics, while others just keep expanding Nodes until the "Goal Node" is found, at which point the route is returned – no matter how good or bad it might be. Both kinds of algorithms will be discussed here, along with their advantages and disadvantages.

All map-tiles and Node/Way data is stored locally. This is because it was too risky to rely on third parties to keep their servers running at all times while the system was being developed. The author's internet-connection was also quite unreliable at times, so making the system rely on online-data would make this project very vulnerable to further delays. Third party mapping-services that were considered but decided against for various reasons: GraphHopper [7], Mapbox [10], Leaflet [29], Cloudmade [2], Omniscale [14], OpenLayers [15], and Google Maps [6].

2.2 Classes and methods

NOTES: - DELETE THIS -

- Classes split into separate packages for database-indexing, route-planning, and running the system.
- Interface for Nodes and Ways? Or are the Node and Way classes the top-level? Double-check.
- Interface for uninformed search.
- Interface for Informed search implements interface for uninformed Search.
- Each search-algorithm inherits all its functionality from those two interfaces - only implements one method: search(or something similar).

Every class has been sorted into one of three packages for database-indexing, route-planning, or running the system. Mapsforge provides the system with many new packages to handle the map, but as these were made by a third party, and accessed via a *.jar*-file, they won't be mentioned here.

The Node- and Way-classes implement an interface to ensure that any new Node/Way classes implemented in the future will still be compatible with the rest of the system, while still allowing them to implement new functionality or store more data.

There is also an interface for *"Uninformed Search"*, which provides every search-algorithm with the methods they need to expand Tower Nodes, change start/goal positions, keep track of run-times and memory usage, and unwind the path found by the algorithms (which only includes Tower Nodes) so that it also contains every Pillar Node on that particular path.

Additionally: There is another interface for informed search-algorithms, which extends the aforementioned *"Uninformed Search"*-interface to also keep track of path-costs, and provide methods to estimate the total path-cost to a goal Node.

There is one class for every search-algorithm in the system, each of which inherits its methods either from the *"Uninformed Search-"*Interface or the *"Informed Search-"*Interface. Because most of the functionality required to run a search-algorithm is already provided by the interfaces, the search-algorithm-classes only need to implement a single method for defining how the search is carried out.

There are two enums which define what constitutes an accessible or inaccessible Way, as well as an enum that defines which Ways represent larger areas or structures (eg. buildings and parking-lots) rather than a concrete path (eg. footways and roads).

2.3 tests

NOTES: - DELETE THIS -

- Test Driven Development (**TDD**).
- Every class has an associated test-class. Each test-class tests every method in the class it is made for.
- Custom test-graph. Ensures completeness and optimality.

This project has been developed using Test-Driven Development (TDD), which means that unit-tests have been written for every class and method in the system. To make it easier to run and keep track of all of these tests, separate test-classes have been created for every *"normal"* class, in which all of its associated tests are kept.

There is also a separate top-level test-class made for running every test-class in the system together, so that the system's functionality can be tested as a whole, rather than testing individual parts of it.

A custom test-graph has also been developed, which is meant to verify that the search-algorithms implemented in the system do not plan inaccessible paths, and are Optimal and Complete (if they are supposed to be); See Figures 4.1, 4.2, and 4.3.

2.4 Routing-data

NOTES: - DELETE THIS -

- Data read from *.osm* file downloaded from OpenStreetMap [24]. Data stored in two separate Arrays for Nodes and Ways (The type: Relations is not stored). XML-reader by Vogella [30].

  Why not store connections/relations and type-information inside the Nodes? No need to store Ways then.

- Data not downloaded from OSM server as the program runs (online functionality) because this has been deemed too risky. Unstable internet connections and risky to rely on 3rd party hosts when developing the software.
- All Nodes and Ways stored, except for islands/isolated data.
- Pillar & Tower Nodes.
- Relations not stored because they represent more abstract areas - i.e bus-routes (suggest as improvement to current system?).

The routing-data used by this system has been downloaded from OpenStreepMaps (OSM) [24], and is free to use for the public as long as we adhere to their license-regulations [16].

As all of the routing-data from OSM is stored as text in an *.osm* file, the system has to copy the relevant data into memory to make sure that it can be accessed quickly. Because *.osm* files are formatted as XML, the process of reading and copying the routing-data is done by an XML-reader made by Lars Vogel [30]. This particular XML-reader is free to use for anyone.

The system is more than capable of reading and using data from other sources though – as long as the file is formatted similarly to the OSM data currently used; see Table 2.1. The tags used to identify Nodes and Ways, as well as those used to identify the fields contained within them, can easily be changed in a designated enum, where all the tags have been hard-coded.
Let's say that a hypothetical data-set from Google was used to build the dataset of Nodes and Ways, and their xml-file called Ways 'Relations', then the label 'way'can easily be changed to 'relation'in the aforementioned enum to let the system know that this particular field has a different name, bust should still be handled in the same way.

After the relevant information in the *.osm* file has been copied into the system's memory, three filters are applied to this data, each of which is defined in its own enum. One filter removes any Ways that are deemed to be inaccessible (eg. steps), another filter removes any Ways that are not relevant for route-planning (eg. fences, bushes, trees, etc.), and the third filter removes any inaccessible doors in buildings, as well as all isolated Nodes and Ways without any connections to the rest of the data-set. These three filters remove a large chunk of unusable data, which frees up a lot of memory, and makes Node-expansion much faster as there are fewer children and alternative routes to explore.

As mentioned already: It is possible to dynamically download routing-data while the system is running, based on which Nodes the algorithms want to expand. This requires less data to be stored locally on the system, and ensures that the data is always up-to-date. The downside to downloading routing-data from an external source is that Node-expansion may be significantly slower per Node, and any downtime on the third party's servers or interruption to our internet-connection[2] would bring the entire system to a halt. The third party may be able to provide a guarantee that their servers will be available over extended periods of time if they get paid for their services, but as this system is not going to generate any money: paying for data is out of the question.

[2]The author's internet connection has been unreliable during the development of this system, which is another reason why all routing- and map-data is stored locally.

Node	`<node id="1" lat="52.4" lon="-4.0"/>`

Node	`<node id="2" lat="1.0" lon="1.0">`
	`  <tag k="entrance" v="yes"/>`
	`  <tag k="wheelchair" v="yes"/>`
	`</node>`

Way	`<way id="1">`
	`  <nd ref="1"/>`
	`  <nd ref="2"/>`
	`  <tag k="highway" v="footway"/>`
	`</way>`

Table 2.1: The file read by the system has to be in the XML-format, and its Nodes and Ways have to contain the data listed in this table. The fields are allowed to have different names, but all of the data has to be present; information other than that listed here is simply ignored.
Nodes can be formatted in either of the two ways shown here, but only the tags listed are currently accepted – with the exception of 'v="designated"', and 'v="limited"'acting as variations of 'v="yes"'after 'k="wheelchair"'.

2.5 Algorithms

NOTES: - DELETE THIS -

- A Star (**A***), Breadth First Search (**BFS**), Depth First Search (**DFS**), Greedy Best First Search (**GBFS**).
- Hierarchical Path A Star (**HPA***) or similar "abstraction(?)"-algorithms most commonly used in real route-planning software. Better (time & memory complexity) for longer routes. A* is ok on smaller areas like the AU Penglais campus.
- O( )-notation [27, P.82 & P.1037].
- Time, Space, (+ two more?).
- Pillar & Tower Nodes.
- path-cost and goal-distance calculated using formula (show it)

2.6 Map and Coordinates

NOTES: - DELETE THIS -

- Slippy map/Tiled web map/rastertile map.
- Map-API: Mapsforge [11].
- Data: OpenStreetMap [24].
- Map-tiles: Geofabrik [4] Sure about this? I think Mapsforge [12] was used instead.

- Considered: GraphHopper (Prominent attribution + Publicly accessible application), Leaflet, pure OSM?, , making my own map.
- Mention how to use the map? How to change start/stop locations, etc.? Or is this described elsewhere, eg. a separate UI section?
- Nodes expanded starting at parent used for entry into the Way; ensures correct path-costs.
- Parent Node is closest Tower Node between child and entry-Node, or entry-Node if there is no Tower Node between them.
- Data not downloaded from external server as the program runs (online functionality) because this has been deemed too risky. Unstable internet connections and risky to rely on 3rd party hosts when developing the software; tried a little, but free service was slow (probably intentionally).

The map used in this project has been provided by Mapsforge [11,12], as it is free, open source, and easy to implement. Many other map-providers were considered as well [2,4,6,7,10,14,15,26], but many of them cost money, were not written in Java, or provided too little/much functionality. GraphHopper [7] in particular was discarded because it is itself based on Mapsforge [11], but required that a prominent attribution to GraphHopper be displayed to the user, and required that the system be made publicly-accessible via a web-page or app-store– which this system is unlikely to be.
Creating a map without using third-party code was also considered, but ultimately decided against because of how mammoth this task seemed to be.

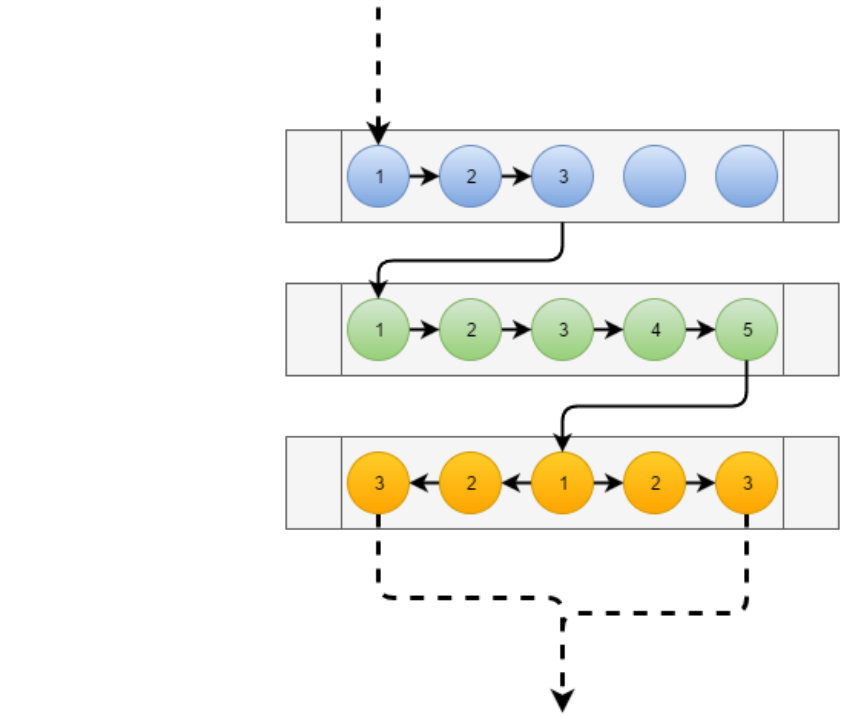
Routes are calculated either from two sets of latitude- and longitude-coordinates representing the start- and goal-Nodes, or from a location on the map clicked on by the user. The start- and goal-Nodes are not placed on the exact coordinates specified by the user however, but rather on the Node closest to those coordinates. This means that the system is robust to coordinates placed outside the expected range of Latitude: $-90 \rightarrow +90$, and Longitude: $-180 \rightarrow +180$.

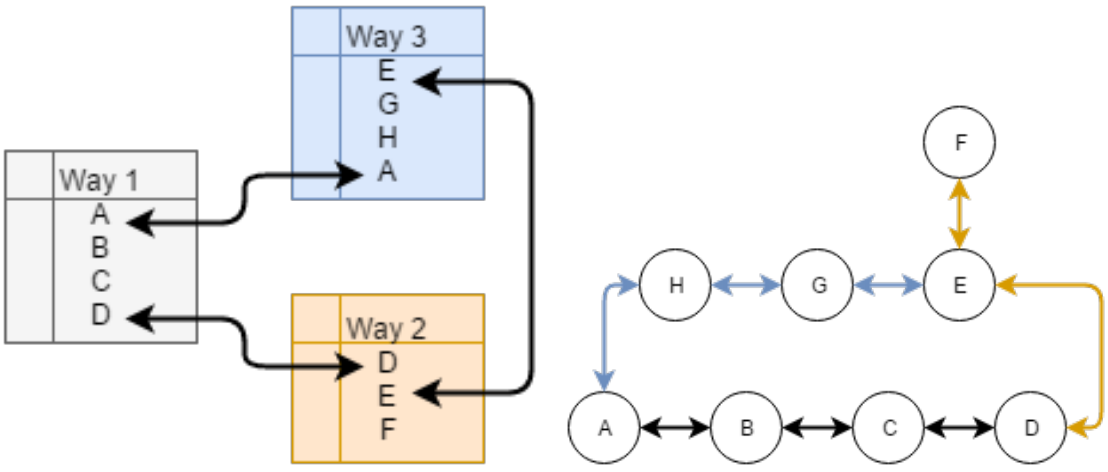
2.7 Programming

NOTES: - DELETE THIS -

- Only really know Java. Java used on many devices & Mapsforge is written in Java - try to spin this to make it sound positive.
- Tried to achieve loose coupling. Everything written in separate modules; changes in one module should not break another. Error/exception handling.
- Heavily commented + JavaDoc.
- Hash map/table: O(1) time-complexity as opposed to O(n) in Arrays (with unknown index), or O(log n) in binary. Could not make it work. Needs good hash-function, or O( ) might become horrible. O(2n)? memory vs O(n) in both Array and binary? (Reference to cheat-sheet?)

Figure 2.1: This illustration shows the order in which Nodes may be expanded inside a Way. Each Node points to the Node it was expanded from, and so on. Note how Nodes are expanded starting from the entry-point into the Way, which is not always the first Node in the list, and that Nodes may be expanded going in either direction.

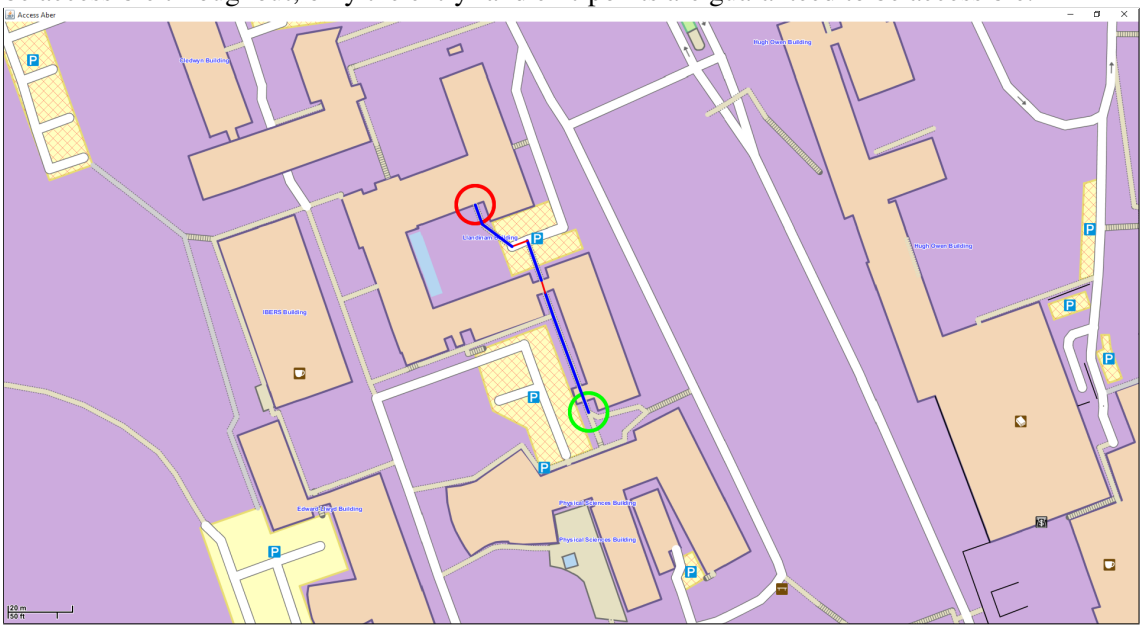




(a) Individual Ways (b) Graph

Figure 2.2: These images show how Ways may be connected by their Nodes. Note that a connection does not have to originate from the first or last Node in the list.

Figure 2.3: This image shows how the route is drawn as a thin red line whenever it passes through a larger area like a building or parking-lot. These red lines are meant to indicate that the actual path through the area is unknown, may be longer than shown on the map, and cannot be guaranteed to be accessible throughout; only the entry- and exit-points are guaranteed to be accessible.



- Heap: Maximally efficient Priority Queue - O(log n) vs O(nk). This may be a quote; careful. O(nk) refers to BSc sort?
- Measure of problem difficulty: Size of the state space graph: $|V| + |E|$. V=Set of Nodes. E=set of links/connections [27, Page: ?]. E=2(Nodes in Way-1) OR E=2(nN-nW). Check this.
- Empty String: 40 Bytes. Reference/Pointer: 32/64 bit. Cache-miss?
- PQ did not reorder when the value of its contents changed - resulted in a big bug.
- PQ and expansion list retained Nodes between runs - resulted in sub-optimal path, but faster runtime. (Mention the importance of path re-use).

Chapter 3

Implementation

3.1 Database

NOTES: - DELETE THIS -

- Nodes and Ways downloaded from OpenStreetMap [24], and stored in a *.osm* file. Size of downloadable area restricted.
- All data read by an XML-reader made by Vogella [30], and stored in two separate Arrays for Nodes and Ways (The type: Relations is not stored). Data filtered afterwards in order to reduce memory-requirements, and to speed up route-planning.

3.2 Search

NOTES: - DELETE THIS -

- Is this where I describe my algorithms, or do I do that in chapter 2?
- A Star, Breadth First Search, Depth First Search, Greedy Best First Search.
- PQ did not reorder after the values of its contents changed (after being sorted).
- PQ and expansion list not cleared between runs. Used old routes. (Touch on how this can be a good thing if done properly; time/space complexity).
- Only tower Nodes used for navigation. Very useful as some ways can have ¿100 Nodes, but far fewer connections to other Ways.
- Three enums: 'Permitted tags', 'Disallowed tags', and 'Area tags' dictate the way in which Nodes are expanded, discarded and handled afterwards.

Figure 3.1: This image shows what can happen to a route if the algorithms are working with bad data. The path between the start and goal positions is not included in the .osm-file, so the search-algorithms were forced to find a much longer route.

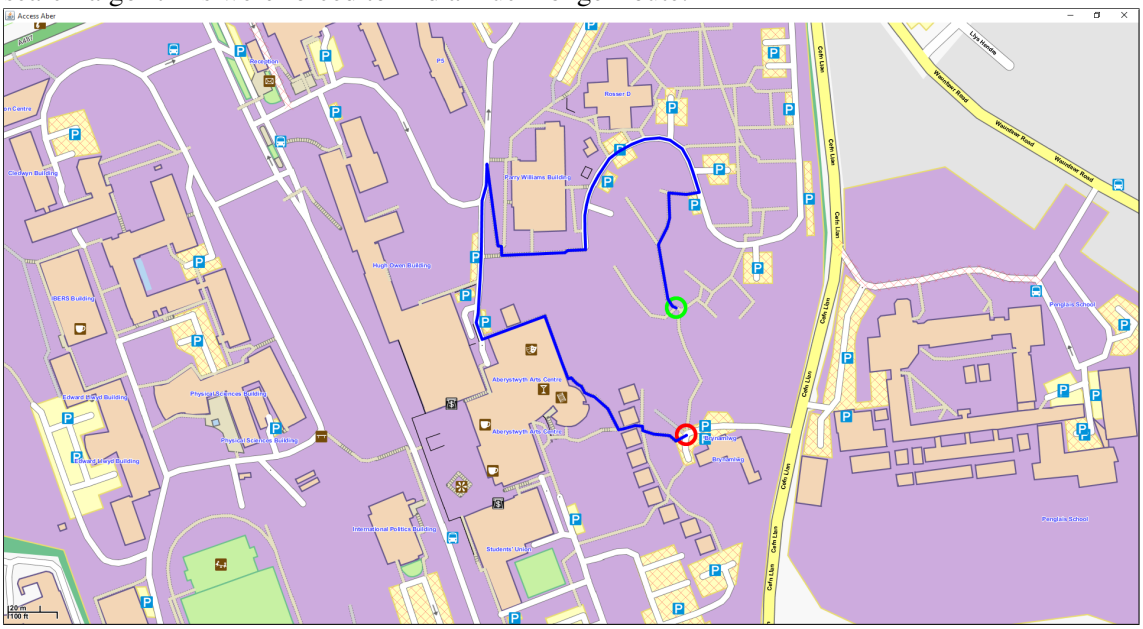


Figure 3.2: Algorithms avoid stairs

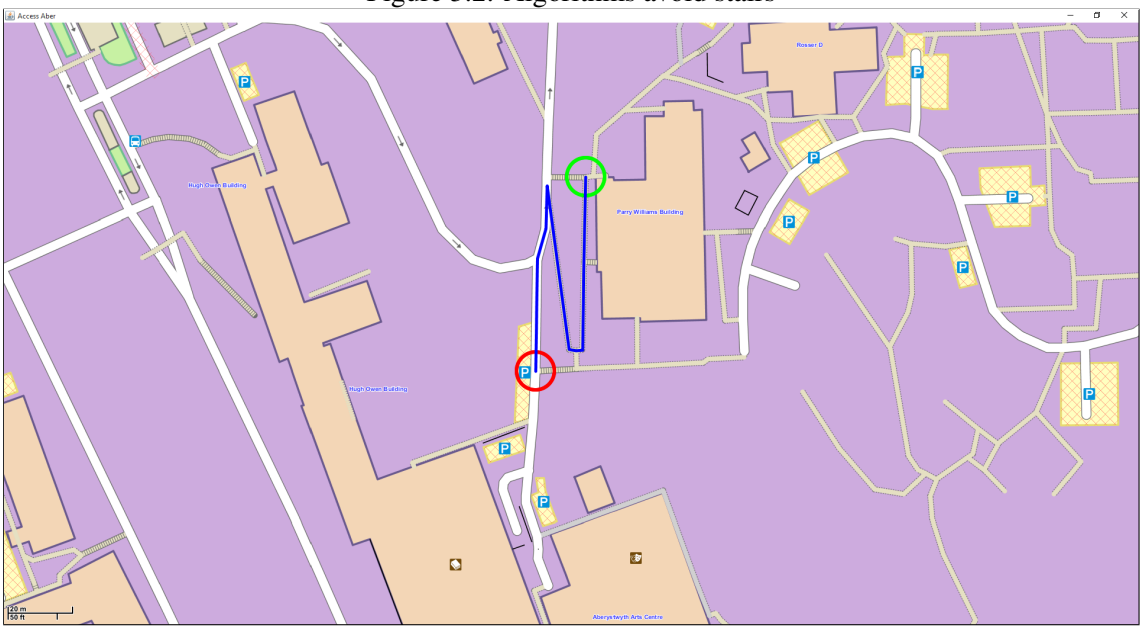
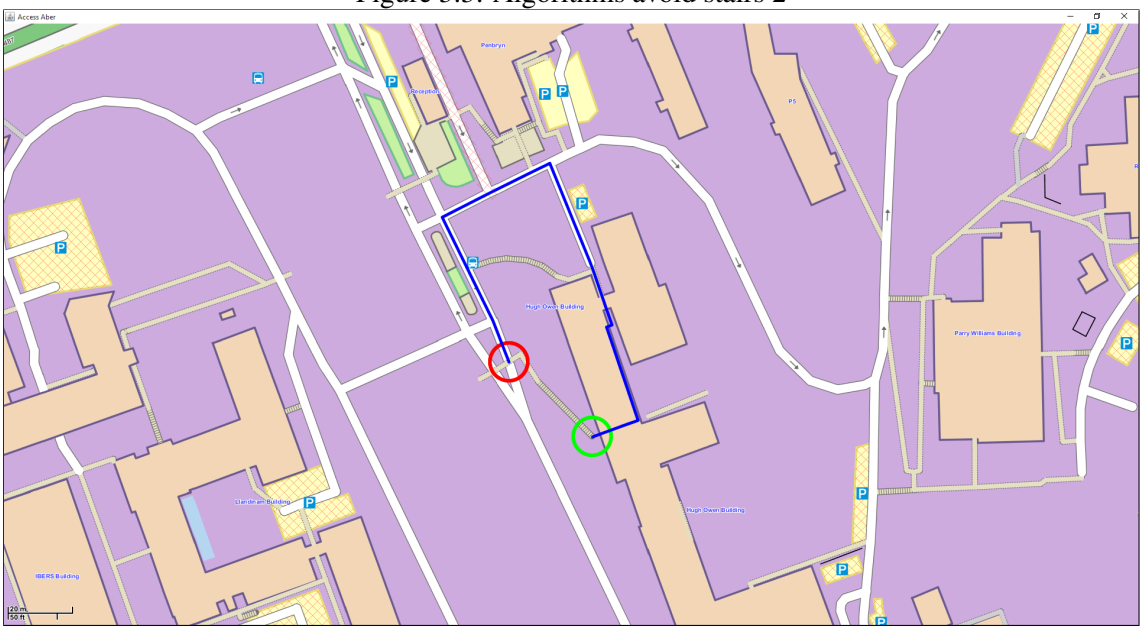


Figure 3.3: Algorithms avoid stairs 2



3.3 Map

NOTES: - DELETE THIS -

- Map-API from Mapsforge [11]. GraphHopper considered; they also use Mapsforge.

3.4 Fulfilment of requirements

NOTES: - DELETE THIS -

- Storage of OSM-data in separate Arrays for Nodes and Ways.
- OpenStreetMap database not updated with new and/or more accurate information.
- Good routing-algorithm found; A* guarantees optimality and completeness. Other algorithms are able to use less memory and find routes faster, but A* works fine on a small area like the Penglais campus of Aberystwyth University.
- Created filters able to easily exclude many Nodes/Ways from the search. Very easy to add and remove labels in the filters. Filters made for wheelchair-users only.
- No localisation (eg. GPS) implemented, but added functionality to find the Node closest to some coordinates and set it as the start/goal position, thus making it relatively easy to use localisation to set the user's position in the future. The coordinates need to use the same coordinate-system as the OSM-data though.

Chapter 4

Testing

4.1 Overall Approach to Testing

NOTES: - DELETE THIS -

- Test Driven Development (**TDD**).

  Tests have been written for every class and every method within those classes.

  Tests were usually written before or right after a new method was implemented.

- Top-level test-class for running every test together.

4.2 Automated Testing

NOTES: - DELETE THIS -

- JUnit-tests created for every class and method. Top-level test-class used to run every test automatically.

4.3 Integration Testing

NOTES: - DELETE THIS -

- Is this where I talk about the custom graph made for testing optimality and completeness?
- Nodes not visited in the custom graph-illustrations are meant to represent loops and dead ends.

4.4 User Testing

NOTES: - DELETE THIS -

Figure 4.1: Inaccessible path in the custom graph. This route passes through a Way with the tag: *highway=steps*, and is therefore inaccessible. It is the shortest path in the custom graph, but should never be expanded by any algorithm.

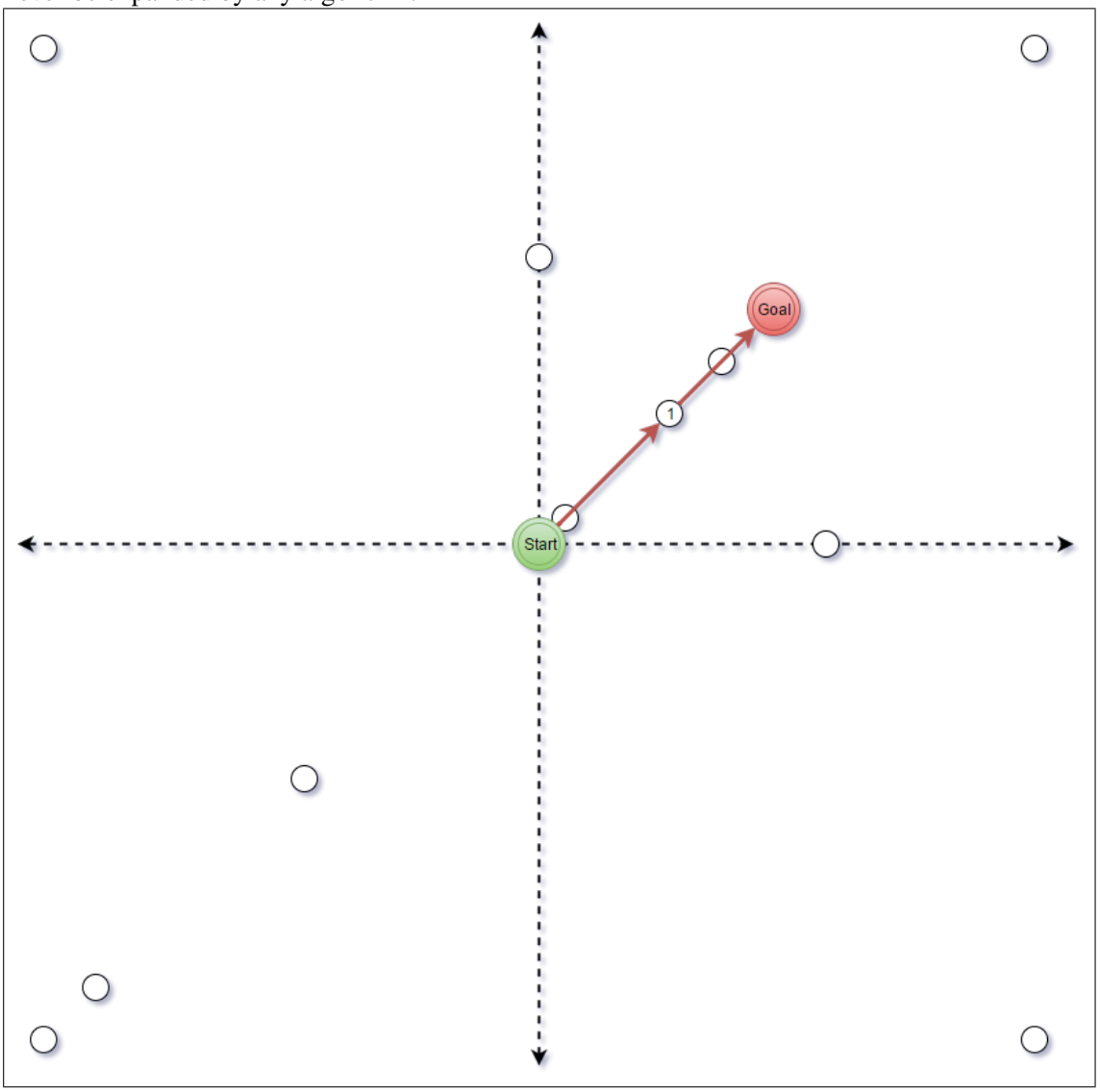


Figure 4.2: The longest path in the custom graph. This route is the longest with respect to distance, but passes through fewer Nodes than Figure 4.3. Uninformed search-algorithms like BFS and DFS, and greedy algorithms like GBFS are likely to follow this path.

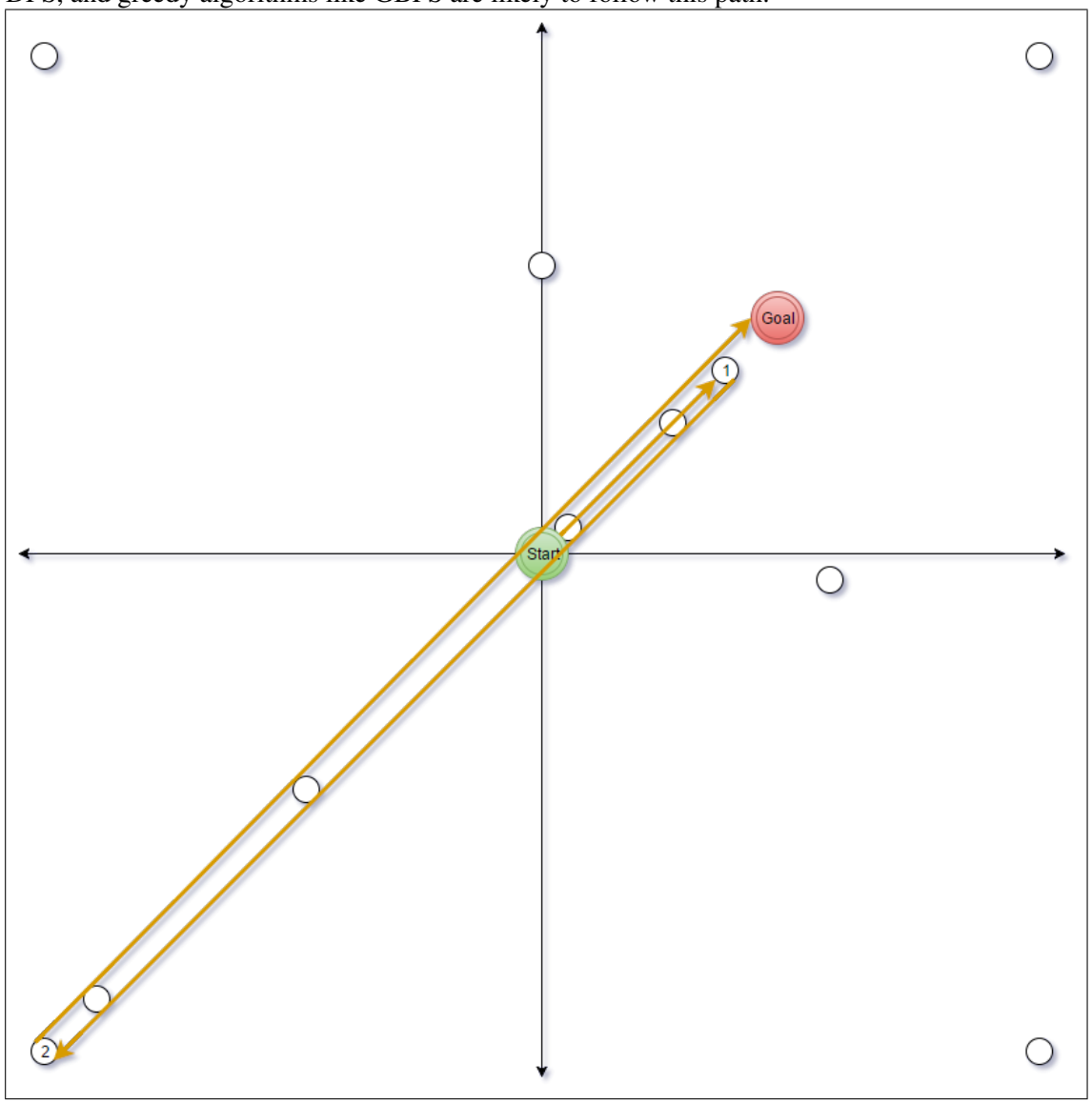
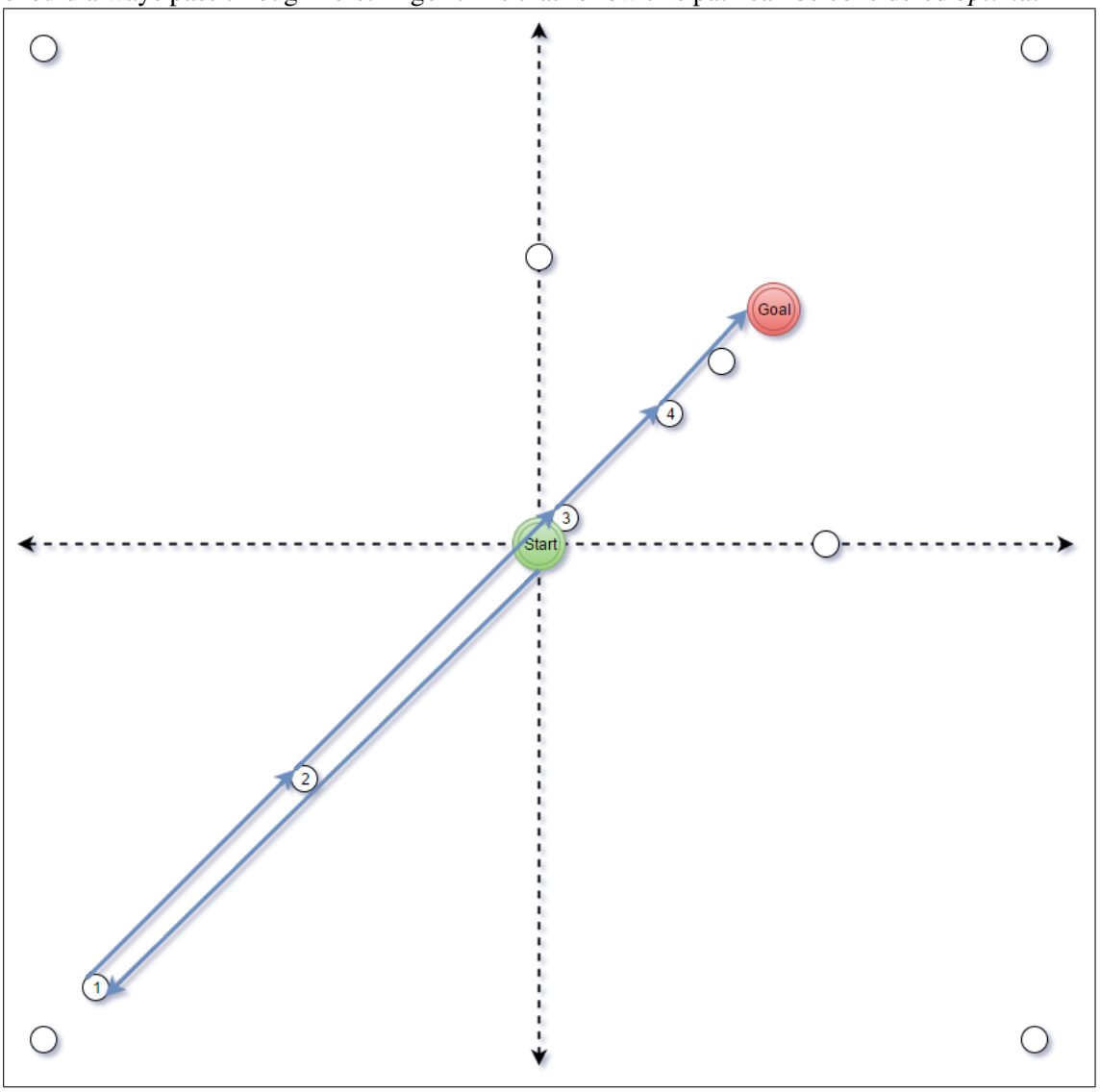


Figure 4.3: The optimal path in the custom graph. This route is the shortest with respect to distance, but passes through more Nodes than Figure 4.2. Informed search-algorithms like A* should always pass through here. Algorithms that follow this path can be considered *optimal*



- The system has not been tested together with any PRMs.

Chapter 5

Evaluation

5.1 Completed work

NOTES: - DELETE THIS -

- Occasional NullPointerExceptions (and a few other exceptions) when running the system. Problem originates in Mapsforge's code, not mine. Example: unable to load map-tile...
- Project started a month later than expected; unable to find supervisor(s) - only two (three?) potential supervisors replied to emails.
- No contact with supervisor for a month.
- Routes not always correct; caused by incorrect labelling in the OSM database. (Show Llandinam as example?).

5.2 Future work

NOTES: - DELETE THIS -

- Display route-distance on map and improve the GUI.
- Automatic retrieval of Nodes and Ways from OSM.
- Store data in better data-structures for faster indexing and lower memory-requirements.

  Hash table/map?

  Eliminate the need to store Ways by filtering out inaccessible Ways, and then storing Node-connections within the Nodes themselves? Way-type might not be needed after the filtering.
- Let user choose their disability/method of locomotion - eg. Manual/Motorised wheelchair, crutches, foot, bike, etc.
- Implement more/better search-algorithms.

- Update/add data to the OpenStreetMap databases to improve the usefulness and accuracy of my system's suggestions.

  Aberystwyth University should be able to provide records of accessible entrances, wheelchair-ramps, etc. on campus. That might be a good place to start.

- Implement localisation (eg. GPS) to make it easier for people to use the system.

- Add search-functionality to let users search for buildings and places rather than forcing them to click on the map; the user might know what the place they want to go to is called, but not necessarily where it is located.

- Include the data-type "Relations"in route-planning; especially bus-routes.

- Use services like [4,26] to download larger map-areas than what is possible using the export-option on openstreetmap.org (Which is what I currently do).

Appendices

Appendix A

Third-Party Code and Libraries

Appendix B

Code samples

2.1 Examples of test-classes

Annotated Bibliography

[1] A. Botea, M. Müller, and J. Schaeffer, "Near Optimal Hierarchical Path-Finding," *Journal of Game Development*, vol. 1, no. 1, pp. 7–28, 2004.

[2] CloudMade. Bespoke routing-service provider. [Online]. Available: cloudmade.com/solutions/maps-portals/

[3] Creative Commons. CC BY-NC-SA 3.0 DE. [Online]. Available: http://creativecommons.org/licenses/by-nc-sa/3.0/de/deed.en

[4] Geofabrik GmbH. Geofabrik. [Online]. Available: download.geofabrik.de

[5] Google. Google maps - terms of service. [Online]. Available: https://developers.google.com/maps/terms

[6] Google. (2015) Google Maps. https://maps.google.com/.

[7] GraphHopper GmbH. Graphhopper. [Online]. Available: graphhopper.com

[8] JGraph ltd. free-to-license web graph-drawing application. [Online]. Available: draw.io

[9] A. Kishimoto, A. Fukunaga, and A. Botea, "Scalable, Parallel Best-First Search for Optimal Sequential Planning," in *Proceedings of the International Conference on Automated Scheduling and Planning ICAPS-09*, Thessaloniki, Greece, 2009, pp. 201–208.

[10] Mapbox. Mapbox. [Online]. Available: mapbox.com

[11] Mapsforge. Mapsforge - homepage. [Online]. Available: mapsforge.org

[12] ——. Mapsforge - map-tiles. [Online]. Available: download.mapsforge.org/maps/

[13] National Geospatial-Intelligence Agency. Wgs84. [Online]. Available: http://earth-info.nga.mil/GandG/wgs84/

[14] Omniscale GmbH & Co. KG. Omniscale en. [Online]. Available: omniscale.com

[15] Open Source Geospatial Foundation. Openlayers 3. [Online]. Available: openlayers.org

[16] OpenStreetMap Foundation. Openstreetmap license. [Online]. Available: www.openstreetmap.org/copyright

[17] ——. OSM - Features. [Online]. Available: wiki.openstreetmap.org/wiki/Map_Features

[18] ——. OSM - Nodes. [Online]. Available: wiki.openstreetmap.org/wiki/Node

[19] ——. OSM - Relations. [Online]. Available: wiki.openstreetmap.org/wiki/Relation

[20] ——. OSM - Tags. [Online]. Available: wiki.openstreetmap.org/wiki/Tags

[21] ——. OSM - Ways. [Online]. Available: wiki.openstreetmap.org/wiki/Way

[22] ——. OSM - XML file format. [Online]. Available: wiki.openstreetmap.org/wiki/osm_xml

[23] ——. Wgs84 conversion in openstreetmap. [Online]. Available: wiki.openstreetmap.org/wiki/Converting_to_WGS84

[24] ——. (2004) OpenStreetMap Homepage. http://wiki.osmfoundation.org/wiki/Main_Page.

[25] Ordnance Survey. A guide to coordinate systems in great britain. [Online]. Available: http://www.ordnancesurvey.co.uk/docs/support/guide-coordinate-systems-great-britain.pdf

[26] OSRM. Open source routing machine. [Online]. Available: project-osrm.org

[27] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson Custom Library, 2014, pearson New International Edition. 978-1-292-02420-2.

[28] The Eclipse Foundation. Eclipse public license - v 1.0. www.eclipse.org/legal/epl-v10.html.

[29] Vladimir Agafonkin of Mapbox. Leaflet. [Online]. Available: leafletjs.com

[30] L. Vogel. Java and xml - tutorial. [Online]. Available: www.vogella.com/tutorials/JavaXML/article.html

[31] S. Yang and A. K. Mackworth, "Hierarchical shortest pathfinding applied to route-planning for wheelchair users," in *Proceedings of the Canadian Conference on Artificial Intelligence, CanadianAI 2007*, Montreal, PQ, May 2007, pp. 539–550.