NTNU – Trondheim
Norwegian University of
Science and Technology

Finding the Nearest Wheelchair

the Application of AI Search Methods to
Logistical Problems in the Hospital

Aleksander Rognhaugen

Master of Science in Informatics
Submission date: June 2014
Supervisor: Pieter Jelle Toussaint, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Ever since the invention of the Global Positioning System (GPS) and its slow entry into the public domain in the 80s and 90s, location tracking has become an increasingly important tool for scientific research, surveillance, and commerce. The potential for increased efficiency and quality of care has resulted in an increase in Indoor Positioning System (IPS) technology uptake by the healthcare sector. The main focus of this master's thesis has been on the application of search methods from Artificial Intelligence (AI) literature to a logistical problem in the hospital, facilitated by the use of location tracking technologies.

Using a Requirements Engineering (RE) framework the thesis explores use cases where time spent looking for wheelchairs can be reduced. The three use cases: *"View Wheelchair Location"*, *"Find Available Wheelchair"*, and *"Find Nearest Wheelchair"* was identified, where finding the nearest wheelchair was found to be most suitable for further investigation. The thesis proposes a search framework for describing the problem of finding the nearest wheelchair given the current position and task information of a hospital porter. Furthermore, the RE study proposes a set of requirements that the search framework must adhere to. A literature study of AI search methods such as the well known Dijkstra's and A* algorithm along with different kinds of acceleration methods were performed, and a prototype of the search framework was built. The search framework and all the proposed search methods have been evaluated by performing a series of experiments on two different kinds of datasets made by the author.

Preface

The work with this master's thesis has been performed at the Department of Computer and Information Science (IDI) and has been submitted to the Norwegian University of Science and Technology (NTNU) as the final part of a two-year master's degree in Informatics.

The original work of this thesis was to explore one or more use cases of Indoor Positioning Systems (IPS's) and perform an evaluation of the IPS at St. Olavs Hospital to see if it could support the aforementioned use cases. Due to circumstances beyond our control the work of the thesis was changed to the exploration, implementation, and evaluation of the use case: finding the nearest wheelchair.

I would like to thank my supervisor Professor Pieter Jelle Toussaint for valuable comments and guidance throughout this work. I would also like to thank Associate Professor Magnus Lie Hetland and Professor Keith Downing for invaluable brainstorming sessions. A further thank you is in order for our domain contact at St. Olavs Hospital, Hans Kottum, and Wireless Trondheim, Thomas Jelle, as well as the porters Sten Erik Rønning and Grete Muller who took the time to help me with the use case studies; this work could not have been done without you.

Trondheim, 2014-05-31

Aleksander Rognhaugen

Contents

List of Figures

List of Abbreviations

AI Artificial Intelligence.

BFS Breadth-First Search.

BPMN Business Process Model and Notation.

DFS Depth-First Search.

GIS Geographic Information System.

GPS Global Positioning System.

ICT Information and Communications Technology.

IPS Indoor Positioning System.

IREB International Requirements Engineering Board.

MDA Model-Driven Architecture.

MOF Meta-Object Facility.

NTNU Norwegian University of Science and Technology.

OMG Object Management Group.

RE Requirements Engineering.

RFID Radio-Frequency Identification.

RQ Research Question.

RSSI Received Signal Strength Indication.

RTLS Real-Time Locating System.

SRS Software Requirements Specification.

TDOA Time Difference of Arrival.

TSP Travelling Salesman Problem.

UML Unified Modeling Language.

WLAN Wireless Local Area Network.

Chapter 1

Introduction

St. Olavs Hospital is a university hospital located in Trondheim, Norway, and serves as the local hospital for the population of Sør-Trøndelag with approximately 300 000 inhabitants (09.09.2012)[1]. The hospital has several regional and national tasks that span the whole population of central Norway (Møre and Romsdal, Nord-Trøndelag and Sør-Trøndelag)[2]. The university hospital is tightly integrated with the Norwegian University of Science and Technology (NTNU), carrying out patient treatment, research, and education.

In 2002 the Norwegian Government decided to build St. Olavs University Hospital at Øya in the central part of Trondheim. The development is managed by the independent project Helsebygg Midt-Norge (The Hospital Development Project for Central Norway), which is owned by both the Ministry of Health and the Ministry of Education and Research[3]. According to the current plans the last phase, 2-2, of the development is set to conclude in 2013. This last phase mainly consists of the construction of The Knowledge Centre.

The main aim of Helsebygg is: " To develop the university hospital as an organization of high quality, efficiency and professionalism. This means a hospital based on teamwork in an integrated health service, with medical expertise, nursing and care focused on the patient." [4]. The implementation of state-of-the-art Information and Communications Technology (ICT) has been a priority area in order to fulfill this aim. The ICT solutions include, among other things, some 1700 access points for wireless communication, wireless IP phones, and smart cards for controlling access to both buildings as well as computers[5].

1.1 Motivation

Ever since the invention of the Global Positioning System (GPS) and its slow entry into the public domain in the 80s and 90s, location tracking has become an increasingly important tool

for scientific research, surveillance, and commerce. In recent years there has been a reduction of cost in software and hardware equipment necessary for location tracking. This in turn have increased the demand for positioning systems in businesses and institutions, such as hospitals.

GPS promises a worst case accuracy of about 7.8 meters[6], and has been used for efficient localization of objects such as cars and people in many outdoor applications; however, as the object of interest moves indoors the GPS accuracy decreases significantly. This problem is due to obstacles that absorb, diffract, reflect, refract, and scatter GPS signals, making it *nearly* impossible to determine the position of objects that reside indoors. Indoor Positioning Systems (IPS's) and Real-Time Locating Systems (RTLS's) attempts to alleviate some of these issues using technologies such as ultrasound, radio frequency, and Wireless Local Area Network (WLAN), among others.

The potential for increased efficiency and quality of care has resulted in an increase in IPS technology uptake by the healthcare sector. Sun et al. (2008) explores the notion of " To Err is Human" [7] by proposing a method for reducing the risk of medication errors by integrating barcodes and Radio-Frequency Identification (RFID) tags[8]. Emory A. Fry and Leslie A. Lenert describes an integrated software-hardware RFID tracking system, MASCAL, designed to enhance the management of hospital resources during a mass casualty event[9]. Other interesting examples of applications utilizing positioning systems includes automatic timestamp documentation for patient tracking[10], tracking of patients suffering from dementia[11], linking position data and clinical data for improved patient processing[12], and reducing loss of expensive medical equipment by way of location tracking[13].

One of the new ICT technologies used across St. Olavs Hospital is the IP phone system, which is heavily used by the hospital logistics group. For instance, an operator is able to assign tasks to porters, such as helping a patient traverse the hospital, based on their location, rather than forcing porters to return to base after each task. In many cases, these tasks involve bringing a wheelchair to the patient. In general, the logistics group is interested in minimizing the time spent searching for wheelchairs; however, this might not always be easy as there are no fixed space where available wheelchairs can be found. The purpose of this master's thesis is to apply search methods from Artificial Intelligence (AI) literature on the problem of finding wheelchairs which lie on the way to tasks. A requirements engineering study will be performed in order to recognize potential use cases where time spent looking for wheelchairs might be minimized, and a literature study and review of search methods will be performed based on the use cases from the requirements engineering study. The result of the thesis will be a concept proposal that reduces the time spent by porters looking for wheelchairs by utilizing the new IPS at St. Olavs Hospital.

1.2 Objectives

The following are the main research objectives of this thesis:

- **Objective 1:** *Explore and understand use cases of IPS for reducing the time spent by porters looking for wheelchairs.*
- **Objective 2:** *Research and compare search methods from AI literature in terms of optimality, runtime, and explored search space.*

The following Research Questions (RQs) were formulated in order to achieve said objectives:

- **RQ1:** *How can the time porters spend looking for wheelchairs be reduced?*
- **RQ2:** *What is considered to be the nearest wheelchair?*
- **RQ3:** *Which search methods are most appropriate for reducing the time spent looking for wheelchairs in terms of optimality, runtime, and explored search space?*

1.3 Scope

The main research objectives of this thesis are answered in the context of healthcare. Relevant ethical issues are beyond the scope of this thesis, and even though they might be mentioned it will not be the focus of this thesis. There is currently a lot of interesting research going on in areas of IPS applications such as Google's Project Tango[14], most of which are beyond the scope of this thesis. For clarity, everything originating from a third party will be attributed to its original creators.

1.4 Research Methodology

Design science research is a problem solving process, and in order to ensure proper use of design science this thesis will rely on the research framework proposed by Hevner et al.[15]. The core concept of design science is the creation of artefacts[15, 16], and can be seen summarized in the research framework in Figure 1.1.

Starting at the far left of Figure 1.1, a set of business needs will be gathered by performing a Requirements Engineering (RE) study in the context of logistical problems at St. Olavs Hospital. RE theory and a literature study on AI search methods will be drawn from the knowledge base of scientific literature. The identified business needs and applicable knowledge from the knowledge base will facilitate the development of a search framework along with a set of potential search methods. The proposed search framework, as well as the potential search

methods, will be assessed and refined by instantiating a prototype and running simulations on it. Finally, observations and conclusions are to be extracted from the simulations, structured, and added back to the knowledge base in the form of this master's thesis. A demonstration of the results of the master's thesis will be presented to our domain contacts; however, due to our limited access to the IPS at St. Olavs hospital, we will not be able to demonstrate the search framework appropriateness, in the environment.

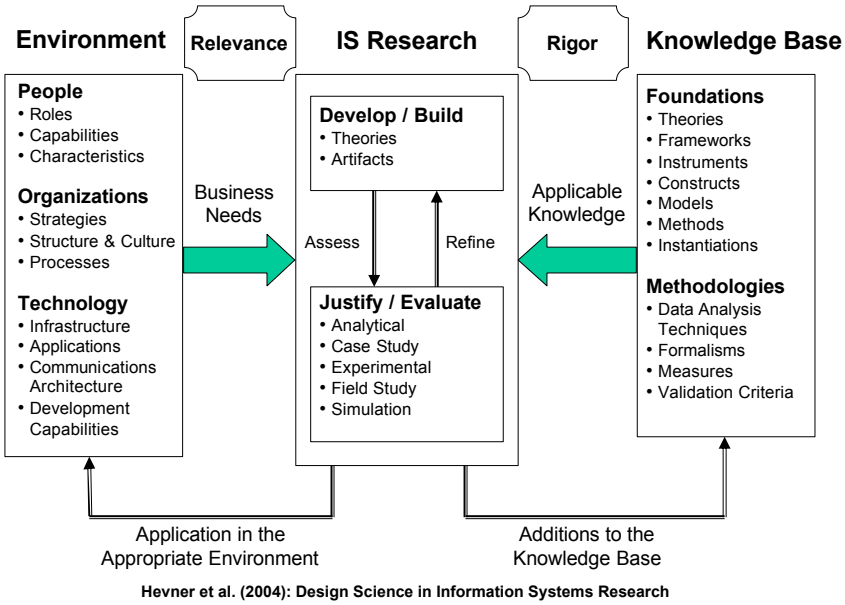


Figure 1.1: Information systems research framework.

The above steps are inherently iterative, meaning that no single step in the research framework, for instance gathering knowledge from the knowledge base, is done *only* once.

1.5 Thesis Outline

The remainder of the thesis is arranged as follows. Chapter two presents background and related work necessary for understanding later chapters. Chapters three draws on the theoretical background from chapter two, and aims to show results concerning **RQ1** by describing the requirements engineering methodology used and the results gained. In the fourth chapter methods and implementational issues pertaining to the proposed system is described. Chapter five describes experiments and results done on the datasets and search methods from chapter four pertaining **RQ2** and **RQ3**, while chapter six includes some general observations and discussions. The thesis concludes in chapter seven, containing a conclusion and future work. Additional information can be found in the appendix.

Chapter 2

Background

This chapter introduces some necessary concepts before we continue on with our methodology. As our thesis combines theory from both software engineering and pathfinding, we will approach various key concepts from both Requirements Engineering (RE) and combinatorial optimization.

2.1 Requirements Engineering

The importance of RE is often misunderstood; however research has shown that the difference between good and bad RE can mean the success or failure for a project. In a report from 2004 by the Standish Group on information systems of American companies it was shown that the top three reasons for overdue projects is lack of user input, imperfect requirements, and requirements that are modified late in the development cycle. The report shows that project failure caused by requirements reaches 48.1%[17](original report not available). In general, a solid RE process will ensure a firm foundation for the rest of the project.

2.1.1 Definition

Requirements engineering can be seen as an exercise of human communication, and a proper definition constitutes of two key components. Requirements can be seen as specification of the software to be built, and describes properties, constraints, and behaviours that the finished system should display. The human element of RE comes in the form of the collaborative activites that has been shown to be a crucial factor in determining the success of the software process[18]. Stakeholder is the name that is commonly used to refer to a person or an organization that is involved or has an interest in the system. They have either direct or indirect influence on the requirements of the system, and it is therefore important to identify the

potential stakeholders early in the process[19, 20]. It is critical to understand that stakeholders are not only the users, the requirements engineer(s), and administrators of the system, but also make up the legal entities, management, hackers, and developers that at some point in time may influence the requirements of the system.

Now that we have described both what requirements and stakeholders are, we can define what we mean by requirements engineering. The definition is loosely based on the definition given by Pohl and Rupp, International Requirements Engineering Board (IREB),[20] and a literature review of the different international standards of RE.

Definition 1. ***Requirements engineering*** *is a sub-discipline of software engineering and is a set of core activities that carries out a systematic and disciplined specification of requirements. Key to this is achieving a consensus among the stakeholders so that the understanding and management of requirements minimize the risk of delivering a system that satisfies the stakeholders' needs and desires.*

A brief overview over the different parts that make up RE will be presented in the following sections.

2.1.2 Requirement Types

In order to fulfill the systematic and disciplined elements of the RE definition above we need to formalize requirement types as they appear in a requirement specification, often called a Software Requirements Specification (SRS)[21, 22]. While there are several international standards, such as ISO/IEC/IEE 29148:2011[23] and ISO/IEC 25010:2011[24], that define requirement types slightly differently, they can more or less, be compartmentalized into functional and non-functional requirements. The first type concerns itself with what the system must do, while the latter can be seen as how the system must adhere to the encompassing business context. All requirement types limit the solution space. Below is a limited set of requirement types adapted from [20] by IREB.

Definition 2. ***Functional requirements*** *are mutable properties that concerns itself with how functions of the system should behave.*

Definition 3. ***Quality requirements*** *are mutable properties that concerns itself with desireable qualities that the system should exhibit. Typically, this is properties such as: availability, performance, and scalability.*

Definition 4. ***Constraints*** *are immutable properties that the system must adhere to.*

2.1.3 Context

An important part of RE is the identification of relevant parts of the environment where the resulting system will reside. These parts are usually immaterial and abstract properties that define already existing systems that can have a huge impact on the scope of the new system. Pohl and Rupp defines the system context as relevant, often immutable, parts of the system environment that is important for the understanding of the SRS. The system interfaces with the context via a system boundary that separates the mutable system to be developed from parts of the environment that cannot be altered by the by the development process. A context boundary separates the relevant and irrelevant parts that has to be considered during RE[20, 21].

Ensuring that all stakeholders agree on how existing systems affect the new system – either implicitly or explicitly – is an important factor in order to reduce the risk of unwelcome surprises during or after the development and deployment of the new system. Generally, it is difficult to assign a proper definition of the system and context boundaries during the early stages of the RE process; therefore, so called *gray zones*[20] may appear that represent boundary areas that are less understood. One way of ensuring that any boundary growth or decrease does not influence the understanding of the SRS is to extend both boundary types beyond what is initially assumed.

2.1.4 Core Activities

Gathering, documenting, verifying, and managing requirements are the commonly performed activities during the RE process; however, the actual definition of each activity may differ depending on, for example, the software development methodology (or process model). In a comparative study from 2013 by Batool et al.[25] it was found that for traditional process models that incorporate RE such as the waterfall model, the main bulk of attention is given to the creation of detailed documentation. On the other hand, in agile approaches such as Scrum and Extreme Programming (XP), more attention and effort was made to face-to-face communication with the customer throughout the development process. Increasing the focus on customer interaction naturally enhances the mutual trust between customers and developers[17]; moreover, this ensures that requirements can be suggested, discussed, validated, or even thrown away rapidly. In a paper from 2012, de la Vara et al. argues that new research is necessary for moving towards customer-based RE[26]. A set of research questions are proposed which includes how customers can and should be more involved in the analysis and validation of a SRS. Other variations include goal-oriented requirements engineering (GORE)[27] where every activity concerns itself with different levels of goal abstractions, e.g. business survival and employee needs, and approaches where the focus is on the integration

of RE and software process models[28, 29]. The latter has been predicted to be of great importance in modern software development[29].

Figure 2.1 illustrates the core activities in RE and how they relate to each other. Notice how the management of requirements is not something that is done after a set of condensed requirements are gathered, but rather an integral part in each activity.

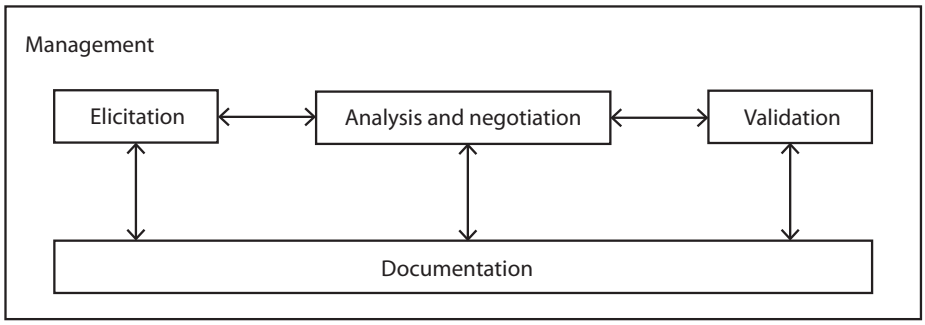


Figure 2.1: Core activities in requirements engineering.

A brief look on techniques and methods for each core activity will be presented in the next few sections.

Elicitation

Requirements elicitation is the process of gathering requirements from stakeholders. In the last decade the acknowledgment of the importance of customers and other stakeholders have increased steadily. This has been especially evident in the agile RE communities. Below is a brief look on the most commonly used approaches for requirements eliciation.

Survey techniques are usually driven by questions and requires that the stakeholder is able to explicitly express knowledge and wishes about the system to be developed. Questionnaires are one such technique that is useful for eliciting requirements from a large number of participants. A questionnaire can contain both open and/or closed questions, however having predetermined answers may reduce misunderstandings. Thus, questionnaires may be useful for gaining initial impressions and opinions[30]. Questionnaires can also be used for asking whether or not the respondent is willing to attend an interview. Interviews is another survey technique that can be either structured, semi-structured, or unstructured. During a structured interview a series of questions are asked, while during an unstructured interview a multitude of topics or scenarios can be explored in a discussion. During semi-structured interviews open-ended questions are asked while the interviewer steers the course of the conversation.

Unstructured interviews are quite similar to brainstorming and focus groups, where groups

of stakeholders collectively discuss and comment ideas. These types of techniques are creative in nature and help facilitate a large number of ideas in a short amount of time. Difficulties in creative techniques often arise when participants do not get along, e.g. one or more participant dominates the conversation, ideas are ridiculed, or there is little to no contribution from the participants. Many of these issues can be remedied by carefully planning the session, e.g. which scenarios to discuss, and having a moderator.

Observational techniques consists of indirect and direct observation[30]. In indirect observation techniques such as activity logging is used in order to analyze scenarios that are relevant to the project. In direct observation trained observers attempts to understand the nature of tasks in the context in which they are performed. Field study and apprenticing are two direct observation techniques where an observer gains in-depth knowledge of one or more tasks.

As stated by Mishra et al. (2008)[31] the state-of-the-art practice in RE is mostly ad hoc, with little to no planning. Thorough planning of requirements elicitation can be beneficial and may save additional costs by reducing errors earlier in the development process. A good understanding of RE techniques may increase the likelihood of high customer satisfaction, which, in turn, will improve the efficiency of the RE process[31]. Most RE techniques are context-sensitive, which means that there is no silver bullet that suits every project and every stakeholder. In the paper by Mishra et al. a combination of different approaches that help consolidate the large amount of information from stakeholders seem to yield the best result.

The characteristics of new requirements may change depending on the presence or absence of existing system architectures. Miller et al. (2009) found an inverse relationship between user needs and technological needs when eliciting requirements[32]. In the presence of a system architecture the requirements engineer tends to elicit approximately 10% more technologically oriented requirements, while in the absence of a system architecture about 10% more user-needs oriented requirements are elicited. This is important information when deciding which eliciation technique/approach is most appropriate. The study was performed by giving a set of eliciation tasks for a fictitious bank to 24 groups, where half of the groups had access to the system architecture and vice versa.

Analysis, Validation, and Negotiation

Raw observations, requirements, and other information from the requirements eliciation process has to be analyzed, validated, and negotiated into a set of requirements that all stakeholders can agree on. Analysis may include throwing away, keeping, or even combining different ideas and opinions. Validation ensures that all requirements abide by the same quality criteria. Negotiation is a general activity that is performed throughout all other activities in an effort

to reduce conflicts between stakeholders during RE.

The international standard IEEE Std 830-1998[22] presents a set of good SRS characteristics. These requirement quality criteria will be summarized in brief here.

- *Correctness* – An SRS must meet the wishes of the stakeholders, and every requirement must be one that the system shall meet.
- *Unambiguous* – Every requirement in an SRS can have only one interpretation. This is important for the whole software life cycle, be it design, implementation, or maintenance.
- *Complete* – All relevant requirements must be documented and each requirement must contain all necessary information. No requirement can be described as "to be determined".
- *Consistent* – It must be possible to implement every requirement in the SRS, there can be no contradictory requirements.
- *Ranked for importance and/or stability* – Each requirement should have been carefully considered by the relevant stakeholders in order to rank its importance and/or stability. Not all requirements are equally important, some can even be optional.
- *Verifiable* – An SRS is verifiable if it is possible to define an acceptance and test criteria for every requirement. A requirement should be removed or revised if it is not verifiable.
- *Modifiable* – An SRS is modifiable if all requirements can be easily modifed while retaining a coherent SRS structure and style. Good qualities for modifiability is no redundancy and a coherent SRS structure.
- *Traceable* – A requirements is traceable if it is possible to trace it over the course of the system life cycle. This may include having a unique name or reference number for each requirement so it can be referenced at a later date.

Conflict management consists of four tasks: conflict identification, conflict analysis, conflict resolution, and documentation of the conflict resolution[20]. A multitude of different conflicts can arise during the RE process, e.g. contradictory requirements from different stakeholders, and the requirements engineer must pay attention to potential conflicts so that they can be resolved or even avoided. There are different types of conflicts such as relationship conflicts and conflicts of interest, and it is important to identify the correct conflict type in order to be able to apply the appropriate conflict resolution method. Common methods for resolving conflicts include compromise, voting, overruling, and consensus, where the last method is

preferable as it is the least likely to create new conflicts. The last task, documentation of conflict resolution, is useful to be able to quickly resolve new conflicts that are similar to previous conflicts.

Documentation

Requirements documentation is an activity that is performed together with eliciation, analysis, and validation of requirements. It consists of systematically representing the condensed set of requirements, satisfying a set of quality criteria. The result of the documentation activity is often referred to as a SRS[21, 22], and may include requirements in natural language and/or conceptual models.

Requirements are frequently documented in natural language, and has the added benefit that requirements can be easily written down and easily read/understood by stakeholders. The disadvantage of documentation in natural language is the potential ambiguities due to its subjective nature. The personal perception of natural language might vary from person to person and can easily lead to uncorrect and unambiguous requirements. Using requirement templates, i.e. a blueprint of the syntactic structure of requirements, is one way of reducing these issues[20]. The following is a minimal example template from [20]: *The system shall/should/will/be able to [insert process verb]*. This template makes it clear who should perform the *process verb*, e.g. display report, and also what kind of obligation the system has.

Conceptual modelling is the idea that anything can be represented using models, where a model is defined as an abstraction of reality. In the context of computer science these abstractions are often modelled with Unified Modeling Language (UML) or Business Process Model and Notation (BPMN). All models adhere to a domain which describes a bounded field of interest or knowledge[33], e.g. a business domain. The so called metamodel is a formalization of the aforementioned domain, and defines the building blocks, relations, constraints, and rules that govern a modelling language. In other words, meta-modelling defines the abstract syntax, e.g. UML description, and the static semantic of a modelling language, but not the concrete syntax, e.g. UML graphical notation. Because a metamodel is a relative concept it needs its own metamodel, called the meta meta model, which describes concepts that the metamodel is allowed to use. UML is a metamodel instantiated from a meta meta model called Meta-Object Facility (MOF), an Object Management Group (OMG) standard for model-driven development[33][1]. UML is often talked about in context with Model-Driven Architecture (MDA)[33]; however, the graphical notation UML uses is often ideal for documenting requirements as models. Using a modelling language like UML has many advantages

[1]Ironically, UML was defined before MOF and MOF uses concrete syntax from UML, making it even more convoluted.

such as the ease of understanding complex and domain-specific knowledge. One typically distinguishes between three types of perspectives when using conceptual modelling to document requirements[20]. Entity-Relationship diagrams and UML class diagrams are often used to model requirements in the data perspective. In this perspective data is modelled as entities that are connected via relations. UML provides many building blocks, such as association and generalization relations, that makes it possible to model data efficiently. UML activity diagrams and UML use case diagrams are often used to model requirements in the functional perspective. UML use case diagrams relate use cases, e.g. *download report*, with actors, e.g. *accountant*. It is possible to convert graphical use case diagrams to textual ones. BPMN and UML state diagrams are typically used to model requirements in the behavioural perspective. These types of diagrams describe how different states are activated using transitions.

In general, it is usually wise to combine modelling languages with natural language, thus minimizing the disadvantages that either one might have.

Management

Requirements management is what ties all the core activites of RE together, ensuring, for example, that all requirements can be viewed by the relevant stakeholders. As can be seen in Figure 2.1, requirements management is not an activity that is done by itself, but rather a group of methods and techniques that support all the other activites. This includes risk management, ensuring traceability, change request handling, and proper versioning.

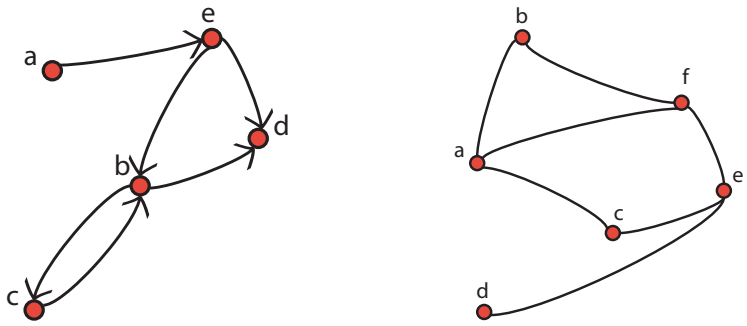
2.2 Graph Theory

The theory of graphs is a major topic in the area of discrete mathematics, and was first described by the Swiss mathematician Leonhard Euler in 1736[34](see page 513). Unlike continuous graphs, the graphs that we introduce here are finite, and as we shall see later are appropriate when trying to visualize and search on maps.

Definition 5. *A graph is denoted $G = (V, E)$, where V is a finite non-empty set of vertices, or nodes, and E is a finite set of cross products between vertices taken from V. The cross product, or cartesian product, between two sets can be defined as $A \times B = \{(a, b)|a \in A, b \in B\}$, thus E can be formally defined as $E \subseteq \{V \times V\}$. V and E are commonly known as the vertex and edge sets of G, respectively[34].*

Depending on whether or not E is a set of ordered or unordered pairs of vertices from V, G is either called a *directed graph* (ordered set), or an *undirected graph* (unordered set). For the rest of the thesis, unless otherwise stated, G will be assumed to be an undirected

graph. Figure 2.2 exemplifies the differences between an undirected and directed graph. (a) – representing a directed graph – defines the vertex set as: $V = \{a, b, c, d, e\}$, and edge set as: $E = \{\{a, e\}, \{e, b\}, \{e, d\}, \{b, d\}, \{b, c\}, \{c, b\}\}$. Conversely, (b) – representing an undirected graph – defines the vertex set as: $V = \{a, b, c, d, e, f\}$, and edge set as: $E = \{\{a, b\}, \{a, f\}, \{a, c\}, \{b, f\}, \{c, e\}, \{f, e\}, \{e, d\}\}$, with the additional property that the order of vertices in the edge set is irrelevant.



(a) Directed graph (b) Undirected graph

Figure 2.2: Directed and undirected graph types.

2.2.1 Graph Metrics

In addition to pure visualization, graphs also allows us to perform an array of measurements, some of which are introduced here.

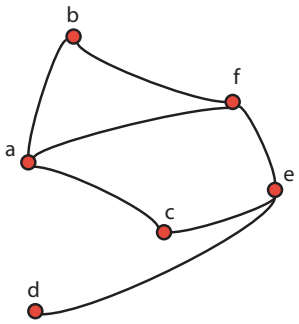
Cardinality describes the length of the vertex set, termed $|V|$, or edge set, termed $|E|$, and is useful in combination with other metrics in order to reason about graph size.

Degree, or vertex degree, represents the sum of edges that each vertex is an endpoint of, loops are counted twice[34]. The minimum degree of a graph G is denoted $\delta(G)$, while the maximum degree is denoted $\Delta(G)$. In a directed graph, vertex degree is termed either in-degree or out-degree, depending on whether or not the edge is incoming or outgoing.

Graph centrality is a concept that governs the relative importance of each vertex in V. Four of the most commonly used centrality measurements will be briefly desribed here. The first centrality measure is called degree centrality, denoted C_D, and it relates the degree of each vertex as it pertains to the whole graph. A large degree centrality symbolizes that the vertex has a high probability of being included in whatever flows through the graph. Closeness centrality on the other hand illustrates how close a certain vertex is to all other vertices, i.e. a higher closeness value means a lower total distance to all other vertices. Betweenness centrality quantifies the amount of times a vertex will be included in a shortest path between two vertices. The last centrality, eigenvector centrality, was first introduced by Bonacich in 1972[35, 36]

and have later been adapted as a central part of Google's PageRank algorithm[37]. It is similar to the degree centrality, however instead of weighing each edge equally it weighs each edge according to its centralities.

A visual example of the above metrics can be found in Figure 2.3, which uses the undirected graph from Figure 2.2.



(a) Undirected graph

$\|V\|$	6
$\|E\|$	7
$\Delta(G)$	*a*, *e*, and f: 3
$\delta(G)$	*d*: 1
$max\{C_D\}$	*a*, *e*, and f: 0.6
$max\{C_C\}$	*e*, and f: 0.714
$max\{C_B\}$	*e*: 0.45
$max\{C_\lambda\}$	f: 0.516

(b) Measurements

Figure 2.3: Graph metrics.

Note that the definitions and terminology that has been introduced here is just a small subset of the total amount that exists in literature (see chapter 11 of [34] for a thorough introduction).

2.2.2 Graph Theory and Positioning Systems

As previously hinted at, graph theory can be very useful for problems related to the representation and search of maps or networks. Coincidentally, the modelling of both indoor and outdoor spaces commonly use a graph based model to bring useful information to the user.

In a paper from 2011, Michael Worboys[38] argues how the current focus on outdoor space when modelling Geographic Information System (GIS) technologies should be extended to allow for formal models that benefit indoor spaces. Worboys presents the current state taxonomy of indoor space models as one of semantic and spatial models – where the latter includes topological, hybrid, and hierarchical models. Models that deliver a rich semantic

representation of the environment often employ different labelling approaches in order to classify (indoor) regions into higher-order concepts and domain ontologies, e.g. transitions such as doors, stairs, and lifts that separate regions[39]. Spatial models on the other hand, e.g. topological models (as well as hybrid and hierarchical models), are concerned with connectivities, and are well suited to be modelled by graphs. Worboys makes the distinction between subtypes of adjacency that are relevant when interpreting the distribution of edges. In a *connectivity graph* an edge between two vertices occur when there is a physical connection between the vertices (representing regions), e.g. a door between two boundary rooms. On the other hand, in a *accessibility graph* an edge between two vertices only occurs if the the two regions (or vertices) are accessible from another.

2.3 Combinatorial Optimization

In the most general case, mathematical optimization relates to either the maximization or minimization of a real function, i.e. finding a global value such that $f(x_i) \geq f(x)$, or $f(x_i) \leq f(x)$ for all x in A – defined as: $f : A \to \mathbb{R}$. In one special case of mathematical optimization, called combinatorial optimization, we are concerned with finding the *best* solution from a set of discrete (or discretized) feasible solutions. The *best* solution is usually called an optimal solution to the optimization problem.

Some examples of problems that are classified as combinatorial optimization problems includes the knapsack problem, Travelling Salesman Problem (TSP), minimum spanning tree, and shortest path problem. While there does exists efficient exact algorithms for solving many combinatorial optimization problems, such as the shortest path problem, many of them are only solveable – in reasonable time – by approximation algorithms. The knapsack problem and TSP are two problems where the optimization problem has been shown to be *NP-hard*, i.e. there is no known polynomial algorithm which can tell whether or not a solution is optimal, unless *P=NP*[40]. However, with approximation algorithms it is possible to find satisfying solutions for both of these problems[40](chapter 35).

2.3.1 Pathfinding

As mentioned in the previous paragraph, pathfinding or shortest path problem as it is commonly known, is classified as a combinatorial optimization problem. The problem can be formulated as finding the shortest path, or route, between two vertices. In this thesis, these two vertices belongs to a weighted graph, meaning that each edge $\in E$ is associated with a *weight*[2].

[2]Edge weights represents the cost of travelling along the edge, e.g. distance.

There are several variants of the shortest path problem, and they are usually classified as being the shortest path between one of the following:

- One source and one destination vertex (single-pair)
- One source vertex and multiple destination vertices (single-source)
- Multiple source vertices and one destination vertex
- All pairs of vertices

As we will see in later chapters the focus in this thesis will be on the single-pair shortest path problem – sometimes called point-to-point shortest path problem[41]. Incidentally, the single-pair shortest path problem can be solved by solving the singe-source problem by setting the same source vertex. The result of this is that we can utilize algorithms such as Dijkstra's algorithm for single-pair pathfinding.

Dijkstra's Algorithm

Dijkstra's algorithm is named after its creator Edsger W. Dijkstra's which first described the algorithm in 1959 as a problem of finding the path of minimum total length between two given nodes P and Q (in a graph G)[42].

Dijkstra's algorithm solves the single-source shortest path problem for a weighted graph given that all weights are nonnegative. Pseudocode for the general implementation of Dijkstra's algorithm can be found in Algorithm 1.

The general algorithm presented above, without using a min-priority queue has a worst case runtime of $O(|V^2|)$, which is caused by the linear search for the vertex with the smalles $dist()$ from the queue, Q. By implementing the queue as a Fibonacci heap the worst case runtime can get as low as $O(|V|log|V| + |E|)$[40](page 662). Note that if there is no path from s to v then $dist(s, v) = \infty$.

If we are only concerned with finding the solution to a single-pair problem, then the single-source version of Dijkstra's algorithm in Algorithm 1 can be transformed into a single-pair version by terminating the search algorithm if the next vertex in the queue, i.e. u, is equal to the destination vertex. This can be useful if we are searching in very large graphs.

One property that can be inferred from the relaxation step[3] in Algorithm 1 is the triangle inequality. The triangle inequality property states that for any edge $\in E$, we have that $dist(s, v) \leq dist(s, u)+w(u, v)$. The importance of the triangle inequality will become apparant in later chapters.

[3]A relaxation step tests whether or not we can improve the shortest path from s to v by going through u, i.e. $dist(u) + w(u, v) < dist(v)$.

Algorithm 1 Dijkstra's Algorithm: single-source shortest path

Input: A directed or undirected graph, $G = (V, E)$, a list of edge nonnegative weights, $w_e = \{e \in E : w_e \geq 0\}$, and a source vertex, $s \in V$

Output: A record of distances where $dist(u)$ represents the distance from s to u, and a record of shortest paths where $prev(u)$ represents the parent vertex of u

```
1:  procedure DIJKSTRA(G, w, s)
2:      for each v ∈ V do
3:          dist(v) ← ∞
4:          prev(v) ← undefined
5:      end for
6:
7:      dist(s) ← 0
8:      Q ← create a queue from V
9:      while Q is not empty do
10:         u ← pop vertex with smallest dist() from Q
11:
12:         for each edge (u, v) ∈ E do
13:             if dist(u) + w(u, v) < dist(v) then
14:                 dist(v) ← dist(u) + w(u, v)
15:                 prev(v) ← u
16:             end if
17:         end for
18:     end while
19:
20:     return dist, prev
21: end procedure
```

A* Search Algorithm

The A* algorithm was first described by Peter E. Hart, Nils J. Nilsson, and Bertram Raphael in 1968[43], and is a commonly used search algorithm in AI. The algorithm is often recognized as an extension of Dijkstra's algorithm, and combines information about the problem domain with a formal mathematical approach in order to find minimum cost paths. Search algorithms that exploit knowledge about the problem domain, in order to make *intelligent* choices in search space, utilize what is known as *heuristic* information, which is computed by a heuristic function. The heuristic function makes a rough estimate of the distance from any given vertex to the destination vertex, and is often symbolised as $h(v)$ where v is a vertex and the destination is implicitly stated.

Compared to search algorithms such as Depth-First Search (DFS) and Breadth-First Search (BFS), which ignore any problem-specific information and simply expand[4] vertices given a

[4]Expanding a vertex means to visit the vertex and look at its neighbours.

basic protocol. In the case of BFS, the protocol is to always expand vertices in the order that they where discovered, conversely the protocol for DFS is to expand newly discovered vertices first. A third option, the *best-first search* approach, is the common name given to search algorithms that exploit heuristics by gradually piecing together a shortest path by expanding the *best* vertices.

A* is the most commonly used *best-first search* algorithm, and it methodically selects the *best* vertices for expansion by evaluating each vertex, as seen in Equation 2.1.

$$f(v) = g(v) + h(v) \tag{2.1}$$

Here, $f(v)$ evaluates and estimates the expected cost or distance of a solution path from source, s, through vertrex v, to a destination vertex. $g(v)$ represents the cost from source, s, to vertex v, while $h(v)$ represents the heuristic, or estimate, from vertex v to a destination vertex.

Pseudocode for the A* algorithm can be seen in Algorithm 2.

Most of the general A* algorithm is quite similar to Dijkstra's algorithm, however there are some subtle differences. In addition to the bookkeeping necessary for including the heuristic function – mainly g_score and f_score – the algorithm makes a distinction between fringe vertices[5] and explored vertices. The reason for this will become clear when we discuss heuristic properties.

The EVALUATE(u, v) can be seen in Algorithm 3, and updates the $g(v)$ and $f(v)$ with respect to the heuristic function.

It is necessary to introduce two properties pertaining to the heuristic function in order to ensure the optimality of A* – a full proof can be found in [44]. The key result is that the tree search version of A* is optimal if and only if $h(v)$ is admissible, while the graph search version is optimal if and only if $h(v)$ is consistent/monotone.

Definition 6. *An **admissible heuristic** never overestimates the actual cost from v to the destination vertex, meaning that it represents an optimistic estimate, or lower bound, of the remaining distance from the current vertex, v. As $g(v)$ represents the actual cost from the source to the current vertex, v, it follows from Equation 2.1 that $f(v)$ will never overestimate the actual cost from the source to the destination vertex via v.*

Admissible heuristics are often found by relaxing, or simplifying, the problem to be solved[44]. One commonly used heuristic that is useful when the problem resides in a grid based Euclidean metric space is the city block, or Manhattan, distance. This heuristic computes a distance metric by considering the Von Neumann neighbourhood[6], of the current vertex v.

[5] The set of all vertices that have been seen but have not been visited yet is called the fringe

[6] A Von Neumann neighbourhood is defined in 2D as the vertices that orthogonally surround a vertex.

Algorithm 2 A* Algorithm: single-pair shortest path

Input: A directed or undirected graph, $G = (V, E)$, a list of edge weights, $w_e = \{e \in E : w_e \geq 0\}$ a source and destination vertex, where $s, d \in V$.

Output: A record of shortest paths where $prev(u)$ represents the parent vertex of u

```
1:  procedure ASTAR(G, w, s, d)
2:      fringe ← {s}, explored ← ∅
3:      prev ← ∅
4:      g_score(s) ← 0, f_score(s) ← g_score(s) + h(s)
5:
6:      while fringe is not empty do
7:          u ← pop vertex with lowest f_score value from fringe
8:          add v to explored
9:          if u = d then
10:             return prev
11:         end if
12:         for each neighbour v ∈ V of u do
13:             if v ∉ fringe and v ∉ explored then
14:                 EVALUATE(u, v)
15:                 add v to fringe
16:             else if g_score(u) + w(u, v) < g_score(v) then
17:                 EVALUATE(u, v)
18:                 if v in explored then
19:                     PROPAGATE-G-VALUES(v)
20:                 end if
21:             end if
22:         end for
23:     end while
24: end procedure
```

Definition 7. *A **consistent heuristic**, often called monotone heuristic, ensures that for every vertex v and every successor v_i' of v, the estimated cost of reaching the destination vertex from v is no greater than the cost of getting to v_i' from v plus the estimated cost of getting from v_i' to the destination[44]. In other words, once vertex v is explored, then $g(v)$ is considered to be the lowest possible cost from source to v. Equation 2.2 neatly encapsulates the consistency property by utilizing the triangle inequality.*

$$h(v) \leq d(v, v_i') + h(v_i') \qquad where \qquad h(destination) = 0 \tag{2.2}$$

A consistent heuristic is also admissible[44].

Many heuristics are found by relaxing problems, which basically adds edges to the search space. Norvig et al. argues that an optimal solution in the original problem is, by definition,

Algorithm 3 A* Algorithm: evaluate current and neighbouring vertex

1: **procedure** EVALUATE(u, v)
2: $g\_score(v) \leftarrow g\_score(u) + w(u, v)$
3: $f\_score(v) \leftarrow g\_score(v) + h(v)$
4: $prev(v) \leftarrow u$
5: **end procedure**

also a solution in the relaxed problem; however the relaxed problem may have even better solutions. Thus, the cost of the optimal solution in the relaxed search space is an admissible heuristic for the original problem[44](page 105). Moreover, seeing as this heuristic is an exact cost for the relaxed problem, it must also obey the triangle equality (Equation 2.2) and is therefore also a consistent heuristic. This shows that it might be difficult to find heuristic functions which are admissible but not consistent, however, if the heuristic does not fit this classification it is possible to visit already explored vertices via a path that yields a lower g value. One such situation can be seen in Figure 2.4, where vertex k is being explored by vertex v which has a lower $g(v)$ than the current parent of k, namely u.

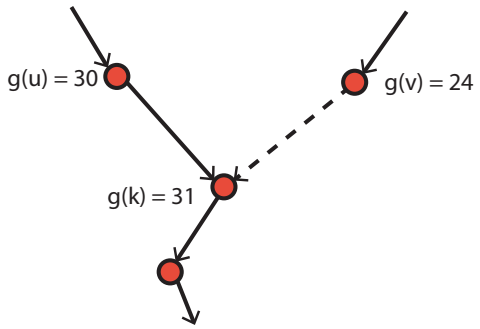


Figure 2.4: Vertex k has been found to be a neighbour of v which yields a lower $g(k)$ value (with u as the parent vertex). In this example every edge has a weight of 1.

Thus, in order to maintain the correct g values throughout the solution path the A* algorithm has to not only change the g value of the current vertex, but also all vertices that are influenced by this vertex. This is handled by the procedure PROPAGATE-G-VALUES(v) which is depicted as pseudocode in Algorithm 4.

Two of A*'s main drawbacks is its time complexity and memory usage. The runtime, or time complexity, is determined by the heuristic function, meaning that in problems that require a complex heuristic, it might be useful to accept suboptimal solutions to lower the total time complexity. On the other hand, the memory issues of A* is caused by keeping every vertex in memory, which means that A* might run out of space before it runs out of time. For this reason it can be useful to give up speed to overcome memory issues in very large-scale problems, e.g.

Algorithm 4 A* Algorithm: propagate g values along the path

```
1: procedure PROPAGATE-G-VALUES(u)
2:     for each neighbour v ∈ V of u do
3:         if g_score(u) + w(u, v) < g_score(v) then
4:             g_score(v) ← g_score(u) + w(u, v)
5:             f_score(v) ← g_score(v) + h(v)
6:             prev(v) ← u
7:             PROPAGATE-G-VALUES(p)
8:         end if
9:     end for
10: end procedure
```

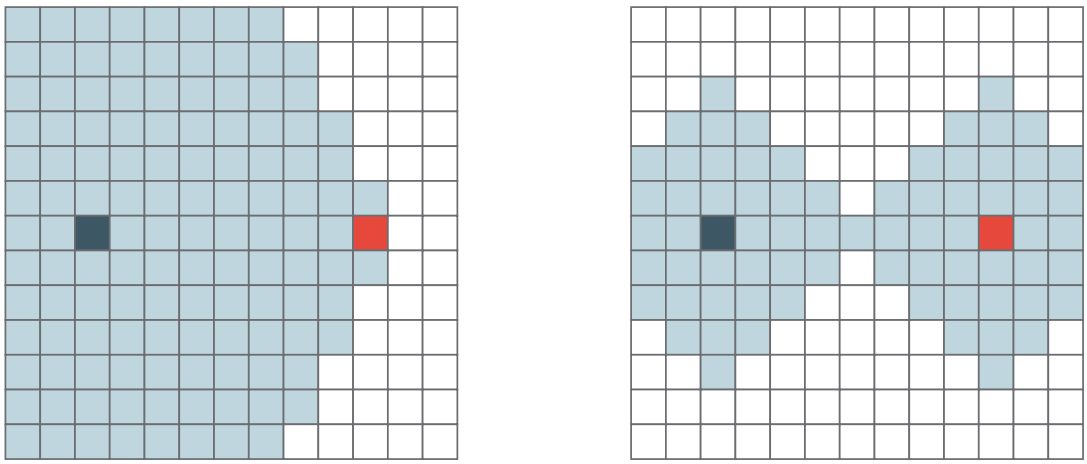
by the use of memory-bounded heuristic search algorithm such as MAWA* [45].

Bi-directional search

A common, but often difficult to handle, way of reducing the time complexity of search is to apply the idea of bi-directional search. The concept is simply to run two searches simultaneously – the first from the source vertex to the destination vertex and the second from the destination vertex to the source vertex. If or when, the two searches intersect we have a solution. Norvig et al.[44] visualizes this as two expanding trees – one from the source vertex and one from the destination vertex – effectively halving the time complexity. Assuming the time complexity of an uninformed search to be $O(b^d)$, with a branching factor b and distance from source to destination d, the time complexity for bi-directional search turns into $O(b^{d/2}) + O(b^{d/2})$, which is considerably less than $O(b^d)$. A visualization of how two such search spaces can look can be seen in Figure 2.5.

Stopping the search when an intersection at vertex u, between the *forward* and *backward* search, is found might at first glance seem like a reasonable stopping criterion. However, there is no guarantee that a path through u represents the shortest path between source and destination vertex. Extra care will therefore have to be given in order to ensure the optimality of the candidate shortest path.

An informed search version of bi-directional search using heuristics was first described in 1969 in a PhD thesis by Ira Pohl[46]. As with the uninformed version of bi-directional search, it can be implemented quasi-simultaneously by alternating between the forward and backward A* search. It requires two distinct heuristic search functions, one for the forward A* search and another for the backward A* search. The stopping criterion suggested by Pohl is based on the shortest path seen so far, μ. When either the forward or backward search explores a vertex v,



(a) Uninformed search

(b) Uninformed bi-directional search

Figure 2.5: Visited vertices during (a) uninformed and (b) bi-directional (uninformed) search. Dark blue represents the source vertex, red represents the destination vertex, and light blue represents the explored search space.

which has already been explored by the opposite search, then $\mu = min\{g_f(v) + g_b(v), \mu\}$[7]. The algorithm can thus terminate as soon as $max\{t_f, t_b\} \geq \mu$, where $t_f = min\{f_f(v)|v \in fringe_f\}$ and $t_b = min\{f_b(v)|v \in fringe_b\}$. This stopping criterion guarantees an optimal shortest path for any admissible heuristic function[47]. An alternative stopping criterion is presented by Ikeda et al.[48] and utilizes the average of forward and backward heuristic functions.

2.3.2 Acceleration Methods

The classic shortest path search methods discussed so far all have one problem in common: they do not scale with the ever increasing size of graphs. Graphs representing, for example, road maps, social networks, and even internet networks can have several million vertices and billions of edges[49]. Computing exact solutions in a timely manner can in many cases be difficult using search methods such as Dijkstra's algorithm or A*. In order to not have to rely on approximate solutions we can resort to the use of acceleration methods, most of which rely on some sort of preprocessing. Two acceleration methods will be briefly introduced here, another – the landmark method – will be described in detail in the next section.

Arc-Flag acceleration[50, 51] is a modification of the standard Dijkstra's algorithm, and avoids the exploration of unnecessary paths. The method works by partitioning a graph into sets of vertices called regions, $r \in R$, and by preprocessing a set of flags, or labels, for each arc (or edge), $a \in A$. Each arc consists of a binary vector $a(R_i)$ of size $|R|$, one for each region R,

[7] Subscript f and b represents forward and backward search, respectively.

where the flag for region R_i is set to $true$ if and only if a is either inside R_i or is on a shortest path to at least one vertex in R_i. Thus, Dijkstra's algorithm can be modified to only traverse arcs where $a(R_i)$ is $true$. The somewhat abstract notion of arc-flags allows for any kind of region partitioning scheme, e.g. using a regular grid or a k-d tree, and also allows the storage of extra information for each arc. This means that it can be applied to many different kinds of search problems.

Reach-based pruning is an acceleration method originally described by Ron Gutman in 2004[52] and was later improved by Goldberg et al. in 2006[53]. Let v be a vertex that lies on a path P with source s and destination t, then the reach of v with respect to P is defined as $min\{dist(s,v), dist(v,t)\}$. The reach of v with respect to the whole graph is defined as the maximum reach of v for all shortest paths P containing v. Reaches can be used to prune vertices when performing, for example, Dijkstra's algorithm by doing a check. Assume we are currently processing edge (u,v), then we can prune v if the reach of v is less than $min\{dist(s,v) + dist(u,v), LB(v,t)\}$[41], where LB represents a lower bound based on, for example, Euclidean distances[52] or landmarks[53]. The idea of reach is motivated by road networks where vertices on highways exhibit high reach, while vertices on local roads have low reach. The problem with reach-based pruning is the enormous preprocessing time for exact reach computation, ranging from multiple hours to years (without using heuristics).

The different types of methods discussed here represent only a fraction of the total amount of acceleration methods out there. In the last few years a great deal of research has gone into methods that exploit graph hierarchies, such as the previously mentioned reach-based pruning, highway hierarchies[54], contraction hierarchies[55], and hybrid methods[53]. A survey of these and other state-of-the-art route planning algorithms can be found in a paper by Delling et al. from 2009[56].

Landmark Method

A landmark is a simple, yet powerful data structure that does not rely on any domain-specific knowledge. A set of landmarks can be computed by storing shortest path distances between all vertices and each of the landmarks. Thus, a landmark can be seen as the result of a single-source shortest path algorithm, such as Dijkstra's algorithm. In essence, landmarks provide search methods a way to peek at large areas of the search space in constant time.

Proper landmark selection is important in order to ensure a robust set of landmarks[57]. In many situations it might be beneficial to select landmarks based on domain-specific knowledge however, this is not mandatory. In general, we are interested in a constant set of landmarks, $\ell \in V$, that represents points of interest in a graph. For example, a simple landmark selection algorithm could be n randomly selected vertices from a graph. Assigning n vertices with the

highest degree or lowest closeness centrality as landmarks is another selection algorithm that ensures better distance estimates[49]. Tretyakov et al.[49] suggests selecting landmarks that *covers* as many vertex pairs as possible. A landmark ℓ is said to be *covering* a vertex pair (u, v) if ℓ lies on the shortest path between u and v; this is a classic *NP-hard* optimization problem in computer science. The approach presented by Tretyakov et al. is a greedy strategy that *(i)* samples M vertex pairs and *(ii)* selects vertices that are present in most paths of the sample. Other landmark selection algorithms often apply different partitioning approaches that try to balance performance with robustness[58].

Landmark A*, often called ALT, was first presented in 2005 by Goldberg et al.[57] and combines A*, landmarks, and the triangle inequality in order to compute optimal shortest paths. The technique applies the triangle inequality in order to compute tight lower bounds based on landmarks. The lower-bounding technique does not rely on Euclidean space and can therefore be used in many different kinds of search problems. Let s and t represent the source and destination vertex, respectively, and let u be the current vertex, and finally let l be the nearest landmark to u. A lower bound, or heuristic, from u to t can then be computed as $max\{dist(l,t) - dist(u,l), 0\}$. This is a guaranteed lower bound on the length of the shortest path from s to t because from the triangle inequality we have that $dist(l,t) \leq dist(s,l) + dist(s,t)$, and thus $dist(s,t) \geq dist(l,t) - dist(s,l)$.

Chapter 3

Requirements Engineering

This chapter is more or less divided into two parts. The first part describes the RE methodology used for gathering requirements based on the background knowledge from chapter 2. The second part presents the result of the RE process.

3.1 Methodology

The focus of our RE study is to explore and understand use cases where time spent looking for wheelchairs can be reduced; this is encapsulated by **RQ1** from chapter 1. We established two domain contacts early on in the research process. The first domain contact, Hans Kottum – Head of Section of the porters at St. Olavs Hospital – served as our contact with St. Olavs Hospital and allowed us to perform an extensive requirements eliciation study. The second domain contact, Thomas Jelle, is the CEO at MazeMap and Wireless Trondheim which provides St. Olavs Hospital its IPS.

The RE study was performed using the RE methodology defined in chapter 2, and consists of the core activities illustrated in Figure 2.1. The techniques and methods used for each of the core activities will be detailed in the remained of this section.

Two eliciation techniques were used in the eliciation activity. The first is an observational techique called field study – a direct observation technique – where the author of the thesis assumed the role of a porter for two days. The first day started of with an introduction to what the work of a porter consists of, and during the rest of the day I accompanied a few different porters around St. Olavs Hospital on different jobs (or tasks). The second day was mainly used for talking to different porters and inspecting different base stations around the hospital. Base stations are a new initiative at St. Olavs Hospital where wheelchairs can be

borrowed by hospital visitors by inserting a coin[59] – much like grocery shopping carts[1] Notes on ideas and observations was written down during and after both field study days. The overall goal of the field study was to identify wheelchair availability and usage from the perspective of porters. The second eliciation technique we performed was a series of semi-structured interviews with both of the domain contacts. The goal of the interviews with our domain contact at the hospital was to elicit wishes, needs, and other desirable qualities a wheelchair location tracking system should or must have. The goal of the interviews with our domain contact at Wireless Trondheim was to recognize constraints of the IPS and other needs. While most of the eliciation focused interviews were performed early on in the research process (August-September of 2013), some of them were also performed at a later date for demonstration and validation purposes.

The raw requirements, notes, constraints, desirable qualities, and other data from the eliciation activity were condensed into a set of use cases and requirements by assessing and refining the elicitation data. This process mainly consisted of organizing the data into requirements, using natural language, and use cases, using conceptual modelling. The condensed set of requirements and use cases was then presented to our domain contacts for feedback. The processed data were managed by keeping requirements in a table format using the spreadsheet application LibreOffice Calc, while the conceptual models were drawn using UML notation in the vector graphics editor Inkscape and the UML tool UMLet[2] The rest of this chapter details the results of the RE activities.

3.2 Context and Stakeholders

We recognize three groups of stakeholders that may influence the proposed system during its life cycle. The first group are the porters, administrators, and other staff members at St. Olavs Hospital, where the porters are the main users of the system. The second group is the IPS provider, that in our case is Wireless Trondheim. The third and last group is the patients or visitors of the hospital, and represent an important part of any evaluation of the system. Any system that is able to reduce the time porters spend looking for wheelchairs will also reduce the waiting time of patients and visitors, thus the members of the last group will be able to *feel* the effects of the system indirectly through the porters. In a general implementation the stakeholders are the hospital, IPS provider, and hospital visitors.

A simplified overview of the system and sociotechnical context boundaries are depicted in Figure 3.1. Any proposed system using the IPS at St. Olavs Hospital must reside within the

[1] There are currently a total of three base stations at St. Olavs Hospital. They are located at the main entrances of Gastrosenteret, Nevrosenteret, and Akutten og Hjerte-lunge-senteret.

[2] LibreOffice, UMLet, and Inkscape are free and open source.

context of the IPS, hence the placement of the system in Figure 3.1.

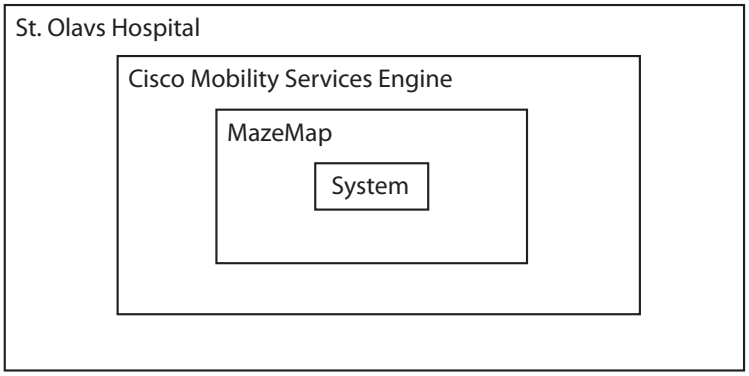


Figure 3.1: System and context boundaries.

MazeMap is an IPS provided by Wireless Trondheim and uses trilateration to find WLAN devices using wireless access points. It is currently available at NTNU, St. Olavs Hospital, and University of Tromsø - The Arctic University of Norway (UiT). A mapping of all buildings, and a-to-b walking paths based on access levels are among some of the features MazeMap provides its users. In a customer case study from 2013 it was found that MazeMap reduces the stress of patients, visitors, and employees and reduces the workload at information desks by making it easier to navigate buildings[60]. MazeMap relies on geometric positioning data from the Cisco Mobility Services Engine (Cisco MSE) and provides its users with a graphical user interface on desktop and mobile devices. Cisco MSE is a system that determines the location of a person or object with an active Wi-Fi device, i.e. a WLAN device, or an RFID tag by utilizing the already existing wireless infrastructure. It uses the two technologies Received Signal Strength Indication (RSSI) and Time Difference of Arrival (TDOA) to track the location of Wi-Fi devices in indoor and outdoor environments, respectively. RSSI determines the location using the perceived signal strength of three or more access points, while TDOA determines the location based on the difference in time of arrival of signals as seen by three or more synchronized access points.

The outermost boundary in Figure 3.1 describes the context in which both the IPS and the proposed system is intended to be used. Information systems that operate in hospitals and healthcare are sociotechnical systems where multiple people with different responsibilities work together. The importance of recognizing the sociotechnical aspects of a system can be visualized in the triangular representation by Steven Alter called the work system method[61]. In this method a system can be understood as a separation of the process, i.e. participants such as nurses, porters, and doctors, information and data, and technologies, versus the outcome of the process, e.g. patient care. It is quite common to think on information systems in isolation,

therefore, disregarding the impact the system has on its participants (directly or indirectly) and the information is uses or produces.

3.3 Requirements Elicitation

A summary of the requirements eliciation will be split up into two parts. The first part will present observations and information from the field study and interviews with our domain contact at St. Olavs Hospital. The second part will present information from the interviews with our domain contact at Wireless Trondheim.

St. Olav Hospital has around 9600 employees (9584 in 2012) and is a 1008-bed hospital with 7 outpatient centers in Trondheim. In 2012 it had a total of 554083 outpatient consultations and 131547 inpatients. 37 of the hospitals employees are porters that handle (on average) 500 logistical jobs/tasks each day. A logistical task may include but is not restricted to helping visitors and patients arrive on time for their appointments, answering emergency calls, and other logistical tasks not involving patients. Porters receive tasks via a IP phone system in real-time from the porter base station at the information desk on the first floor of Akutten og Hjerte-lunge-senteret. One or more operators are responsible for receiving and assigning tasks from patients and visitors to porters based on their current location. The current job dispatching system is provided by Imatis[3] and uses a separate location tracking infrastructure, based on IP phones, from MazeMap. Tasks are received by porters in the form of textual messages that describe the building, floor, department, corridor, room, and bedpost that the patient or visitor is located and where he or she should be transported.

There is no common repository of wheelchairs, and each department is responsible for purchasing, maintaining, and keeping track of their own wheelchairs. However, wheelchairs are used between departments by either department staff or by porters that need a wheelchair for their current task. Compared to hospital beds, the number and condition of wheelchairs are not known and difficult to determine; hospital beds also have a system in place for tracking their movements. Regardless of this the porters try to take responsibility and coordinate wheelchairs in order to minimize the chances of missing wheelchairs. Despite this it is not uncommon to lose track of wheelchairs[4], thus many department employees find it difficult to lend out wheelchairs across departments. Wheelchairs that have been lost track of are usually not found again. The typical life cycle for wheelchairs is the following: In most cases a wheelchair is requested and delivered to the polyclinic where they disappear for a couple of hours before resurfacing somewhere again. Most wheelchairs resurface at the Nevrosenteret

[3] Imatis is a software company and delivers solutions designed for the healthcare industry.

[4] It is not unheard of that hospital visitors take wheelchairs with them home, thus stealing a wheelchair either knowingly or unknowingly.

as it is a popular pickup area – taxis can be found here.

From the interviews with our domain contact, Hans Kottum, we were able to gather that the IP phone system used by the porters has a poor temporal resolution. It might take a few minutes, one minutes and more was frequently observed, before a porter receives the task sent by the operator; a time difference which invalidates the position observed by the operator. The current granularity, or spatial resolution, of task descriptions can be up to a room level however, hospital branch and corridor level is considered good enough by the porters we talked to. The newly implemented wheelchair base stations mentioned in section 3.1 have alleviated some of the issues related to difficulties in finding wheelchairs, however, due to the skewed demand of departments some of the base stations are more likely to be empty. While the exact number of wheelchairs is not known our domain contact expects that there are a couple of hundred wheelchairs that are of location tracking interest. i.e. they are in working condition. Our domain contact estimates that each porter uses 10 to 15 minutes on each shift for procuring wheelchairs – this is mainly done by guesswork. There are around 25 available porters each day, meaning that a total of 312.5 minutes or around 5 hours are used looking for wheelchairs each day on average.

MazeMap receives geometric positioning data from Cisco MSE relative to building floors, e.g. (x, y) raw position data for floor F. This data is combined with GPS information in order to improve accuracy of user requests that are done outside. The combined positioning data from Cisco MSE and GPS is filtered and then presented to the user using proprietary techniques. Best case spatial accuracy is typically less than 2 meters and on average it is 5 to 10 meters, however, it can get up to 20 meters depending on access point and WLAN device location. MazeMap currently supports map display, the ability to find your own position, a-to-b walking paths, and area usage statistics reports. The current system is able to display any item on their map as long as it has some sort of WLAN device, e.g. smart phone; however, in order to track the location of specific items a module that is able to query them would have to be designed and implemented.

3.4 Use Cases

Analyzing the information from the requirements elicitation we propose three use cases that can be valuable for reducing time spent looking for wheelchairs. The use cases can be seen in Figure 3.2. It became evident early on that porters can save unecessay time spent looking for wheelchairs by having them plotted on a map. Finding those wheelchairs that are *near* and *available* can be seen as an extension of this. For simplicity, all use cases are drawn independently to illustrate that they can be implemented separately.

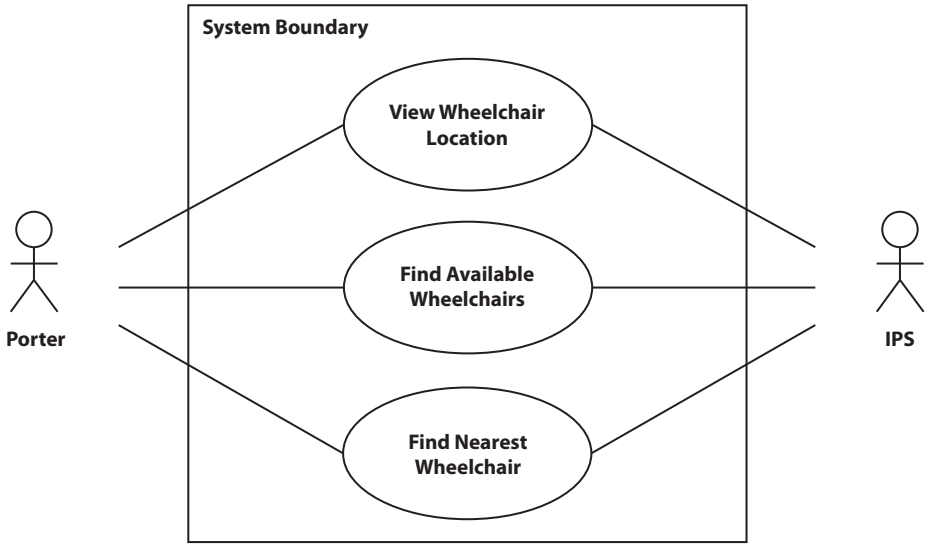


Figure 3.2: Use case diagram for location tracking of wheelchairs.

3.4.1 Actors

Two actors can be seen in Figure 3.2, the porter and IPS. The first actor, the porter, is the typical user that is interested in the location, status, and nearness information about wheelchairs in and around a hospital. The IPS is the entity which is able to answer the questions asked by the porter. In our case the IPS is MazeMap by Wireless Trondheim and the porters handle logistical tasks in the context of St. Olavs Hospital.

3.4.2 View Wheelchair Location

The first use case that became apparent to us during the requirements elication was the *View Wheelchair Location* use case, and it allows porters (and other relevant stakeholders) to view the location of *all* wheelchairs on a map. This use case was deemed highly desirable by the porters, and our porter domain contact predicted that this would greatly decrease the time spent looking for wheelchairs. Such a system would also grant beneficial side effects, such as reducing the amount of wheelchairs that are not in the correct hospital branch, stolen, or otherwise lost.

During a meeting with our domain contact at Wireless Trondheim we were informed that the IPS at St. Olavs Hospital already has the ability to display items that have a WLAN device, e.g. smart phone or a computer. We came to the conclusion that we can classify the use case as trivial to implement, however, the initial cost of WLAN devices – one for each wheelchair – is yet to be determined.

3.4.3 Find Available Wheelchairs

The second use case from the the top of Figure 3.2 describes the ability to find *available* wheelchairs, for example, by displaying them on a map such as MazeMap, or by getting textual information via the IP phone system used by the porters. We will formally define *available* wheelchairs as a wheelchairs that can be used, i.e. does not require repair, and is not currently in use by, for example, other patients or visitors. This use case has a more narrow focus than the one presented above as it prunes away wheelchairs that are either in use or otherwise deemed unusable. However, it can answer questions such as: are there any wheelchairs in the process of leaving the hospital grounds? As we found out during the eliciation, the theft of wheelchairs either knowingly or unknowingly, is not unheard of, thus it can be valuable to be able to answer such a question.

The problem of finding *available* wheelchairs is a difficult one, and it is not simply the case of finding out if the wheelchair is not motion or not. Even humans, with access to visual information, can have great difficulty with deriving the correct status of a wheelchair. On the other hand, machines, or computers, have great difficulty with deriving even the most basic understanding of sensory information, let alone being able to do this in real-time for, potentially, thousands of wheelchairs. There are a great many questions that have to be answered before we can say with confidence that this wheelchair is available. Is the wheelchair moving? Is it moving with or without a passenger? Is the passenger sitting still in the wheelchair, for example, watching TV? Is the wheelchair in the ownership of a patient, but currently not used, e.g. it is standing beside a bed? While some of these questions migth be answerable by recognizing the context of where the wheelchair in question is currently positioned, it does not diminish the fact that we (porters) must know the answers to these questions before we decide to take the wheelchair.

3.4.4 Find Nearest Wheelchair

The third and last use case in Figure 3.2 describes the ability to find the nearest wheelchair to a given location. Note that this question does not say anything about the availability of the nearest wheelchair; however it can be combined with, for example, the use case that finds available wheelchairs. This use case offers an endless amounts of customizability where the

nearest wheelchair can be found given, for example, personal preference for each individual porter and wheelchair type. Due to the way the job dispatching system at St. Olavs Hospital works it is natural to consider the problem of finding the nearest wheelchair relative to the current porter job. If a porter is on his or her way to a patient or visitor and is asked to bring a wheelchair it would be beneficial to pick up a wheelchair *on the way* to the task. Wheelchairs are typically only needed by porters during a task, thus we predict that if the porters not only have an overview over where the wheelchairs are but also have information about which wheelchair would be beneficial to use, then the time spent looking for wheelchairs could be reduced even further than the *"View Wheelchair Location"* use case. In addition, this use case would also gain all of the beneficial side effects of the *"View Wheelchair Location"* use case as it requires most, if not all, of its functionalities. Such a system could send wheelchair suggestions, including their location, along with the textual information that is already sent over the IP phone system. An alternative solution is a path display system that draws paths on, for example, the MazeMap frontend; this could be displayed to porters by using smart phones.

Due to the numerous benefits that accompany this use case and its scaleable complexity we argue that this concept is suitable for further investigation; therefore, this section, and the rest of the thesis, will be dedicated to the analysis of this use case.

3.5 Requirements

Based on the description of the use cases above, a set of requirements have been compiled in Figure 3.3. The requirement name is constructed using a slightly modified version of the briefly mentioned requirement template from chapter 2. As mentioned by the previous paragraphs the thesis will concern itself with the use case *"Find Nearest Wheelchair"*, thus only requirements for this specific use case will be itemized here.

A list of the system requirements pertaining to the *"Find Nearest Wheelchair"* use case can be seen in Figure 3.3. Notice that the requirements does not specify in which way the results of the system should be presented to the user, i.e. the user interface. Beyond what was mentioned in section 3.3, any discussion about the user interface is beyond the scope of this thesis.

The most obvious requirement, Req-1, is necessary in order be able to deliver a functioning system – it is taken for granted. Req-1 is the only requirement that uses the term *"will be able to"*, hence it is also the only mandatory requirement; all the other requirements supplement and enhance Req-1 to the users' delight. Req-1 defines the quality of the solution as the *most efficient path*, it therefore follows that the path or paths the system outputs to the user should be an exact, or at least a close approximate, solution.

Developers must be able to utilize the current state-of-the-art acceleration and search

Identifier	Name	Type
Req-1	The system will be able to find the most efficient path from the current position of the porter to the task end point via one of a set of wheelchairs	*functional*
Req-2	Due to the rapidly growing literature on new and better search methods the system should provide the system developers with a clear interface that separates the search from the search method	*functional*
Req-3	The system shall be able to rank paths according to their efficiency	*functional*
Req-4	The system should be able to find the nearest wheelchairs within a maximum of 30 seconds	*quality*
Req-5	The system should scale well with increasing hospital sizes and increasing amounts of wheelchairs	*quality*
Req-6	The system should be able to finish any necessary preprocessing offline	*quality*
Req-7	The system should be able to work with the current IPS at St. Olavs Hospital	*constraint*

Figure 3.3: System requirements.

methods and not be restricted to the current status quo. Req-2 ensures that there exists a clearly defined interface that separates the search framework from the search method.

Req-3 provides porters with the ability to select paths from personal preference and experience rather than forcing a path upon them. The system should act as a tool by helping porters find suitable wheelchairs and not force them to select any one specific wheelchair[5]. In other words, the porter should always be in control over which wheelchair is selected. Req-3 is designed to decrease the likelihood of frustration and increasing the system usage after implementation.

Req-4 is perhaps the most important of the quality requirements and is an aspect of the usability of the system, i.e. the system's ease of use. Porters are always on the move and are not able to wait around for wheelchair suggestions that take too long to compute when they could have just as easily found one by themselves. The current IP phone system used for distributing tasks to porters already has a considerable delay, during which the porter is likely to either (i) wait around for another task or (ii) make their way to the base station. As was

[5]This can be defined as striking a balance between informal and formal information systems in software engineering literature. Informal information systems are often recognized as very chaotic and allows each individual to choose for themselves. On the other hand, formal information systems are very restrictive and does not allow for any individual choice.

mentioned in section 3.3, porters were quite frequently able to return, or almost return to, the porter base station before receiving a new task. Another motivation for Req-4 is the temporal resolution problem, where the longer the computation takes the greater the results margin of error becomes. In other words, the margin of error starts to increase from the time the system recognizes the porter's position to the results have been computed – a porter's position may change within this time.

Req-5 refers to the system's ability to handle growing amounts of work such as, large hospitals and increasing amounts of wheelchairs. System scalability is directly related to Req-4 as it influences the system's ability to solve the task set in Req-1 in a timely manner. Req-5 only specifies that the system should scale *well*, thus leaving it up to the developer to decide the actual implementation, allowing the system to use the current state-of-the-art acceleration and search methods.

Req-6 is a complementary to Req-5 and is important for any hospital where system down-time should be minimized. Req-6 ensures that any preprocessing necessary for the system to work, should be able to run *offline*. By offline we mean that the preprocessing step should not interfere with Req-1, i.e. preprocessing runs asynchronously of the system. Furthermore, Req-6 acknowledges the need for acceleration methods as a means to ensure system scalability.

The last requirement, Req-7, ties the system proposed here and the already existing system at St. Olavs Hospital together. It is the only constraint requirement and is important in order to correctly place the system in the context described in section 3.2. The system can be generalized for *any* hospital by relaxing Req-7 and ensuring that the system should work with any IPS at any hospital; however, for the purposes of this thesis we will restrict our scope to St. Olavs Hospital.

Chapter 4

Methods and Implementation

This chapter describes the problem of finding the nearest wheelchair and proposes a search method, or algorithm, that can solve it. A discussion of the term *length* will be given alongside our explanation of the datasets that will be used to evaluate our approach.

4.1 Problem Description

The problem description will be based on the use cased termed *"Find Nearest Wheelchair"* and the discussion presented in chapter 3.

Porters receive jobs/tasks via a job dispatching system that is controlled by one or more operator. Tasks are dispatched given porter's position and will in most cases include (i) the starting point of the task, e.g. the location of a hospital bed, (ii) the destination point of the task, e.g. a specific hospital ward, and (iii) any extra information such as *"bring wheelchair"*. If this is the case then it would be beneficial to pick up the wheelchair on, or close to, the path to the patient/visitor. Thus, the problem can be reformulated as a minimization over the set of all wheelchairs subject to the shortest path from the porter to the starting point of the task.

Let $W = \{w_1, w_2, ..., w_n\}$ be the set of all wheelchairs, s be the current position of the porter, t be the starting point of the task, and $P_{s,t} = (v_1, v_2, ..., v_m)$ be the shortest path between s and t, then the nearest wheelchair, w^*, with respect to $P_{s,t}$ can be defined as the wheelchair that minimizes the sum in Equation 4.1.

$$w^* = \min_{w} \sum_{i=1}^{n} (d(P_{s,w_i}) + d(P_{w_i,t})), \qquad \forall_i \, w_i \in W = \{w_1, w_2, \ldots, w_n\} \tag{4.1}$$

P_{s,w_i} is the shortest path between s and w_i, $P_{w_i,t}$ is the shortest path between w_i and t, and w^* is the optimal, or nearest, wheelchair with respect to $P_{s,t}$.

The correctness of Equation 4.1 can be proved by considering the triangle inequality: $d(P_{s,t}) \leq d(P_{s,w_i}) + d(P_{w_i,t})$. The closer w_i lies with respect to $P_{s,t}$, the more equal $d(P_{s,w_i}) + d(P_{w_i,t})$ becomes to $d(P_{s,t})$. If $d(P_{s,w_i}) + d(P_{w_i,t})$ should become equal to $d(P_{s,t})$ then w_i must lie on the shortest path between s and t itself. Thus, w^* must represent the nearest wheelchair to $P_{s,t}$.

A visual representation of the problem can be seen in Figure 4.1. It becomes self-evident that any path via one of the wheelchairs from W must be equal to or greater than P.

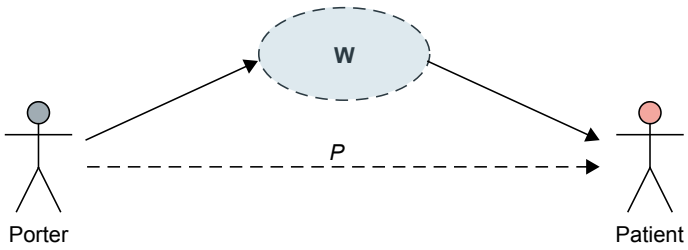


Figure 4.1: Visualization of the nearest wheelchair problem.

The nearest wheelchair problem can be generalized to the Travelling Salesman Problem (TSP) by relaxing the problem constraints and reformulate the problem to ask for the shortest path via *all* wheelchairs in W. As mentioned in chapter 2 the optimization version of TSP has been shown to be in NP-hard but not necessarily in NP, so it is not in NP-complete[62]. With that said, due to the constraints on the nearest wheelchair problem it might not be necessary to resort to approximation algorithms in order to achieve complete and optimal solutions in a timely manner.

4.1.1 Nearest Wheelchair Algorithm

In this thesis we propose a direct approach for solving Equation 4.1. By letting Equation 4.1 be our search framework the minimization can be solved by selecting an appropriate search method that computes P, the shortest path between two vertices. This means that the framework runs each search method twice for every wheelchair. Referring back to the requirements in Figure 3.3, any algorithm solving the problem must be able to deliver exact, or near approximate solutions (Req-1), must separate the search framework from the search method (Req-2), and must be able to rank paths according to their efficiency (Req-3). This search algorithm fulfils Req-3 by making the ranking of wheelchairs a natural part of the framework, Req-2 is achieved by letting the actual shortest path search method be up to the developer. Req-1 will in this case be directly dependent of Req-2, thus the *most efficient path* will be relative to the

selected search method. Furthermore, the lack of constraints put on the selection of a search method means that just about any state-of-the-art search method is applicable.

Our approach to solve the nearest wheelchair problem requires that the IPS is able to (i) query the location of all wheelchairs in real time and (ii) has a sufficiently high enough spatial resolution. The first demand placed on the IPS concerns the quality of the temporal resolution, which indicates how long time the IPS needs to detect changes to the location of WLAN devices. This can have considerable effect on the quality of the solution given by the algorithm such as being able to select wheelchairs that are already in use. The second demand concerns the level of detail the IPS is able to report. This can be important in edges cases where the wrong level of detail can lead to considerable detours.

Some implementational issues will not be considered by this thesis, such as the constraint that a porter should not have to use stairs after having retrieved a wheelchair. Such problems are often easy to solve and usually boils down to simple checks in the implementation.

A prototype of the search method has been implemented in Python, version 2.7, and a technical overview of the implementation, along with an example, can be found in appendix A. The rest of this section will consider strengths and weaknesses of potential search methods, and they will be put up for scrutiny with respect to the quality requirements in Figure 3.3 – Req-4, Req-5, and Req-6 – in later chapters.

Dijkstra's Algorithm

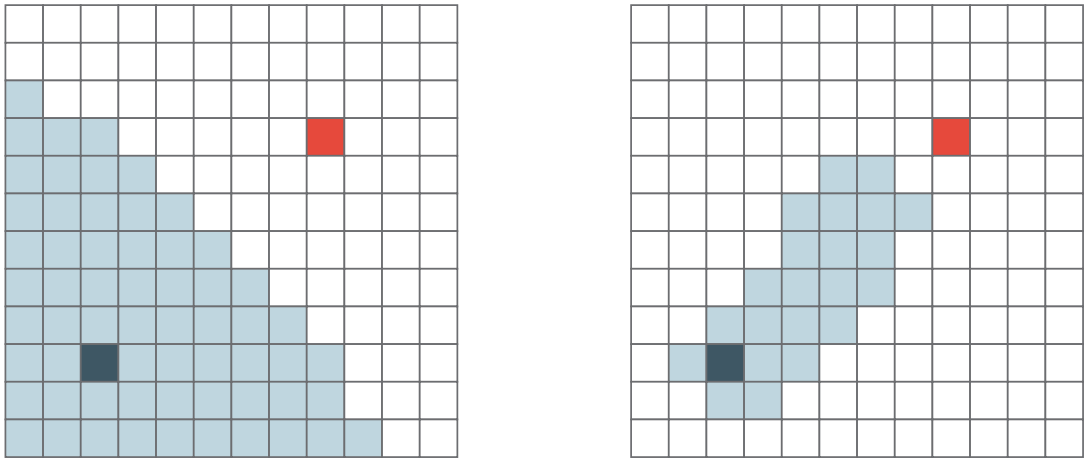
Dijkstra's algorithm is the obvious choice for any shortest path problem, and the single-source version can easily be transformed into a single-pair version by terminating the search after the target has been found. Dijkstra's algorithm falls into a class of algorithms known as uninformed search algorithms, hence no distinction is made between vertices.

Dijkstra's Algorithm with Landmarks

Dijkstra's algorithm can easily be extended with the landmark acceleration method by considering each landmark as structure that allow the search algorithm to quickly jump around the search space. Each landmark consists of shortest path information to every vertex in a graph from the landmark vertex, this means that a landmark also contains the shortest path to the target vertex. Dijkstra's algorithm with landmarks can therefore be implemented by *jumping* automatically to the target vertex if a landmark is encountered, i.e. if it is the currently inspected vertex. The problem with this approach should be obvious as even though the path from the landmark to the target is optimal, the path from the source vertex to the landmark is not. Our implementation tries to reduce these types of errors by terminating the search only after a second, better landmark is encountered.

A* Algorithm

The A* algorithm is an informed version of Dijkstra's algorithm and is implemented alongside a heuristic function that estimates the distance from the current vertex to the target vertex. The heuristic function differentiates the vertices that the search can branch to from the current vertex. This is different from Dijkstra's algorithm which in the naïve case treats all vertices equally. Consequently, the explored search space differs for uninformed and informed search algorithms. This can be seen in Figure 4.2 where the uninformed search algorithm expands vertices in a sphere around the source vertex, while the informed search algorithm expands vertices *towards* the target vertex. Note that the behaviour in Figure 4.2 (b) depends on the properties of the heuristic function being used. Strictly speaking, no correlation between explored search space and runtime exist[1], however it is assumed that informed search algorithms have better scalability and thus perform better in larger search spaces.



(a) Uninformed search

(b) Informed search

Figure 4.2: Visited vertices during (a) uninformed and (b) informed search. Dark blue represents the source vertex, red represents the destination vertex, and light blue represents the explored search space.

As mentioned in chapter 2 the A* algorithm is optimal as long as the heuristic function is admissible and consistent for tree and graph search, respectively. Three different heuristic functions were implemented in this thesis.

The first heuristic we implemented is based on the ALT algorithm (A* and landmarks) first proposed by Goldberg et al.[57] in 2005 that we have termed *TE1*. The technique uses the triangle inequality on landmarks in order to compute lower bounds on distance estimates. The

[1]The runtime, or time complexity, of the A* algorithm is constrained by the heuristic function, i.e. a computationally heavy heuristic function will yield a higher runtime, and vice versa.

heuristic function can be seen in Equation 4.2. $h(u, v)$ is the heuristic estimate from u – the current vertex – to v – the target vertex – and l is the closest landmark to u. Finding l can either be done by (i) a linear search of all landmarks in ℓ or (ii) by pre-computing the nearest landmark for every vertex in the landmark generation phase. Our implementation uses the second option. *TE1* is both admissible and consistent due to the triangle inequality, the proof of which can be seen in the last section of chapter 2.

$$h(u, v) = max\{d(l, v) - d(u, l), 0\} \tag{4.2}$$

Our second heuristic termed *TE2*, is a novel extension of *TE1* and produces an ever tighter lower bound by considering the ideas of *TE1* symmetrically. The heuristic function can be seen in Equation 4.3. The definitions are the same as for *TE1* except for l which in *TE2* is defined as l_u and l_v which represents the closest landmark to u and v, respectively. Admissibility and consistency are both natural properties for *TE2* as they are for *TE1*.

$$h(u, v) = max\{d(l_u, v) - d(u, l_u), d(l_v, u) - d(v, l_v), 0\} \tag{4.3}$$

The third and final heuristic is the commonly used Euclidean distance heuristic, termed *ED*, and naturally only works in Euclidean space. In other words, compared to *TE1* and *TE2*, *ED* only works if vertex coordinates are available. The heuristic function for a two-dimensional Euclidean space can be seen in Equation 4.4. Notice that we decided to keep the expensive square root operation in order to preserve admissibility. If the square root operation is removed the distances returned from $h(u, v)$ will be much higher than the $g(u)$ cost function which in turn will cause overestimating the actual distance. The Euclidean distance is multiplied by $(1 + 0.00001)$ in an effort to avoid situations with equal edge costs. By scaling the heuristic slightly upwards the informed search will prefer to expand vertices closer to the target. As long is the upward scaling is small enough the admissibility property will still apply. The *ED* heuristic function is naturally consistent.

$$h(u, v) = h(v, u) = \sqrt{(v_x - u_x)^2 + (v_y - u_y)^2} \times (1 + 0.00001) \tag{4.4}$$

Bi-directional Search Algorithms

A bi-directional search executes two searches, one from the source to the target called a forward search and one from the target to the source called a backward search. The general idea is that the two searches will meet in the middle somewhere more quickly than a forward search will reach the target. We have implemented a bi-directional version of both the regular Dijkstra's algorithm and the A* algorithm. The stopping criterion is the same for both

algorithms, where the algorithm terminates if the currently expanded node has been explored by both the forward and backward searches. The algorithms have been implemented to run quasi-simultaneously by alternating between the forward and backward search.

4.2 Landmarks

As previously described, a landmark is a data structure that contains single-source shortest path information for a set ℓ of n vertices from V. Landmarks have a number of applications, including a way to compute heuristics for non-Euclidean space as seen in the previous paragraphs. Our implementation builds n landmarks given a graph $G = (V, E)$ by running the single-source version of Dijkstra's algorithm for n vertices and storing the solution in an associative array, or dictionary, consisting of a collection of key-value pairs. A second dictionary is pre-computed to keep the nearest landmark for each and every vertex in G. The second dictionary speeds up the operation of finding the nearest landmark for vertex u and v in the heuristic function *TE1* and *TE2*. Dictionaries allow for easy access with an average *get item* time complexity of $O(1)$ and amortized worst case of $O(n)$[63]. Two questions have to be answered in order to use landmarks:

- Which landmark selection algorithm should be used?
- How many landmarks should be constructed?

The first question will be described in more detail in the next paragraph, while the second will be thoroughly tested in the next chapter.

4.2.1 Selection Algorithms

Arguably the most important aspect of any landmark data structure is the landmark selection algorithm. For example, *TE1* and *TE2* will most likely calculate worse distance estimates if every landmark in ℓ are close together. A landmark should represent a robust point of interest in a graph, and can in some cases be constructed using domain-specific knowledge. There are two important aspects of landmark selection algorithms, the first is the landmark building runtime or time complexity, and the second is the landmark quality or robustness. Finding a suitable selection algorithm must involve a careful balancing between these two aspects. There might not be a correlation between landmark build time and robustness, thus it is important to test selection algorithms in order to find the one most suitable for the current problem. We have implemented four different kinds of landmark selection algorithms, each of which will be thoroughly tested in the next chapter.

The first and most basic form of landmark selection we have implemented is a uniform selection of random landmarks, and is implemented by selecting n unique vertices from V uniformly.

The second selection algorithm we have implemented is the selection of n vertices with the highest vertex degree, i.e. $\Delta(G, n)$. This is a common landmark selection algorithm and can be found in the paper by Tretyakov et al.[49].

An intuitive way of selecting points of interest on a graph is to select the n vertices with the highest centrality, and one of the most suitable of the centralities for this task is the eigenvector centrality. Equation 4.5 describes the eigenvector centrality of a vertex x in two ways, as an (adjacency) matrix equation and as a sum[36]. λ is the largest eigenvalue of A and n is the number of vertices in V, where the eigenvector centrality of a vertex x is proportional to the sum of the centralities of its neighbours. Our third landmark selection algorithm selects the n vertices with the highest eigenvector centrality.

$$Ax = \lambda x, \qquad \lambda x_i = \sum_{j=1}^{n} a_{ij} x_j, \qquad i = 1, \ldots, n \tag{4.5}$$

Tretyakov et al. describes a selection algorithm that samples a set of M vertex pairs, computes the shortest path for each pair, and selects the vertices that are present in most paths of the sample[49]. The idea behind the algorithm is to select the vertices that covers most of the shortest paths, hence the name BEST-COVERAGE. Our implementation is an extension of this and works by sampling $2n$ vertices, uniformly, computing the single-source shortest paths for these vertices, and selecting the vertices that are present in most paths of the shortest paths. This is an intuitive, yet slow, algorithm that computes a set of robust landmarks.

4.3 Datasets

As discussed in chapter 2 graphs are a useful way of modelling both indoor and outdoot environments. In this thesis we will be using two types of datasets that model hypothetical indoor environments slightly differently. The datasets will be used in order to evaluate the aforementioned search methods. Both dataset graph types can be classified as connectivity graphs, according to Michael Worboys' taxonomy[38], where edges occur between two vertices if there is a physical connection between the two, i.e. a connection of regions. However, one of datasets will reside in Euclidean space while another will not. In the Euclidean dataset each vertex will be associated with a position – a two-dimensional vector (X, Y) – whereas edges will represent Euclidean distances between vertices. In the non-Euclidean dataset vertices will not be associated with any valus, however, edges will represent travel time (in seconds).

4.3.1 Akutten og Hjerte-lunge-senteret

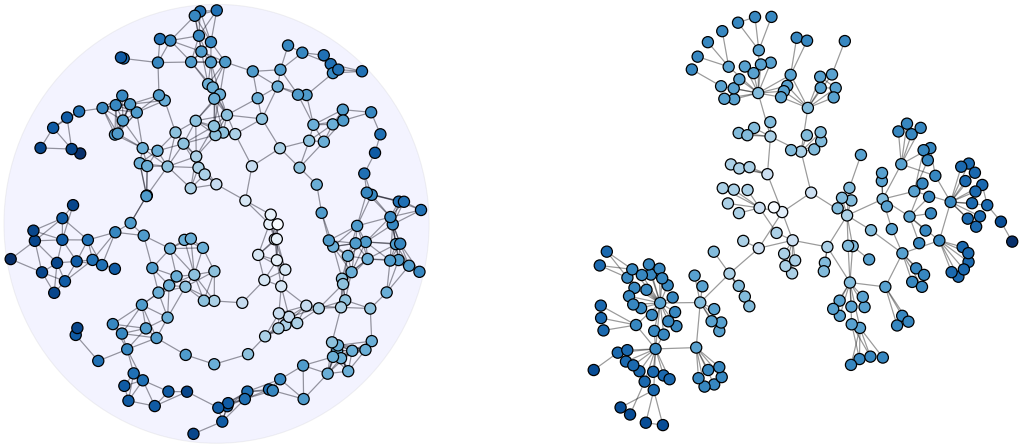
The first dataset models the first floor of Akutten og Hjerte-lunge-senteret (AHL) at St. Olavs Hospital, and represents the time-dependent non-Euclidean dataset. It is time-dependent because every edge is associated with a a travel time instead of a Euclidean distance, which for our purposes is randomly generated depending on the type of regions that are connected. A modified version of the MazeMap frontend map can be seen in Figure 4.3 and is used as a base for our graph. Each room, corridor, stair, and lift of the first floor of AHL has been assigned one of four prefixes and a number that together uniquely labels each region. The prefixes determines what kind of region it is, e.g. room, corridor, stair, or lift, and is used in a random number generator to generate plausible travel times for the edges that connect the regions. For example, if one of the regions is a lift and another is the corridor outside then the travel time will between the two will be very low. The graph of Akutten og Hjerte-lunge-senteret is suitable for testing search methods that utilize landmarks, and seeing as it is in non-Euclidean space it can not be used with the Euclidean distance heuristic seen in Equation 4.4.

Figure 4.3: A modified version of the MazeMap visual representation of Akutten og Hjerte-lunge-senteret.

In an effort to reduce the manual work needed to create such a graph we have decided to only model the first floor of AHL. The first floor consists of 203 vertices connected with 231 edges and can be seen in Figure 4.4 (b). Larger versions of the non-Euclidean graph are obtained by connecting multiple versions of the same floor via vertices classified as stairs and lifts. This lets us generate non-Euclidean graphs with $203n$ vertices and $231n + 15(n-1)$ edges. There are 15 lifts and stairs on floor one, hence the $15(n-1)$ part.

4.3.2 Random Geometric Graph

The second dataset is a random geometric graph and represents a non-time-dependent Euclidean graph. The graph is generated by creating a k-d tree of randomly sampled points, or vertices, from a circle with a user defined radius. An edge between two vertex pairs, quickly queried from the k-d tree, is created if the distance between the two is less than a threshold value. A visual representation of a random geometric graph with 203 nodes sampled from a circle with radius 0.6 can be seen in Figure 4.4 (b). Notice that the number of edges is completely determined by the threshold and cannot be defined by the user. The random geometric dataset can be used to evaluate every search method and heuristic mentioned above, however, it will be especially useful for assessing the *ED* heuristic as this is not possible with the AHL dataset.



(a) A random geometric graph with $|V| = 203$ and $|E| = 568$ sampled from a circle with a radius of 0.6.

(b) The first floor of Akutten og Hjertelunge-senteret with $|V| = 203$ and $|E| = 231$.

Figure 4.4: Visual representation of the datasets used for evaluating the system.

Chapter 5

Experiments and Results

In this chapter we explore the experiments and the results obtained from our search approach presented in the previous chapter. The experiments are split up into four scenarios, each focusing on the different qualities that make up our search method. Only a brief set of observations will be presented in this chapter, a more detailed discussion will be found in the next chapter.

5.1 Scenarios

The first scenario focuses on the landmark selection algorithms. The scenario is split up into two experiments that evaluate both the landmark build time and the achieved search runtime of different number of landmarks and selection algorithms. The scenario is designed to recognize strengths and weaknesses of the different selection algorithms. The rest of the scenarios utilize the results from this scenario in order to reduce the dimensionality of the experiments – a fixed selection algorithm and number of landmarks is chosen.

The second scenario explores the efficiency, or optimality, of the different search algorithms that can be used to find the nearest wheelchair. The scenario is split up into two experiments that test for two different amounts of wheelchairs. The classic Dijkstra's algorithm is used to measure optimality due to its guaranteed optimality when the input graph contains no negative edge weights. In addition to counting solution inaccuracies the experiments also output weighted inaccuracies, or cumulative distance errors.

The third scenario focuses on the search runtimes for different amounts of wheelchairs and graph sizes. The scenario is split up into four experiments that test the two datasets on each of the scenario types, i.e. graph size and wheelchair amounts.

Similarly to the third scenario, the fourth scenario is split up into four experiments that

test the two datasets on each of the two scenario types; however, the fourth scenario measures the fraction of explored search space instead of runtimes.

For simplicity, algorithms and heuristics will sometimes be abbreviated in order to reduce clutter. Dijkstra's algorithm with landmarks will be referred to as *DL* and bi-directional versions of algorithms will be referred to as *BD*, e.g. *BD A* TE1* refers to the bi-directional version of the A* algorithm using the regular triangle inequality heuristic.

5.2 Environment

As with the prototype of our search approach, all experiments in the thesis were developed in the Python programming language, version 2.7. The CPython implementation of Python was used as the interpreter of our experiments as it is the default and most widely used implementation of Python[64]. The choice of programmming language was due to a combination of the author's knowledge and previous experiences, the possibility of rapid prototyping, and the abundance of useful Python packages. The following third-party Python packages were used to implement the experiments: *NumPy*, a package for doing scientific computing with Python, *matplotlib*, a plotting library, and *mpltools*, a set of tools for matplotlib[65, 66, 67]. virtualenv[68] was used to create isolated and predictable Python environments.

5.2.1 Dataset

The experiments will use the datasets described in chapter 4 to evaluate the search methods and heuristics. The A* algorithm and the bi-directional version using the *ED* heuristic will only be tested on the random geometric graph. The first two scenarios evaluating the landmark selection algorithms and the search method optimality will use datasets generated on the fly, while the last two scenarios will be using pre-generated datasets. The reason for this is simply to save time as the last two scenarios will be testing a large number of datasets that can range from around 800MB to up to 1GB in size. The pre-generated datasets will be stored on the author's computer using the Python object serialization module, `pickle`.

5.3 Landmark Selection Efficiency and Effectiveness

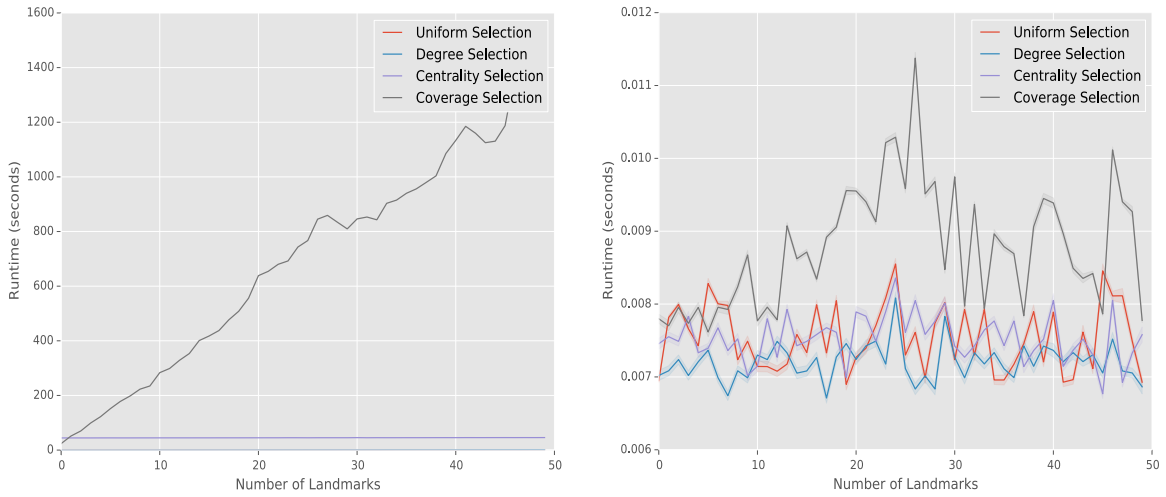
The first scenario measures the efficiency, or landmark build time, and effectiveness, or search time, for the four landmark selection algorithms: uniform, degree, eigenvector centrality, and coverage selection algorithms. Both of the effectiveness experiments will be applying the A* algorithm with the regular *TE1* triangle inequality heuristic on a dataset of connected floors of Akutten og Hjerte-lunge-senteret. 500 searches will be run and then averaged for each number

of landmark, and the sample variance will be shown as fill colour in the search time plots. A formula for the sample variance can be seen in Equation 5.1, where n is the sample size, x_i is the ith sample value, and m is the sample mean.

$$s^2_{n-1} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - m)^2 \tag{5.1}$$

Experiment 1: All Landmark Selection Algorithms

The first experiment uses a dataset with 20 connected floors, consisting of 4060 vertices and 4905 edges. The dataset does not change for each of the experiments, thus edge travel time remains the same for every experiment. A maximum of 50 landmarks was tested over a period of 1 day. The resulting graph plots can be seen in Figure 5.1.



(a) Landmark build time.

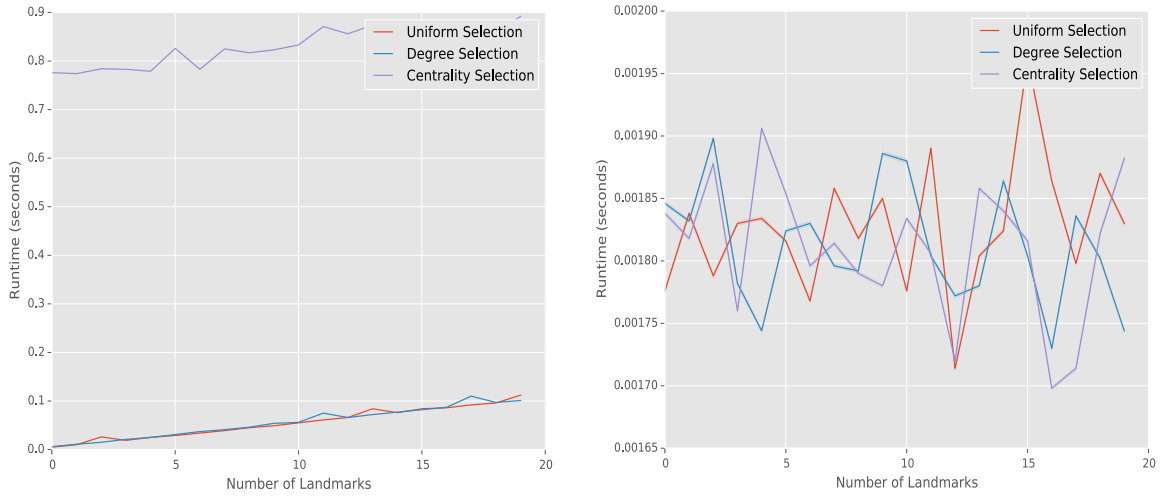
(b) Landmark search time.

Figure 5.1: Build and search time for all landmark selection algorithms. Up to 50 landmarks are tested.

In Figure 5.1 (a) the coverage selection algorithm dwarfs every other selection algorithm where it can take up to 30 minutes to build 40 landmarks. This can easily be explained by the NP-hardness of the coverage computation that performs an exhaustive single-source shortest path search and counting procedure for $2n$ vertices, e.g. 50 landmarks times 2. The next experiments omits the coverage selection algorithm, so the uniform, degree, and eigenvector centrality build times should be more visible. The search times for the selection algorithms in Figure 5.1 (b) are quite variable, with the coverage selection algorithm, unexpectedly, performing worse than the rest of the algorithms, at least for larger numbers of landmarks.

Experiment 2: Uniform, Degree, and Centrality Selection Algorithms

The second experiment uses a dataset with 5 connected floors, consisting of 1015 vertices and 1215 edges. The purpose of this experiment is to gain a better understanding of the uniform, degree, and eigenvector centrality build and search times that were dwarfed by the coverage selection algorithm in the first experiment. The results of the experiment can be seen in Figure 5.2, where the maximum number of landmarks is 20.



(a) Landmark build time.

(b) Landmark search time.

Figure 5.2: Build and search time for uniform, degree, and eigenvector centrality selection algorithms. Up to 20 landmarks are tested.

In Figure 5.2 (a) we can see the landmark build times much more clearly than in Figure 5.1 (a). Landmarks constructed using uniform and degree selection increase at a slow pace, and lie around 0.1 seconds for up to 20 landmarks. Eigenvector centrality landmark building increase at a slower but more varying pace and usually take around a second to build. This is considerably less than the coverage selection algorithm which increase at a much more rapid rate. The A* search time for the respective algorithms in Figure 5.2 (b) are comparable to the first experiment. Uniform selection, predictably, displays the most variable results, degree selection displays average performance, while eigenvector seem to perform just a bit better than degree selection.

For the rest of the experiments we will be using 15 landmarks created with the eigenvector centrality selection algorithm. As we saw in Figure 5.1 and Figure 5.2, this is a good compromise between efficiency and effectiveness.

5.4 Search Efficiency

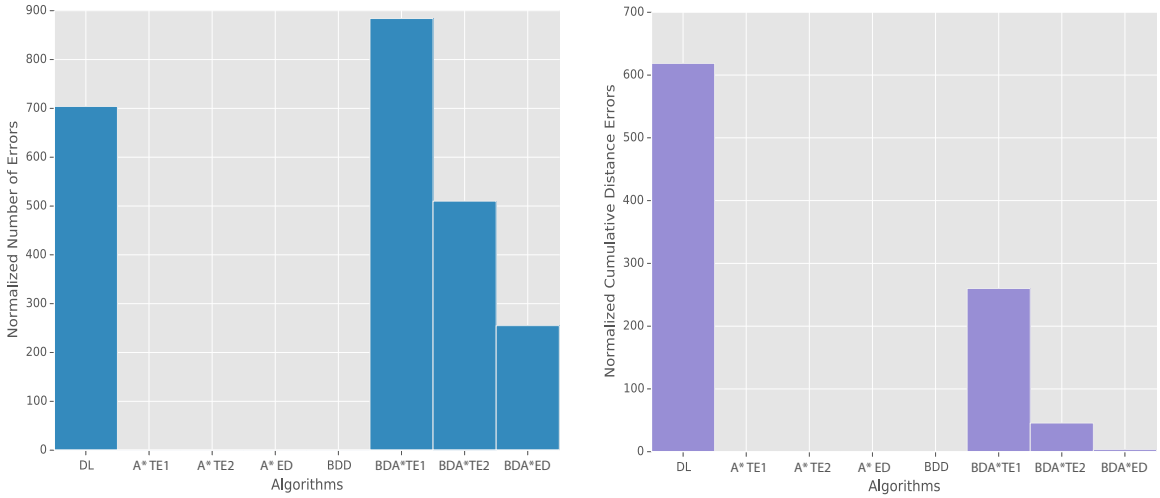
The second scenario measures search efficiency, or optimality, for each search algorithm with a varying number of wheelchairs. The first experiment will be testing for the base case of one wheelchair, while the second experiment will test for ten wheelchairs. Each algorithm is run 1000 times with a constant number of 15 landmarks created with the eigenvector centrality selection algorithm. The random geometric graph dataset is used in order to be able to test the regular and bi-directional version of the A* algorithm with the Euclidean distance heuristic (*A* ED* and *BD A* ED*). The random graph will consist of 1015 nodes sampled from a circle with radius 1.1 and 4051 edges, which is equivalent to the AHL dataset with 20 connected floors.

The number of errors and distance errors are calculated by comparing against a guaranteed optimal solution calculated using Dijkstra's algorithm. An error is defined as a solution path and accompanying distance that does not match the optimal solution. The number of errors is normalized by dividing the errors with the number of wheelchairs, thus the maximum number of errors is 1000. The calculation of the normalized cumulative distance errors can be seen in Equation 5.2 and is defined as the absolute of the difference between distances of the inspected and optimal solution. n is the number of wheelchairs while $x_{i,d}$ and $x^*_{i,d}$ is the distance for the ith wheelchair of the inspected and optimal solution, respectively.

$$Err(x) = \frac{1}{n}\sum_{i=1}^{n} |x^*_{i,d} - x_{i,d}|, \tag{5.2}$$

Experiment 1: All Search Algorithms, One Wheelchair

The first search efficiency tests all search algorithms with one existing wheelchair. The result of the experiment can be seen in Figure 5.3. In Figure 5.3 (a) we see that Dijkstra's algorithm with landmarks (*DL*) and the bi-directional A* algorithms are the only algorithms that consistently return the wrong answer. However, even though the bi-directional A* algorithm has errors in almost 90% of all searches, the severity of the errors is almost non-existent compared to Dijkstra's algorithm with landmark shortcuts; as can be seen in Figure 5.3 (b). For example, bi-directional A* with the Euclidean distance heuristic have almost zero distance errors even though over 20% of the solutions contain errors. The general observation between the heuristics seem to be that there is an inverse correlation between the quality heuristic estimate and the number of errors; however, we will have to test and see how the explored search space looks before confirming this.



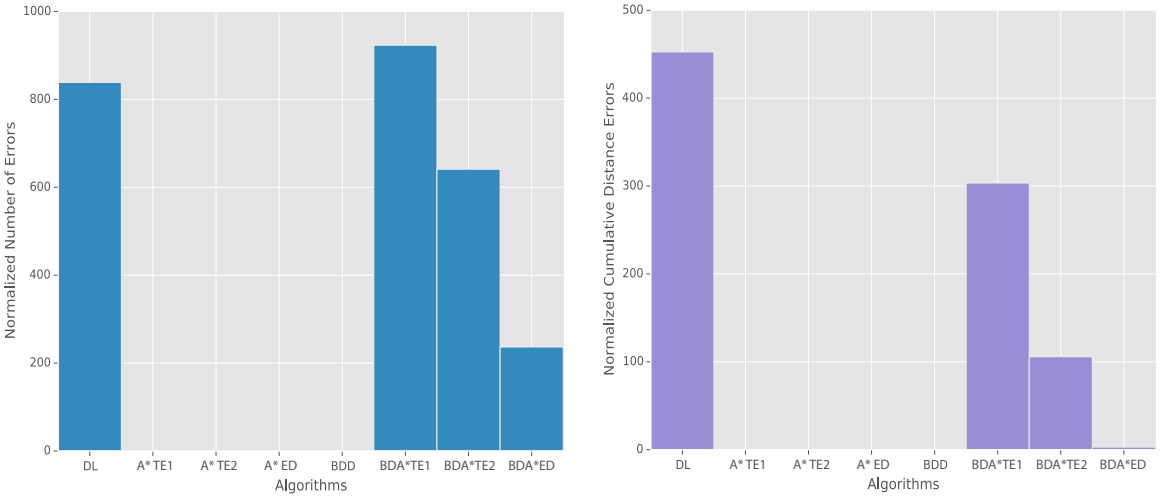
(a) Normalized number of errors.

(b) Normalized cumulative distance errors.

Figure 5.3: Search inaccuracies and cumulative distance errors with one wheelchair.

Experiment 2: All Search Algorithms, Ten Wheelchairs

The second search efficiency tests all search algorithms with ten wheelchairs. The result of the experiment can be seen in Figure 5.4.



(a) Normalized number of errors.

(b) Normalized cumulative distance errors.

Figure 5.4: Search inaccuracies and cumulative distance errors with ten wheelchair.

The number of errors in Figure 5.4 (a) is comparable to Figure 5.3 (a), the only difference being the slightly less number of errors for the bi-directional A* with the *TE2* heuristic, and

a slightly higher number of errors for Dijkstra's algorithm with landmarks. Overall, we see a marginally better performance for all algorithms that display errors in Figure 5.4 (b). Notice that bi-directional A* with the *ED* heuristic exhibit almost no distance errors in both Figure 5.4 (b) and Figure 5.3 (b) meaning that the errors observed in Figure 5.4 (a) are minimal at best.

5.5 Search Runtime

The third scenario measures search runtime for either an increasing graph size or an increasing number of wheelchairs. The first two experiments will test how each of the dataset types fares with an increasing graph size, while the latter two experiments will test how they perform with increasing amounts of wheelchairs. As before, every dataset will be using a constant number of 15 eigenvector centrality landmarks. Every search algorithm is run 5 times with a new random source and target vertex for each graph size or number of wheelchairs.

Experiment 1: Increasing Graph Size, Non-Euclidean Dataset

The first experiment uses 40 pre-generated datasets of AHL from 1 floor to 40 connected floors (a maximum of 8120 vertices). A constant number of 100 wheelchairs is used for each graph. The results can be seen in Figure 5.5. Notice that the *ED* heuristic can not be used on this dataset.

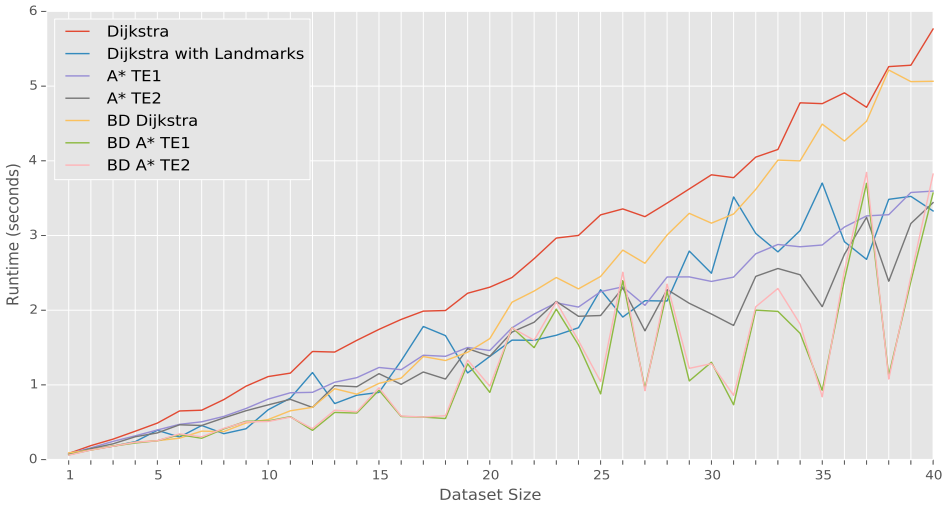


Figure 5.5: Search time for increasing non-Euclidean graph sizes.

We can see that the bi-directional versions of Dijkstra's algorithm and A* perform much better than the regular versions; however the runtime of the bi-directional A* algorithms tend

to oscillate more the larger the graph size is. The *TE2* heuristic also seem to perform better than the *TE1* heuristic for every graph size. Furthermore, the runtime of Dijkstra's algorithm with landmarks can be compared to the A* algorithm, but with more inconsistency.

Experiment 2: Increasing Graph Size, Euclidean Dataset

The second experiment uses 40 pre-generated datasets of the random geometric graph and has exactly the same number of vertices as the AHL graph in the first runtime experiment. A constant number of 100 wheelchairs is used for each graph. The results can be seen in Figure 5.6.

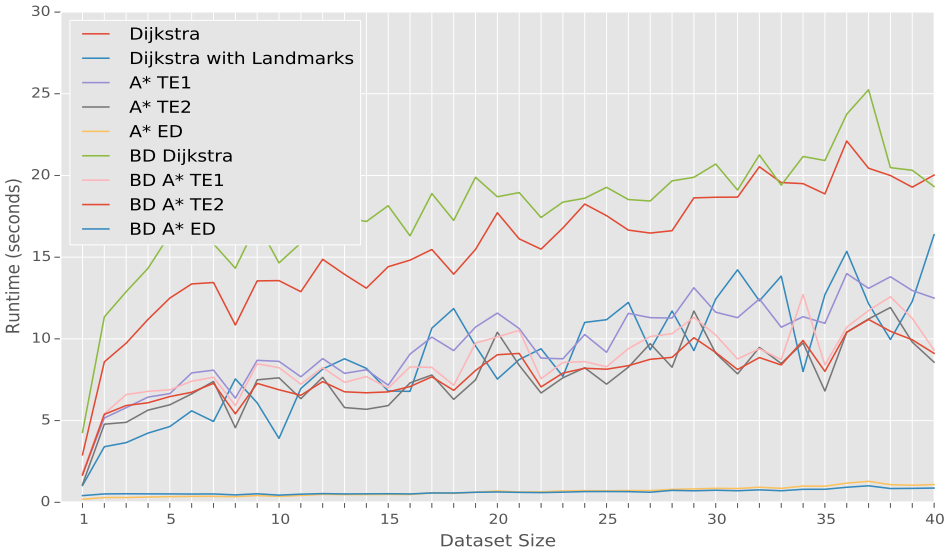


Figure 5.6: Search time for increasing Euclidean graph sizes.

Surprisingly, every algorithm and heuristic beside *ED* perform considerably worse compared to the AHL dataset. However, as we can see by the yellow and dark blue lines at the bottom of Figure 5.6, the *ED* heuristic performs significantly better than the others. Another, yet surprising, observation is that the bi-directional version of Dijkstra's algorithm performs consistently worse than the regular version. This behaviour might be caused by the fact that the random geometric graphs tend to have more edges connecting the vertices than the non-Euclidean datasets.

Experiment 3: Increasing Number of Wheelchairs, Non-Euclidean Dataset

The third experiment uses an increasing amount of wheelchairs on a pre-generated graph of the AHL dataset with 20 connected floors (4060 vertices and 4905 edges). Due to the fact that

differences in runtime between n and $n+1$ wheelchairs do not add up to much, we decided to increase the amount of wheelchairs by 100 for each search up to 2000 wheelchairs. The results can be seen in Figure 5.7. Notice that the *ED* heuristic can not be used on this dataset.

Compared to the the previous two experiments we can see that the number of wheelchairs have a much bigger impact on how fast the search methods perform. The bi-directional version of A* *TE1* and *TE2* perform significantly better than the regular versions, with a difference of up to 10 seconds, compared to the rest of the algorithms. Both the regular and bi-directional version of Dijkstra's algorithm do not scale well with increasing numbers of wheelchairs.

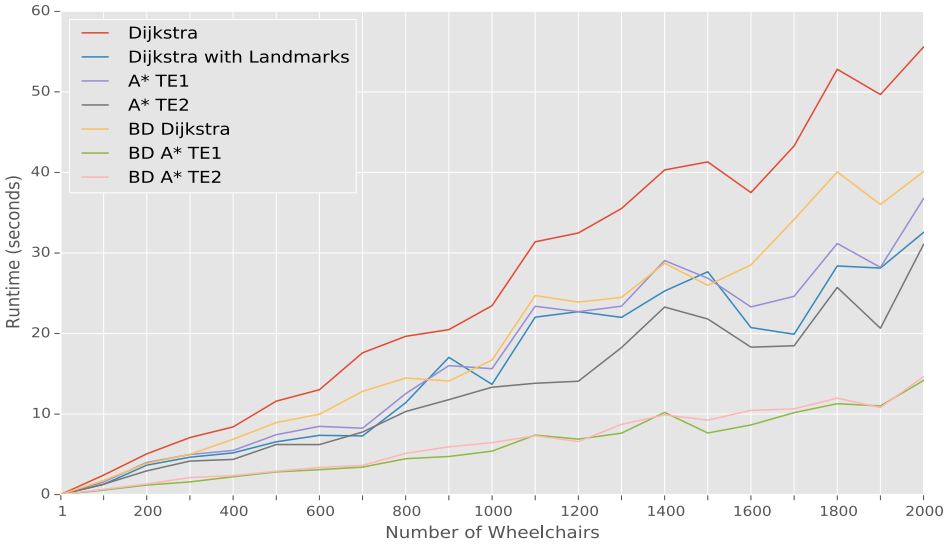


Figure 5.7: Search time for an increasing number of wheelchairs on a non-Euclidean graph.

Experiment 4: Increasing Number of Wheelchairs, Euclidean Dataset

The fourth and last experiment for measuring search runtime uses the same setup as the third experiment, however, a random geometric graph dataset with 4060 vertices is used instead. The results can be seen in Figure 5.8.

As with the second runtime experiment, every algorithm and heuristic besides *ED* perform much worse compared to the AHL datset, with runtimes lasting as long as 4 or 5 minutes in the worst cases. The bi-directional and regular version of A* with the *ED* heuristic can barely be seen at the bottom of the graph search times ranging from 0 to 20 seconds. We can again see that the bi-directional Dijkstra version performs worse than the regular version.

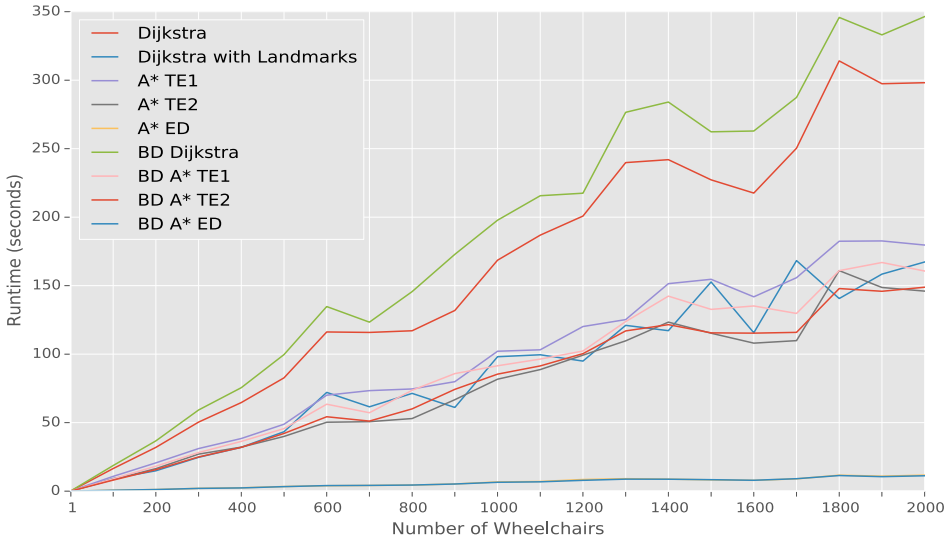


Figure 5.8: Search time for an increasing number of wheelchairs on an Euclidean graph.

5.6 Explored Search Space

The fourth and last scenario measures explored search space by counting up the number of explored vertics and dividing it with the total number of vertices, thus the range of the explored search space is between 0.0 and 1.0. These experiments ran simultaneously with the search runtime scenario, thus the first two experiments tests how the datasets perform on increasing graph sizes, and the last two experiments test how the datasets perform with increasing amounts of wheelchairs. As before, every dataset will be using a constant number of 15 eigenvector centrality landmarks. Every search algorithm is run 5 times with a new random source and target vertex for each graph size or number of wheelchairs.

Experiment 1: Increasing Graph Size, Non-Euclidean Dataset

The results of the first experiment can be seen in Figure 5.9, it uses the same exact setup as the first experiment in the search runtime scenario.

Dijkstra's algorithm explores, unsurprisingly, more of the search space than any other algorithm. We can also see the significance of the *TE2* heuristic as even the regular A* algorithm perform better or equal to the bi-directional A* algorithm with the *TE1* heuristic. Interestingly, the algorithms, with the exception of Dijkstra's algorithm, all explore less than 50% of the search space; however, the explored search space does not seem to go down with larger search spaces.

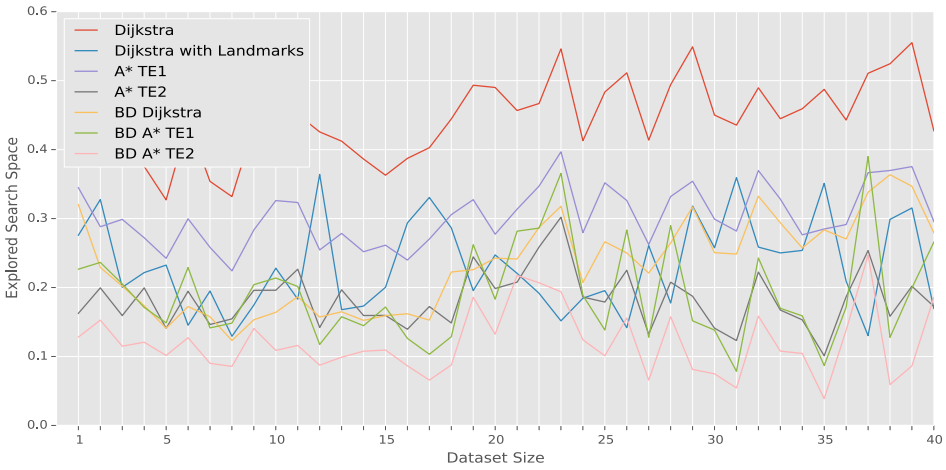


Figure 5.9: Explored search space for increasing non-Euclidean graph sizes.

Experiment 2: Increasing Graph Size, Euclidean Dataset

The second experiment uses the same setup as the second experiment in the search runtime scenario.

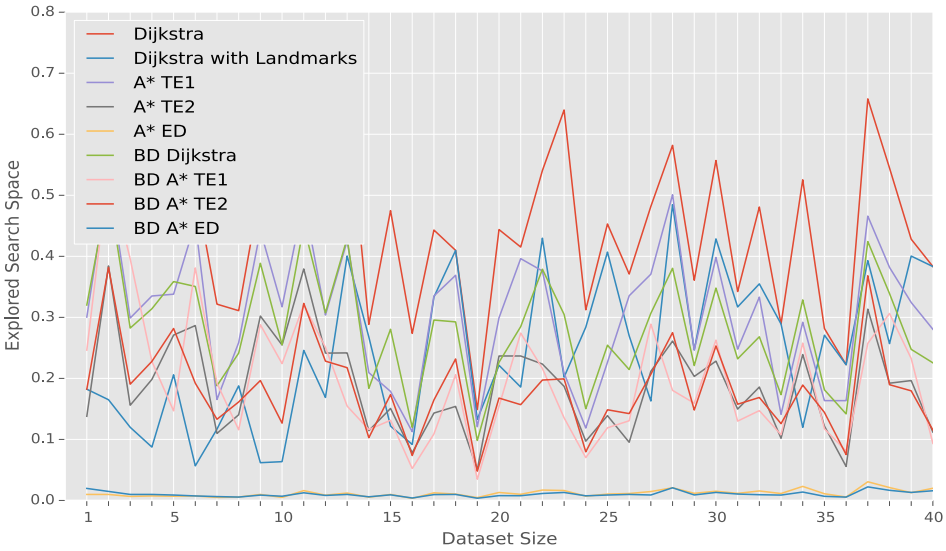


Figure 5.10: Explored search space for increasing Euclidean graph sizes.

As we can see from Figure 5.10, the performance of the search methods is comparable to the results in Figure 5.9, however, except for the versions of A* that use the *ED* heuristic, the size of the explored search space is quite variable. There seems to be no obvious pattern or reason for the variable results, however, it is clear that the *ED* heuristics perform much better

on the Euclidean datasets. In fact, the *ED* heuristic does not even explore 10% of the search space, it is more close to 1% or 2% of the total search space. As we saw in Figure 5.9 the size of the explored search space neither seem to decrease or increase with increasing graph sizes, but rather seem to oscillates between high and low values.

Experiment 3: Increasing Number of Wheelchairs, Non-Euclidean Dataset

The results of the third experiment can be seen in Figure 5.11, it uses the same exact setup as the third experiment in the search runtime scenario.

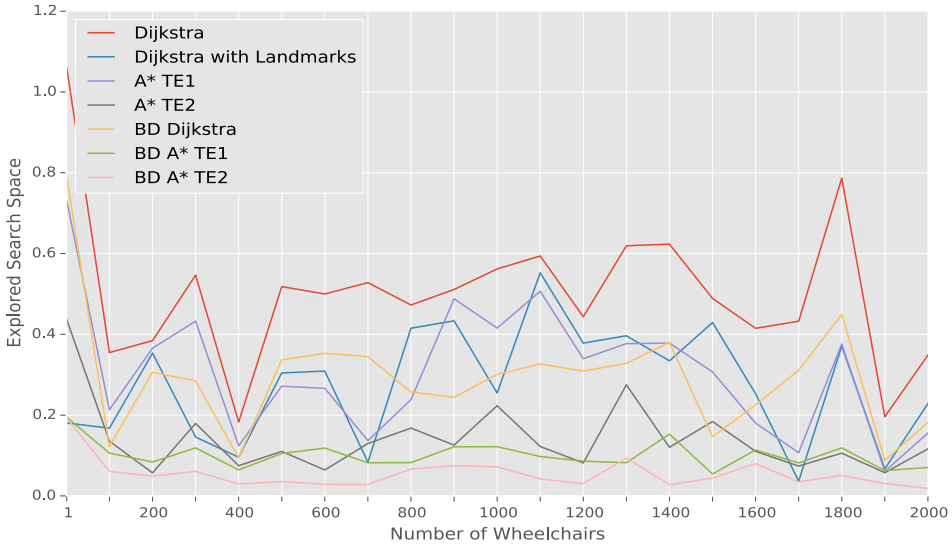


Figure 5.11: Explored search space for an increasing number of wheelchairs on a non-Euclidean graph.

Compared to the third and fourth experiment in the search runtime scenario we do not see any extreme increase, or decrease, of the explored search space. Compared to the first and second search space experiments it actually seems that the number of wheelchairs is irrelevant with respect to the size of the explored search space. As before only Dijkstra's algorithm seem to crawl past the 50% explored search space mark, with most of the algorithms staying well below 40% of the explored search space, on average.

Experiment 4: Increasing Number of Wheelchairs, Euclidean Dataset

The results of the fourth and last experiment can be seen in Figure 5.12, it uses the same setup as the fourth experiment in the search runtime scenario.

The graph exhibits the same oscillatory behaviour as the second experiment, which from what we have seen from earlier experiments might have something to do with the Euclidean

dataset itself. As we saw in Figure 5.11, the number of wheelchairs do not seem to have any effect on the size of the explored search space, and just like the second search space experiment the search methods using the *ED* heuristic seem to only explore about 1% or 2% of the total search space.

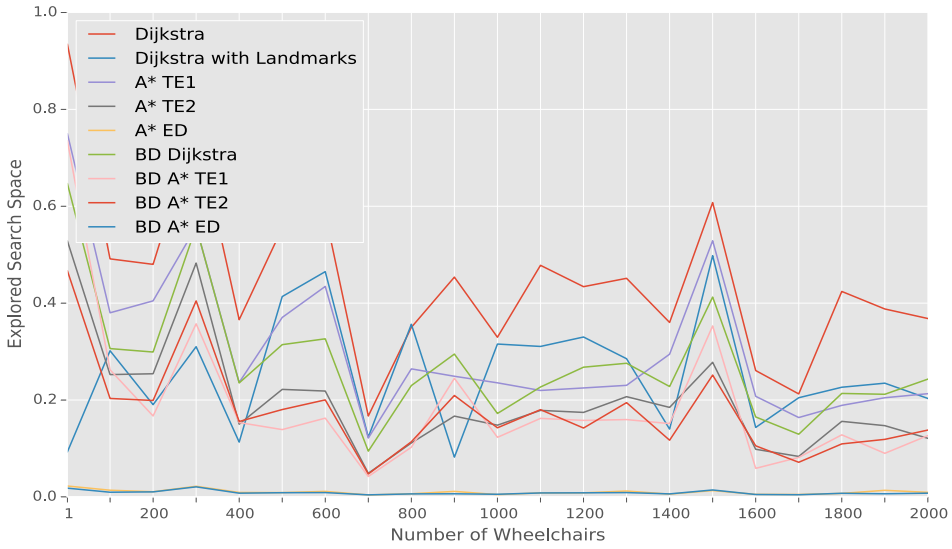


Figure 5.12: Explored search space for an increasing number of wheelchairs on an Euclidean graph.

Chapter 6

Discussion

In this chapter we discuss the results from chapter 3 and chapter 5, while also drawing upon the work presented in chapter 4. The chapter is split up into two parts. The first part discusses the results from chapter 5 by evaluating the performance of the search methods, while the second part discusses the search framework and search methods from chapter 4 with respect to the requirements from chapter 3.

6.1 Performance

The discussion of the experimental results will be split up into two parts. One part discussing the landmark selection algorithms, while another part discusses the search methods

6.1.1 Landmark Selection

There are both strengths and weaknesses to the four landmark selection algorithms we have prototyped here, and the performance or quality of the selected landmarks can have a significant impact on the heuristics and algorithms that use them. The most obvious result is the fact that more landmarks lead to longer preprocessing times, with the coverage selection algorithm having the longest build time, by several orders of magnitude. Compared to the other selection algorithms, the build time for uniform, degree, and eigenvector centrality selection uses, at most, just a few seconds, meaning that they can even be run online without having a major impact on the total search time. Uniform selection proved to be the most unstable selection algorithm, whereas the coverage selection algorithm yielded disappointingly high search times compared to the more simple algorithms. We can only speculate that the uniform sampling of the two times the number of landmarks vertices coupled with the variable performance of the random selection of candidate landmarks only lessened the coverage landmark robustness.

Degree and eigenvector centrality selection generated comparable results, with eigenvector centrality having just a bit better search times for increasing numbers of landmarks.

6.1.2 Search Method

The bi-directional and regular version of Dijkstra's algorithm do not scale well for neither increasing graph sizes or increasing numbers of wheelchairs. While displaying decent runtime performance for small graph sizes and wheelchair amounts, the runtime quickly increased for larger numbers. From the users' perspective, even though these algorithms do guarantee optimal solutions it might be better to accept a solution with *some* flaws instead of having to wait up to one or more minutes for an optimal solution. Our attempt at combining Dijkstra's algorithm with landmarks as shortcuts proved to be a disappointment, with runtimes comparable to the regular A* algorithm, unpredictable runtimes, and a significant error rate. In fact, it displayed the highest amount of weighted distance errors compared to the other algorithms.

On the other hand, the A* algorithm displayed the most promising results, regardless of heuristic. The bi-directional version of A* has better runtimes compared to the regular version; however, we can observe that speed is indeed traded for correctness. Optimality aside, it is clear that the bi-directional version of the A* algorithm scale much better than the alternatives. In retrospect, we should have implemented a stronger stopping criterion, such as the average of heuristics presented by Ikeda et al.[48]. Interestingly, there seems to be a strong case for an inverse correlation between the tightness of the heuristic estimate and the correctness of the search, i.e. a better heuristic estimate from both sides increases the chance of the two searches intersecting at the optimal path.

The *TE2* heuristic performed better than the regular *TE1* heuristic in all cases on the non-Euclidean dataset, even though it also has to find the nearest neighbour of the target vertex in order to be able to exploit the ideas of *TE1* symmetrically. An interesting property with both *TE1* and *TE2* is that the lower bound becomes better the more landmarks are present in the graph, meaning that there is an trade-off between landmark preprocessing and lower bound tightness for sufficiently many landmarks. This property follows naturally from the triangle inequality and the fact that both heuristcs rely on the *nearest* landmark, i.e. the closer the landmark the better the heuristic estimate. The *ED* heuristic is a clear winner for the Euclidean dataset where it performs several orders of magnitude better than the other two heuristic functions.

The behaviour experienced on the runtime and explored search space experiments using the random geometric graph, or Euclidean dataset, can partially be explained by the increased amount of edges compared to the non-Euclidean dataset. An example of this can be seen in Figure 4.4 where the number of edges on the random geometric graph is over double the

size of the number of edges on the AHL graph. This is an implementational issues as edges are created between two vertices if the distance between the pair of vertices is less than a threshold value. This means that there is large chance that the Euclidean distance between any two vertices will prove to be a good heuristics. With that said, the *ED* heuristic will only work on graphs with Euclidean distances and not on graphs modelling travel time. More on the topic of distance types will be covered in the last section.

The explored search space proved to neither increase or decrease with regards to the shifting graph and wheelchair parameters for any of the search methods that we tested. However, it did show that there is a correlation between search methods and heuristics where, for example, the methods applying the *TE2* heuristic performs consistently better than the *TE1* alternative with regards to both runtime and explored search space.

6.2 Appropriateness

In this section we will consider each of the requirements in Figure 3.3 and see if any of the proposed search methods manages to fulfil them.

The first requirement, Req-1, states that the system should be able to find the most *efficient* solution to the problem. As previously discussed, both the regular and bi-directional version of Dijkstra's algorithm and the regular version of the A* algorithm is able to guarantee correct solutions, in our prototype. However, depending on the type of graph it is clear that the best search method would be the A* algorithm with either the *TE2* heuristic function for graph models that do not have any vertex coordinate information, or the *ED* heuristic function for graph models with vertex coordinates. The choice of graph type depends on how, or in what way, the environments are modelled. The obvious and most commonly used type is the use of vertex coordinates such as latitude and longitude, where distances between vertices are measured using, for example, meters. While this might give an accurate description of the environment itself, it does not give any indication as to how long it takes to travel from A to B. Alternatively, the environment can be modelled as physical areas connected together by edges describing travel time, for example, in seconds. This gives an accurate description on how long it takes to travel from A to B, and resolves questions such as: is it quicker to use the stairs or the lift? In the first case, the *ED* heuristic function is a good choice, while the *TE2* heuristic function is a good choice for the latter.

The search framework presented in Equation 4.1 describes a natural way of separating the search framework from the search method, thereby fulfilling Req-2.

Every search method presented in this thesis is able to rank paths according to their efficiency, thus fulfilling Req-3. However, in practice it should be possible to relax this requirement to only require, for example, the k best wheelchair choices, with k being less than the number

of wheelchairs. In a real life situation a porter would probably be satisfied with, for example, a list of the 5 best wheelchairs choices instead of a ranking of several hundreds of wheelchairs.

For several thousands of wheelchairs in a fairly large hospital only the bi-directional version of the A* algorithm and the regular A* algorithm with the *TE2* heuristic function is able to find the nearest wheelchair within a maximum of 30 seconds. In the case of St. Olavs Hospital, where there are just a few hundred wheelchairs of location tracking interest, the suggested search method will have no problem finding the nearest wheelchair. Regardless, Req-4 is fulfilled for any small to medium sized hospital and partially fulfilled for large hospitals depending on the number of wheelchairs. There is a trade-off between correctness and speed which will need to be considered for larger hospitals and larger amounts of wheelchairs.

The search framework proposed by this thesis follows directly from Equation 4.1 which describes the nearest wheelchair as the wheelchair with the shortest path from source to target via a set of wheelchairs. This approach ultimately suffers due to the exhaustiveness of the search as each wheelchair will have to be checked, regardless. If the focus on keeping a set of ranked solutions is relaxed to simply the k best wheelchairs or removed completely it would be possible to make use of algorithms that yield approximate solutions, such as genetic programming, simulated annealing, and ant colony optimization. Another approach would be to prune away large parts of the set of wheelchairs by disregarding those that lie in, for example, another building given that both the porter and patient is in the same building. Req-5 is partially fulfilled as the system scales well with increasing hospital sizes but does not scale well with large numbers of wheelchairs.

Not only is the landmark preprocessing able to run offline, thereby fulfilling Req-6, the best landmark selection algorithm, using eigenvector centrality, is quick enough to be used online.

Req-7 states that an implementation of the nearest wheelchair problem must be able to work with the current IPS at St. Olavs Hospital. As discussed in chapter 4, our search approach requires that the IPS is able the query wheelchair locations in real time and has sufficiently high enough spatial resolution. From the requirements eliciation we found that MazeMap, the IPS at St. Olavs Hospital, is able to display items, such as wheelchairs, and compute paths between two points on the map. During one of our meetings with Thomas Jelle at Wireless Trondheim we found that there should not be anything in MazeMap that limits the implementation of a system such as the one proposed in this thesis.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The main focus of this master's thesis has been on the application of search methods from Artificial Intelligence (AI) literature to a logistical problem in the hospital. First, a Requirements Engineering (RE) study was performed to recognize potential uses of IPSs for the reduction of time spent looking for wheelchairs, where finding the nearest wheelchair was found to be suitable and beneficial for further investigation. Next, a search framework based on the triangle inequality was developed that leverages location information from an IPS. Then, using existing search methods from AI literature as starting points, a set of potential search methods have been developed and evaluated with respect to efficiency and effectiveness.

The main contributions of this thesis is the concept of minimizing time spent searching for wheelchairs by considering the current task of the porter. In addition, a novel extension of the commonly used triangle inequality heuristic function (*TE1*), termed *TE2*, has been developed by applying the ideas of *TE1* symmetrically. The proposed *TE2* heuristic function outperforms *TE1* by computing a better heuristic estimate.

7.1.1 Answers to Research Questions

The following section elaborates on how the RQs in section 1.2 have been answered by the thesis:

RQ1: How can the time porters spend looking for wheelchairs be reduced?

A RE study was performed in order to recognize a set of use cases and requirements that answer this RQ. Chapter 3 proposes three use cases that have the potential to reduce time spent

looking for wheelchairs. The three use cases are: *"View Wheelchair Location"*, *"Find Available Wheelchair"*, and *"Find Nearest Wheelchair"*, where finding the nearest wheelchair, with respect to the current task of the porter, was found to be most suitable for further investigation. Furthermore, the RE study also resulted in a set of requirements for the aforementioned use case.

RQ2: What is considered to be the nearest wheelchair?

A search framework defining the *nearest* wheelchair was presented in chapter 4 and defines the nearest wheelchair as the wheelchair with the shortest path from source to target via a set of wheelchairs. In addition, a discussion on the definition of distance and its effects on users were presented in chapter 4 and chapter 6. The discussion recognizes two useful ways of modelling distances. The first, considers distances between physical areas in the traditional time-independent Euclidean way by, for example, the use of meters. The second, considers distances as time-dependent by modelling connections between adjacent physical areas as, for example, travel time. Both approaches offer different strengths and weaknessess, where the traditional approach offers an accurate description of the physical environment, while the time-dependent approach offers an accurate description on how long it takes to travel from A to B.

RQ3: Which search methods are most appropriate for reducing the time spent looking for wheelchairs in terms of optimality, runtime, and explored search space?

A prototype of the proposed search framework was implemented in order to evaluate potential search methods and heuristic functions from AI literature. The evaluation of the search methods was performed in chapter 5 and a discussion with respect to optimality, runtime, explored search space, and the answers of **RQ1** took place in chapter 6. It shows that the regular and bi-directional version of the A* algorithm using the *TE2* or Euclidean distance heuristic functions outperforms the other search methods. With *TE2* being a good choice for models modelled in a time-dependent manner, and the *ED* heuristic being a good choice for models modelled in a traditional time-independent manner.

7.1.2 Research Objectives

RQ1 and partially **RQ2** achieves the first research objective of exploring and understanding use cases of IPS for reducing the time spent by porters looking for wheelchair. **RQ2** and **RQ3** achieves the second research objective of researching and comparing search methods from AI literature in terms of optimality, runtime, and explored search space.

7.1.3 Limitations

The prototype of the proposed search framework implements only rudimentary functionality with the purpose of evaluating the search framework and potential search methods. However, the findings and results of the thesis should be applicable to most IPSs, as none of the proposed theory is IPS-specific. We were not able to perform a proper demonstration of the search framework in the hospital environment due to our limited access to the IPS at St. Olavs Hospital; however, this should not have had any impact on the answers to the RQs.

7.2 Future Work

This section presents possible future work.

The most obvious work is to instantiate the proposed system in the context of the IPS at St. Olavs Hospital, in collaboration with Wireless Trondheim. This would involve purchasing WLAN devices for a set of wheelchairs, implementing the necessary modules into the IPS, and assessing the usefulness of the system.

Another avenue for future work is usability, or ease of use, of the system. While functional, a textual description of nearby wheelchairs on a smart phone is not exactly aesthetically pleasing. It would be pertinent to investigate the user experience (UX) of the system by considering a user-centered design approach using, for example, ISO 9241-210:2010[69].

The final point would be to extend the problem of finding the nearest wheelchair by generalizing it. This could prove to be very useful as there are many logistical and other problems that are similar to TSP, such as the problem of picking up a limited set of different types of items while travelling from A to B. When generalizing the nearest wheelchair problem it also becomes natural to explore and develop approximate approaches to the the problem.

Appendix A

Technical Overview

A.1 Introduction

The following appendix briefly describes the Python package `nearest_wheelchair` which implements a prototype of the system proposed by the thesis. The code itself can be found in the accompanying attachment of the thesis.

The appendix is split up into three parts. The first part lists the prototype requirements, while the second part presents an overview of the implementation itself. Part three presents examples of how the package can be used and also presents the accompanying demonstration file.

A.2 Requirements

The system is implemented in the Python programming language, version 2.7, and has been tested with the CPython interpreter[64]. We decided to use the Python package NetworkX[70] in order to avoid having to implement a graph module from scratch, and due to the flexible implementation of NetworkX we were able to extend it with extra features. The third-party Python packages NumPy, matplotlib, and SciPy are required by NetworkX and are also used in other areas of the prototype[65, 66, 71]. For example, the k-d tree implementation in SciPy is used by the random geometric graph module in order to build random graphs faster.

A.3 System Description

The package `nearest_wheelchair` is comprised of several modules, each of which is responsible for different parts such as dataset generation, graph definition, and search algorithms. A

UML class diagram of the system can be seen in Figure A.1, for simplicity the modules have been split up into two sets of modules, namely `algorithms` and `datasets`. The demonstration module and third-party packages have been omitted to reduce clutter. A description of how to use the package is found in the next section.

`algorithms` contains an implementation of different variations of Dijkstra's, A*, and bi-directional variants, that can be used by the nearest wheelchair search algorithm. The `bidirectional` module is associated with the `astar` module in order to gain access to the heuristics TE1, TE2, and ED. The `landmarks` module contains both an implementation of the landmark acceleration method and four landmark selection algorithms. Some of the selection algorithms utilize static functions from the `statistics` module, however, most of the contents of `statistics` is used for experimentation purposes. The `quickest_path` module contains the algorithm for finding the nearest wheelchair given a source and target vertex. The five methods below `find_quickest_path()` are wrapper functions and makes sure that every algorithm conforms to a common interface.

The set of modules in `datasets` are responsible for generating datasets, or graphs, that we search on. Graphs are defined by the `dataset_definition` module, and is an extension of the graph data structure provided by NetworkX. Our extension provides methods for placing n wheelchairs, setting random edge weights, drawing the graph using matplotlib, and selecting random vertices. The module also contain a definition of vertices and wheelchairs and also two static functions. The `ahl_graph` and `random_geometric_dataset` modules are able to generate graphs. The first generates a manually defined graph of the first floor of Akutten og Hjerte-lunge-senteret at St. Olavs Hospital, where multiple floors can be connected via vertices labelled as lifts and stairs. The second generates a random geometric graph with n vertices sampled from a circle with a user defined radius; an edge between two vertices is created if the distance between them is less than the threshold value. The `datasets` module acts as a wrapper and simplifies the dataset creation process. Writing and reading graphs to and from disk is also supported.

A.4 Examples

This section introduces an example of how the `nearest_wheelchair` package can be used to find the nearest wheelchair with respect to the shortest path between a porter and a patient. Additional examples can be found in the accompanying *demo.py* file. The file can be run from the command-line if all the required Python packages are installed.

Only the `nearest_wheelchair` package has to be imported in order to be able to use its modules.

```
import nearest_wheelchair as nw
```

As previously seen in the UML class diagram, a graph can be generated from the `datasets` module. In this example a random geometric graph is generated by sampling 400 vertices from a circle with a radius of 0.4. The `datasets` module forces graph reachability automatically[1].

```
graph = nw.get_euclidean_dataset(n=400, radius=0.4)
```

Landmarks can be built by instantiating the `Landmarks` object. In this example, ten landmarks will be built using the eigenvector centrality selection algorithm.

```
landmarks = nw.Landmarks(graph)
landmarks.build_landmarks(10, nw.centrality_selection)
```

The code snippet belows randomly distributes 40 wheelchairs among the graph nodes.

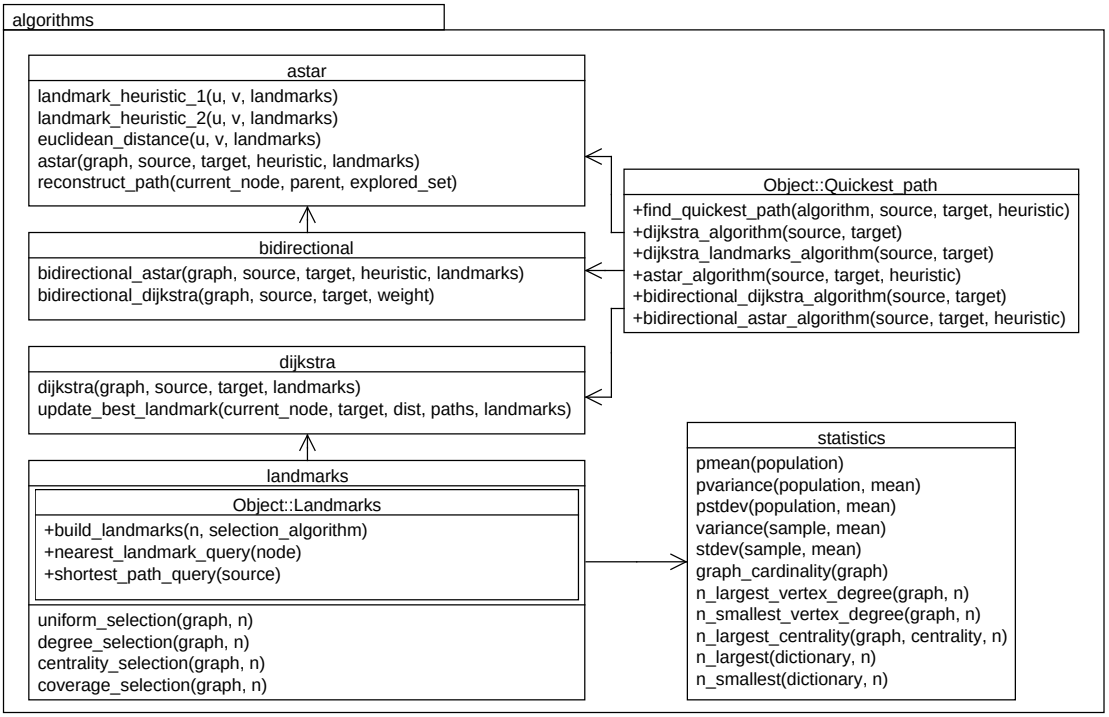
```
wheelchairs = graph.place_n_wheelchairs(40)
```

Finding the nearest wheelchair with respect to the shortest path between a porter and a patient can done by instantiating the `Quickest_path` object and calling the `find_quickest_path()` method. In this example we will be using the A* algorithm with a regular triangle inequality heuristic (commonly known as ALT). A source and a target vertex has been selected randomly.

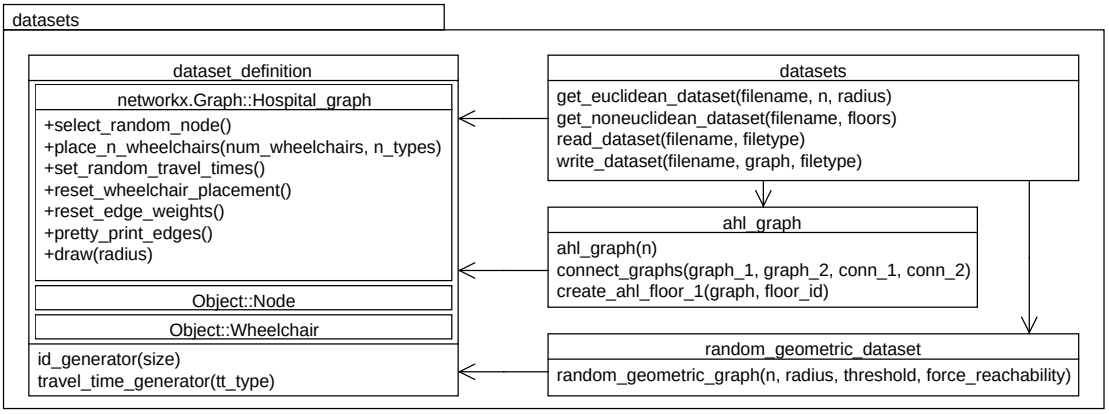
```
porter = graph.select_random_node()  # source
patient = graph.select_random_node()  # target
qp = nw.Quickest_path(graph, wheelchairs, landmarks)
results = qp.find_quickest_path(qp.astar_algorithm,
                                porter,
                                patient,
                                nw.landmark_heuristic_1)
```

The result is a sorted list of Python dictionaries, where each element contains a reference to the current wheelchair, distance and path from the source to the target vertex via the current wheelchair, the number of explored vertices, and runtime information. The list is sorted in ascending order with respect to the path distance, meaning that the first element is the nearest wheelchair.

[1] Vertex u is reachable from vertex v if there exist a path between the two vertices. By forcing graph reachability we ensure that it is possible to get from every vertex to every other vertex.



(a) A set of modules that handle the landmark generation and nearest wheelchair computation.



(b) A set of modules that handle dataset generation.

Figure A.1: A UML class diagram of the prototype system. Together, (a) and (b) make up the `nearest_wheelchair` package.

Bibliography

[1] ØYVIND VIKAN *Fakta og tall* 2012 URL: http://www.stfk.no/no/Fylket_vart/Fakta_og_tall/ (visited on 2013-09-19) (cit. on p. 1)

[2] ASTRID HAUGEN *St. Olavs Hospital, Trondheim University Hospital, integrated with NTNU* 2013 URL: http://www.stolav.no/en/About-the-hospital/ (visited on 2013-09-19) (cit. on p. 1)

[3] HELSEBYGG MIDT-NORGE *Project Owners* 2012 URL: http://www.helsebygg.com/owners/ (visited on 2013-09-19) (cit. on p. 1)

[4] HELSEBYGG MIDT-NORGE *The Patient's Perspective* 2012 URL: http://www.helsebygg.com/concepts/ (visited on 2013-09-19) (cit. on p. 1)

[5] HELSEBYGG MIDT-NORGE *ICT in the new hospital* 2011 URL: http://www.helsebygg.com/aboutus/ (visited on 2013-09-19) (cit. on p. 1)

[6] U.S. GOVERNMENT "GPS Standard Positioning Service (SPS) Performance Standard" in: (2008) (cit. on p. 2)

[7] JANET M. CORRIGAN LINDA T. KOHN and INSTITUTE OF MEDICINE MOLLA S. DONALDSON EDITORS; COMMITTEE ON QUALITY OF HEALTH CARE IN AMERICA *To Err Is Human: Building a Safer Health System* The National Academies Press, 2000 (cit. on p. 2)

[8] P. R. SUN, B. H. WANG, and F. WU "A new method to guard inpatient medication safety by the implementation of RFID" in: *Journal of Medical Systems* 32.4 (2008), pp. 327–332 DOI: 10.1007/s10916-008-9137-9 (cit. on p. 2)

[9] E. A. FRY and L. A. LENERT "MASCAL: RFID tracking of patients, staff and equipment to enhance hospital response to mass casualty events" in: *AMIA Annual Symposium Proceedings* (2005), pp. 261–265 (cit. on p. 2)

[10] R. A. MARJAMAA, P. M. TORKKI, M. I. TORKKI, and O. A. KIRVELA "Time Accuracy of a Radio Frequency Identification Patient Tracking System for Recording Operating Room Timestamps" in: *Anesthesia and Analgesia* 102.4 (2006), pp. 1183–1186 DOI: 10.1213/01.ane.0000196527.96964.72 (cit. on p. 2)

[11] C. C. LIN, M. J. CHIU, C. C. HSIAO, R. G. LEE, and Y. S. TSAI "Wireless Health Care Service System for Elderly With Dementia" in: *Information Technology in Biomedicine, IEEE Transactions on* 10.4 (2006), pp. 696–704 DOI: 10.1109/TITB.2006.874196 (cit. on p. 2)

[12] TIMO STÜBIG, CHRISTIAN ZECKEY, WILLIAM MIN, LAURA JANZEN, MUSA CITAK, CHRISTIAN KRETTEK, TOBIAS HÜFNER, and RALPH GAULKE "Effects of a WLAN-based real time location system on outpatient contentment in a Level I trauma center" in: *International Journal of Medical Informatics* (2013) DOI: 10.1016/j.ijmedinf.2013.10.001 (cit. on p. 2)

[13] M. HAGLAND "Follow that pump. As medical devices grow more expensive, tracking them becomes absolutely imperative" in: *Healthcare Informatics Magazine* 26.3 (2009), pp. 42, 80 (cit. on p. 2)

[14] GOOGLE *Project Tango* 2014 URL: https://www.google.com/atap/projecttango/ (visited on 2014-03-15) (cit. on p. 3)

[15] R HEVNER VON ALAN, SALVATORE T MARCH, JINSOO PARK, and SUDHA RAM "Design science in information systems research" in: *MIS quarterly* 28.1 (2004) (cit. on p. 3)

[16] KEN PEFFERS, TUURE TUUNANEN, CHARLES E GENGLER, MATTI ROSSI, WENDY HUI, VILLE VIRTANEN, and JOHANNA BRAGGE "The design science research process: a model for producing and presenting information systems research" in: *Proceedings of the first international conference on design science research in information systems and technology (DESRIST 2006)* 2006, pp. 83–106 (cit. on p. 3)

[17] LIU JUN, WANG QIUZHEN, and GAO LIN "Application of Agile Requirement Engineering in Modest-Sized Information Systems Development" in: *Software Engineering (WCSE), 2010 Second World Congress on* vol. 2 2010, pp. 207–210 DOI: 10.1109/WCSE.2010.105 (cit. on pp. 5, 7)

[18] L. MACAULAY "Requirements Capture as a Cooperative Activity" in: *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on* 1993, pp. 174–181 DOI: 10.1109/ISRE.1993.324820 (cit. on p. 5)

[19] H. SHARP, A. FINKELSTEIN, and G. GALAL "Stakeholder Identification in the Requirements Engineering Process" in: *Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on* 1999, pp. 387–391 DOI: 10.1109/DEXA.1999.795198 (cit. on p. 6)

[20] KLAUS POHL and CHRIS RUPP *Requirements Engineering Fundamentals* O'Reilly Media, Inc, 2011 (cit. on pp. 6, 7, 10–12)

[21] "IEEE Guide for Developing System Requirements Specifications" in: *IEEE Std 1233, 1998 Edition* (1998), pp. 1–36 DOI: 10.1109/IEEESTD.1998.88826 (cit. on pp. 6, 7, 11)

[22] "IEEE Recommended Practice for Software Requirements Specifications" in: *IEEE Std 830-1998* (1998), pp. 1–40 DOI: 10.1109/IEEESTD.1998.88286 (cit. on pp. 6, 10, 11)

[23] "Systems and software engineering – Life cycle processes – Requirements engineering" in: *ISO/IEC/IEEE 29148:2011(E)* (2011), pp. 1–94 DOI: 10.1109/IEEESTD.2011.6146379 (cit. on p. 6)

[24] "Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models" in: *ISO/IEC 25010:2011* (2011), pp. 1–34 (cit. on p. 6)

[25] A. BATOOL, Y.H. MOTLA, B. HAMID, S. ASGHAR, M. RIAZ, M. MUKHTAR, and M. AHMED "Comparative Study of Traditional Requirement Engineering and Agile Requirement Engineering" in: *Advanced Communication Technology (ICACT), 2013 15th International Conference on* 2013, pp. 1006–1014 (cit. on p. 7)

[26] J.L. DE LA VARA, L. HOYOS, E. COLLADO, and M. SABETZADEH "Towards Customer-Based Requirements Engineering Practices" in: *Empirical Requirements Engineering (EmpiRE), 2012 IEEE Second International Workshop on* 2012, pp. 37–40 DOI: 10.1109/EmpiRE.2012.6347680 (cit. on p. 7)

[27] S. ANWER and NAVEED IKRAM "Goal Oriented Requirement Engineering: A Critical Study of Techniques" in: *Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific* 2006, pp. 121–130 DOI: 10.1109/APSEC.2006.38 (cit. on p. 7)

[28] D. PANDEY, U. SUMAN, and A.K. RAMANI "An Effective Requirement Engineering Process Model for Software Development and Requirements Management" in: *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on* 2010, pp. 287–291 DOI: 10.1109/ARTCom.2010.24 (cit. on p. 8)

[29] I. SOMMERVILLE "Integrated Requirements Engineering: A Tutorial" in: *Software, IEEE* 22.1 (2005), pp. 16–23 DOI: 10.1109/MS.2005.13 (cit. on p. 8)

[30] YVONNE ROGERS, HELEN SHARP, and JENNY PREECE *Interaction design: beyond human-computer interaction* John Wiley & Sons, 2011 (cit. on pp. 8, 9)

[31] D. MISHRA, A. MISHRA, and A. YAZICI "Successful Requirement Elicitation by Combining Requirement Engineering Techniques" in: *Applications of Digital Information and Web Technologies, 2008. ICADIWT 2008. First International Conference on the* 2008, pp. 258–263 DOI: 10.1109/ICADIWT.2008.4664355 (cit. on p. 9)

[32] J.A. MILLER, R. FERRARI, and N.H. MADHAVJI "Characteristics of New Requirements in the Presence or Absence of an Existing System Architecture" in: *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International* 2009, pp. 5–14 DOI: 10.1109/RE.2009.45 (cit. on p. 9)

[33] THOMAS STAHL, MARKUS VÖLTER, and KRZYSZTOF CZARNECKI *Model-driven software development: technology, engineering, management* John Wiley & Sons, 2006 (cit. on p. 11)

[34] RALPH P. GRIMALDI *Discrete and Combinatorial Mathematics; An Applied Introduction, 5th Ed.* Pearson, 2003 (cit. on pp. 12–14)

[35] PHILIP BONACICH "Factoring and weighting approaches to status scores and clique identification" in: *Journal of Mathematical Sociology* 2.1 (1972), pp. 113–120 DOI: 10.1080/0022250X.1972.9989806 (cit. on p. 13)

[36] PHILIP BONACICH "Some unique properties of eigenvector centrality" in: *Social Networks* 29.4 (2007), pp. 555–564 DOI: 10.1016/j.socnet.2007.04.002 (cit. on pp. 13, 41)

[37] AMY N LANGVILLE and CARL D MEYER *Google's PageRank and Beyond: The Science of Search Engine Rankings* Princeton University Press, 2011 (cit. on p. 14)

[38] MICHAEL WORBOYS "Modeling Indoor Space" in: *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness* 2011, pp. 1–6 DOI: 10.1145/2077357.2077358 (cit. on pp. 14, 41)

[39] A. RITUERTO, A.C. MURILLO, and J.J. GUERRERO "Semantic labeling for indoor topological mapping using a wearable catadioptric system" in: *Robotics and Autonomous Systems* 62.5 (2014), pp. 685–695 DOI: 10.1016/j.robot.2012.10.002 (cit. on p. 15)

[40] THOMAS H. CORMEN, CHARLES E. LEISERSON, RONALD L. RIVEST, and CLIFFORD STEIN *Introduction to Algorithms, Third Edition* The MIT Press, 2009 (cit. on pp. 15, 16)

[41] ANDREW V. GOLDBERG "Point-to-Point Shortest Path Algorithms with Preprocessing" in: *SOFSEM 2007: Theory and Practice of Computer Science* vol. 4362 Springer Berlin Heidelberg, 2007, pp. 88–102 DOI: 10.1007/978-3-540-69507-3_6 (cit. on pp. 16, 23)

[42] E.W. DIJKSTRA "A Note on Two Problems in Connexion with Graphs" in: *Numerische Mathematik* 1.1 (1959), pp. 269–271 DOI: 10.1007/BF01386390 (cit. on p. 16)

[43] P.E. HART, N.J. NILSSON, and B. RAPHAEL "A Formal Basis for the Heuristic Determination of Minimum Cost Paths" in: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968), pp. 100–107 DOI: 10.1109/TSSC.1968.300136 (cit. on p. 17)

[44] S. J. RUSSEL and P. NORVIG *Artificial Intelligence: A Modern Approach, 3rd Ed.* Prentice Hall, 2009 (cit. on pp. 18–21)

[45] S.G. VADLAMUDI, S. AINE, and P.P. CHAKRABARTI "MAWA* - A Memory-Bounded Anytime Heuristic-Search Algorithm" in: *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 41.3 (2011), pp. 725–735 DOI: 10.1109/TSMCB.2010.2089619 (cit. on p. 21)

[46] IRA POHL "Bi-directional and Heuristic Search in Path Problems" PhD thesis 1969 (cit. on p. 21)

[47] MICHAEL N. RICE and VASSILIS J. TSOTRAS "Bidirectional A* Search with Additive Approximation Bounds" in: *SOCS* AAAI Press, 2012 (cit. on p. 22)

[48] T. IKEDA, MIN-YAO HSU, HIROSHI IMAI, S. NISHIMURA, H. SHIMOURA, T. HASHIMOTO, K. TENMOKU, and K. MITOH "A fast algorithm for finding better routes by AI search techniques" in: *Vehicle Navigation and Information Systems Conference, 1994. Proceedings., 1994* 1994, pp. 291–296 DOI: 10.1109/VNIS.1994.396824 (cit. on pp. 22, 58)

[49] KONSTANTIN TRETYAKOV, ABEL ARMAS-CERVANTES, LUCIANO G. B., JAAK VILO, and MARLON DUMAS "Fast Fully Dynamic Landmark-based Estimation of Shortest Path Distances in Very Large Graphs" in: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management* 2011, pp. 1785–1794 DOI: 10.1145/2063576.2063834 (cit. on pp. 22, 24, 41)

[50] ULRICH LAUTHER "An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background" in: *Geoinformation und Mobilität* vol. 22 2004, pp. 219–230 (cit. on p. 22)

[51] EKKEHARD KÖHLER, ROLF H. MÖHRING, and HEIKO SCHILLING "Fast point-to-point shortest path computations with arc-flags" in: *IN: 9TH DIMACS IMPLEMENTATION CHALLENGE* 2006 (cit. on p. 22)

[52] RONALD J. GUTMAN "Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks" in: *ALENEX/ANALC* 2004, pp. 100–111 (cit. on p. 23)

[53] ANDREW V. GOLDBERG, HAIM KAPLAN, and RENATO F. WERNECK "Reach for A*: Efficient point-to-point shortest path algorithms" in: 2006, pp. 129–143 (cit. on p. 23)

[54] PETER SANDERS and DOMINIK SCHULTES "Highway Hierarchies Hasten Exact Shortest Path Queries" in: *Algorithms – ESA 2005* vol. 3669 Springer Berlin Heidelberg, 2005, pp. 568–579 DOI: 10.1007/11561071_51 (cit. on p. 23)

[55] ROBERT GEISBERGER, PETER SANDERS, DOMINIK SCHULTES, and DANIEL DELLING "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks" in: *Experimental Algorithms* vol. 5038 Springer Berlin Heidelberg, 2008, pp. 319–333 DOI: 10.1007/978-3-540-68552-4_24 (cit. on p. 23)

[56] DANIEL DELLING, PETER SANDERS, DOMINIK SCHULTES, and DOROTHEA WAGNER "Engineering Route Planning Algorithms" in: *Algorithmics of Large and Complex Networks* Springer-Verlag, pp. 117–139 DOI: 10.1007/978-3-642-02094-0_7 (cit. on p. 23)

[57] ANDREW V. GOLDBERG and CHRIS HARRELSON "Computing the Shortest Path: A Search Meets Graph Theory" in: *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms* Society for Industrial and Applied Mathematics, 2005, pp. 156–165 (cit. on pp. 23, 24, 38)

[58] ALEXANDROS EFENTAKIS and DIETER PFOSER "Optimizing Landmark-Based Routing and Preprocessing" in: *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science* 2013, 25:25–25:30 DOI: 10.1145/2533828.2533838 (cit. on p. 24)

[59] FRODE NIKOLAISEN *Håper dette gir bedre forhold* 2013 URL: http://www.stolav.no/no/Nyheter/Haper-dette-gir-bedre-forhold/ (visited on 2013-08-30) (cit. on p. 26)

[60] CISCO "Norway University Hospital 'Mobilizes' Patients with Cisco and MazeMap" in: (2013) (cit. on p. 27)

[61] STEVEN ALTER "The Work System Method: Systems Thinking for Business Professionals" in: *Proceedings of the 2012 Industrial and Systems Engineering Research Conference, Orlando, Florida* 2012 (cit. on p. 27)

[62] ROD HILTON *Traveling Salesperson: The Most Misunderstood Problem* 2012 URL: http://www.nomachetejuggling.com/2012/09/14/traveling-salesman-the-most-misunderstood-problem/ (visited on 2014-04-20) (cit. on p. 36)

[63] PYTHON WIKI *CPython: Time Complexity* 2012 URL: https://wiki.python.org/moin/TimeComplexity (visited on 2014-04-21) (cit. on p. 40)

[64] *CPython* URL: https://www.python.org/ (visited on 2014-03-16) (cit. on pp. 45, 64)

[65] *NumPy* URL: http://www.numpy.org/ (visited on 2014-03-16) (cit. on pp. 45, 64)

[66] *matplotlib* URL: http://matplotlib.org/ (visited on 2014-03-16) (cit. on pp. 45, 64)

[67] *mpltools* URL: http://tonysyu.github.io/mpltools/ (visited on 2014-03-16) (cit. on p. 45)

[68] *virtualenv* URL: https://virtualenv.pypa.io/ (visited on 2014-03-16) (cit. on p. 45)

[69] "Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems" in: *ISO 9241-210:2010* (2010), pp. 1–32 (cit. on p. 63)

[70] *NetworkX* URL: http://networkx.github.io/ (visited on 2014-03-16) (cit. on p. 64)

[71] *SciPy* URL: http://www.scipy.org/ (visited on 2014-03-16) (cit. on p. 64)