Maption - making a scalable accessibility instrument

Thesis (KISPECI1SE)
GitHub: `https://github.itu.dk/trhj/Master-s_Thesis.git`

Troels Hjarne - trhj@itu.dk (18542),
Anders Skaaden - anska@itu.dk (18619),
Johannes Bertoft - olbe@itu.dk (18620),
Christian Andersen - cpan@itu.dk (18613)

Supervisor:
Maria Sinziiana Astefanoaei

IT UNIVERSITY OF COPENHAGEN

Software Design
IT University of Copenhagen
Denmark
June 2022

Contents

Abstract

The negative side-effects of urbanization and motorized transport have increased attention to concepts such as the 15-minute city, promoting walking-distance neighborhood access to basic amenities. Research on accessibility suggest that there are conflicting opinions on the usefulness of general metrics, and a lack of coherence between objective indicators and subjective perceptions of accessibility. A variety of accessibility instruments have been developed to aid planning in this context, but are often confined in geographical coverage, pre-computed metrics and a limited selection of amenities. This thesis analyzes and implements a proof-of-concept of a scalable instrument that can overcome these limitations. This is done by modelling and pre-processing OpenStreetMap data and utilizing spatial queries and contraction hierarchies to fetch-and-create user-defined networks. These can be used for computing fast on-demand distance queries with any selection of amenities, which are visualized in a web-GIS application. The results show that the architecture can enable more globally scoped accessibility instruments, providing both urban planners and regular citizens with customizable accessibility metrics.

Keywords - Accessibility, Contraction hierarchies, OpenStreetMap, Street networks, walkability, 15-minutes city

1 Introduction and motivation

1.1 Context

In recent years, there has been an increased amount of focus toward urban city-planning, in response to the growing population density in major cities and high amounts of daily commuters.

As of today, about 55 pct. of the world's population is situated in an area that is considered urban. The ongoing urbanization growth is expected to continue. According to forecasts, the urban population will grow to 5 billion by 2028 and 6 billion by 2041, accounting for 60 pct. and 68 pct. of the world's estimated population [74].

Our transportation networks, travel behaviour and spatial patterns have consequently changed in many European cities over the last decades. Some of these changes have resulted in increased average travel distances and level of car dependence as a daily transportation mode [10]. Mobility has become an important element of our daily lives, but also a factor defining the functionality of our cities and urban regions.

There is an increased awareness of the multiple negative side effects that these transportation habits bring, such as congestion, air pollution, noise and potential social exclusion based on your place of residence [4].

One of the more promising solutions within sustainable transportation, is shifting from motorized individual transport towards non-motorized or so-called *active mobility*, which entails transportation either by walking or cycling.

In relation to this, there is an increased attention to concepts such as the '15-minute-city', which is an urban-planning concept based on designing cities around people and their everyday needs, ensuring their accessibility to a variety of different services and amenities [13].

Given the above, an increasing amount of accessibility instruments have been developed to assist practitioners in decision-making processes when planning for active mobility during the last decade, in contrast to conventional transport modelling tools which focus on distribution of traffic [9].

1.2 Making a scalable accessibility instrument

This thesis project is a continuation of our research project '15-Minute Cities: Personalizing accessibility metrics'[70], and the web-application prototype that it concluded with.The research project focused on assessing different ways accessibility could be analyzed and more specifically the complexity surrounding subjective accessibility. The web-app prototype was developed with a personalized accessibility metric, where defined cities as one spatial unit each could be compared side-by-side. The prototype relied on pre-computed data for each city and created a storage limitation in terms of geographical space being covered and amount of different amenities available.

The research project and the preliminary research work that was done, highlighted some key areas that an improved application should consider while being developed.

Firstly, there has been a contradictory stream of research, on one side there has been a focus on "increased complexity and thoroughness" [33] for accessibility instruments, while on the other also highlighting "the importance of "simple, usable and understandable instruments for planning practice" [33].

Secondly, due to a historic disconnect between the instrument developers and the potential end-users, many technologies have been developed without a shared understanding of what the end-user aim to achieve in a specific planning context. This has led to many instruments being regarded as complex and rigid black boxes [10].

Thirdly, there is a lack of research surrounding accessibility instruments for active mobility. With active mobility comes the challenge of the diversity of people's mobility behavior, which motivated a focus on perceived- and personalized accessibility [52].

A more recent contribution that has addressed these findings is the GOAT (Geo Open Accessibility Tool) from a research group at Technical University of Munich (TUM), and now the startup Plan4Better. This thesis has drawn inspiration from GOAT and similar GIS applications. The identified implementations that are publicly available are generally confined to pre-defined city-boundaries and a limited amount of locations. This is most likely due to local data set dependencies, computational and storage limitations or similar.

In light of the context and problem description, this thesis centers around

developing a prototype web-GIS application for accessibility analysis, focused on active mobility. Its' overarching goal is to account for both shortcomings addressed in academic research, and gaps identified in currently available tools. We therefore design and implement "Maption", a globally scoped application that allow the user to freely define the area of study and selection of amenities, which can provide assistance as a support system when planning for local accessibility. It is built using data from OpenStreetMap (OSM), an openly available, community-built database for spatial data. The tool utilizes only open-source software and openly available data, to make it transparent and available for anyone.

1.3 Approach

The process builds upon an iterative software development process, following three phases.

The first phase consists of a literature review on accessibility and available tools. It resulted in identification of a lack of agreements in terms of metric calculation, lack of instruments that allow for fast on-demand aggregate queries in a user-defined area, and as a result an accessibility computation that is independent of area or data dependencies.

The gaps identified in the first phase were guiding principles in the second phase where iterations of performing technical feasibility analyses of different tool-kits and implementing parts of the prototype. Since building an end-to-end solution required a full stack of tools, the iterations often involved exploratory phases to identify limitations with the set of tools used.

The last phase consists of performance evaluation to ascertain the feasibility of the application according to the requirements specified and defined use-case. This phase included running performance tests of the flows and procedures through all parts of the application and identifying bottlenecks that could serve as a focus area for further development.

1.4 Contributions

We have designed and developed a prototype of an end-to-end, web-based GIS application allowing aggregate network analysis of accessibility, with a focus on walkability. Its' main purpose is to allow the user to define an area

and perform distance queries (such as "distance to k nearest supermarket") for *any* combination of amenities. The results of these queries are used to show the distances from every intersection of walking paths in a web interface. The application use-case can both be applied as an analytical tool for urban-planning and for regular citizens to identify walkable areas in their city.

The core features and the problems they solve:

- Customizable user-defined area - completely area-independent that can be extended globally

  Solves: Previous implementations focused on aggregate analysis are limited in area-coverage. This is either because data sources are area-specific, or because aggregate accessibility metrics have been pre-computed which makes it storage-heavy and limit customizability.

- Computation and data storage designed for fast on-demand network queries.

  Solves: Research suggests aggregate network accessibility tools in the past have been too slow for on-demand computation [10]. Pre-computation can solve this for predefined metrics and areas, but complete extendability and customizability requires on-the-fly computation.

- Unopinionated and versatile Point of Interest accessibility calculations

  Solves: Previous research indicates lack of agreement on accessibility metrics [61, 10], and implementations of tools are often confined to specific amenities or complex scoring schemes. Our implementation enable simple metrics and completely user-selected Points of Interest.

These features have been accomplished by building an accessibility instrument that combines the following components:

- A scalable database design and implementation in PostgreSQL/PostGIS to utilize spatial queries for customized areas

- A GeoJSON compatible API that can be used to perform aggregate network queries utilizing the fast query processing of the Pandana library

- A graph construction algorithm modified from the OSMnet library to preserve the geometries of the street network when querying the OpenStreetMap (OSM) database
- An interactive map-based Single Page Application, to query and visualize the results.

The thesis is structured as follows. In chapter 2, we define accessibility and analyze practices of measuring accessibility indicators. Furthermore, we review existing available accessibility instruments, and the OpenStreetMap (OSM) data ecosystem, that lays the data foundation of Maption. In chapter 3, the scope and requirements of Maption is presented together with an analytical framework from which we evaluate components of the architecture. Chapter 4, analyzes and elaborates upon the architectural choices of Maption. In chapter 5, we give a high-level introduction to the web application interface, followed by chapter 6, where key design and implementation aspects are presented at all levels of the application. Chapter 7 presents experiments conducted to evaluate the performance of different parts of the application. In chapter 8 we evaluate the feasibility and limitations of the architecture in relation to the system requirements, and present ideas for future work. Finally, we summarize the findings in chapter 9.

2 Background

The following section will introduce *accessibility* as a concept, and define the type of accessibility this thesis is based on. The section will also research the associated practices of measuring and computing accessibility and the implications of these metrics and their use cases. From this we derive a suitable metric approach for the rest of the project.

In addition, data within the domain of accessibility will briefly be described, as well as the possibilities and limitations of using spatial data from OSM.

Finally, the section will assess some of the existing accessibility instruments, in regards to their respective designs, functionalities and qualities. Based on this assessment, shortcomings will be identified as key implementation objectives for this project.

2.1 Defining accessibility

The term accessibility has been used in many urban planning contexts. Hansen's seminal paper published in 1959, defined the concept as "the potential of opportunities for interaction" [32]. Since then, the term has been used in studies measuring and analysing both physical distance, behavioral mobility as well as access to social and economic opportunities [24, 31, 61, 38, 39].

As these contexts are different, so is the interpretation of the term accessibility. The type of accessibility in scope of this project can be characterized by using the framework proposed by Michael Batty [8]. This framework identifies two primary types of accessibility.

Type 1 consider accessibility as a general measurement of locational behavior. This type of accessibility focuses on human interaction with places, and social and economical activity. Research into this aspect of accessibility often analyse travel patterns and social aspects of travel behavior, for instance using demographics and aggregate mobility data [31].

Type 2 approaches the physical aspects of accessibility often in the context of spatial network analysis. This type is distance-centric, using physical proximity between locations and Points-of-Interest (POI) as a measure of how reachable places are [8]. It is frequently used in the context of measuring and

improving urban planning. Studies built on this type often promote frameworks using network data models and open source data, for their simplicity and data availability [22, 39].

Given these two contrasting types of accessibility, another stream of research has focused on measuring coherency between physical proximity and social and economic activity, thereby applying both of Batty's accessibility types. For instance, studies of activity-participation have shown how disaggregate models that take both types into account can better model how actual behavior and physical proximity is related [24]. This context has also led to scrutiny of generalized accessibility metrics, and how they fail to account for subjectivity [61].

This thesis builds upon frameworks of network analysis [22, 39], that study accessibility from a **type 2** perspective. However, while the outset is an instrument using physical proximity between places and Points-of-Interest, the aim is also to account for subjectivity by building a flexible system where an element of preference is included in the tool.

2.2 Perceived accessibility

Using accessibility as a measure of distances and social activity is common practice in both research, city-planning, and public transportation [43]. It is often measured based on generic and objective metrics, without accounting for the human aspect and the subjectivity that lies within personal accessibility preferences. Especially when using accessibility of **type 2**, measurements are often based on proximity to a fixed set of amenities. In many use cases this approach has the advantages of being simple and highly comparable across geographical areas [43]. However, computing indicators of accessibility based on these amenities and fixed weights can end up disregarding subjective perceptions of accessibility [61].

Contrary to this approach, research into subjective accessibility attempts to measure accessibility from a human-centric viewpoint, accounting for preferences of individuals living in the geographical area of study [37]. This area of research attempts to improve the analytical measures and city-planning efforts by interpreting results based on perceptions and experiences collected from the population. Because subjective perceptions of accessibility differs across demographic traits such as age and gender, it is more difficult to gen-

eralize.

A study from 2016 by Lätmann [38] showed discrepancy between generic accessibility measures and perceived accessibility measures by questioning a sample of 500 people in the Swedish city Karlstad. The sample of the population was asked to rate the accessibility of public transportation based on a number of questions, stating to what degree they agreed with the statement. The outcome of the study showed that the accessibility score obtained by generic metrics were higher, and thereby gave a false indication of higher accessibility compared to the score of the perceived accessibility metric.

With the identified discrepancies between subjective perception and the objective measures frequently used in various fields of research [38], it is therefore important to clarify different practices of measuring accessibility.

2.3 Practices of measuring accessibility

When measuring accessibility, the aim is to translate the concept of accessibility into an indicator that incorporate one or more aspects from land use, transportation system, economic benefit or human preferences and constraints [10].

Since there is a wide scope of factors affecting accessibility, multiple accessibility metrics have been developed. In general they all are comprised of two basic components, the cost of travel (determined by the spatial distribution of travelers and opportunities) and the quantity of opportunities [51].

Accessibility can be measured from the perspective of the location of origin ('supply of opportunities') or from the standpoint of the destination ('demand of participants') [34]. An online survey completed by 358 land use and transportation practitioners [72] indicated that the accessibility metrics being used are generally based on travel time or distance.

The following sections describe three common approaches to modelling, and measuring, accessibility, namely;

- Distance to nearest location
- Cumulative opportunities
- Gravity-based measures

Finally, we summarise the trade-offs and explain how we approach the choice of methodology.

2.3.1 Distance to nearest location

The distance to nearest location is arguably the most simple model of the three. This model stipulates that among a set of locations L of an individual will always choose the location (e.g. supermarkets) that is closest in distance or travel time [42] Formally, this can be expressed by:

$$A_i = \min\{j \in L | d_{ij}\}$$

Where:

A_i = the accessibility of zone i to the nearest location in L.

$j \in L$ = each POI j in the location set L.

d_{ij} = the distance (in time or distance) from zone i to location j.

With its simplicity come some inherent limitations in terms of modelling the real world. Firstly, it treats all locations of the same type equally, not taking into account the attractiveness of the location. Secondly, it does not account for the cumulative effect of having multiple locations nearby - i.e. it considers it more attractive with 1 restaurant within 500 meters than 2 restaurants 600 and 700 meters away [42].

As a standalone indicator, used by modellers, this may be too simplistic. However, it is argued to have some value as an explanatory variable in residential choice-models [42]. Given the straightforward definition it is also highly transparent, which makes it suitable for an instrument targeting both planners and regular citizens.

2.3.2 Cumulative opportunities (isochrone) approach

The cumulative opportunities approach, also refered to as isochrone analysis, is the most common location-based metric in planning practice. [42] The metric counts all opportunities (e.g. POIs) that are within a travel cost threshold D (a maximum distance or travel time). This metric indicates that accessibility increases if more opportunities can be reached within the

travel cost threshold. An example could be the amount of jobs that are within 45 min of travel time to indicate the level of access to jobs from a specific origin. The accessibility measure can be defined by the equation below [53]:

$$A_i = \sum_j O_j f(C_{ij})$$

Where:

A_i = accessibility of A of origin i
O_j = number of opportunities available at destination j
C_{ij} = cost of travelling from i to j (travel time)
$f(C_{ij}) = 1$ if within threshold D, else $f(C_{ij}) = 0$

The main advantage of the metric is that it is rather simple to compute and easy to understand for the practitioner. Its' simplicity comes with some methodological issues that can arguably limit its general use. Notably, it assumes that the traveler is indifferent between travel cost between competing locations, as long as they all situate within the travel threshold D. Considering a scenario with a set of locations that are 10 min away from zone i (D = 10 min), and the same ones located 30 min away from the zone (D = 30 min), it's natural to conclude that the former option should have a higher accessibility level than the last one.

Similarly, it follows that the measure excludes all locations that are outside the defined threshold when measuring the location i's accessibility score. To have this strict cut-off goes against travel behaviour theory [42], if the level of accessibility immediately drops to 0 when we have reached the threshold in terms of time or distance. This is naturally not the case in real life.

Lastly, the threshold D must be defined and asserted by each respective user, as there are no established standards in research or practice as of now. A common practise however is to use thirty, forty, or forty-five minute thresholds [42], usually based on the assumption that the majority of trips occur within either range, or that there is no indifference between the travellers within the range. Again an assertion that does not have strong grounds within travel behaviour theory.

2.3.3 Gravity-based measures

Conversely, we have the gravity-based measures which are more reflective of true travel behaviour, due to weighing opportunities less that are further away (by distance or time), than closer opportunities. In terms of a behavioural point of view and flexibility, this represents an improvement compared to the strict cut-off threshold from the cumulative opportunity metric. [42]

We can re-use the previous definition, but with an adjusted impedance function. [53]:

$$A_i = \sum_j O_j f(C_{ij})$$

Where:
$f(C_{ij})$ = Impedance function; e $^{\beta}\backslash C_{ij}$
β = a decay function coefficient

The impedance function represents the growing inconvenience of accessing a point the further away it is. In the above case, the impedance function used is a negative exponential, but can be estimated similarly with different functional forms. Frequently used functions are power, negative exponential, logistic and Gaussian functions [28].

The decline will barely be noticeable among locations with similar, shorter distances. The decline will rapidly scale when in the neighbourhood of some threshold (where the travel impedance/cost is becoming increasingly more apparent), until reaching some point where the choice probabilities become negligible small.

By doing so, the measure is more consistent in regards to general travel behaviour, that any noticeable change of travel cost should consequently affect the general accessibility level. Nonetheless, it does not account for the personal characteristics of the individual that might measure the accessibility. All individuals in the same location will be treated equal despite the chance that the possibilities would not apply to them (unqualified for certain jobs for example).

2.3.4 Choosing k-nearest POIs as accessibility measurement

This thesis will use the distance to nearest location approach with a slight modification. Its' simplicity and relatable metric and units can serve both as an entry-point for urban planners and regular citizens. When building an interactive instrument, rather than using the measure as a standalone indicator for research purposes, it may also provide a layer of customization that can partly account for its' limitations as a standalone indicator of accessibility.

We must of course acknowledge the limitations established with the metric, and how cumulative and gravity based approaches can account for these. The cumulative opportunities approach can account for both the amount of locations, and the attractiveness of the locations, but lack the travel-cost aspect, whereas the gravity-based approach is able to account for both of these. Despite this advancement, cumulative-opportunity measures are still easier to generate, interpret and communicate [29], and consequently more commonly used in planning. Although the gravity-based approach could be a better metric indicator for planners, it could at the same time suffer from complexity.

We therefore attempt to take some of limitations into consideration, and adopt the k-nearest location (POIs) variant, to allow the user to see a cumulative effect of having more than just a measure of the nearest location (and rather the k nearest locations). With this choice we build an architecture which could factor in weighting in terms of attractiveness in the future to provide the option of a gravity-based metric. In addition our implementation will focus on simplicity and the element of preference and personalization by the free choice of amenities.

2.4 Modelling the city as a graph data structure

In order to compute the accessibility measures described above, one must first choose a spatial unit of analysis [22]. Zones, or blocks have long dominated in a majority of transportation models because of their computational simplicity, but suffer from a few drawbacks. Firstly, zones have to be manually defined, and can become arbitrary in scope. Secondly, they cannot accurately model micro land-use - this becomes especially apparent in regards to walkability [22]. As argued by Long and Shen [40] zones, or cells, are often

low-resolution in space, sometimes covering several square kilometers in large urban areas.

More recently, there has been a push for using a more natural and accurate representation of urban areas, using urban parcels (piece of land/ground) as the minimal spatial unit - mapped to its enclosing street network [22, 40]. Measures based on street-level geography can provide greater accuracy in cost functions, removing dependence on "as-the-crow-flies" measurements, and can in part be attributed to improvements in the field of graph algorithms to solve a variety of computational problems in the domain of street network analysis [22].

Modelling the city as a network has a long-standing tradition [22]. Thanks to continuous improvements in the study of graph algorithms and the advent of sources such as OpenStreetMap, we now have open access to the data to produce accurate street geography models.

In essence, a network, or *graph* is a data structure which models complex, non-linear relationships between objects. These objects are represented as a set of vertices V, and the relationships between them as a set of edges E. Each edge references a connection between two vertices. Edges can either be directed or undirected, indicating a one-way or unidirectional relationship, and weights can be associated with edges. In a street network, a graph $G = (V, E)$ can simply model intersections as the set of vertices, and street segments connecting these intersections as the set of edges. The weight of an edge is most often used to represent the "cost" of travel between two vertices, by some impedance function, usually related to distance or time, as described in section 2.3.

Foti et al [22], promotes the use of this street-network model where parcels can be thought of as very small traffic analysis zones (TAZ). Each parcel centroid can be mapped directly to the nearest vertex (intersection) in the surrounding street network. The parcels associate agents of the city - e.g. points of interest, households, and buildings, which are linked through the street network. Metrics can thereafter be computed and aggregated upon each street node, e.g. the distance from node s to the node t that associates the *nearest* doctor from the set of doctors T. This model gives an accurate representation of pedestrian-scale accessibility that removes the issues of arbitrary scope of zones [22]. This measurement also enables accurate de-

tection of changes to the environment at a pedestrian scale that can be used in urban planning scenario building [22].

Due to the advantages of the graph data structure it will be the basis of our computational analysis. The next section will describe the underlying data source used to model the graph.

2.5 Using open data-sources

Building a graph data structure representing the street network, will require access to accurate street data to model the nodes and edges. Gathering spatial data across larger geographical areas can be an issue due to different governmental practices and standards [39]. Different standards often entail different formatting of the data, as well as different definitions of spatial phenomena, making it difficult to analyse and compare in spatial analysis [39]. To mitigate this issue, there exist open source solutions such as OSM.

2.5.1 OpenStreetMap

OSM is an open-source, community-built project since 2004, providing global spatial data to thousands of applications ranging from prototypes to enterprise-solutions and is widely known as an open, reliable geospatial data source [39]. OSM is considered one of the most successful VGI (voluntereed geographical information) projects. Anyone can sign up as a contributor to OSM for free. Since January 2022, there are more than 8.3 million registered users, between them there are contributions from at least 1.75 million unique contributors. OSM registers an approximate of 4 million map changes per day, which means the state of their geospatial database is continuously changing in a rapid pace [46].

The OSM database is utilizing a standardized data schema that supports additional data to be effectively inserted into the database. Due to its' wiki-style process where all users can add or edit the data, the project relies on that the majority will check, fix or get rid of data contributions that is either inaccurate, malicious or accidental. Accordingly, there is no guarantee of any accuracy to the physical world. However, this is not necessarily the case with commercial maps either.

One can argue that this is exactly the strength of OSM, allowing for quick

data updates when discrepancies are found or changes occur in the physical world. According to OpenStreetMap, this methodology generally makes their data more up-to-date and of higher quality than other commercial maps when dealing with recently changed streets [46].

However, the methodology also has limitations. Fonte et al [21], argue that the frequent changes to the same features can reduce the overall quality and usability of the geodata. In the case of OSM specifically, the database is composed by several large-scale imports from open-licensed sources where the data structure will be variable and tagging of features is left up to each contributor. As a result, the quality of OSM can vary slightly over space and time.

Another consideration is the skewness of data representation that will occur in VGI-projects such as OSM. In previous research it has been demonstrated a spatial bias of information, where urban areas will naturally have more data collected than in rural areas [21].

Previous research has been conducted to clarify the quality of OSM data to determine to what degree it can be a suitable source for accessibility analysis. Data from the OSM database has in that regard been compared to official sources in cities such as Hong Kong, Olomouc and Belfast [39]. The test consisted of direct comparison of nodes and edges for network analysis from both data sets and even though some discrepancy was present, OSM data was recognized as adequate for accessibility analysis and might be even more suited for analysis than the official data sources. The same research also suggest that OSM had an 88 percent correspondance with Google, based on cross validation with several different Google services [39]. With a global scope, requiring access to an open spatial data source, OSM can therefore be deemed a viable choice, despite its' limitations.

2.6 Existing accessibility instruments

The variety of measurement techniques, and focus areas of accessibility research, has given rise to a wide span of tools that can broadly be classified as accessibility instruments. The tools primarily concern computer models used to give life to data and information about urban systems, through map visualizations or numerical indicators [10].

The following sections describe and analyze five such accessibility instru-

ments. The purpose of this analysis is to identify possible gaps or shortcomings of existing instruments, with the ambition of addressing some of these limitations.

2.6.1 Classifying accessibility instruments

Accessibility instruments have been defined by the European Cooperation in Science and Technology (COST) as tools that *"aim to provide explicit knowledge on accessibility to actors in the planning domain [...] specifically developed to support planning practice (analysis, design support, evaluation, monitoring etc.) by measuring, interpreting and modelling accessibility"* [10].

In other words, if *accessibility* is the general abstraction of the concept as defined in section 2.1, and *accessibility measures* as described in section 2.3 are the quantification of indicators to represent the concept, one can think of *accessibility instruments* as the tools which can convey these measures to an audience (i.e. to provide insights for planning purposes).

Following the COST Action TU1002, the META Accessibility platform was created with the aim to "close the gap between academia and practice" [2]. By collecting, categorizing and reviewing accessibility instruments under one umbrella, the META platform lists tools and resources within the field. Four categorizations are used to identify the purpose of the tools, and thus lay an analytical foundation for the research space and the target areas by which to associate the tools with. These categories are

Mode of transport walking, cycling, public transport, motorized transport, intermodal transport

Spatial unit of analysis street level, building block, grid, district, traffic zone, municipal, other

Tool type web instrument, desktop instrument, software extension, guideline, other

Target group planners, academics, political decision makers, retail/real-estate developers, citizens, other

The coverage of the 31 tools listed on META shows that there are only four tools that are both web instruments and analysing walking as mode of

transportation. Only one tool - GOAT - also uses "street level" as spatial unit of analysis.

Apart from the tools listed on META, four other street-level walkability tools have been identified. This section covers these five accessibility instruments in greater detail to describe some of the previous implementations that have approached a combination of being web-based, supports walking as a mode of transportation, and using street level analysis. The target groups will also be discussed for these five implementations - Walk Score, The 15minute city-map by Here Technologies, 15-minutes.city, ParisCSL and GOAT. The instruments have in common that they operate within the '15 min cities' domain, although having different approaches, use-cases and scopes.

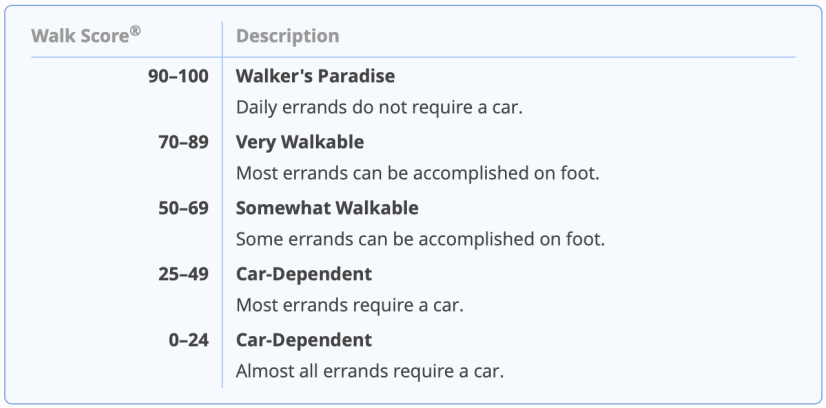
2.6.2 Walk Score

Walk Score is a web-application with the ambition, in their own words to *'make it easy for people to evaluate walkability and transportation when choosing where to live.'* [64]. Walk Score started as a widely open source project, transitioning to a commercial B2B-offering where companies can buy access to their products, which includes access to map implementation, data and API access. The use case of Walk Score is described as being in the realm of real estate, urban planning, government, public health, finance, analysts and researchers. Furthermore, Walk Score uses a variety of data sources, including Google, Factual, Great Schools, Open Street Map, the U.S. Census, Localeze, and places added by the Walk Score user-community [64]

Walk Score algorithm The Walk Score methodology is based on a point system where the distance to a given amenity is rewarded points based on a polynomial distance decay function. The points and distance have a negative relationship, with an upper boundary of 30 minutes, where no points are awarded. Each amenity within the 9 predefined categories (grocery, restaurants, shopping, coffee, banks, parks, schools, books, entertainment) are then weighted, based on the effect a given amenity is deemed to have on walkability. The distance, count and weights determine an area's walkability score and is then normalized to be in the range of 0 - 100 (fig. 1).

Research into Walk Score and walkability as concept, suggest that Walk Score's calculation of an area's walkability has several flaws [68]. The re-

Figure 1: Walk score [64]



Walk Score®	Description
90–100	**Walker's Paradise** Daily errands do not require a car.
70–89	**Very Walkable** Most errands can be accomplished on foot.
50–69	**Somewhat Walkable** Some errands can be accomplished on foot.
25–49	**Car-Dependent** Most errands require a car.
0–24	**Car-Dependent** Almost all errands require a car.

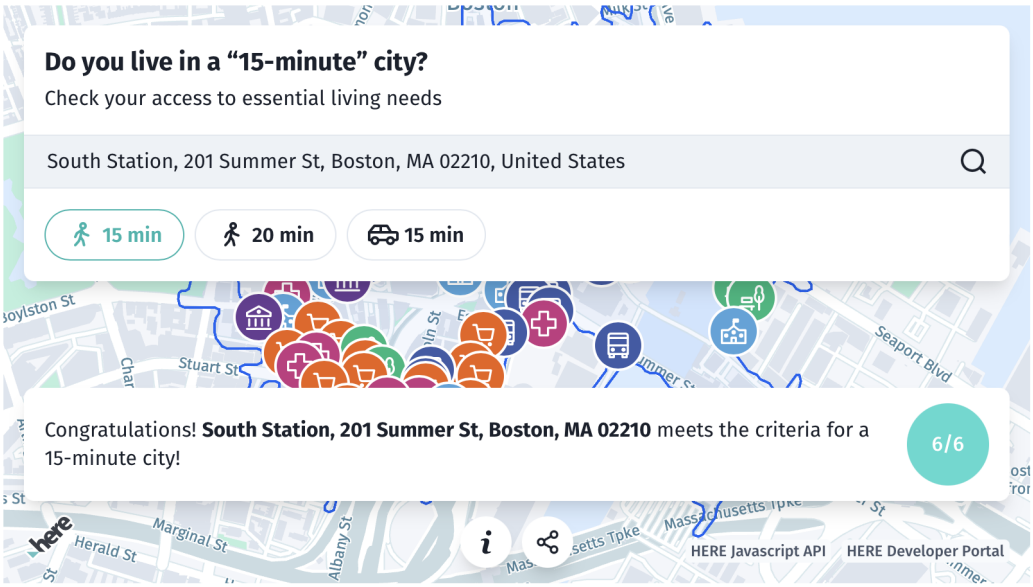
search suggests that there exist a discrepancy between an area's calculated walkability and the actual walkability in terms of how many people actually walk in the particular area. The discrepancy is observed using Google street view, and local knowledge of a given place. The research suggest that the use of smartphone location data, combined with data from cars, can contribute to a more realistic representation of an area's walkability index [68].

2.6.3 15minute city-map by Here Technologies

'15 minute city-map', is a product by the company Here, who operates and is engaged within the realms of location data, smart cities, transportation and logistics [71]. They offer a 15-min map, that based on a given address in the United states, calculates an area corresponding to 15 minute on foot or bike. Furthermore, it displays the existing amenities within that particular area and determines whether or not a given address meets the 15 minute criteria. The '15 minute city-map' is based on what can be defined as a single-source principle, where the accessibility metric is calculated with a single address point as the reference point.

The resulting visualisation can therefore be claimed to be more locally scoped. In contrast, the following two implementations show a more general view of accessibility, using aggregate scores that are computed and visualized for a whole city.

Figure 2: 15minute city-map by Here [71]



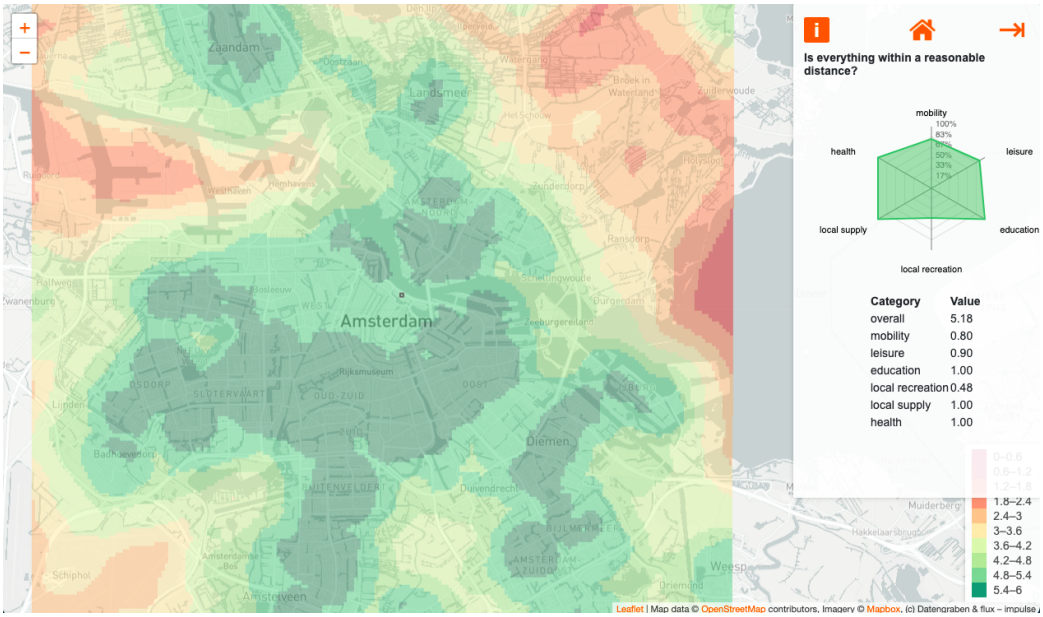
2.6.4 15-minutes.city

The ”15-minutes.city” implementation shows a typical interface for this type of implementation, where an entire city is colored based on measurements, or *scores*, for each area in a grid.

In the case of ”15-minutes.city”, the implementation is a beta version that features similar capabilities as the prototype implemented during the research project preceding this thesis, enabling the overview of a selection of large cities, with pre-defined boundaries. As shown, the implementation also features the possibility of selecting a point on the map, and it details the total score and score by category for the area selected. Categories are defined by the creators as groups of urban amenities that fall under mobility, health, local supply, education, leisure time, and recreation [1].

The differentiation of this tool compared to the tool by ’Here Technologies’ is an overall view of a city, rather than a point-based view. It naturally offers a number of use cases, for instance quickly localizing low accessibility areas that could be improved in an urban planning setting. However, as discussed in section 2.2, there is a need for more flexibility in both the choice of amenities used for analysis, and choice of accessibility metrics. The implementation lacks any filtering possibilities on amenities, and offers little

Figure 3: 15-minutes.city [1]



documentation on *how* the score is computed. This means that the accessibility becomes generalized, and to some extent arbitrary in precision.

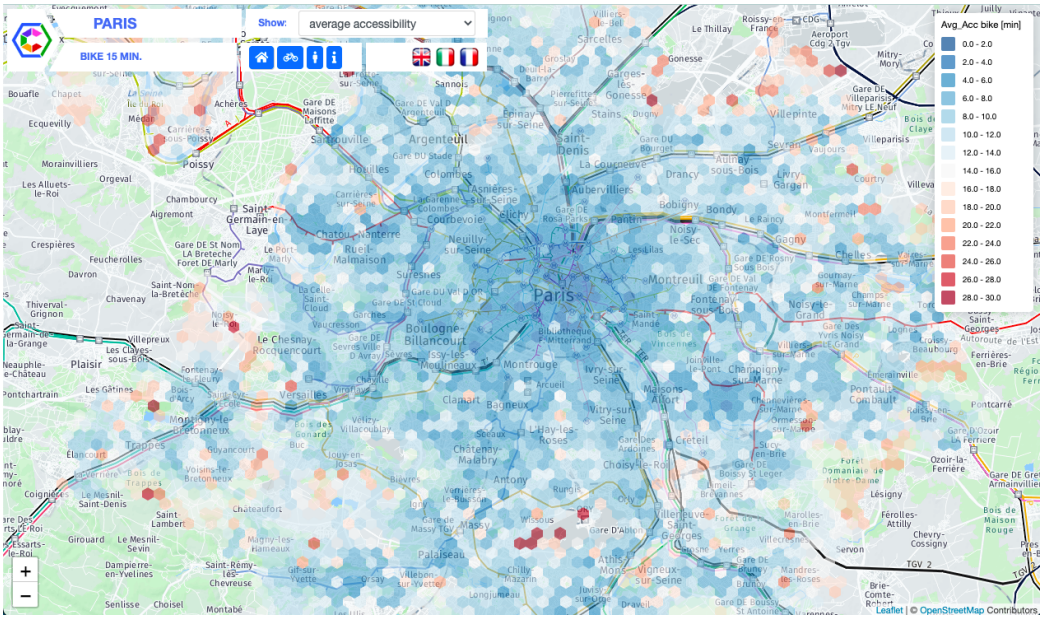
2.6.5 ParisCSL

Another instrument that implements the overall view is "The 15-minute City Platform", developed by Sony CSL Paris. The creators describe the instrument as an aid to re-think and plan organization of urban space, "while also enabling the ability to identify which areas of a city are struggling to offer an enjoyable neighbourhood-experience at an overall acceptable distance" [55].

This instrument is also directly associated to the 15 minute-city concept, promoting active mobility. In the top bar it gives the user the option of visualizing based on either walking or biking. It allows for filtering accessibility based on category of amenity, and rather than compute a score, it shows the average distance for each zone. It also features a radar diagram to visualize how distance metrics differ across categories as shown below.

The instrument claims to leverage open data projects such as OpenStreetMap combined with historical GPS traces, and routing algorithms to allow variations of quantifying accessibility. The instrument currently fea-

Figure 4: 15-minute city by ParisCSL [55]



tures 10 cities in Europe and is continuously growing [55].

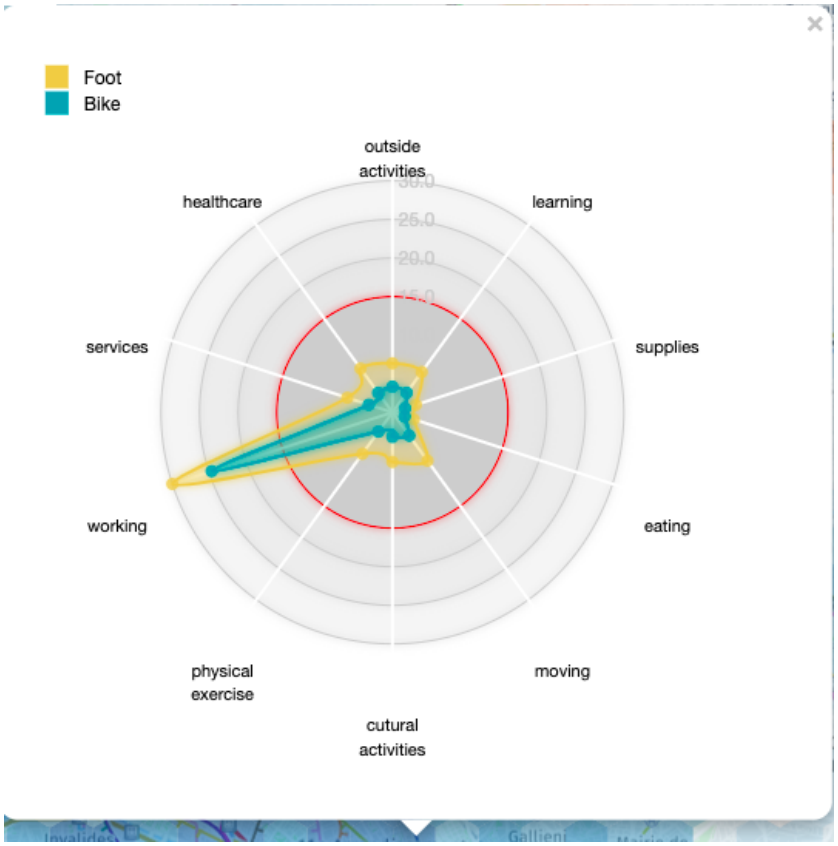
Although being free to use for anyone, the instrument is not open-source and therefore do not provide insight into how the data has been processed and the type of calculations and routing algorithms used to compute the metrics. It is limited in both geographical coverage and the amount of filtering capabilities available.

2.6.6 GOAT

GOAT is arguably the most sophisticated and well-documented of the described instruments. The analysis of GOAT will therefore be covered in greater detail, as they also utilize technologies that will be evaluated throughout the thesis.

GOAT (Geo Open Accessibility Tool) is designed to interactively analyze local accessibility, and act as a planning support system (PSS) for active mobility, which means either by walking or cycling. The mapping tool originates from environmental engineers, GIS developers and transport planners from Technical University of Munich (TUM) and has later developed into a startup called Plan4Better.

Figure 5: ParisCSL Radar diagram [55]



The developers initially conducted research on the suitability of accessibility instruments when planning for active mobility. The research revealed that the available instruments had been significantly further developed in recent years, and an increase in the number of accessibility instruments that contain novel features [52].

Despite this, they attested a clear gap of visualizing and modelling tools that are open-source, supporting interactive scenario building, while having a user-friendly UI. As a result, GOAT was developed and is aimed to be a flexible accessibility instrument which is highly customizable and extendable to multiple study areas.

Their main target group is practitioners within public authorities, planning offices and real estate, who have previous knowledge in the field of transport and urban planning. More specifically professionals that either need a time-efficient solution or do not posses adequate skills to perform

analyses on more advanced GIS software.

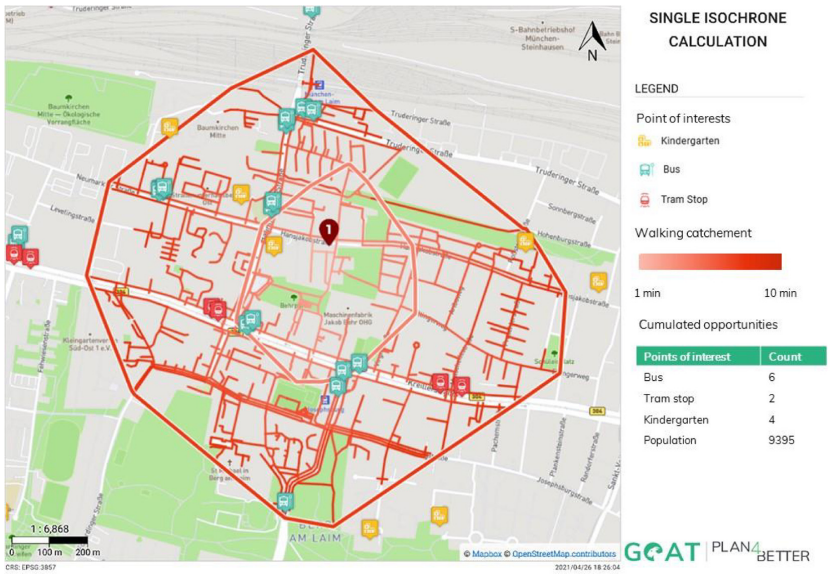
Correspondingly, the GOAT application can be considered as a middle ground between a WebGIS-application for the general public and an advanced fully-featured desktop software.

Both the source code and software used to build GOAT are fully open source. GOAT is mainly built with data from OSM. GOAT relies on PostgreSQL and the spatial extensions PostGIS and pgRouting for database management and routing algorithms.[52, 57]

To provide a user-centric solution which can be implemented in diverse study areas, they have chosen to use robust contour and gravity-based accessibility measures.

Contour-based accessibility measures GOAT is able to calculate and visualize isochrones using a 2D concave hull algorithm, which is implemented in a Javascript library called Concaveman. According to the GOAT-team, this was the 'most performant open source solution found on the web'[52].The library creates isochrones, defining the area from a set of nodes that can be reached in a specified time.

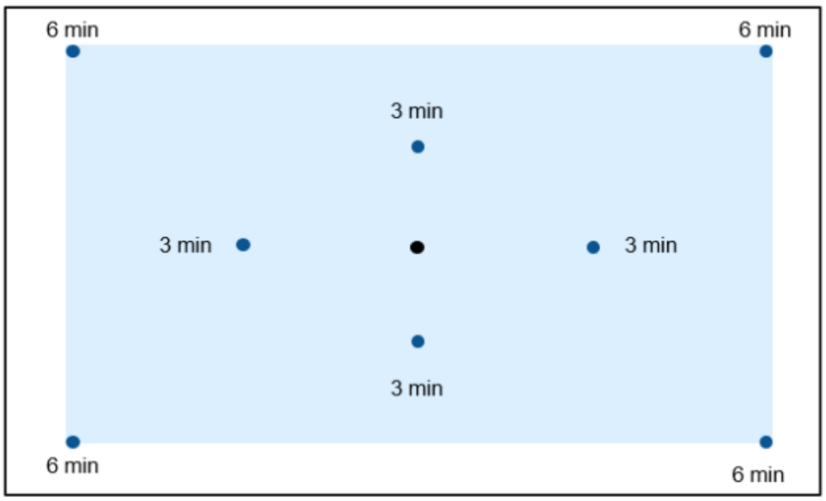
Figure 6: Single-point isochrone with a cumulative opportunity measure [56]



A concave hull is usually a result of a refined convex hull fitting the data more precisely, where the convex hull is a geometry made of the edges of the

most outer points of the area.

Figure 7: Convex hull [56]



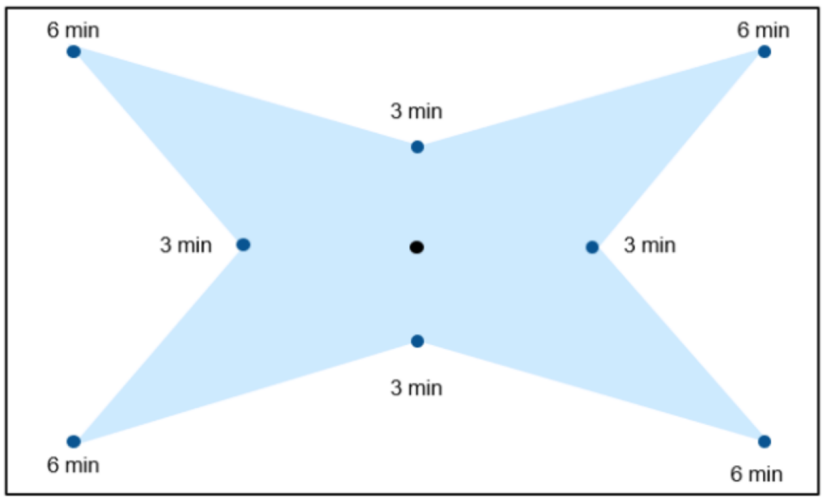
The final shape's (concave hull) precision will be subject to the chosen concavity. An improved precision will result in a higher computation time. There are different parameters that can be used for the calculation, where there will be a compromise between accuracy and computation time. In GOAT's case the main parameters are a concavity of 2 and a length-threshold of 0.006 degrees (EPSG 4326).[52].

Gravity-based accessibility measures GOAT is able to calculate and visualize gravity-based accessibility measures, which are visualized as heatmaps in the shape of hexagonal grids. Based on pre-calculated travel times the heatmap can be adjusted dynamically, based on the selection of the user.

As previously introduced in 2.3, a key factor when calculating the gravity-based measures is the impedance function. GOAT has decided to currently use a modified Gaussian function that was identified as one of the most robust ones for modeling walking accessibility in a conducted study, testing 20 different pedestrian accessibility measures [62]. The exact function is defined below.

$f(t_{ij})$ = modified Gaussian function; $e^{(}t_{ij}^{2}/^{\beta)}$

Figure 8: Concave hull [56]



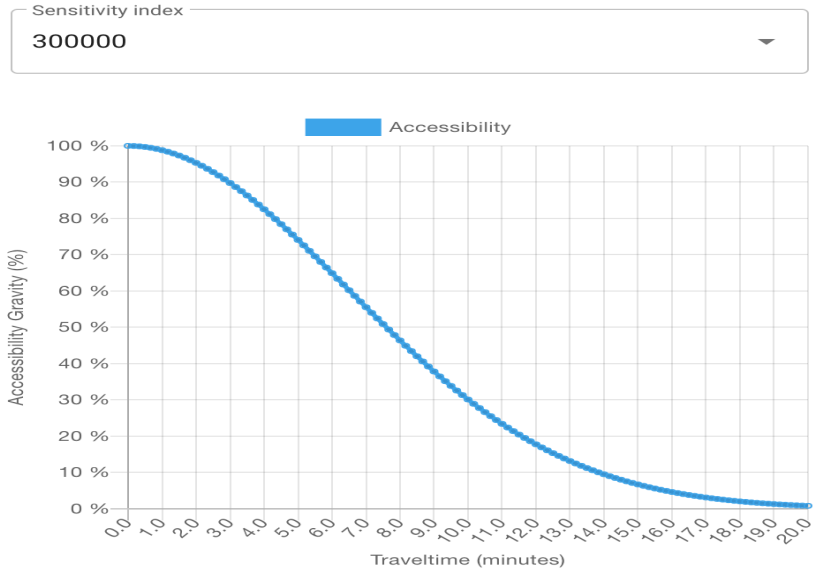
The sensitivity parameter β defines how accessibility changes with increasing travel time, and is consequently decisive when measuring accessibility. For that reason, GOAT allows the user to adjust according to their preference and enables a more personalized accessibility measure. The default β value is 300,000 for all amenity groups. As seen from the modified Gaussian function, the higher the β value, the more in favour of that amenity the user is with a reduced cost over time.

Data and geographical coverage As previously mentioned, OSM accounts as their largest spatial dataset and GOAT's strategy has always been to utilize free and publicly available data wherever possible.

Their routing algorithm from pgRouting has a direct integration with postgreSQL, which allows the database to act as the core of the tool. By integrating the core functionalities within the database, GOAT leverages the high performance of SQL for reading and retrieving structured data, which is essential for a webGIS application that has frequent I/Os.

With the development and advancement over the years, several spatial datasets have been included, either essential for a specific feature in the application or solely for visualization. The other datasets are sourced from statistical agencies, local municipalities or similar. For this reason the application is constrained to city-boundaries where these data implementations

Figure 9: Sensitivity-graph based on impedance function [56]



are available and have been done.

In addition, as the hexagons are based on pre-computed travel times to visualize local accessibility, there will be limitations of the scale of study area and selection of amenities. Both are predefined from the development team. To support a larger geographical area or new graph networks, the classic Dijkstra routing algorithm that is utilized in pgRouting would not be computational efficient enough, as discussed in section 4.4.

2.6.7 Summary of key findings

This chapter has established some of the main theories and concepts that need to be considered when working with accessibility and developing a web GIS-application.

It can be concluded that the customizability and parameter space is wide, ranging from the accessibility definition itself, how to measure accessibility, choice of spatial unit to analyse and different visualization approaches.

Consequently there has been an advancement in amount of accessibility instruments, but with it a growing range of complexity, addressing unique target user groups from novices to field specialists. The chosen implementations that we chose to analyse highlights this case, but have also helped define

what functionalities that seem appropriate to consider in a new application.

The analysis of existing accessibility instruments leads to the below described findings in regards to existing functionalities, design choices and how the notion of accessibility, computation and evaluation of these differs.

For a quick comparison overview, see fig. 10

Figure 10: Accessibility Instruments comparison table

	FEATURES			
	GLOBAL SCOPE	USER-DEFINED AREA	AMENITY-FILTERING	ISOCHRONIC VS. AGGREGATE
WALK SCORE	USA AND CANADA			ISOCHRONIC
15MINUTE BY HERE	USA			ISOCHRONIC
15-MINUTES.CITY	27 SELECTED CITIES			AGGREGATE
15-MINUTE PARIS CSL	66 SELECTED CITIES		✔	AGGREGATE
GOAT	✔		✔	BOTH
TAJC PLATFORM	✔	✔	✔	AGGREGATE

There is identified to exist two overall approaches, the single-source principal (isochronic) eg. an address, and aggregate scores based on larger geographical area (aggregate) eg. a city. Furthermore there seems to be a broad variety of use-cases as well as types of end-users. This is seen to be manifested in terms of how advanced the functionalities the tools offer are, with '15 minute city-map' by Here being very simplistic in contrast to 'GOAT'.

Another interesting finding is the level of transparency and public insight between the different accessibility instruments. '15 minute city-map' by Here and '15-minute.city' offer little or no explanation of how the accessibility metric is calculated, in contrast to 'GOAT' where the user can see how a particular score is computed, but with a higher degree of knowledge required to defer what it means.

The different tools also differ in terms of their coverage - the geographical extent of the application. The discussed accessibility instruments are operating with the following scales: neighbourhood, city, country. There seems to be a gap in terms of coverage with no instruments operating on a global scale currently. The missing global coverage is an interesting finding since it most likely relates to scalability in terms of storage and use of precomputations. The GOAT implementation supports any study area to be added, but must be extended on a case-by-case basis as it requires an implementation effort.

Lastly, it is a common approach to operate with predefined amenity groups which contributes to an easier user experience without a choice overload problem, but reduces the options for a more personalized accessibility-search if desired.

By using OSM data, we are able to develop a tool that covers any arbitrary city, and which can scale up to cover globally, while allowing for a fully personalized metric choice. In addition, the calculated area should be an area defined by each individual user, supporting street-level precision. As the comparison table displays, this will be in contrast to the majority of available implementations, which are rarely precise and flexible in terms of areas with multi-source accessibility measures.

The aim is to create an accessibility tool that has an intermediate complexity level, bringing value to both practitioners but also manageable for general users.

3 Maption - Scope and Requirements

Based on the described gaps of existing implementations in the previous chapter, this chapter will outline the scope and purpose of the instrument being developed. Firstly, we will provide a definition of Maption and the differentiation of the tool in relation to other accessibility instuments. Then, an analytical framework and software qualities from which instrument will be developed is described. Lastly, the requirements of Maption will be presented.

3.1 Purpose and scope

We implement a proof-of-concept, whereby the coverage is limited to all of Denmark, but the scope is global. In other words, it can be extended to any country using the same architecture and codebase. The instrument targets urban planners and regular citizens.

Maption will be built upon accessibility of type 2, which revolves around distances between locations as described in section 2.1. Based on the key findings in section 2.6.7, the purpose of Maption will be to fulfill the following shortcomings of the existing implementations:

- Geographical coverage
- Limited selection of amenities
- Predefined study areas (boundaries)

Maption will accommodate the shortcomings by:

- Creating a scalable architecture
- Providing users the ability to select any combination of amenities
- Allow the users to define specific areas for analysis

3.2 Instrument definition

Maption is a user-friendly accessibility analysis tool in the form of a web-application, which can be used by both novices and professionals. It strives

to provide useful insights on walking-level accessibility by doing network analysis on user requested areas with customizable metrics. The instrument will allow users to draw an area on the map and thereby define any area to be analyzed on street-level.

Maption will provide the option to select any amenity in the database, available within the user defined area. Furthermore the user will have no limitations on how the user can mix amenities to be included in the analysis. To provide greater customization of the metric, users can apply filters to each of the chosen amenities, and thereby only include POIs with certain properties. From the user-based request the instrument will perform network analysis and provide the user with a variety of options on visualizing layers of the results on the map. The user can choose the following:

- See the actual network on the map
- Mark the selected amenities on the map
- See computed distances for all nodes of the network
- See aggregated network analysis

The user can choose any visualization and can freely combine multiple visualizations.

3.3 Analytical Framework and Software Qualities

At a conceptual level *what* the user should be able to do, can be represented by two subsequent *actions*. 1) The user should be able to select an area, and 2) the user should be able to see accessibility metrics within the selected area. The architectural decisions will in the end come down to the options considered in solving these tasks.

Having these in mind, we can here define four key software qualities based on the issues identified in chapter 2, and how they relate to the problem. Together, these build an analytical framework that permeate the evaluation, and subsequent decisions made for which components to include.

precision how precise are the distance measurements when computing accessibility

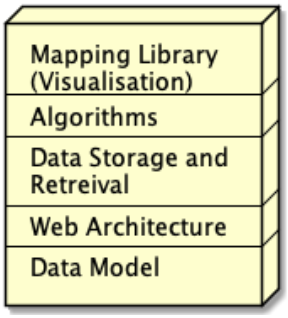
flexibility how adjustable will the system be to user-defined requests and implementation of extended functionality

scalability how can the system be designed to scale well compromising between viable storage size and speed of computation

performance how can the system operate efficiently without compromise on precision and flexibility

The high-level components that need to be evaluated can be represented by building blocks, as seen in Figure 11.

Figure 11: Building blocks



The foundation is the **Data Model** which describes how OSM data is utilized, in section 4.1. In relation to this data model, we assess how the **Web Architecture** can best support this model in section 4.2. This follows with an evaluation in section 4.3 of how to optimize **Data Storage and Retrieval** given the architecture. The possible **Algorithms** and data structures used as the application logic is then described in section 4.4, and finally the **Mapping Library**, as the engine of visualisation, is described in section 4.5. Evaluation of the components have been performed through a mix of experimental phases of developing the system, and technical review of literature and documentation.

3.4 Requirements

The scope and software qualities defined above establish a foundation for defining the functional[1] and non functional requirements [2] of the accessibility instrument.

The following sections 3.4.1 and 3.4.2, describes the functional- and non-functional requirements respectively.

3.4.1 Functional requirements

The functional requirements of the project are represented in the format of user stories to adapt to the agile process from which the instrument is developed. The user stories defined below describe critical functionalities that the system must support to accommodate the mentioned shortcomings.

- As a user I want to define an area such that I can analyze specific geographical areas
- As a user I want multiple visualisations of the data such that I can adapt the visualization to the context
- As a user I want to select the POI's for the metric such that it represents my personal preferences
- As a user I want a filter on amenity attributes such that my preferences can be more accurate and specific
- As a user I want the accessibility score to be shown as distance in meters, such that I can easily understand the meaning of the metric
- As a user I want to be able switch between multiple areas on the map, such that I can compare my defined areas.

[1] **Functional** requirements These are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do. [66]

[2] **Non-functional** requirements These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process, and constraints imposed by standards. Non-functional requirements often apply to the system as a whole rather than individual system features or services. [66]

3.4.2 Non-functional requirements

Precision The system should with the available OSM data represent all walking paths and accurate shortest distance metrics within meter precision.

Flexibility The system should support user queries of any number of amenities and filtering options within user-defined areas of sizes <10,000 km^2.

Scalability The architecture should support global coverage without having to change any core software components, and not exceed the storage requirements for the current OSM database (≈1.5 TB uncompressed XML).

Performance The system should support fetch and creation of a Network in less than 1 minute for an area of 10,000 km^2. The system should support "all-nodes" k-nearest POIs queries for an area of 10,000 km^2 in less than 2 seconds.

4 Maption - Analysis and Architecture

The review of currently available accessibility instruments and data, highlights the shortcomings of currently available tools, and *what* a system addressing these shortcomings might consist of. In this section, a deeper analysis of *how* these missing features can be addressed by analysing the components and alternatives viable to make up such a system.

4.1 Data Model

To model a city as a graph, as described in section 2.4, requires source data for two essential components - intersections as vertices V, and street segments as edges E. OSM does not provide data in this format "out of the box", so understanding the OSM data layout is important to describe and assess how it can be manipulated to fit a graph-based model. Access to the OSM data occurs via the Overpass API - described as a 'database over the web'[49], with its own query language and structure, which also becomes an important part of the evaluation in terms of scaling the system.

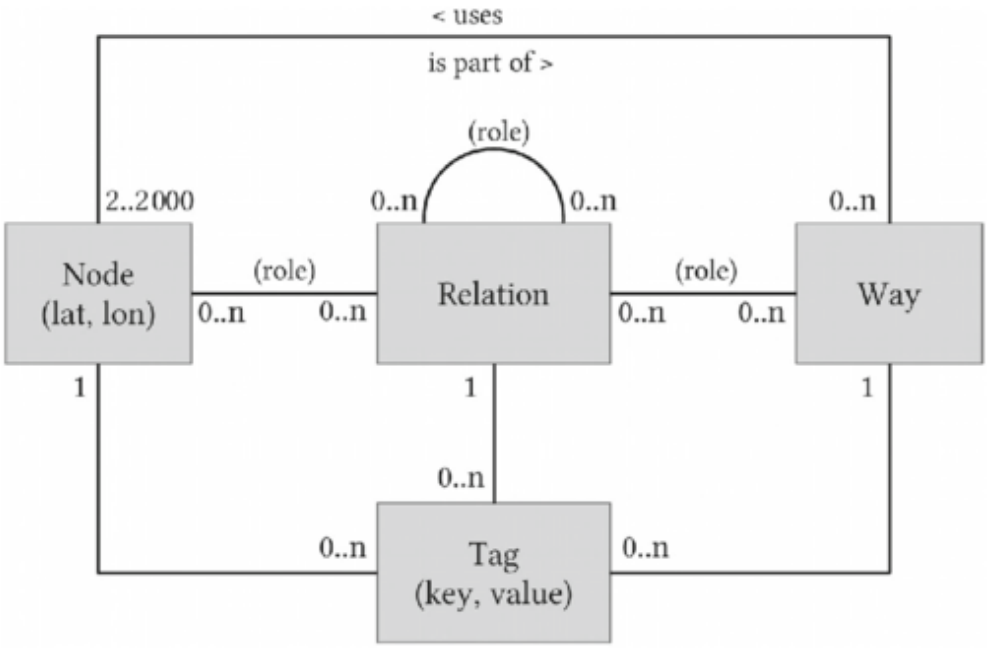
4.1.1 Overpass API

The Overpass API has an extensive interface which makes it possible for a user to send queries to the API and get any data as described by the OSM data model, as response. The Overpass API is read-only and is optimised for this purpose [6]. One of the downsides of fetching data continuously from the Overpass API versus from a self-managed database, is the limitation in regards to optimization. Adding additional indexes is not possible, and hence not an option for optimization. Furthermore query optimization and analysis, as it is possible with PostgreSQL's built-in commands eg. EXPLAIN ANALYZE, is not possible using the Overpass API.

4.1.2 OSM data model

Once the data is fetched with the Overpass API, it is composed of four basic elements, as seen in Figure 12 and described below.

Figure 12: OSM data model [20]



Node *Nodes* are core elements in the OSM data model, having the following properties: latitude, longitude and node id. Additionally elevation, often referred to as the Z coordinate, can be defined as well. *Nodes* are commonly used for representing point features, with a Tag that describes the point feature eg. amenity name. Furthermore, *Nodes* can be used for defining the shape of a *Way* element. As of September 2020, The OSM database contains over 7 billion nodes [46].

Way *Ways* are like *Nodes*, fundamental elements in the OSM data model. *Ways* are often used for representing, for example, roads and rivers. Furthermore, *Ways* can be defined as an ordered list of *Nodes*, since *Nodes* makes it possible to shape a line eg. make it curve. *Ways* can be quite complex and can consist of 2 - 2000 *Nodes* [48].

Tag *Tags* are important properties of the previous described elements (*Node*, *Way*, *Relation*). *Tags* are used for describing the properties of these features, and are represented as key-value pairs.

OSM has agreed on conventions that determine the meaning and the usage of *Tags*. The conventions are not always enforced and it is not uncommon to encounter different interpretations of *Tags* and usage of these.

Relation In previous research and implementation of an accessibility instrument conducted by the thesis group [70], *Relations* was an important and central concept. A *Relation* is defined by OSM as *"a group of members which is an ordered list of one or more nodes, ways and/or relations. It is used to define logical or geographic relationships between these different objects (for example a lake and its island, or several roads for a bus route)"*[47].

Relations have previously been used in combination with the Nominatim API [3], where a relation ID is returned based on a relation name eg. 'Copenhagen municipality'. The ID can then be used to query the Overpass API to obtain POIs and network data (nodes and ways), and simplify queries compared to querying by a polygon e.g. latitude, longitude coordinate sets. Although this approach was very convenient, it also had limitations, since not all locations are defined as relations.

4.1.3 Creating a graph data model with OSM data

At first glance, the OSM *Nodes* and *Ways* seem to have a close correspondence to vertices and edges of a graph, except that *Ways* are not connections between two nodes. A *Way* can be made up of a number of *Nodes* to indicate curvature, but these *Nodes* are not necessarily *intersections* in a practical sense. However, there are a few different ways to modify this data model to represent a graph.

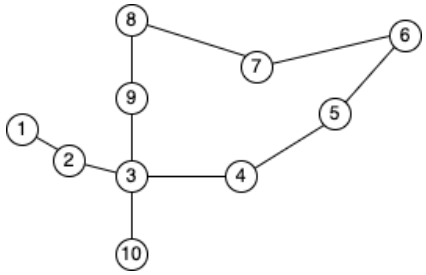
To see an example of this, let us define three OSM *Ways*, A, B and C, each with a set of nodes $A = \{1, 2, 3, 4, 5, 6\}$, $B = \{8, 9, 3, 10\}$, $C = \{8, 7, 6\}$. Using the OSM model, one straightforward approach to model the graph is depicted in figure 13. Every OSM *Node* is represented as a vertex in the graph, and each *Way* is split into the connecting segments, composing the set of edges. Since walking can generally occur in both directions, the graph is a symmetric directed graph, but can for simplicity also be modelled as an

[3]Nominatim uses OpenStreetMap data to find locations on Earth by name and address (geocoding). It can also do the reverse, find an address for any location on the planet. [45]

undirected graph, such that the graph G would be represented by,

$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$
$$E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}, \{6, 7\}, \{7, 8\}, \{8, 9\}, \{9, 3\}, \{3, 10\}\}$$

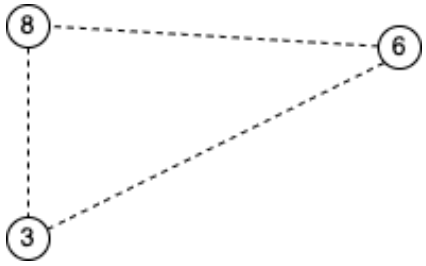
Figure 13: Example graph representation of OSM ways



This would define the size of G by $|V| = 10$ and $|E| = 10$. However, an *intersection* - defined as a place where two, or more, roads meet - would in reality paint a different picture.

A different way to represent the intersection is of course strictly mathematically for each pair of ways. For instance, $A \cap B = \{3\}$, $A \cap C = \{6\}$, and $B \cap C = \{8\}$. With this approach, one could eliminate all nodes that are part of less than two OSM ways, and construct edges between the remaining nodes, as seen in figure 14.

Figure 14: Example graph representation of OSM ways

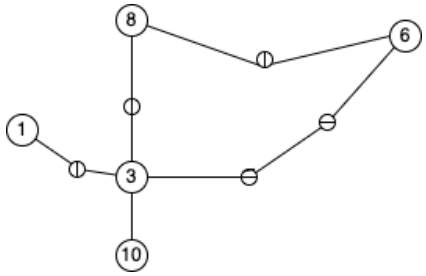


The major advantage with this method is of course a much smaller graph, of only $|V| = 3$ and $|E| = 3$ (resulting in faster search algorithms). However, this would come at the expense of accuracy when modelling the network because it would remove dead-ends (node 1 and 10 are examples of these). Naturally, a POI could be located by these dead-ends, but would then have

to be associated with the next nearest intersection, leading to less precision. Another point to note is of course that each of the distance segments would need to be stored in the new edge (e.g the new distance of edge $\{8, 3\}$ would have to be computed as $l(3, 9) + l(9, 8)$.

A more accurate approach, which still allows for simplifying the graph compared to the first option, would be to define intersections more loosely to also include dead-end streets. This method would thereby include vertex 1 and 10, and the edges $\{1, 3\}$ and $\{3, 10\}$. This is a compromise, with simplification for better performance than the first model, and better accuracy than the second model.

Figure 15: Example graph representation of OSM ways



As seen in figure 14, the edges are drawn with dashed lines, representing an abstract edge, with an associated distance. Removing the nodes between two intersections means that the curvature that these extra nodes represent could be lost, unless the geometric shape of these points in space were preserved. Another improvement in terms of model accuracy would therefore be to preserve the geometric shape of an edge, without storing the OSM nodes (e.g. node 4 and 5) as part of the underlying graph representation.

This improvement was seen as a last step to model the source data, as shown in Figure 15. This leaves a graph data structure with $|V| = 5$ and $|E| = 5$, and concluded as the best trade-off of simplicity and accuracy.

We found no external tools that process OSM data for an architecture supporting fully geo-compatible graph model in this way. The implementation of this algorithm will be covered in section 6.5.2. Given this processing phase of data retrieved from the Overpass API, the data model could be supported by different architectures, either by calling the Overpass API directly in the application or preprocessing and storing the data in a database, or

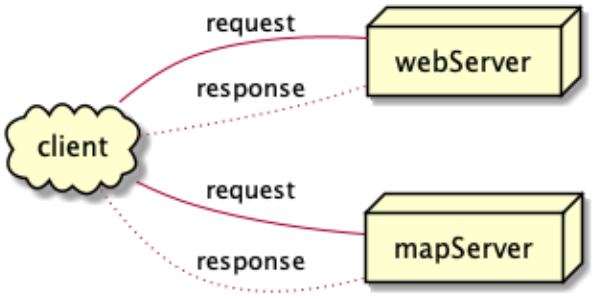
some form of hybrid model. The pros and cons of these architectural choices will be discussed in the next section.

4.2 Technical Architecture

To evaluate different components of the system, an overall system architecture must be discussed. As described in Background chapter 2, there is a lack of an interactive web instrument, with a wider geographical coverage. Web application architecture often follow the client-server architecture, a conceptual model featuring two actors, the client and the server, which communicate over an established protocol. The client-server model is suitable for many-to-one scenarios, where multiple clients want to access information or services that can be centralized at a single site [12].

In a web application, the client is often comprised of the web browser, and the server component is a web server capable of handling the requests from multiple clients. In addition to the classic web architecture, WebGIS architectures typically consist of an additional map server that handle mapping services and geospatial data [5]. The client-server model becomes operable through its core operations: the server *listens* for a client *request*. Upon receiving a request the server sends a *response* for which the client receives, or *accepts* [12].

Figure 16: WebGIS Client-Server Architecture



The client-server model incorporate three conceptual layers; the **presentation layer**, **the application logic**, and the **data storage**. In practice the classic client-server model is often architecturally translated to a 2-tier architecture, or extended to a 3-tier architecture, or N-tier architecture. The differentiation of these architectures are crucial in understanding how the

intended solution might service the user as described through the *actions* above.

In a 2-tier architecture, the system is partitioned in two tiers, where the first is the client handling the **presentation layer**, and the second is concentrated in a server capable of handling both client *requests*, accessing **data storage**, applying **application logic** and serving a *response*. In a 3-tier or N-tier architecture, the three layers are separated in 3 tiers (or more tiers if the **application logic** and **data storage** needs to be divided further). Applications that rely on external services, are therefore examples of N-tiered architectures [12].

The architectural decisions that need to be considered for developing a prototype can therefore be summarized using the following support questions:

- How should the Maption architecture reflect dependencies on external services, especially on source data (from OSM)?
- How should the Maption architecture separate presentation, application logic, and data storage?
- Which options of software components can be utilized to best support the architectural decisions, and guiding software qualities?

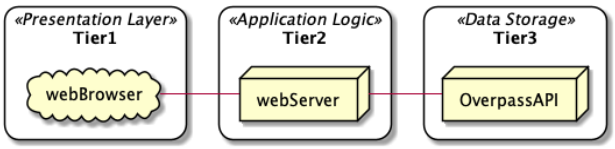
4.2.1 Creating a scalable architecture

The data ecosystem of OSM allows for wide coverage, but also means the architecture needs to take into account how this OSM-data dependency should be handled. During the initial phases of experimenting with the Overpass API, three different varieties of implementations were considered, each of which would have different impact on the system architecture.

No DBMS - complete dependence on OSM API The first option was to have a system where the **data storage** layer would be entirely provided by OSM. Given the above architectural description, this would entail a 3 (or N-tiered) architecture, with the first two tiers implemented with a client communicating with a Web Server responsible for the application logic, with the remaining tier(s) being represented by the OSM service as seen in Figure 17. The Web Server would effectively be both a server - responding to

requests from the browser - *and* a client, requesting resources from the Overpass API as the server in that relationship. Therefore it places a dependency on processing of request-response to OSM.

Figure 17: 3/N-tier architecture based on OSM as a connected external service



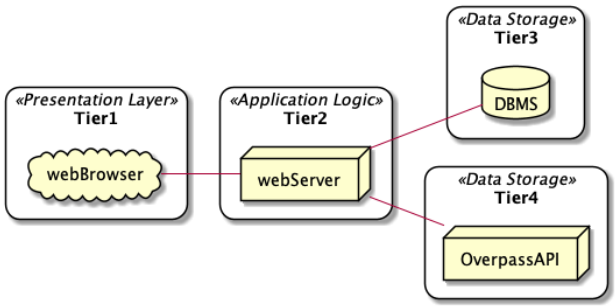
In theory, this architecture has some benefits in terms of the software qualities defined. Firstly, since the data storage layer has been completely "outsourced", there would be no storage requirements posed on the Maption instrument being developed. In terms of scalability, the only concern would therefore be how processing of incoming requests would be handled. Lastly, data maintainability would be completely handled by the OSM ecosystem.

In practice, some critical bottlenecks were quickly identified with this architecture. Because OSM impose rate limits on frequent, large data requests, fetching data directly from the Overpass API would limit performance [6]. In addition, due to the data layout that is used within OSM, as described in section 4.1, there is a significant overhead of processing OSM data into the desired graph model.

Hybrid Solution - Own DBMS and external dependency to OSM
The second option considered was a hybrid option, whereby the system would have an own **data storage** layer that would grow "organically" upon user request. This would entail that the **application logic** should handle request data and distinguish between data within areas that has been pre-processed, and data that falls outside of already pre-processed areas. In the case of data outside of pre-processed areas, it would be fetched externally from OSM. Figure 18 represents this N-tiered architecture where the **data storage** layer would be partitioned between the own DBMS and the OSM external service.

The hybrid solution offers some improvements in regards to the major bottleneck identified in the first option. When data has been pre-processed once, it would for other users requesting that same data, be available for

Figure 18: N-tier architecture with both OSM as external service and own DBMS



efficient retrieval and pre-processing. It would maintain a global coverage, because effectively any data would be available, and storage requirements would follow a supply-demand logic by only storing data that is actually requested by users, allowing storage requirements to *scale* with the number of users. In terms of performance, areas that had not been completely processed, would still require longer loading time, albeit only once, but that would still strain the user experience.

In addition, if a user request involves data that are only partially stored in the database, the hybrid architecture must accommodate fetching only the data not present in the database from OSM. This functionality requires a more complex set of operations to identify the non-existing data.

Furthermore, since the database would be incremented little by little, it would mean the source data would be from different time periods. Consistently maintaining updated data would be cumbersome, since the OSM changes continuously.

Own DBMS - no direct dependency to OSM Due to the limitations of the previous two options, the analysis of architecture concluded with the choice of using our own DBMS for the **data storage** layer when used in connection with the web application. The source data from OSM is pre-processed separately - meaning a request cycle where the client would interact with the web application would not have a direct dependency on the Overpass API.

This solution was evaluated as the best compromise on scalability and performance for several reasons.

Firstly, it would mean that user requests would always be fast relative to the previous two architectures - because graph initialization would be performed on pre-processed data.

Secondly, it would allow complete control of the query logic, because the system would not be constrained by the OSM data model.

Thirdly, it would be easier to scale, for instance country by country, and maintaining the most recent data would not require as complex logic as the hybrid solution.

The drawback would be that, to have global coverage, the system would need to pre-process data for the entire world, which would be both time-consuming and storage-consuming. To keep somewhat recent data, it would need to be processed at regular intervals. However, that processing step would be detached from the request cycles of the web application, which was identified as the primary bottleneck to be avoided. It is estimated that a dump of the entire OSM database is approximately 1580 GB in uncompressed XML format [46], and would increase the storage requirements on the system, but not to unmanageable levels.

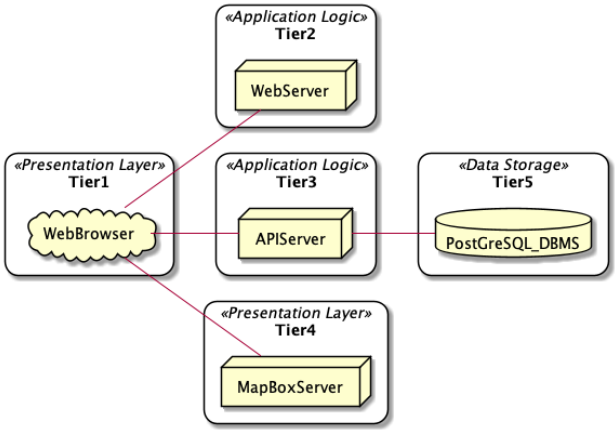
Final architecture summary Our final architecture is presented in figure 19. The chosen architecture can be thought of as a multi-tier architecture, here depicted in 5 tiers. An important difference to note is also the separation of the web server and the API server. The client-side is built as a single-page application (SPA), and therefore creates a clear separation of the client and API Server. The web server becomes solely responsible for serving the static files to the user, in this case, making it highly portable.

In essence, the design of the API is standalone in the sense that a different client application could connect to it to retrieve data. Furthermore, the modularity is enforced by not allowing the client to have direct access to the data storage layer, so that the API server can sanitize client requests. The Mapbox server is also depicted in the diagram to show the external dependency to the mapping service, which is an essential part of the application.

4.3 Data storage and retrieval

Following the decisions made in the previous section which included running our own DBMS, removed necessary interactions with the Overpass API

Figure 19: N-tier architecture with DBMS



in the request cycle. The following sections will analyse our approach of implementing a DBMS to the application.

4.3.1 A flexible DBMS for geospatial data

When developing a GIS (Geographic Information Systems) web-based application, choice of database technology and design become important due to storage requirements of geographically referenced data. A full-scale GIS application must be able to store large amounts of data and make them available on demand. Furthermore, it is expected to be able to integrate data for different fields of study and from multiple sources [23]. The performance of the chosen database therefore has a direct impact upon the performance of the web application itself[3].

GIS applications involve several domains of knowledge. Examples could be urban planning, route optimization, demography, agriculture and epidemics control. Each type of application brings different data, features, and spatio-temporal properties. Different types of applications will lead to distinct types of requirements regarding database, data models, storage and indexing schemes. As a result there is no general-purpose database for GIS that will encompass all the possible considerations [41].

In our case, the implementation is built on one main data-source (OSM) and contains conventional data such as alphanumerical attributes (tags) and geometries.

Some of the main considerations were how the DBMS would index and store the spatial data, but also the available database access methods and spatial operations. This was an important aspect to support our functional requirements, since the application would have to locate all POIs, nodes and edges within a user-defined area.

The characteristics of our data and the necessary functionalities, led us to explore the use-case of PostgreSQL and its PostGIS extension.

PostgreSQL and PostGIS is an Enterprise RDBMS which has been around for more than 25 years as a multi-vendor open source community project. It is highly extensible by design, runs on both small and large platforms and available on any cloud platform [59]. PostgreSQL is the main RDBMS used by OSM.org and their Main API. It is also used in several of the implementations that were identified and analysed during the research work. PostgreSQL comes with the PostGIS extension that adds GIS capabilities to the RDBMS. PostGIS runs alongside PostgreSQL and enables the user to run spatial operations (spatial SQL), store geometries and build spatial indexes. It is a completely free offering that is considered as one of the leading GIS implementations as of today.

A comparison study of free and open source software for geographic information systems indicated that the PostGIS extension was normally the preferred solution, where 15 out of 19 web apps had utilized the combination of PostgreSQL and PostGIS [69]. This can be the result of the claim that PostGIS provides the most extensive implementation of the OGC-SFS standard [67], which is a set of standards that specify a common storage and access model of geographic features. The hosting of the PostgreSQL database will be described in the next section.

Given PostgreSQL and PostGIS broad use in the industry, combined with our initial exploration of the tools, resulted in the choice of these for our implementation. How the database was designed and implemented will be discussed in section 6.5.1.

To have a database continuously available during the development phases, we decided to use a free version of Google Cloud's database offering, as setting up a proper infrastructure can be costly, and not a feasible solution for small scale projects. The cloud database provider option can also offer certain

options in regards to scalability. The specifications can quickly be scaled up or down, if this should be deemed necessary due to change in storage requirements and potential user-traffic in a production setup.

The database server has the following specifications depicted in fig. 20.

Figure 20: DB server specifications

Region	europe-west3 (Frankfurt)
DB Version	PostgreSQL 14.2
vCPUs	1 vCPU
Memory	3.75 GB
Storage	100 GB
Network throughput (MB/s)	250 of 2,000
Disk throughput (MB/s)	Read: 48.0 of 240.0
	Write: 48.0 of 72.0
IOPS	Read: 3,000 of 15,000
	Write: 3,000 of 4,500
Connections	Public IP
Backup	Automated
Availability	Single zone
Point-in-time recovery	Enabled

4.3.2 Efficient data retrieval with spatial indexing

Indexing is a core concept in database optimization and is often a crucial aspect of query optimization. Indexing allows the DBMS to search for data using indexing as pointers to where the data are located on the disk or in memory.

Spatial data sets are traditionally quite large, where the data is often distributed in an irregular way [58]. In a scenario where Maption would have global coverage, the dataset would grow extensively, and the need for implementation of spatial indexing in the database becomes more present.

One of the primary functionalities Maption should support, is the ability for the user to define their own area of interest. In regards to this functionality, the database must be able to efficiently retrieve all relevant data *within*

that area.

Traditional database systems' data structures such as B+-trees were deemed sufficient enough for queries on alphanumeric data types (something that contains letters and numbers). However, the 2- or 3-dimensional spatial data that needs to be retrieved in typical GIS applications, requires other types of data structures for efficient indexing. Indexing of the spatial entities (nodes, edges and POIs), is therefore crucial for a performant application.

PostgreSQL offers three suitable index options (index frameworks) for spatial data, the GIST, SP-GIST and BRIN indexes. The following three paragraphs describe these indexes, that have been used for performance testing of the application (see figure 39).

Generalized tree search (GIST) GIST is a balanced search tree and a generic form of indexing for multi-dimensional data, such as geographic coordinates. [58] In PostGIS it uses an R-tree data structure, where bounding boxes (bboxs) are defined based on nearby spatial objects that are grouped. The bounding boxes are saved to the index pages, instead of saving the geometries.

Since the bbox contains the objects, a query can avoid scanning through all the objects within the defined bbox, by searching in the subtrees of bbox's. The R-tree is built such that the division is coarse and become more granular as the search moves down the tree, with the leaf nodes being pointers to the real location on the disk.

As with all mentioned spatial indexes, GIST is a lossy index method, which means that a secondary check is required to ensure a record matches a particular search condition [58].

Space-Partitioning-Generalized tree search (SP-GIST) Like the GIST, the SP-GIST access method is also a general scheme, but specifically tuned for "spatial partitioning". The root node represents the entire entity, while the partitioning recursively divides the index nodes into two completely distinct domains. Contrary to the GIST index, the resulting tree is a non-balanced tree, and is best suited for data with a natural clustering element and particularly non-overlapping geometries. GIST is less sensitive to the spatial distribution and topology of the geometries, and as a general rule will

outperform SP-GIST if the data has a lot of overlap [58].

Block Range Index (BRIN) The BRIN index is a more simple index, where the list of ranges maps the search space values to database pages. A block range is a group of pages adjacent to each other, where summary information (min value, max value, page number) about all the pages is stored in index. This leads to a very small BRIN index, even for very large tables.

Due to the nature of the blocks, the effectiveness of the index is directly dependent on the data's underlying order. As such, BRIN indexes are suited towards large tables, where the data input is arranged or sorted sequentially to match the query structure.

The advantages of BRIN compared to the other indexes are primarily

- Building a BRIN index is less CPU-intensive and can be 10 times faster than a GIST index over the same data. [58] Due to the speed, it can always be interesting to run a test to see if the BRIN-index is suitable, if the data table is large.
- As it only stores one bbox for each range of table blocks, the disk space spent can be up to a thousand times less than a GIST index [58]

Given the above, each index comes with different properties and use cases. When choosing the best-performing index, the most important indicators will be the spatial distribution (consistently spaced vs. blobs of geometries) and the geometries topology (many overlaps vs isolated distribution).

4.4 Computing distances for "all-nodes" k-nearest POIs

It is now established how a graph data model can be conceptualized, and how data for the model can be stored and retrieved. The next important architectural concern is how the distance data for the "all-nodes" k-nearest POIs should be computed.

One option would be to to pre-compute a full distance table with the k-nearest POIs from every amenity-type and for every node in the database. The other option is to compute the k-nearest POIs upon request from the user. This section will introduce the advantages and disadvantages of these

respective options, and motivate the choice of 'on-demand' computation in light of the requirements.

Furthermore, it will describe the problem in context of classic algorithms to solve shortest distance queries on a graph data structure, and how using contraction hierarchies enables efficient computing of this accessibility metric on-demand.

4.4.1 Precomputed vs. on-demand distance queries

In the previously conducted research project [70], a query for distance to the 5 nearest POIs to each node, for 7 different amenity-types were pre-computed. This was a feasible solution, due to the following: a limited geographical coverage, which only included 20 cities in Europe, with predefined boundaries. Furthermore the pre-selected amenities, effectively limited the number of computations and storage requirements.

By precomputing the distance queries, there are clear benefits in terms of efficiency. When an area is selected, retrieving distance metrics only requires a table "look-up"; no algorithms need to be applied at run-time as these are completely detached from the request cycle. Pre-computed distances can therefore enable a seamless user experience for certain use cases. An example is the instrument made by "15 minutes.city" in section 2.6.4, which allows one selection of a city and show a single "view" of accessibility within a city. With a more extended version, such as ones from ParisCSL or GOAT 2.6.5, the user can also select among a few selected amenity-types, and see the relevant accessibility metrics. However, this methodology has two limitations in terms of scalability and flexibility.

Firstly, pre-computation can be a viable solution only if areas are predefined. This is because the boundaries of the area defines the street network upon which to compute the nearest distances. So unless areas are pre-defined, or pre-computation would be performed on the entire world, the nearest distance queries would *have* to be computed on-demand, since the street network (graph data structure) is a prerequisite for this query step.

Secondly, pre-computation imposes some limitations on the options the user has. It may be feasible to have pre-computed distances for a limited number of areas, and a limited number of amenity-types, but these precomputed values need to be stored, which quickly grows to an unsustainable

level, both in terms of scale and maintainability.

As an example, the rather small country of Denmark has $\approx$ 1.2 million nodes in our database (after pre-processing) and $\approx$ 200 distinct amenities. To pre-compute the 5 nearest POIs, for each node and for each of the 200 amenity-types would require storing 1000 distance entries for each node. Therefore approximately 1.2 billion data points would have to be stored, only for Denmark (in addition to being pre-computed at regular intervals to keep up-to-date).

A system relying on computation on-demand would be able to account for these limitations. In regards to scalability, scaling would only mean to include more source-data such as POIs, nodes and edges - no shortest-distance data would need to be stored, drastically reducing storage requirements. Since computation would happen upon request, any query parameters could be defined at the user's discretion. This means the user could run a query for any set of amenities, and the system would be more flexible to implement new features (e.g. user-uploaded data). Furthermore, maintaining up-to-date data would be much easier. To account for both the flexibility and scalability NFRs, it is therefore necessary to compute the "all-nodes" k-nearest POIs on-demand. The greatest concern of this choice is naturally whether it is possible to compute it efficiently, for reasonably large areas. The next couple of sections will outline how the performance requirement can therefore be accommodated.

4.4.2 "Batched" shortest distance graph problems

Web-based map services have led to a great increase of routing techniques for point-to-point queries, finding the shortest path between a source node s and a target node t [11]. However, in the field of accessibility analysis, more relevant analyses are "batched" queries, that indicate some notion of *reachability*. These graph problems can broadly be generalized by the following query types [11],

One-to-all find the shortest distance from one source node s to all other nodes in a graph G

One-to-many find the shortest distance from source s to all nodes t in a target set of nodes T.

Many-to-many find the shortest distance from each source node $s \in S$ to all target nodes $t \in T$, in essence computing a distance table of $S \times T$.

4.4.3 The classic Dijkstra shortest distance algorithm

The classic algorithm to find the shortest distance from a source node s and a target node t, denoted $dist(s, t)$, is using Dijkstra's algorithm [22]. Dijkstra's algorithm solves the problem by keeping a distance table $d[v]$ tracking the distance to each vertex from s found so far. It initializes $d[s] = 0$ and all other vertices to positive infinity. Thereafter, it maintains a priority queue of unsettled vertices with their distances as keys. Vertices are scanned in increasing order from s, by continuously deleting (settling) the next minimum vertex u from the priority queue, and then use edge relaxation to update distance values of each adjacent vertex v from u. If $d[u] + l(u, v) < d[v]$ the value of $d[v]$ is updated, and v's order in the priority queue is adjusted, if necessary [65]. This preserves the invariant that $d[v] = dist(s, v)$ when v is removed from the priority queue (i.e. no other shorter path can exist from s to v because it is the currently shortest path to any unsettled vertex). The invariant also means that a search can be aborted when deleting t from the priority queue if solving a point-to-point query, or continue to settle vertices in a one-to-many for all target vertices $t \in T$ or until the queue is empty in a one-to-all problem [14].

Solving the k-nearest POIs problem from a single source vertex is thus a special case of the one-to-many problem, whereby the search can be aborted when the k nearest from the set of target vertices T has been settled from the source s [15]. Solving the "all-nodes" version of k-nearest POIs, that we are concerned with, can be thought of as a special case of the many-to-many problem, where the set of source vertices $S = V$ in graph G. In theory, it could therefore be solved either by running $|V|$ single-source k-nearest POIs searches as described above. Alternatively, it could be solved by running $|T|$ "backwards" one-to-all searches from each $t \in T$, meanwhile for each $s \in S$ maintaining a sorted k-sized list of distances to the k nearest $t \in T$.

4.4.4 Limitations and techniques to improve Dijkstra's

Although the "all-nodes" k-nearest POIs problem can be solved using the classic Dijkstra algorithm, for many practical applications where large road networks are used, Dijkstra's algorithm does not scale well for a range of problems [22, 11, 14]. Route-planning optimizations have therefore received a great deal of attention and given rise to a variety of speed-up techniques.

One classic way to speed up point-to-point queries using Dijkstra is to run a bidirectional Dikjstra, which simultaneously runs a forward search from the source, and a backward search (reversing the edges) from the target [16]. Once some node has been settled from both directions, the shortest distance can be derived using the information collected during search. This idea is important because it underpins many of the techniques used in other speed-up techniques.

In the case of static routing (i.e. when edge weights don't change), many techniques have been developed to perform some pre-computations to exploit "hierarchical" structures in the graph to limit search space during queries. Some of these include Small Separators, Multi-Level Techniques, Advanced Reach-Based Routing, Highway Hierarchies (HH), Highway Node-Routing (HNR), Transit-Node Routing (TNR), and Contraction Hierarchies (CH) [16]. Dynamic routing, (i.e. when edge weights are changing) is an active field [16], and also utilize some of these techniques, but will not be considered here as we are limiting the tool to use walking distance - therefore non-changing edge weights.

These techniques have not only been used to develop fast point-to-point queries, but also highly performant batched queries [17, 35], which can also be used to solve the "all-nodes" k-nearest POIs problem.

Based on a flexible system whereby it's possible for the user both to define the area themselves, and to define any queries to be performed on the street-network, it is important therefore to consider how both a pre-processing phase and a query phase can exist as part of the request cycles.

4.4.5 Speed-up with contraction hierarchies

Contraction hierarchies (CH) [25] builds on the formerly mentioned HH and HNR, and many of the techniques to solve problems using these optimiza-

tions, can likewise be applied to CH. Algorithms utilizing CH have become ubiquitous in many routing based applications [16]. It leverages the property that some nodes would be more frequently visited in the network, and therefore includes a preprocessing phase where nodes are *contracted* using heuristics to define some order of "importance". This effectively creates a hierarchical ordering of nodes, every node having their own "level" in the hierarchy [25].

In the *contraction* phase, a vertex u is contracted, removing incoming edges to u, and outgoing edges from u temporarily. If a shortest path from an adjacent node v to an adjacent node w passes through u, a short-cut edge will be added in order to preserve the shortest distances. When all nodes have been contracted, they are placed back in the original graph, together with the original edges and the shortcut edges, creating an overlay graph G^* [25].

The implicit topological ordering of nodes in essence creates two directed acyclic graphs, an "upwards" graph with only edges leading from nodes lower in the hierarchy to nodes higher in the hierarchy, and the opposite "downwards" graph.

In a point-to-point query, when searching from s to t, the idea is to perform a modified bidirectional Dijkstra but with one important difference; the search considers much smaller subgraphs of G^*, because only edges leading outwards to "more important" nodes need to be considered.

In essence, two complete Dijkstra searches, "forward" from s and "backwards" from t are first performed. For all nodes u that are found in both searches, denoted here the set L, $dist(s,t)$ is found by computing the minimum $\{dist(s,u) + dist(u,t)\}$ for all $u \in L$. Since both the forward and backward search considers (relaxes) *only* edges that move increasingly "upwards" in the forward search, and "downwards" in the backward search, the search space is drastically reduced [22, 14]. The slight trade-off of the preprocessing phase can give a significantly faster query phase.

4.4.6 Contraction hierarchies to solve k-nearest neighbor

Since the "all-nodes" k-nearest POIs problem is a variant of the many-to-many problem, one way to solve it would be by running $|S| \times |T|$ distance queries using the above technique for point-to-point search. However, as

discussed by Knopp et al [35], this naive approach does not necessarily lead to significant speed-up [35].

A different approach in the many-to-many scenario is therefore to perform forward searches for each $s \in S$ and backward searches for each $t \in T$ only once, rather than pairwise. This crucial idea is achieved by using *buckets* that holds pairs $(t, d(u))$ associated with each node u, encountered during the backward searches from each $t \in T$ [35]. Then, in the forward search from each $s \in S$, the buckets of each encountered vertex u is scanned [63] and a distance table of $|S| \times |T|$ is updated with $dist(s, u) + dist(u, t)$. After all nodes u are considered, the correct distance is found for s [63]. As documented in several studies, [63, 35, 25] this technique is orders of magnitudes faster than the previously mentioned Dijkstra solutions.

Adapting this solution to the k-nearest POIs solution, formalized by Geisberger [26], only requires maintaining the k closest connections found - not the entire distance table as described above, and can be implemented using a max-oriented priority queue of size k where the max is deleted once a *closer* POI is found [26]. Once the k furthest POI is closer to s than any remaining vertex in the forward search space of s, the search can also be aborted, which means additional speed-up compared to the more general case [22].

If areas are completely predefined, the contraction phase could occur "offline", and only the fast query phase be part of the user request cycle. If areas are user-defined, contraction hierarchies have to be created on a request basis. This is because the graph, in other words, the nodes and edges would be different for each request. It means the contraction phase would also have to occur during the request cycle at some point. But considering the number of queries that can be performed, and the speed-up reported for these techniques, creating contraction hierarchies was considered a necessary trade-off, to meet both the functional and non-functional requirements. The experiments in section 7.2.5 evaluates the performance of this approach, and its inherent limitations will be discussed throughout the paper.

4.5 Visualizing the results

This chapter has covered how Maption can store data using a graph model, and compute the accessibility metrics using the "all-nodes" k-nearest POIs approach.

However, we have not described how to allow the user to define a set of points constituting the area for analysis in a format that can be translated to the backend, and how to visualize the results from the distance queries.

A common format to use in WebGIS applications is GeoJSON [30]. Since PostGIS, and most mapping tools support using the GeoJSON format, it therefore provides a serialization standard to use for data transfers between all layers in the architecture. The GeoJSON format support the following geometries: Point, LineString, Polygon, MultiPoint, MultiLineString and MultiPolygon. The ones primarily used in Maption is

- Point
- LineString
- Polygon

where the Point type is used to represent both nodes of the network and POIs. The LineString is used to represent edges, and the Polygon represents the area defined by the user.

To draw areas for analysis and visualise data, we analyzed some of the mapping tools available. We found several options that are well documented and meets the basic requirements of what is expected from mapping tools.

As it was difficult to find specific limitations of the mapping tools in the documentation in regards to our use case, several libraries were experimented with. Finally, Mapbox GL was chosen for its ease-of-use and detailed documentation.

Mapbox GL JS uses vector tiles which store information as Points, Lines and Polygons laid out in a vector tile set which are rendered client-side and allows for dynamic data modifications. It is fully compatible with GeoJSON data, which can be displayed in different layers on the map interface. A variety of extensions have been built for the Mapbox API, such as the ability to draw onto the map canvas [7]. Therefore it could support the functional requirement of defining an area on the map, and the resulting Polygon type could be sent in requests to the API server. In addition, the results from the 'all nodes' k-nearest POIs queries, could be converted to GeoJSON and transfered as a response to the client, and visualized with Mapbox layers' functionality supporting the GeoJSON format .

The following chapter 5, "User guide", will present the application and its core functionalities in a step-by-step format.

5 User Guide

This chapter serves to guide the user and visualize the functionalities of the application. The purpose of the user guide is assisting users to do specific tasks in the application. The implementation of the functionalities will be covered in section 6

5.1 System description

The application allows users to do accessibility analysis in self-defined areas within Denmark. It offers analysis on any amenity available in the OSM database. Visualization of the analysis results can be adjusted to the user's preferences, whether the user wishes to see aggregated scores, visualization of the POI's, visualization of the network etc. The available user interactions can be categorized into four groups; interacting with the map (drawing, searching, zooming, panning, enter full-screen), creating and navigating areas (creating a new street network, selecting between previous areas), configure and run an accessibility query (select amenities, filter by tags, run query), and interact with layer panel (toggle visibility, change paint properties, select calculated distance properties, and recalculate scores).

5.2 Tour

When accessing the web-application, the option to take a tour of the application is presented.

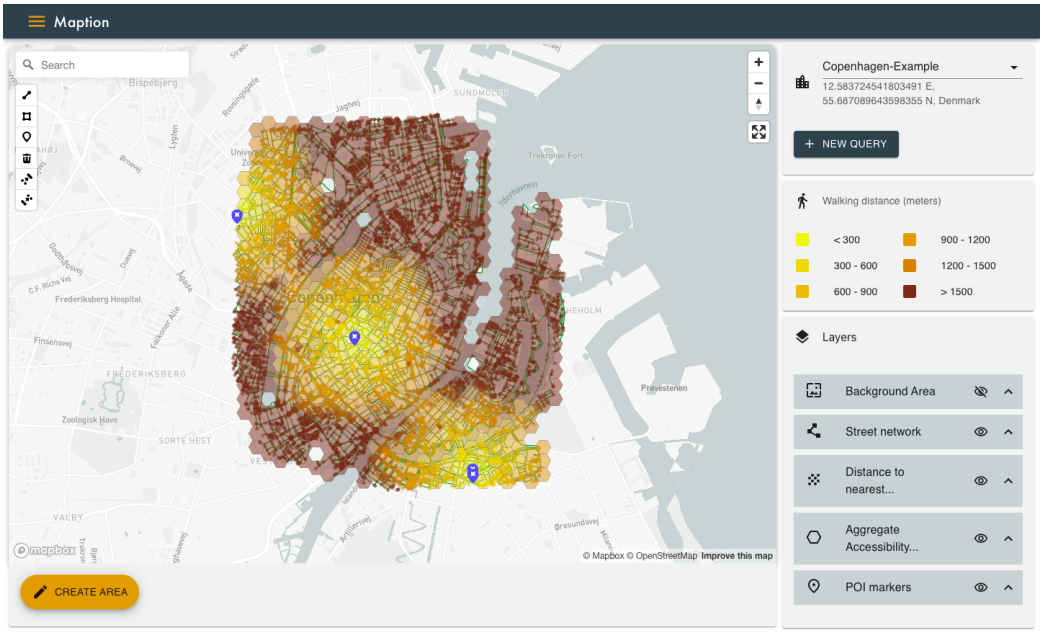
5.2.1 Take a Tour

Clicking the 'Take a Tour' button starts the tour which presents three essential steps consisting of 'drawing an area', 'selecting an area' and 'adding layers'. When the tour is finished, the application will be available for use.

5.2.2 Skip tour

For skipping the tour, press the 'No' button to directly access the application.

Figure 21: Maption interface



5.3 Map

The map serves as the presentation layer of the the data analysis. Here the results of the queries will be visualised in the requested format.

5.3.1 Navigate the map

There are three possible ways to navigate the map in its current state. First option is to use the mouse to drag the map in the desired direction. Second option is to use the arrow-keys on the keyboard to move the map in the desired direction. Third option is to use the search-bar in the top-left corner of the map. In the search-bar it is possible to enter the name of the area one wants to analyse, and the map will then 'fly' to the location of that area.

5.3.2 Zoom on the map

It is possible to zoom in on the map to get a closer view of a specific area. Likewise, it is possible to view an area further away by zooming out. To use the zoom function, click the zoom buttons in the top-right corner of the map.

5.3.3 Full-screen map

To enter the full-screen mode of the map, press the full-screen button on the top-right side of the map. To exit full-screen mode press the button again, or press exit on the keyboard.

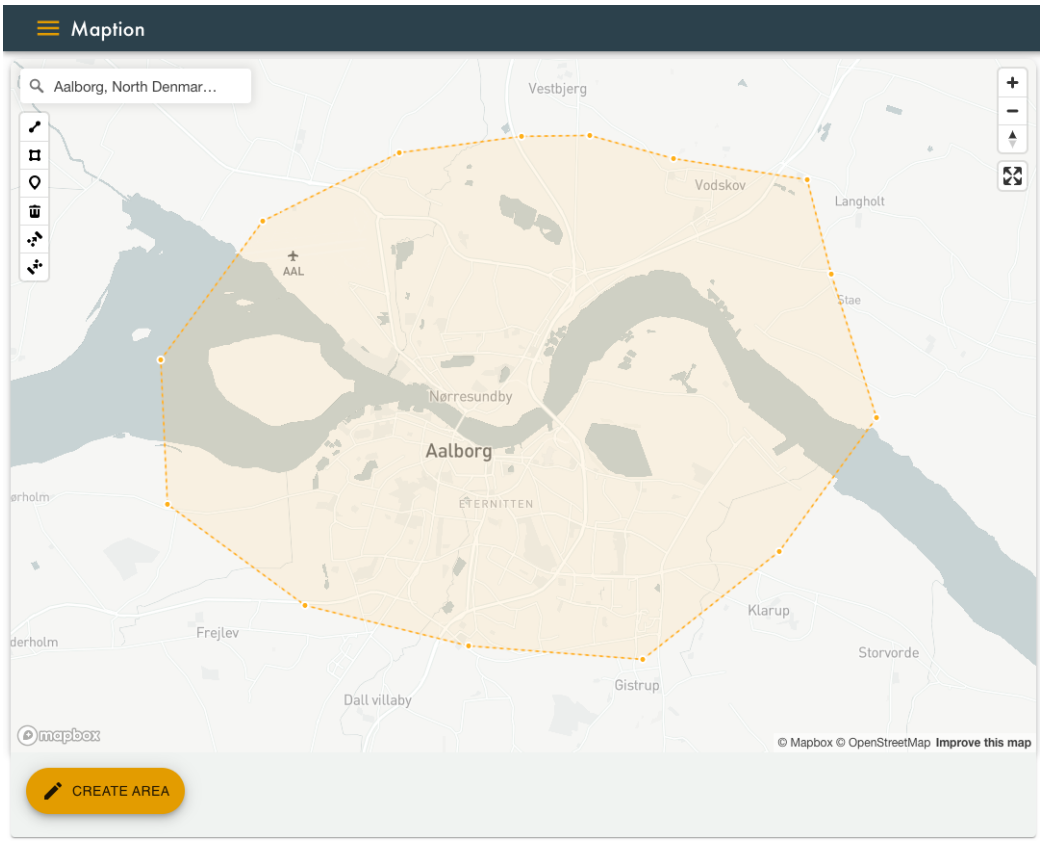
5.4 Create area

To create a desired area for analysis use the following steps

5.4.1 Step 1: Draw the area

Select the draw tool on the top-left side of the map, and mark the boundaries of the area.

Figure 22: Draw the area



5.4.2 Step 2: Register the area

When the drawing is done, select the 'Create Area' button on the bottom-left side of the page. The system then checks if the data within the covered area is available. If the data is available, name the area for later use and save the area. If not, a prompt appears suggesting the user to redefine the area.

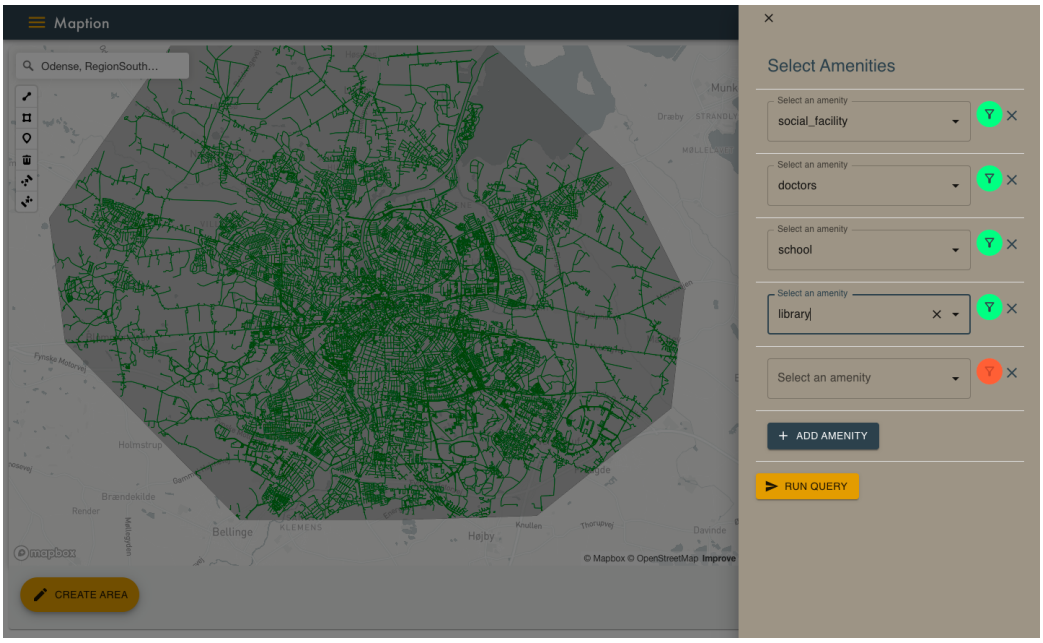
5.4.3 Select other areas

On the top-right side of the page, it is possible to browse through previously defined areas. Click the drop-down menu displaying the name of the current area, to choose a different area.

5.5 Create a query

To create a query and request analysis of amenities, click the 'New Query' button.

Figure 23: Select amenities



5.5.1 Select Amenity

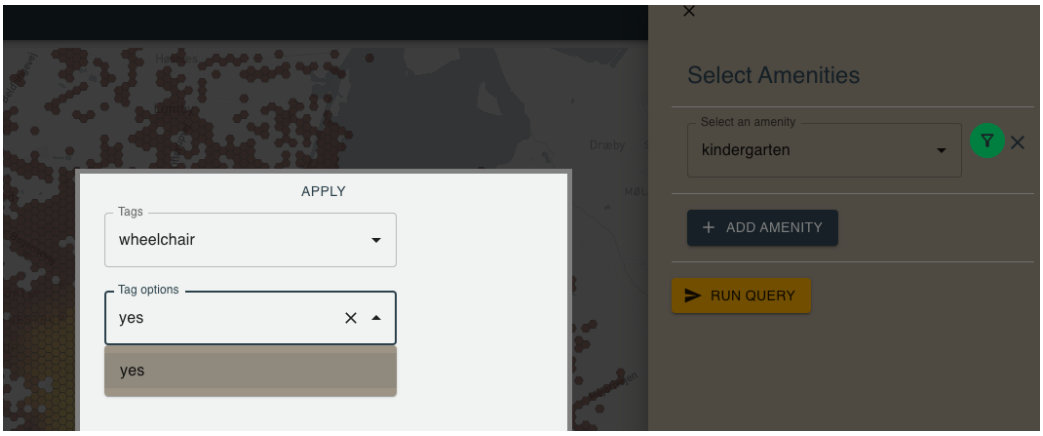
Choose the desired amenity to analyse in the drop-down section 'Amenities'. The menu only displays amenities available ind the requested area.

It is possible to analyse multiple amenities by adding more menus on the 'Add amenity' button.

5.5.2 Amenity filtering

When an amenity is selected, the filter icon will change color to green, signaling that filtering is available. Clicking the icon prompts a window for filtering.

Figure 24: Amenity filtering



Tags The first drop-down menu displays the tags available of the chosen amenity in the requested area. To choose filtering on a tag, select the tag.

Options The second drop-down shows the options available for filtering on the chosen tag. To choose an option, select the option.

Apply filters When tag and option are selected, the filter can be applied to the query by clicking the 'Apply' button.

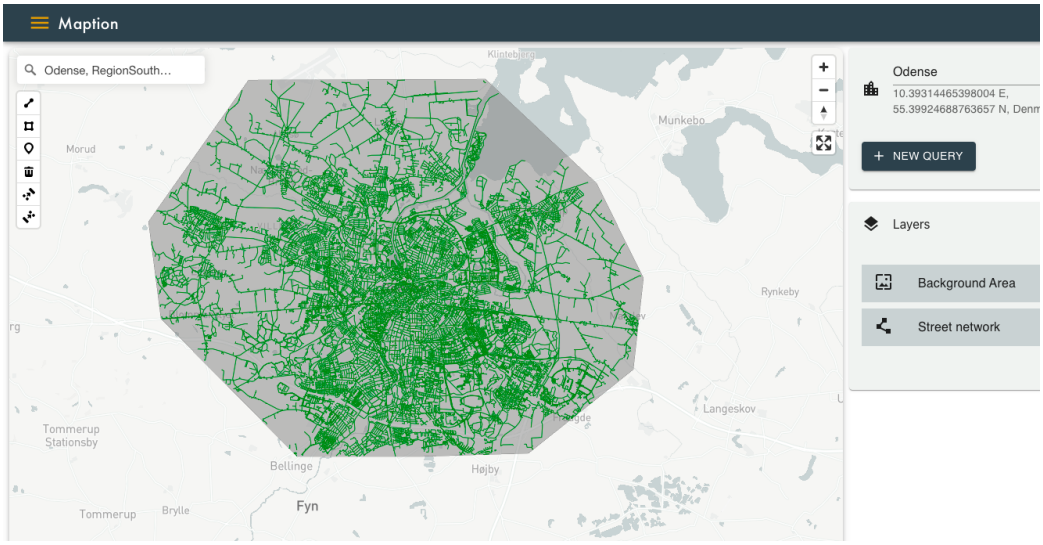
5.5.3 Run the query

When the desired amenities and filters have been applied, the query can be initialized by clicking the 'Run Query' button.

5.6 Layers

In the layers sections it is possible to configure the visualization of the analysis in the area. When no query has been made, the only two layers shown are the Background area and the Street network.

Figure 25: Street network



5.6.1 Background area

This layer shows the boundaries of the selected area and can be toggled on and off.

5.6.2 Street network

This layer shows the street network in the chosen area. Additional layers are available when a query has been added to the analysis.

5.6.3 Distance to nearest

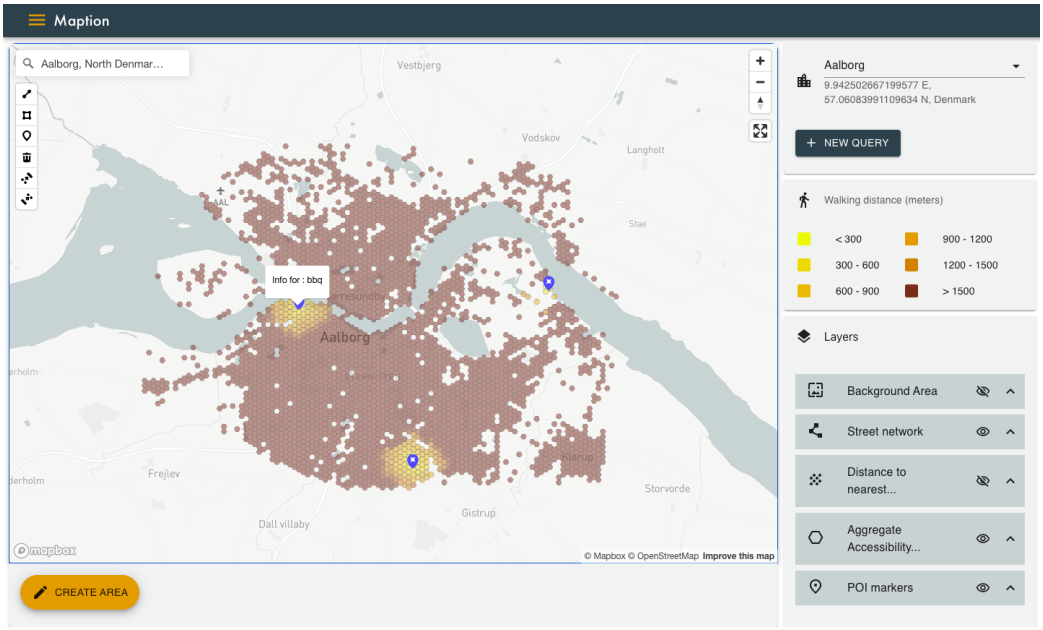
This layer represents the distance to the nearest amenity. The data is visualized by color-code on each of the intersections in the area. It is possible to select how many of the the nearest amenity the layer shall visualise. The size of the intersections can be modified, as well as the amenity it shall visualise the distance to. Only amenities from the query are available.

Figure 26: Distance to nearest

5.6.4 Aggregated accessibility

This layer divides the area into a grid and displays the aggregated score in each of the grid sections.

Figure 27: Aggregate layer and POIs



5.6.5 POI Markers

This layer visualises the location of the selected amenities in the query.

6 Maption Design and Implementation

In the chapter 4. we concluded on a foundation for our architecture. This section will cover the design and implementation of the different parts of the application. Section 6.1 describes the software libraries and frameworks used. Section 6.2 lists the components implemented in the frontend, and section 6.3 describes a typical user interaction flow. In section 6.4 the request-response interactions triggered between different layers of the stack is outlined. Finally, section 6.5 describes the design and implementation details of the back-end, covering the database and the API.

6.1 Software libraries and frameworks

In addition to using PostgreSQL for the database, the back-end of the application is written in Python, and the front-end in JavaScript. This section describes some of the central third-party libraries and frameworks which have been used in the implementation of Maption. We provide a brief introduction to the functionalities they provide in the context of the system.

GeoPandas The properties of GeoPandas function as a connection between backend operations in Python and the geo-spatial data stores in the Postgres DBMS extending PostGIS. GeoPandas provides a functionality that allows handling of data in GeoJSON, as it is stored in the database. GeoDataFrames simplifies transferring data between the backend and the database, as it has methods that insert data in the format of a GeoDataFrame directly into the PostgreSQL database [19].

Pandana Pandana is a network analysis library which utilises contraction hierarchies to perform shortest path searches quickly. The decision to use the Pandana library for network search, was both due to previous experience with the library, its simple API with an interface to the pandas library, yet powerful parallelized computation using C++ as the engine. The efficiency of the library can enable the system to perform searches within a reasonable time upon user request and provides contraction hierarchies and the k-nearest algorithm in one library. [54].

Flask Flask is a framework for web developing. It offers easy and quick implementation of web servers and expose endpoints. There is an active community developing open source libraries built on top of Flask, adding a lot of useful functionalities, middleware and automated documentation which is useful for the current project [75]

React React is a JavaScript library used for development of the front-end. React uses the Node package manager (npm) which offers easy integration of broad variety of open source libraries [27].

Redux Redux is third party library, used for state handling. Due to the nature of React where properties need to be propagated through the components, using Redux can ease the task of sharing properties between components. Redux exposes states to the components that are wrapped in its context, such that they can be accessed globally [60]

Turf.js Turf is a spatial analysis library written in JavaScript and provides functions for analysis using GeoJSON, making it compatible with the data formatting of the system [73]. Turf is used to calculate accessibility scores in the visualisation of the hexagon layer of the system, by creating a grid of the marked geographical area and calculates scores for the grids individually. Some of the more heavy computation done on the client-side are performed when adjusting this layer. Selecting and deselecting amenities to be included in the metric, require recomputing the score for the data source. In addition, if the user wishes to resize the hexagons, a new spatial join of points within a hexagon and recalculating the aggregates are performed. It was decided to use Turf.js to perform these spatial operations, and do the calculation on the client-side directly, to minimize latency on the visual display of the layers.

6.2 User interface

The application and UI is built as a single page application (SPA). With a SPA, the presentation logic is rendered on the client-side. When loading the application, all necessary code is retrieved by the browser with a single page load. When interacting with the SPA, requests to the Maption API will retrieve necessary data from the server and the application can render the

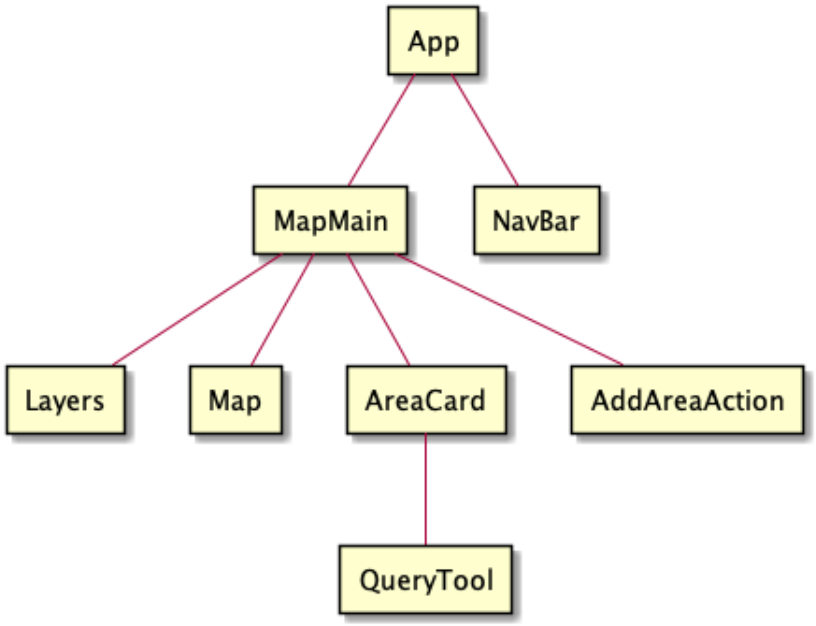
associated parts of the page, instead of loading a new page. This functionality enables for the biggest advantage of using a SPA, which is a swift loading speed and creates the feeling of a native app.

This is especially helpful when the application aims to display a rich user interface with many features.

6.2.1 Components

The user interface can be decomposed into a component tree which displays the relationship and dependencies of the components on the front-end. For simplicity, this section will only cover the main components that support the core functionalities of the application.

Figure 28: Main React.JS component dependencies



App App Component is the parent component in React which functions as container for all other components.

MapMain MapMain Component is a parent component for all map-related components within the application and covers the whole window except the upper navigation bar.

NavBar NavBar Component functions as the menu-bar and can be toggled by the user. A slide-in panel will appear when toggled.

Map Map Component is an imported component from the react-map-gl library. It displays an interactive and customizable vector map with minimal code needed. Within the component the user can search for any location, zoom in and out, toggle full-screen mode and draw polygons to define an area.

Layers Layers Component allows the user to toggle which data and layers should be visible after defining an area. The user can also define the amount of a specific amenity that should be considered in the metric calculation (1-5) and hexagon size.

AreaCard AreaCard Component displays the current area (coordinates and name) the user is analysing. It also allows the user to select previously defined areas and will automatically transport the user to the corresponding location on the map.

QueryTool QueryTool Component displays all available amenities and allows the user to define which amenities should be considered in the accessibility calculation. The user can also filter further based on any available tag-option, such as name, opening-hours, WIFI, wheel-chair accessibility etc.

AddAreaAction AddAreaAction Component acts as an action-button after an area has been drawn. When clicked, it triggers the request flow to generate a network. The application will check if the area is covered in the database. If successful, the user will be asked to name and save the area. If the area is not available to be analyzed, the user will receive a prompt indicating that the area is not covered.

6.3 User interaction flow

The following section shows how the flows and interactions described in the user guide (section 5) are designed. The functionalities also show the interaction between components in the client-server architecture.

Figure 29: Main user interaction flowchart

The activity diagram shows the main user flow from accessing the web application from the web browser.

Step 1 is where the user starts by interacting with the Map component to define an area using the draw tool. The draw tool is an extension built on the Mapbox API, and when used sets the user in different modes (`draw`, `select`, `deselect`). Once finished drawing the boundaries of the area of interest, the user can exit the `draw` mode and enter a `select` mode.

Step 2 In `select` mode a new area can be created by clicking the AddAreaAction component. Clicking the button triggers a request to generate the network from street data within the defined boundary. If the area does not contain data, the user will be prompted to draw a different area. This can happen if the area is outside of Denmark, only covering water, or fields where there are no streets/walking paths. When the request is complete, the new area is presented to the user in the AreaCard component, and map layers with associated tools in the Layers component are created for the background and streets.

Step 3 The QueryTool gets mounted for the new area, and the user can query the network for accessibility metrics. After clicking on "new query" the QueryTool appears in the sidebar, including all available POIs. The user must select at least one amenity to run a query. When finished configuring the query, the run button can be clicked which sends parallel requests for 1) the data associated with amenities selected, and 2) the results of running the accessibility queries. With a successful response, new map layers are created and displayed on the map. These layers are based on the source data received, and associated tools are mounted and presented to the user in the Layers component.

Step 4 The user can now interact with the Layers component. A variety of options are presented to the user. For instance toggling the eye icon (visibility of layer), adjust size of points, and select amenity to base the coloring on. These interactive elements changes the layout- and paint-properties of the underlying Mapbox layers or source data, and are subsequently re-rendered on the map.

6.4 The request cycles

Fig. 30 depicts the core request cycles in the system. The client-side of the architecture is represented by the `Web Browser`. The client primarily embodies the *presentation layer*, but also contain a small part of the *application logic*, running smaller calculations and manipulation of source data in the browser.

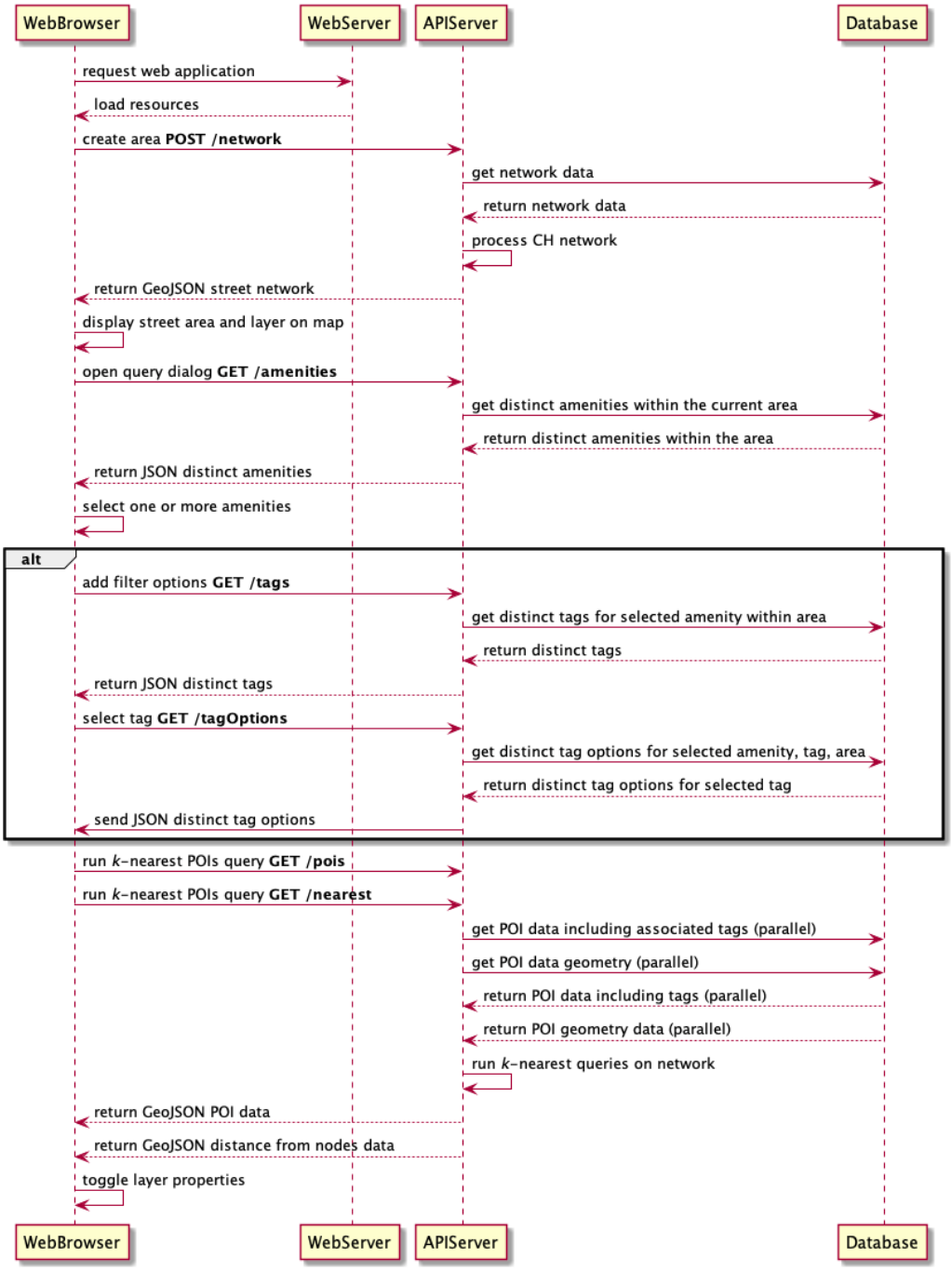
The client is built as a single page application. This means that all web resources (HTML, CSS, Javascript, images) are served by the `Web Server` upon request *once*, and all further interaction with the application happens between the browser and the `API Server` directly. Interacting with the map tools happens on the client without any interaction with the `API Server`. The map itself, and associated extension tools (e.g. GeoCoder) have interactions with Mapbox servers to load tiles for the map and for searching geocoded locations. These interactions are external dependencies and for simplicity, not depicted here. The server-side is composed of the `Web Server`, the `API Server` and the `Database Server`. The following paragraphs describe the interactions for the different request cycles triggered upon user interaction.

Creating a new area The interaction with the server-side starts when an area drawn by the user is requested. An HTTP POST request is sent from the `Web Browser` to the `API Server` for constructing the network. Upon receiving the request, the `API Server` sends a query to the `Database Server` for the node and edge data used for processing the street network. After receiving the node and edge data the `API Server` runs the pre-processing of the network (generates the contraction hierarchies (CH)) and stores the street network in memory.

Then the `API Server` sends the response back to the `Web Browser`. Depending on success or failure, the `Web Browser` displays the new layers or an error message accordingly.

Configure a k-nearest query The next sequence of interactions occur when the user opens the query tool. A request to get the distinct amenities within the currently selected area is sent from the `Web Browser`. The `API Server` queries the `Database` for this information and returns to the `Browser`.

Figure 30: Sequence diagram



The user can thereafter select among the distinct amenities to configure the query parameters. Similarly, if a user selects an amenity and opens the filter

tool, a request for tags related to the selected amenity is triggered. When a tag is selected, the associated tag options are retrieved in a similar fashion.

A different implementation we tried, fetched all amenities within a user defined area, with all its tags and tags options when the query tool was opened. The idea behind this implementation was that all user options would be available at once in the query tool, such that no loading time in regards to tags and tags options would be required after selecting a given amenity. This approach was feasible when the selected area was relatively small, but deemed infeasible with relatively larger areas such as Copenhagen, due to an extensive loading time.

Therefore, it was split up in three different steps. The **alt** section of the diagram indicates that it is optional if the user wants to have filters on a particular amenity.

Run the query Once the query parameters has been set and submitted by the 'Run query' button, a set of parallel HTTP GET requests are sent from the `Web Browser` to the `API Server`. The k-nearest query request is handled by first retrieving the POIs specified by the user from the `Database`. Then, using the pre-computed network, the multiple k-nearest POIs queries on each of the amenity categories specified is computed. This means that the "heavy" lifting of constructing the street network (generating the contraction hierarchies) has already been performed, and the queries run fast on a pre-computed network. The design creates, in essence, a cached layer on the `API Server`. That allow the so-called query phase in this part of the sequence, to be run multiple times within the same area, without having to reconstruct the street network. Once the query is executed on the `API Server`, the response is sent to the `Web Browser`, with data for all nodes in the network, and the associated nearest POI measurements, in GeoJSON format.

Show the layers Upon receiving the response, the `Web Browser` displays the necessary front-end elements and renders the layers on the map. From there on, all interaction with the layers directly occur on the client-side. These include showing the difference for the kth nearest amenity, toggling visibility of layers and setting other layout properties.

6.5 Backend Design and Implementation

The following section will describe Maption's database design, the related data model and API.

6.5.1 Database design

The data model for Maption's database consist of three relations, Edges, Nodes and POIs, as depicted in the ER-diagram, fig. 31 below. Furthermore, the implications of the designed data model, in terms of the functionality it offers will be briefly described.

Edges The Edges relation consist of four columns : *from*, *to*, *geom* and *osm_wayID*.

'from' and *'to'* are IDs referring to a NodeID in the *'Nodes'* relation. This means that an edge will be related to exactly 2 nodes. The foreign key constraint also means that the *'to'* and *'from'* ID cannot refer to a node that is not stored in the *'Nodes'* relation. The geometry of an edge is stored in the *'geom'* column as a LineString. This column is of type geometry which makes it possible to efficiently retrieve these with use of spatial queries. The osm_wayid is a unique OSM ID retained from the original OSM data model.

Nodes The *'Nodes'* relation consist of only a NodeID (unique ID) and the *'geom'* column storing point geometries. This column allows similar spatial querying as for the edge geometry.

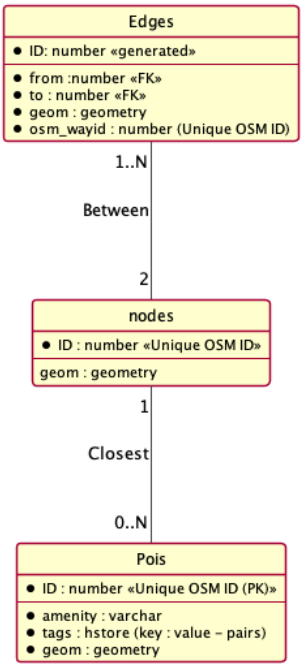
A Node is part of 1.N edges, which effectively means that isolated nodes will be discarded, since they are not considered being part of the street network.

POIs Consist of three columns : *'Amenity'*, a label describing the type of POI, eg. *'restaurant'*. The *'Geom'* column which stores the point geometry and *'Tags'* of type hstore, which is a key:value store for tags and their associated options. Having the hstore *'Tags'* column makes it feasible storing all tags in the *'POIs'* relation within one column. One alternative would be to store one tag per column, which potentially would create a lot of NULL values.

In the ER diagram, there is a relation between the *'POI'* and the *'nodes'*. This is because each POI must be associated with its nearest node, when running the k-nearest POI search. However, this relation is actually not implemented with key constraints or a separate table because the association is made at runtime of the k-nearest search.

Lastly, a node in OSM can be both a Node and a POI. In the project's database, this same entity can not be present in both relations, both will be represented as unique entities.

Figure 31: ER-diagram



Spatial queries The database design makes it possible to perform the needed queries without having to perform any joins, which can be costly. Fig. 32 depicts a query with no joins.

Figure 32: Example of spatial query with no joins

```
SELECT e.from, e.to, e.osm_wayid, e.geom,
↪ st_length(e.geom::geography) as distance
↪ FROM edge_new e WHERE st_within(e.geom,
↪ st_geomFromGeoJSON('{"coordinates":
↪ [[[12.544572615041034, 55.70040386853063],
↪ [12.550237440480515, 55.70204832311907],
↪ [12.555215620411758, 55.69711475177277],
↪ [12.544572615041034, 55.70040386853063]]], "type":
↪ "Polygon"}')
```

6.5.2 Pre-processing of OSM data

To process the data to the database, such that it complies with the data model, an initial pre-processing step was implemented.

The below paragraphs describe the preprocessing algorithm, which is also presented as pseudo code in fig. 33

Fetch raw data from OSM The pre-processing step initially declares the area to be pre-processed which in the case of this project is Denmark, but could be any relation in the OSM database. The method fetches all OSM 'nodes' and 'ways' in the declared area which is the subject for pre-processing. When all nodes and ways are fetched, they are split into two arrays and can then be pre-processed separately. From the nodes array, a point geometry is created based on the initially fetched X- and Y-coordinates.

Identifying nodes to delete The first important pre-processing logic is to identify the nodes to be discarded as described in 12. These are the nodes that are not part of more than one way, and are not the start- or end-node of a single *way*. However, the coordinates of these nodes need to be preserved to keep the curvature of the edges. The algorithm achieves this by saving the coordinate set of the node as part of the LineString geometry for the edges.

When the geometry of edges is preserved, the nodes have no purpose and can therefore be deleted to minimise the network and more importantly the time it takes to create the network.

Splitting ways into edges The second important logic is to identify the way-segments that can be regarded as edges, in other words splitting an OSM way into corresponding edges. The algorithm iterates through all the *ways*. For each *way*, it iterates through all the nodes and identifies 'from' nodes, and 'to' nodes that should be stored in the database. These are either start- or end-nodes or intersection nodes. It then creates edge objects connecting these 'from' and 'to' nodes, storing the geometry of each edge as a LineString.

Inserting into the database From the array of edges, a GeoDataFrame is created of all the processed edges, and finally the GeoDataFrame with nodes, filters out all the nodes that should be discarded. Finally it inserts the nodes and edges into its corresponding tables in the database.

Fetching POIs The second part of the pre-processing is to fetch and handle the POIs. Since POIs are represented as nodes in OSM, the initial step is to fetch all nodes in OSM with an 'amenity' tag. All POIs have tags assigned to them which can be used for filtering. The tags can be attributes such as 'opening hours', 'name', 'wheelchair access' etc. These tags have to be formatted correctly so they can be utilized in the database when the API fetches data.

The initial format of the tags is JSON, and is slightly adjusted to fit the hstore format, which allows storing the tags in a single column. The formatting is done by doing a series of string operations and basic regular expressions. When the tags have been formatted, the POIs are added to the database along with the pre-processed nodes and edges.

6.5.3 The Maption API

The pre-proccessed data which has been inserted into the database, will be retrieved by the Maption API.

The API exposes a set of endpoints, each being responsible for a certain operation, related to fetching data needed for the accessibility map visuali-

Figure 33: Processing algorithm

```
SET arrayOfEdges
SET arrayOfNodesToRemove
SET arrayOfWays
SET arrayOfGeometries


FOREACH way in arrayOfWays:

    SET first node as the beginning of way
    ADD geometry of node to arrayOfGeometries
    SET edge
    ADD beginning node to edge

    FOREACH node in way

        ADD geometry of node to arrayOfGeometries

        IF node is last OR node is part of other ways
            ADD node as ending node to the edge
            SET geometry from arrayOfGeometries
            ADD geometry to the edge
            ADD ID of the way to the edge
            ADD edge to arrayOfEdges
            RESET edge
            ADD node to edge as the first node of next way
            EMPTY arrayOfGeometries
            ADD node geometry to arrayOfGeometries

        ELSE
            ADD node to arrayOfNodesToRemove
```

sation. The class design of the backend API is simplistic in the sense that it only has a few classes that communicate upon a request.

The endpoints of the API are listed below:

- POST:network

- GET:pois
- GET:tagValues
- GET:tags
- GET:amenities
- GET:nearest

At each of the endpoints, the Controller class acts as the guardian. It parses the input parameters of the request and depending on request endpoint, delegates the task to the respective classes, and handles any errors that may propagate.

The class diagram below shows a static representation of the internal structure of the API.

Below is a short description of each class and its primary responsibility.

`WebAPIServer` responsible for listening to incoming requests at designated endpoints, and serve the response to the requesting entity.

`DatabaseEngine` responsible for providing the interface between the application logic and the database/storage layer

`DatabaseConnection` handles connectivity between application server and database server, transporting requests and response between layers.

`Controller` responsible for parsing incoming request data, passing the queries to the correct entity for processing, parsing response data and handling errors.

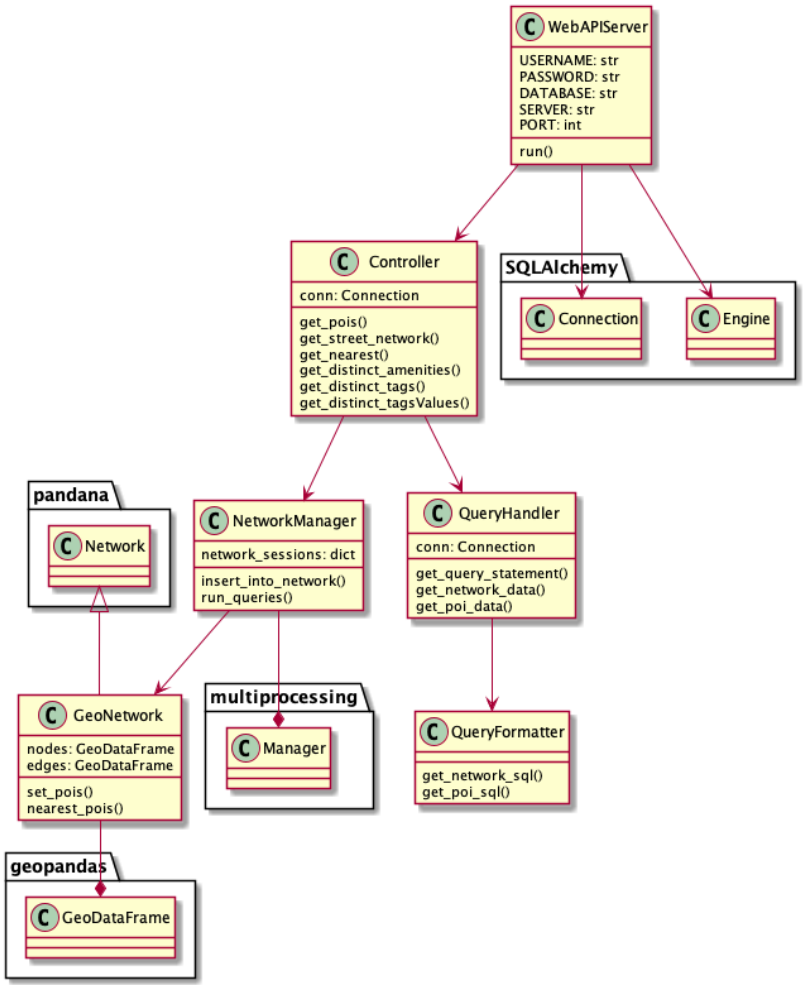
`QueryHandler` handler for all the database queries that may be performed, parsing query data, and retrieving data from the database layer.

`QueryFormatter` constructs SQL statements based on query parameters

`GeoNetwork` an extension to the `Network` to provide an interface to the distance query graph with Geo formatted data functionality

`Network` the graph data structure with distance query capabilities

Figure 34: Class Diagram of API



`GeoDataFrame` a DataFrame class with a column for geometry types

`NetworkManager` creates instances of `GeoNetwork` and also functions as a "cached layer" data store, managing access to the pre-processed `GeoNetwork` instances that can run distance queries.

`Manager` responsible for handling concurrent access to the "cached" data store

The `WebAPIServer` class represents the framework and class that define the endpoints and request methods. The association arrows to the SQLAlchemy

package indicate that it is also responsible for maintaining the database engine and connection to be used during the server runtime. The `Controller` is designed to be the first point of contact from the `WebAPIServer` class. A number of methods are exposed from the Controller to use as a distribution method for processing the request. The directed arrows pointing from the `Controller` to the `NetworkManager` and the `QueryHandler` indicate an association whereby the `Controller` *controls* communication to the two classes.

The design of these relationships have an important implication for separation of concerns. The two primary operations that this relates to occur when the `Controller` is handling a POST request from the 'network' endpoint or a GET request from the 'nearest' endpoint. If a network is to be created through a POST request, the `Controller` can communicate with the `QueryHandler` to retrieve the nodes and edges data. When receiving the data, the `Controller` may pass this onto the `NetworkManager`, which creates the `GeoNetwork` to ensure that it can be preserved between requests. When handling a GET request on the 'nearest' endpoint, the `Controller` retrieves the POI data from the `QueryHandler` and passes this to the `NetworkManager`. The `NetworkManager` can then run the k-nearest distance queries on the preprocessed `GeoNetwork`.

6.5.4 Network creation - extending Pandana

Pandana is intended for general network computation, and doesn't support geoformatted data (even though it does include X- and Y-coordinates for the nodes). Also, edges are only abstract representations of connections, they do not have an inherent shape. Therefore, as shown in the class diagram in figure 34 we implemented `GeoNetwork`, a subclass of the pandana `Network` class. This was implemented to use all the existing Pandana computations, but extending some of the functionality to simplify working with standardized geoformatted data. In short, this implementation uses `GeoDataFrame` instances for nodes and edges as argument inputs to the constructor, and similarly retrieve distance query results in `GeoDataFrame` format. Furthermore, it would also enable many useful extensions - e.g. aggregating distance data within polygons by performing spatial joins on the computed node-data.

6.5.5 Network in-memory vs. persisted Network

An issue that was encountered when using Pandana and Flask as the web framework was how to retain a `GeoNetwork` instance between user requests. Since the cost related to the creation of the network itself is relatively expensive, saving it and allowing for fast queries afterwards would naturally give a more pleasant user experience. The idea behind storing it between requests is to *amortise* the cost of creating the contraction hierarchies. Thus, after a user has created an area of interest, the resulting network should be available such that it can be reused repeatedly for distance queries, e.g. if the user chooses new amenities and filters.

The options considered to preserve a contracted network could be to save it to persistent memory in the database, or to cache it, either using Flask's session functionality, or some external cache tool such as Redis. However, each of these options require that objects to store are serializable - meaning they can be converted to a byte stream, and thereby persisted, and when retrieved deserialized to create the object again. This functionality is unfortunately not available with Pandana.

This meant that a different solution had to maintain the `Network` objects in-memory between requests, a design choice which introduces questions that is present in most distributed systems, which involves concurrent processes, race conditions and deadlocks. Using Flask's built-in development server, it is possible to utilize a global variable to hold the objects. For a production environment however, a WSGI server interface would be used, running multiple workers and therefore the application instances in multiple processes, without access to shared memory.

Therefore the `Network Manager` was included in the design to control the `GeoNetwork` objects, and was implemented using the `Manager` class from the standard library multiprocessing module in python - whose purpose is to provide an access layer to shared memory objects to other processes. In essence, the `Network Manager` was implemented such that the request processes from the `API server` can serve the user with data, and whenever a request needs to use or create a `GeoNetwork` instance, it can do so via the `Network Manager` running as a separate process, and controlling concurrent access between the processes. The inherent limitations with this design will be discussed in section 8

6.6 Summary

This chapter has covered the main design- and implementation options that were considered during the development efforts, to address and satisfy the functional requirements from section 3.4.1. This was made possible by utilizing or make necessary alterations of existing libraries such as Pandana, Flask and PostGIS for back-end, and React, Turf for front-end.

The chapter has also showcased some of the technical functionalities and complexity of the Maption instrument, and the related implications that had to be addressed.

In the next section we will test and benchmark the components of the system and evaluate how they line up to the defined non-functional requirements from section 3.4.2.

7 Experiments

Testing the performance of Maption system components is especially important in regards to the non-functional requirements described in section 3.4.2. The experiments strive to test with different sized areas and to test the system under different conditions, such that critical bottlenecks can be identified.

Experiments and evaluation will be conducted on the following components/processes of Maption:

- Network creation and fetching of data, by querying the Maption API and the Overpass API. This is relevant for evaluation of Maption's architecture vs the considered architectures described in section 4.2.
- Performance testing of indexes described in section 4.3.2. Index performance is relevant to evaluate due to the performance requirement described in section 3.4.2. Testing and evaluating the indexes' performance on different sized search areas and edge densities, can help indicate if the current database design and the chosen indexes can support the fulfillment of the performance requirement.
- Fetching of network data using Geopandas built-in PostGIS functions. This was identified to have potential overhead and was therefore evaluated in regards to performance requirements. It can verify indications of which technologies to find alternatives for or keep for further development.
- Creation of contraction hierarchies. Testing this operation relates to the performance requirement, and what potential restrictions should be imposed on area size. By testing the related algorithm on different hardware, it can help indicate where to scale in terms of hardware to improve the run-time.
- k-nearest POIs search in Network. Testing k-nearest POIs within different sized networks, relates to the performance requirement. By testing the k-nearest algorithm on different sized areas, number of POIs and different hardware, a more adequate estimation of the system limits can be accounted for in the user interface.

7.1 Experiment setup

Eight different experiments have been designed to evaluate the above parts of the system. All performed experiments are conducted using real data, either processed or raw OSM data.

All experiments are based on spatial queries, where a geographical area (polygon) is used as a spatial filter for requesting data within an area. Each experiment has been run using 5 iterations for each polygon in the input set used for a specific experiment. Mean run-time and standard deviation are reported for each of the experiments and input set, and can be found in the appendix 10. The plots in the following sections are log-log scaled, and run-times are in some cases normalized, as specified in each section. The experiments have been performed using the following hardware.

Computer	Processor	Memory	OS
Laptop	1,4 GHz Quad-Core Intel Core i5	8 GB	MacOS Monterey
Desktop	Intel(R) Core(TM) i5-9600KF CPU 3.70 GHz	16 GB	Windows 64-bit

The experiments which rely on the database and/or the Overpass API, only report the desktop results. The database used has the specifications detailed in section 20.

For the experiments in section 7.2.5 and 7.2.6, results for both hardware instances are reported, since these experiments are isolated from the database hardware and network connections, and therefore can show hardware-specific performance comparisons of vital operations of the API.

A brief description of each type of test input (numbers correspond to 'Input ID' in table 1

1. Nine areas of varying size within the range 8.9 - 10,721.8 km^2, (see smallest and largest area in figure 36)
2. Nine areas of varying size within the range 8.9 - 10,721.8 km^2, (see smallest and largest area in figure 36) and with fixed number of POIs (amenities) (229 - 290)

3. Seven areas of varying density, within range (40 - 722 (edges/km^2))
4. Fixed area of size 548.9 km^2 (see figure 35), varying number of POIs within the range (21-2780)

Input ID	experiments
1	Network creation (including fetch data) comparison
1	Fetch data comparison
1	Comparison of PostGis indexes and no index
3	Comparison GIST and SP-GIST runtime on density (edges/sqKm)
1	Comparison of fetch methods
1	Comparison of creating contraction hierarchies
2	Comparison k-nearest POIs by number of POIs
4	Comparison k-nearest POIs runtime by size of network

Table 1: Inputs related to experiments.

Figure 35: input experiment fixed area (548 km^2)

7.2 Experiment results

The resulting data output for each experiment can be found in appendix 10

Figure 36: Input experiment different areas

(a) smallest (8.8 km^2)

(b) largest (10,721 km)

7.2.1 Pandana/OSM fetch and create network vs. Maption fetch and create network

Fig.37 compares network creation, including data fetching, for the Maption API and the Pandana built-in function using the Overpass API, in respect to the mean run-time (seconds) and the size of the geographical request area (km^2).

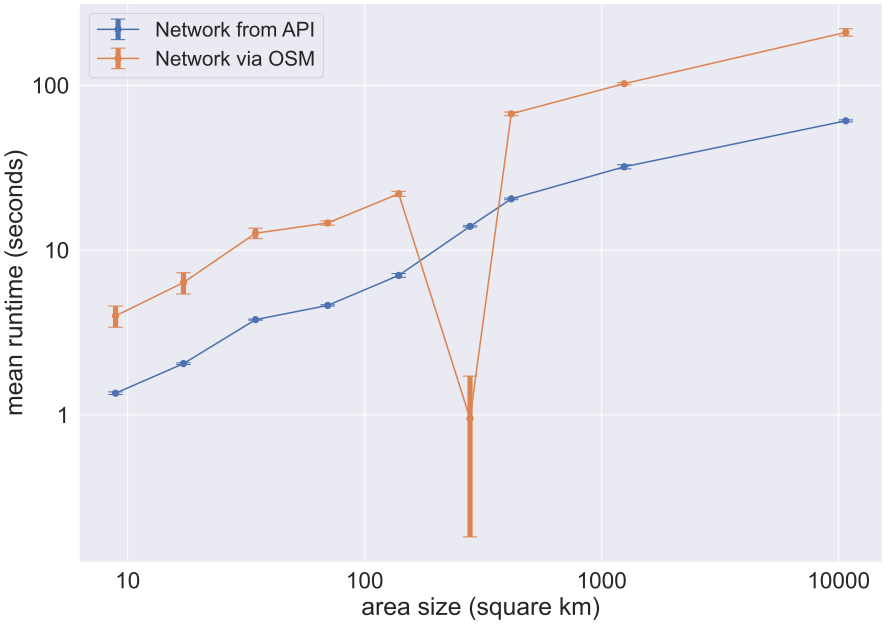
It can be observed that the mean run-time tends to increase somewhat linearly with the area size. It can furthermore be observed, that given the hardware used in this experiment, creating a network using the Maption API will have a better mean run-time for each of the input areas, compared to using the Overpass API.

It should be noted that the data response size for the Maption API and the Overpass API are not equal. Therefore, the comparison will solely function as motivation and argumentation for why using a self managed database is a sensible solution.

The Overpass API imposes rate limits and priority request queues. These factors create difficult test conditions, and must be taking into consideration while during the assessment. One attempt to overcome this issue, was to implement a timeout of 1 minute between experiment runs, which reduced the occurrence of fetch errors.

In fig. 37, one anomaly can be observed for data point 6 in the OSM graph, with a sudden decrease in mean run-time. The cause of this anomaly

Figure 37: Network creation (including fetch data) comparison



has not been completely uncovered. The experiment's log give some indication that the anomaly can be related to the following Overpass error message: **WARNING:osmnet:Server at www.overpass-api.de returned status code 429 and no JSON data. Re-trying request in 23.00 seconds**. The error indicates that too many requests are sent, when multiple queries are sent from one IP.

7.2.2 Fetching from Overpass vs. fetching from API

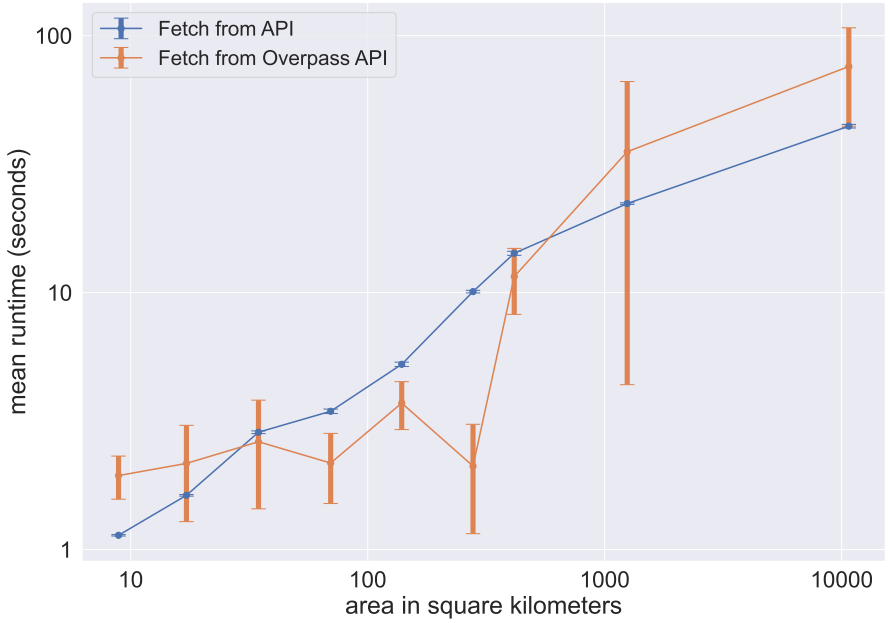
Fig.38 depicts the mean run-time (seconds) of fetching network data, in respect to the size of the geographical request area (km^2).

It can be observed that the Overpass API in general has a greater standard deviation compared to the Maption API. The large standard deviations can partially be explained by the limitations and restrictions posed by the Overpass API described in the above section.

The Maption API's mean run-time can be observed to scale with the area

size, as expected. In case of the Overpass API this pattern is less obvious because of the large standard deviations.

Figure 38: Fetch data comparison



7.2.3 Index testing

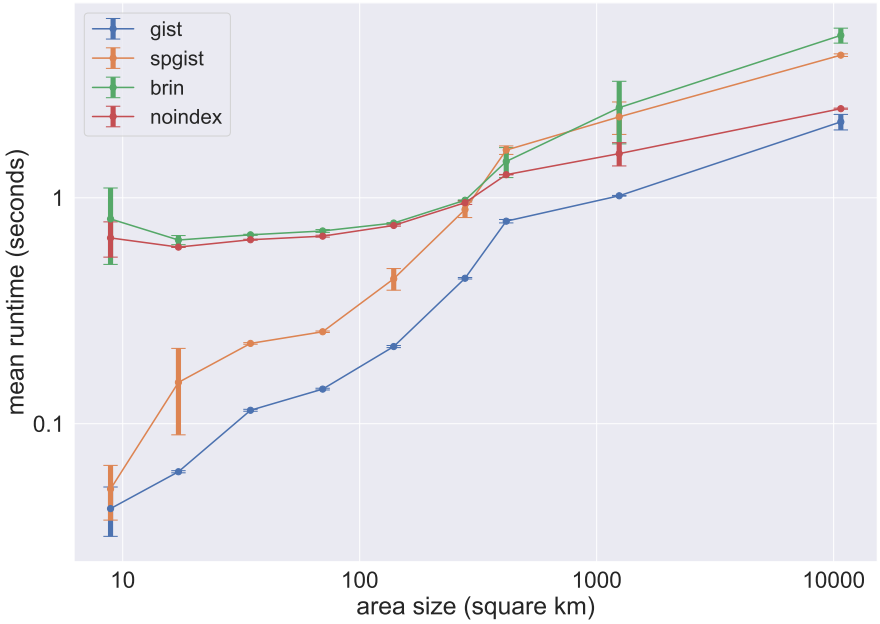
In section 4.3.2, three indexes were described and discussed: GIST, SP-GIST and BRIN. Given the differences in how these indexes are constructed, there are several factors that could potentially influence the comparative performance of the indexes; e.g. area size, complexity of polygon, type of geometry, amount of data records within an area and so on. Two experiments were performed on the edge table here to evaluate potential differences. Execution time is recorded on the database server (excluding query planning and network latency). This experiment was performed using the Postgres built-in "explain analyze" command, and recording the reported execution time.

Run-time per edge size Figure 39 shows the mean run-time for each of the polygons detailed in section 7.1, plotting number of edges on the x-axis.

Experiments were conducted for queries with each of the three indexes, as well as without index. The results indicate that GIST has the best performance of the three. Both GIST and SP-GIST shows orders of magnitude better performance for smaller queries, than both BRIN and using no index. This is expected, since the database is designed to avoid complex joins, and can utilize the index with the `ST_Within` function to quickly locate the rows to fetch.

For larger queries (in the 500K-1M range), the performance gains are less obvious, and SP-GIST performs even worse than using no index. However, since these row counts borders on the entire table size, it is expected that a sequential scan would then have similar performance. The BRIN index performs the worst, without performance gains over using no index. Since the BRIN index works with a specific sorting order of the table, the slow execution could indicate that the configuration used for this experiment has not been appropriate, or that BRIN is not applicable for our data layout.

Figure 39: Comparison of PostGIS indexes and no index

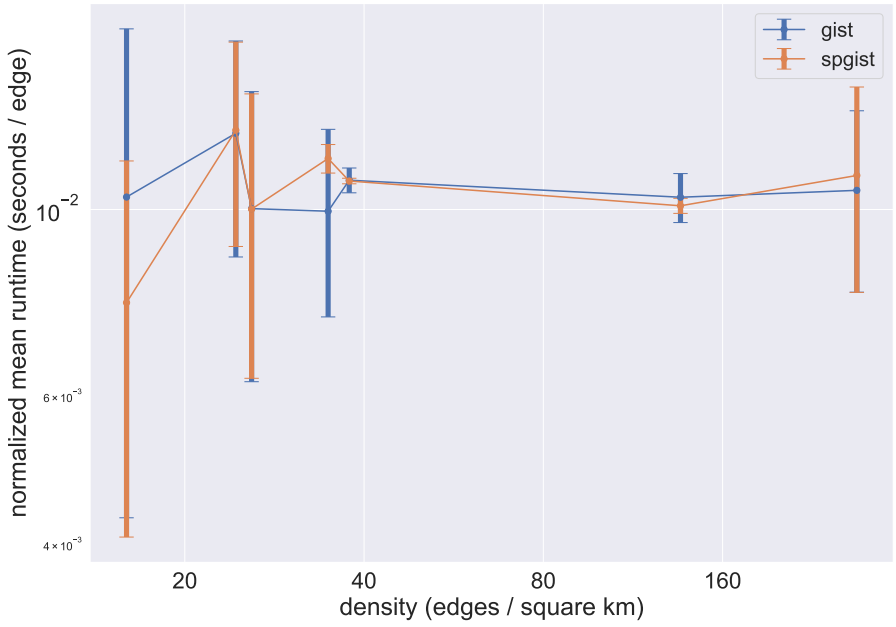


Comparing query performance for GIST and SP-GIST on density (edges/km^2) Figure 40 compares the two best-performing indexes, GIST and SP-GIST, using density as parameter. Density can be thought of here as the number of edges per square kilometer. The set of input polygons are used, fixing the area size, but moving the polygon around between dense and less dense areas.

Given that more records need to be fetched for denser areas, the mean run-time (and standard deviation) is normalized by number of edges to show how the query performs relative to how dense the area is.

Although the normalized mean run-time looks fairly stable, for both GIST and SP-GIST, the results show large standard deviations, especially for less dense areas. It is therefore difficult to conclude any trends based on the density of the area, and comparative advantages of either index.

Figure 40: Comparison GIST and SP-GIST runtime on density (edges/km^2)

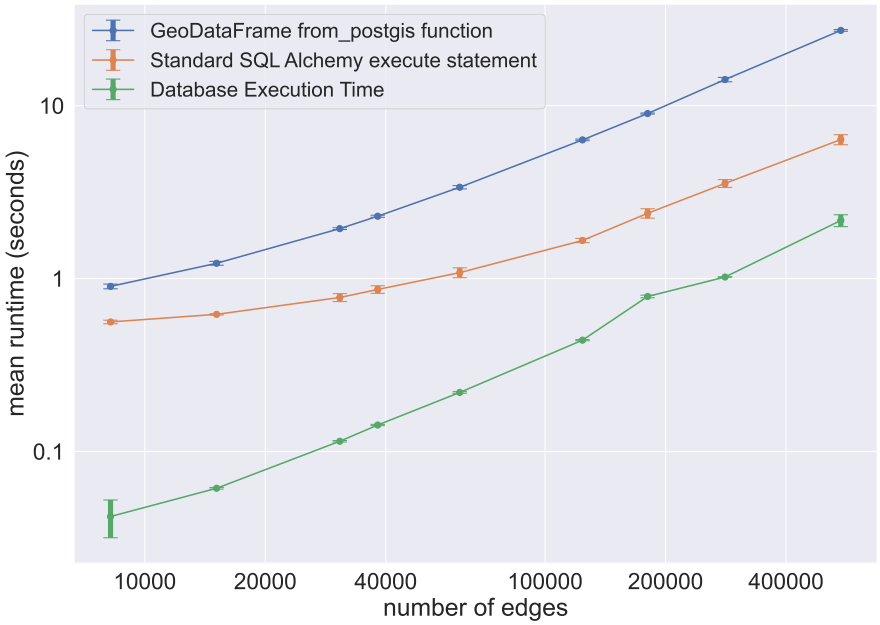


7.2.4 Testing the GeoPandas PostGIS function

A large discrepancy can be seen in the run-time between fetching the data from the Maption API in figure 38 and at the query execution time at the database level in figure 39. For instance, for the same areas where database query execution runs in sub-second time, the entire fetch process run in much longer times (5-10 seconds). It could indicate either network latencies of large data transfers, or slow data processing times of the functions in the backend used as middle layer to fetch from the database.

The discrepancy motivated an experiment to identify a potential bottleneck. In the current implementation GeoPandas built-in `from_postgis` function is used to retrieve data from the database. Fetching using this function was therefore compared with using standard fetch methods from the SQL Alchemy library in Python. For comparison purpose, the database execution time with the GIST index from figure 39 is also depicted here in figure 41

Figure 41: Comparison of fetch methods



The results are a clear indication that the fetch method using GeoPandas is a bottleneck. For small, sub-second queries it may not have a significant effect. There, the difference from the database execution time could also very well be due to network latency.

However, for larger queries where the number of edges is more than 100K, it shows a more increasing growth rate in running time compared to using SQL Alchemy standard fetch methods. While the database execution is still well below 1 second, the GeoPandas function is using almost 10 seconds in total to process the query. Even though there may be some network latency involved, the big difference between GeoPandas and SQL Alchemy points to some other functionality in the GeoPandas processing that takes more time. Since GeoPandas is an external dependency, this would have to be investigated further in future improvements of the instrument.

7.2.5 Generating contraction hierarchies

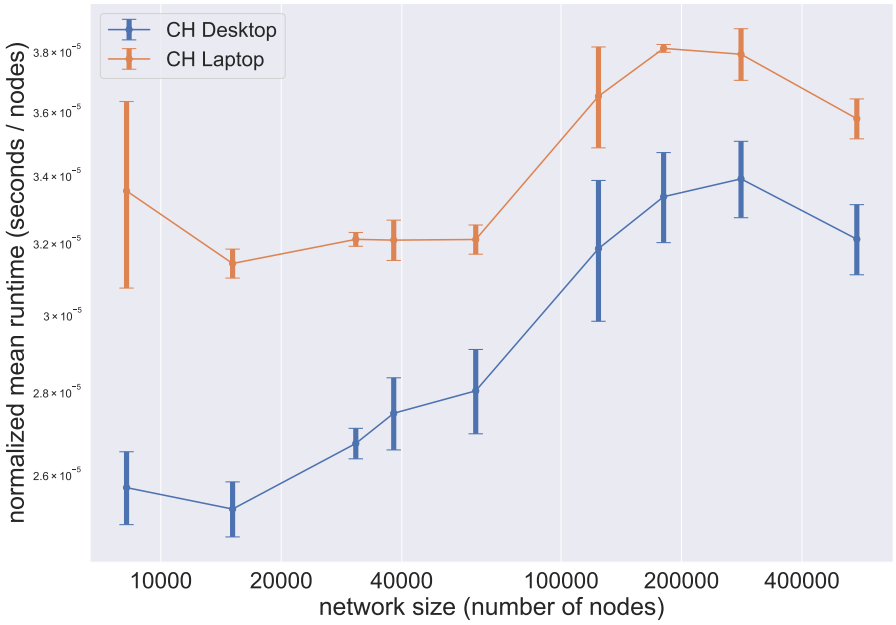
Running experiments on the construction of contraction hierarchies also becomes an important indicator of potential limitations of Maption. Since the instrument supports user-defined areas, the contraction hierarchies are created on-demand, rather than in an offline phase.

Given the theoretical limits of this rather expensive operation as discussed in section 4.4.6, the run-times can indicate how large areas (in terms of graph size) can be supported, to meet performance requirement.

The implementation of the contraction hierarchies in Pandana relies heavily on utilizing the available hardware for parallel computation. Experiments were therefore conducted on two computers with different hardware specifications to show which extent this also can become a factor - it would especially be relevant to consider in a future deployment setup.

Figure 42 shows the mean run-times for constructing networks of varying size (in terms of nodes).

Figure 42: Comparison of creating contraction hierarchies



The results show that making contraction hierarchies can be supported for graphs of input sizes with well over 100K nodes in less than 10 seconds. From the normalized plot there are some interesting results in terms of run-time per node, with first an increasing trend, but then reaching a break-point where the run-time actually decreases per node. This could perhaps be attributed to gains from the parallel execution paying off more in the long-term since there are costs associated with initializing threads, but this would need further investigation.

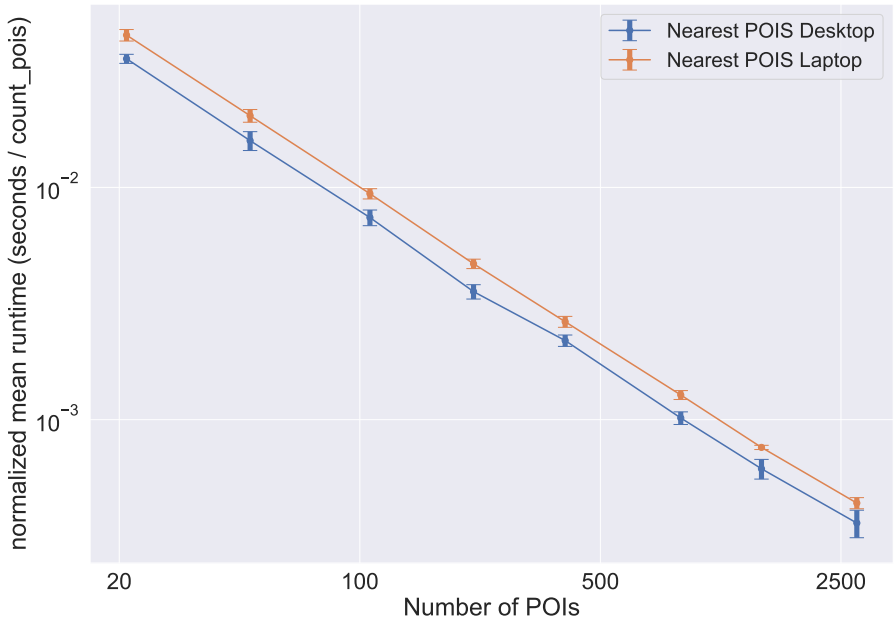
The difference in hardware can also be seen, where the Desktop performs with a factor of approximately 1.2 better than the Laptop. Although this is not a great increase, it shows that additional hardware resources is certainly a factor to consider when scaling the application. Further investigation would also have to look into this relationship in greater detail, testing on a variety of hardware to assess the feasibility and limitations when deploying.

7.2.6 Querying the network

As explained in section 4.4.6, creating contraction hierarchies should give a significant speed up in the query phase. The bucket-based approach with backward search and forward search occurring only once for each $s \in S$ (in this case S is all nodes in the network) and $t \in T$ (all POIs queried for in the network) rather than pairwise, should make the search phase highly scalable. To verify this assumption, two different experiments were run, in all cases using the "all-nodes" k-nearest POIs algorithm described in section 4.4 with $k = 5$.

Fixed area, varying POIs The first experiment fixes the area, and varies the number of POIs within that area. In other words, S (in this case equal to V or "all nodes") is held fixed, while T (set of POIs to search k-nearest to) varies - approximately doubling $|T|$ for each run.

Figure 43: Comparison k-nearest POIs by number of POIs



The results of this experiment is shown in Figure 43. Normalizing the

run-time by number of POIs = $|T|$, gives a clear indication that the search phase scales well, with a steeply decreasing trend in run-time per POI. In essence, this means that the queries will not be highly affected by how many POIs is queried for. For instance, there are 2780 benches in the test input area, and mean run-time for this query is ≈ 1.21 seconds. In contrast there are 21 doctors, with a mean run-time of ≈ 0.95. Normalized per POI, the run-time orders of magnitude faster for the larger (bench) input. The results show only a slightly better performance for the desktop. Compared to the contraction phase, it is also evident that hardware is not as crucial a factor in the query phase.

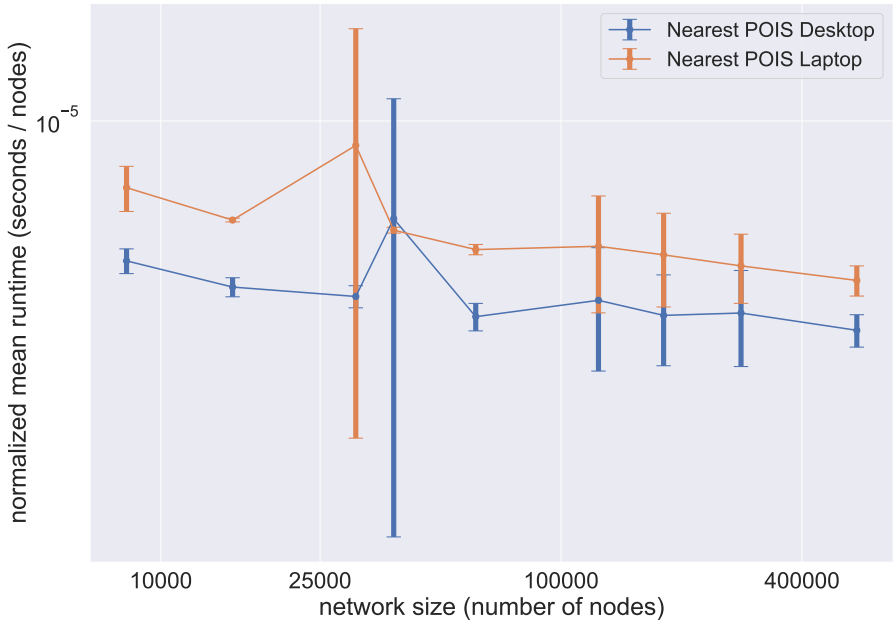
Fixed number of POIs (approximate), varying area size The second experiment checks the query performance on increasingly larger input areas (in terms of nodes). This varies S, so the size of the network, expressed in $|S|$ is doubled for each run, while holding T (approximately) fixed. This was achieved by selecting different types of POIs that had approximately the same count ($229 < |T| < 290$) within very different areas. Similar to the first experiment, the expectation is that the query phase will still scale well when increasing the network size.

The results shown in figure 44 show a normalized mean run-time with slightly decreasing trend. These results also show higher variance, and less scalable results compared to the previous experiment. However it shows still a sub-linear relationship the size of the network, and can support $k = 5$ nearest distance queries for the ≈ 250 POIs on networks with sizes up to 500K nodes in 2.53 seconds. This means it would not be a major bottleneck, because with these sizes of networks there are much greater bottlenecks as identified earlier, and some limitations would most likely need to be imposed on the size of networks in a production-ready setup.

7.2.7 Summary and comparison of steps

The results overall have demonstrated that the architectural choice of using our own DBMS provides a more stable and performant request process for "fetch and create network", compared to directly querying the Overpass API using the Pandana built-in function. This was expected, given that the create network process is not completely comparable, because as explained,

Figure 44: Comparison k-nearest POIs run-time by size of network



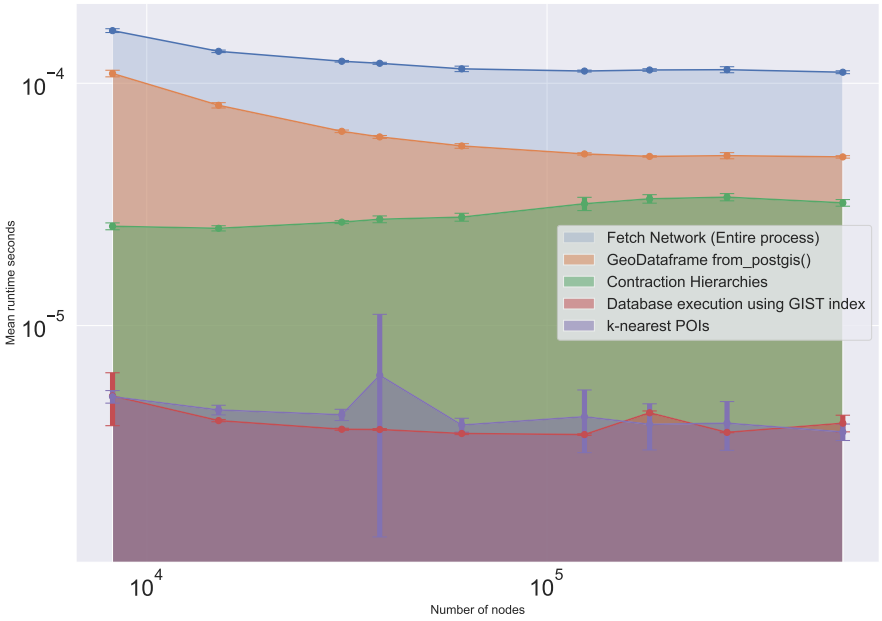
the Maption API has already processed the data into a graph data structure ahead of time.

However, the performance comparison when only considering the data fetch (excluding processing for network creation) between Maption API and OSM are not as clear-cut. This indicated that there is some part of our API fetch that is not performing well. Figure 45 shows the comparative normalized run-time of different parts of the process; the blue line represents the entire fetch and create network process.

The experiments showed that the database execution, using spatial indexes are very performant, as expected. However, the GeoDataframe function used to read the data is slow, which creates an overhead on the entire process. Removing this overhead would leave a more clear performance advantage over fetching from Overpass, and would be an important focus area for future work.

Creating the contraction hierarchies are also taking a significant portion of the request time. Given the theoretical assumptions presented in 4.4.6

Figure 45: Comparison of normalized run-time all components



this was expected. However the results showed that hardware could be a considerable factor for performance in this part of the process. Finally, the k-nearest POIs experiments also verify the assumptions that the number of POIs in a search scales well, whereas the size of the network is a more limiting factor - this is also in line with the theoretical assumptions. Since there are much fewer POIs than nodes ($|T| << |S|$) the backward searches from POIs are relatively "cheap" in comparison to the forward searches from nodes.

8 Discussion

The following chapter will discuss the central aspects of the implementation of Maption. It is structured such that section 8.1 will first evaluate and consider the limitations of the instrument from all levels of the stack. Secondly, possible improvements and important considerations for future work will be covered in section 8.2.

8.1 Evaluation and Limitations

This section will assess the architectural design and implementation choices, as well as limitations with both our current implementation and external dependencies. The section evaluates the system based on the software qualities and non-functional requirements defined in section 3.

8.1.1 Precision

The system should with the available OSM data represent all walking paths and accurate shortest distance metrics within meter precision..

To accommodate the precision requirement, it was decided to improve precision by keeping certain nodes, otherwise discarded by Pandana's built-in functions. These nodes essentially represent 'dead-ends'. Furthermore the system save the geometries of nodes that are only part of one *'Way'* which creates a more accurate display of streets on the map.

In the Network data object these nodes are discarded, for graph simplification reasons. This approach poses some limitation on the precision, in terms of k-nearest POIs queries. Since the discarded nodes are no longer represented in the graph representation, a POI cannot be assigned to these nodes, which means they will instead be assigned to the nearest intersection, potentially causing more imprecise distance computations. The simplification of the network graph, will add to the error of precision but is necessary for optimized performance and storage.

Another limitation related to the precision of the computed distances is that the constructed network can be a product of disconnected components. The impact of having a network of disconnected components is that parts of

the network will be unreachable from other disconnected component. In such scenarios, the distance from a node to a POI will be set to the max distance that is used as a threshold for the k-nearest POI function in pandana. Such a scenario is depicted in fig 46.

The impact of this issue is that an area will potentially get a much worse accessibility score, even though the average distance to a POI in reality is much less than computed.

Figure 46: disconnected components

One way to solve this issue would be to connect the disconnected components. By creating a LineString based on the shortest euclidean distance between the components, this could in many cases improve the precision.

This solution is rather naive, since the euclidean distance in its nature is 'optimistic'. The euclidean distance will always be less than or equal to the actual distance of a potential real connection. Furthermore it doesn't account for components that are in reality disconnected, such as islands see fig. 47.

Despite many great use cases of the OSM data, we have also identified limitations in regards to availability of elevation data for the included OSM elements [46]. The data included in Maption, does not include elevation data (z coordinates). Without elevation data, the created network will therefore model the street network in a two dimensional space, using the distance formula for two points (euclidean distance)

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

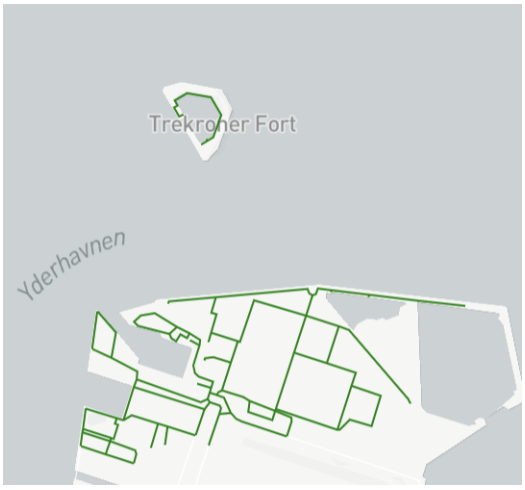


Figure 47: disconnected components

when computing the distance from a node to a POI. The implications of the missing elevation data, is that it effectively adds to the error of real world precision. Variation in the topography as well as the infrastructure (tunnels, bridges, buildings) will therefore impact the precision of the computed distance and hence the accessibility metric score. Including elevation data such that distance between two points in a 3-dimensional space, can be computed, using the formula [36]:

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

would also enable the possibility of adding more personalized options in regards to distance computation and personalized networks. The accessibility instrument GOAT (see section 2.6.6) makes it possible for a user to select disability as an option, which will impact the created network [52]. This is an interesting feature, since it enables some degree of personalized networks.

Due to the above-mentioned limitations, the *Precision* requirement can only be said to be partially fulfilled. One option for improvement would be to compute elevation data for each geometry in the database, by use of a digital elevation model (DEM) [18], but implementing this has not been in the scope of this thesis.

8.1.2 Flexibility

The system should support user queries of any number of amenities and filtering options within a user defined areas of sizes $< 10000\ km^2$

Several efforts has been made to meet the requirement of flexibility. The first key decision was to use the PostGIS extension. How this relates to flexibility is in respect to the element of user defined areas. As PostGIS allows for storing the geometry of nodes and edges, it therefore enables Maption to fetch data within any specified area by using the built-in spatial query functions. Combined with the 'drawing' functionality provided by Mapbox GL, the instrument enables the user to define any geographical boundaries, making the area-request completely customizable for the user.

The second key decision is the integration of all amenity-types of OSM. This is a direct consequence of the architectural choice, as no pre-computations need to be stored in the database. Pre-computing distance to all POIs would put limitations on either the amount of amenities the user could select, or the available area from which the user could request data. This would cause the storage requirements to exceed the limits set by the scalability NFR. From Maption's current design there are no immediate limitations to how large the selection of different amenities can be.

The third key decision was to utilize the **hstore** data type. This alone made it possible to query amenities with certain properties without populating the database with cells of NULL values. Since the hstore extension provides its own sub-query-language, tags can be stored as a dictionary-like type in a single column.

The fourth key decision was to use GeoJSON throughout the entire request flow. In a future implementation, this design choice could enable the option of allowing users to provide their own POIs. As all POIs in the database have the same format, any POIs can be added as long as it is uploaded in the GeoJSON format, making the instrument even more flexible.

Though these decisions and implementations have improved flexibility, there are also limitations. Some amenities in OSM are not denoted as nodes but as *Ways*. These amenities are typically larger, such as hospitals and parks. Due to their size they are sometimes denoted as *Ways* because ways can cover a larger geographical area, and thereby be a better representation

of the amenity. Transforming these OSM elements into POIs in the Maption API is possible and is not an unreasonable task. Transforming these could be done by finding the centroid of the way-geometry. However, this method could pose limitations in precision, e.g. for a park that can be accessed from many entry-points. Alternatively, the way-geometry could therefore be split into several POIs, but it has not been accounted for in the thesis, and it is therefore a limitation of Maption in its current form.

Another limitation is the inconsistency in the convention of tagging, since not all POIs of the same type have the same tags. Consider a filter on restaurants open after 8pm. All restaurants without an 'opening hour' tag would not be included in the fetch, and that way many possibly relevant restaurants would be excluded due to the missing tag. This limitation is more difficult to eliminate as it would require adding additional data sources for the missing tags.

8.1.3 Scalability

The architecture should support global coverage without having to change any core software components, and not exceed the storage requirements for the current OSM database (1.5 TB uncompressed XML).

A primary purpose of building the Maption instrument was to show how a scalable architecture could be designed. While scalability can entail the capacity to adjust in size, number of users, functionality or performance, it was in this context primarily focused on extending geographical data coverage.

From the review of previous implementations, as seen in table 10 the geographical scope of applications are often limited to "selected cities". In addition these instruments often rely on pre-computed metrics, and/or area-specific data; making a geographical expansion more cumbersome. As noted in section 2.5 official data from administrative regions can be hard to come by, and adhere to varying standards, making data integration a non-trivial task. In addition, the use of contraction hierarchies with pre-processing overhead, may not be feasible for all use-cases, as GOAT developers claim [52]. But without faster graph processing algorithms, on-demand queries in the style "all-nodes" k-nearest POIs are not feasible - putting a limit on geographical

coverage.

We therefore identified the three essential components to accomplish a scalable instrument that could still meet the other **NFRs**.

- A geographically uniform data model and open data source
- A simple, yet precise graph data model
- Performant algorithms for on-demand computation

The **NFR** described for scalability therefore attempts to address an architecture which would be so simple that a new country could be directly integrated into the current architecture, without storage requirements growing far beyond what is currently available in OSM.

The prototype is in essence a proof of concept to this requirement. The database currently only holds data for Denmark, but the database design, OSM graph data processing algorithm, API implementation and front-end application are all designed to support integration of new data, without having to change any components in the codebase. Furthermore, because we have steered clear of any pre-computations for the distance queries, the storage requirements should with a geographical expansion not exceed the current size of the OSM database. Because rather than to pre-compute distances, or add *more* data in some way, Maption only stores data that in some way relate directly back to the OSM database.

A different take on scalability concerns how the architecture scales with the number of users. The experiments have shown that the architectural choice of implementing the API with a database, rather than directly relying on OSM, are sensible for a user-facing application where a lot of data need to be retrieved frequently. The portability of the application enables horizontal scaling of the application as the number of users grow.

However, a major limitation with the current implementation is related to how instantiated networks are stored between user requests. Although a solution was found to store networks in-memory, this solution is not viable for a large-scale production setup. Firstly, in a multi-user environment, these networks would have to be stored in a shared memory data structure. That would introduce a much greater complexity, and could also come with significant overhead related to performance in order to design the application such that thread-safety can be ensured.

Secondly, this would place great demands on memory specifications, and the current implementation would not be able to handle such an increase in memory consumption. It is therefore a limitation with the external dependency to the Pandana library, which in the current version cannot serialize the `Network` object. And it is a shortcoming of the current implementation, not being built to handle this scenario.

8.1.4 Performance

The system should support fetch-and-creation of a Network in less than 1 minute for an area of 10,000 km2. The system should support "all-nodes" k-nearest POIs queries for an area of 10,000 km2 in less than 2 seconds.

Computing accessibility queries based on a user-defined area with any selection of amenities *at run-time*, motivates a discussion on the limitations of performance.

Currently, no restrictions are enforced in the system to control the size of the area a user can request. Given the precision requirement, street-level detail can mean network size can become unmanageable for large areas. This can become problematic both in terms of load time of fetching data, query-time to compute the accessibility metrics, memory management and rendering capabilities of the web interface.

Therefore the requirement was set up to assess how the system could handle areas in the order of the largest metropolitan areas in the world. Using a classic Dijkstra to compute k-nearest POIs could remove the overhead of creating contraction hierarchies. However, the actual distance queries would not be possible within seconds, or even minutes [14, 35]. The measures taken to fulfill this requirement therefore include the initial data processing step, the implementation and use of Postgres/PostGIS, and the application of contraction hierarchies. The experiments show that the Maption API can process (fetch data and create contraction hierarchies) a Network of over 10,000 km^2 in less than a minute. Relying on OSM directly for the same process had a mean run-time of more than 200 seconds.

Since Maption should ideally support both urban planners and regular users, one may think that the load time of 1 minute is too long, but for smaller areas the fetch-and-create takes significantly shorter time. Moreover,

if study areas are not changed too frequently, the flexibility and performance of queries can allow for a much more interactive instrument once a network has been created; a trade-off worth making.

A limitation with this approach and **NFR** is naturally that the density of areas often vary. This begs the question on how one could impose restrictions to the size of areas a user can request. In the experiments the smallest input area was chosen from central Copenhagen, and roughly doubled in size, expanding outwards. This meant that larger areas covered increasingly sparse (rural) land areas. However, metropolitan areas of much greater size than Copenhagen (e.g. New York or Tokyo), would possibly be much denser, which would likely push the run-time for these areas over the **NFR** limit.

To restrict the user based on the size of the network, would however require querying the database for the number of nodes or edges after an area has been defined, which may not be optimal in terms of user experience. The easiest way to control the area the user is able to request, would most likely be to have a restriction on size in square kilometers. This could be calculated directly on the client, without interacting with the API. To implement this, heuristics on density could be calculated to find a suitable limitation, but this would require more experiments to be run.

The second part of the performance NFR addresses the effort of supporting "all-nodes" k-nearest POIs queries for an area of 10,000 km2 in less than 2 seconds. This is mainly possible due to the bucket-based search algorithm that Pandana adopts.

The experiments showed that the run-time pr. POI decreases when more POIs are queried for. Although queries for which there is a larger set of POIs has a longer run-time, the difference is relatively small due to the decrease in run-time pr. POI. This makes the search-algorithm that Pandana has implemented a feasible design choice.

The expected run-time of a multi-amenity query (e.g. restaurants and preschools) is not primarily determined by the number of POIs of each of these amenity types. Instead it is determined by the number of different amenity-types included in the query, because a separate query must be performed for each amenity type.

In summary, the use of contraction hierarchies, and the bucket-based k-nearest POI algorithm, the experiments showed that a query for a single

amenity-type with ≈ 250 POIs could be performed in an area of 10,000 km^2 in less than 2 seconds. However, the requirement must be said to only be partially fulfilled, as it is not tested on more dense areas, with possibly a higher POI count.

8.2 Future work

This section will describe how some of the limitations elaborated upon in the above section, can be solved, by developing existing features or adding additional functionalities.

8.2.1 Improving the UX

Categorized amenities: Users may not always have a clear idea of their specific preferences in terms of amenity-types. To improve the usability of the instrument and minimize required user interactions, pre-categorised amenities would be a useful feature. By grouping some amenities into categories such as 'Food', 'Entertainment', 'Culture' etc. the user can select all amenities within that category instead of having to add all amenities themselves (example: 'Restaurant', Cafe', 'Diner' under 'Food' category). This could be done by adding a category column in the POI table containing a categoryID and an index table of the categories. Such categorizations are often seen within the domain of accessibility.

UI and user testing: One of the objectives when developing Maption was to create a transparent and easy-to-use accessibility instrument. While that is somewhat accomplished in the sense of transparency of the analysis and metric scores, the current UI implementation requires a certain degree of experience on the platform. To eliminate this barrier, a user-oriented design of the instrument could be beneficial.

By running a heuristic evaluation with both first-time users and practitioners, we would get a more nuanced perspective and insights of the UI and usability of the application. The evaluation usually involves an independent walk-through by each user, assessing established heuristics such as visibility of system status, user control, error prevention, aesthetic and minimalist design and application-specific heuristics [44]

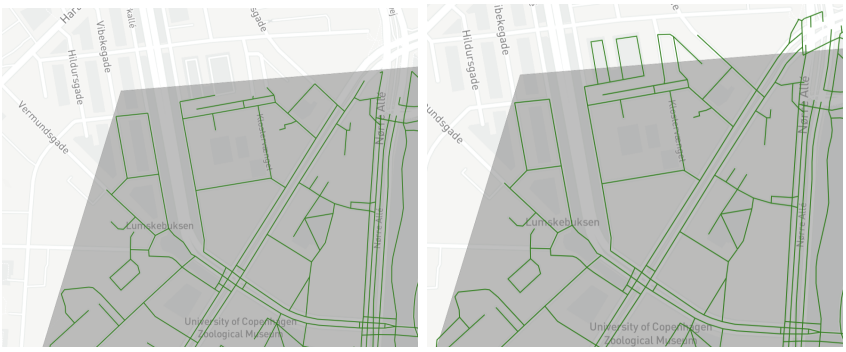
8.2.2 Computation

Isochronic analysis: In its current implementation Maption performs aggregated analysis in the form of "all-nodes" k-nearest POIs. To expand the use-case of the instrument it would be ideal to implement additional types of analysis such as isochronic analysis which is implemented in some of the existing accessibility tools. The current map implementation supports the functionality of placing a point on the map and returning coordinates for the point. To implement isochronic analysis, one could draw inspiration from the GOAT platform's implementation, using a concave hull algorithm to define the isochrone [56].

Buffering of the search area In the current implementation, all spatial queries are based on the PostGIS `ST_Within` function. The `ST_Within` function returns TRUE if geometry A is completely inside geometry B, corresponding to nodes and edges being inside a user defined area. Using `ST_Within` effectively discards edges that intersects with the user defined polygon. This approach has two problems. Firstly, it can potentially discard an edge that 'curves' outside of the search area. Secondly, it does not account for a POI that may be one of the k nearest, if it is located outside the drawn area.

One way to address this problem would be to implement a buffer functionality, for instance as shown in Figure.48, including data outside of the drawn gray area.

Figure 48: Network : no buffer & buffered



8.2.3 Extending libraries

As previous explained in section 6.1, the Pandana and GeoPandas libraries pose some limitations in regards to persisting the Networks and performance related to using GeoPandas' GeoDataFrame.

To overcome these limitations/barriers, the thesis group strives to do the following in the future work.

Pandanas current implementation doesn't allow serialization. The reason for this could be the C++ dependency which makes standard python serialization methods (such as pickling) more complicated.

The first option would be to identify how serialization could be implemented as an extension to the Pandana library.

The second option would be to find an alternative to the Pandana library. Although there exist plenty of open source libraries that offers 'shortest path' functionalities, we have not found other efficient alternatives using contraction hierarchies in Python. However there are libraries available in other languages that could provide this functionality, such as 'contraction-hierarchy-js', that should be considered [50].

GeoPandas - GeoDataFrame The reason for using GeoPandas' GeoDataFrame is due to the `from_postgis` functionality, allowing direct fetch from the data base into a row/column based format. Furthermore using GeoDataFrame also make conversion to GeoJSON format easy using the GeoDataFrame's build in functionality for this purpose. Although using these functionalities are very convenient, it also have some downsides in regards to the systems overall performance, as elaborated upon in the Experiments section 41.

When running experiments on data fetching it was identified that GeoPandas could be a bottleneck, as its built-in functions for fetching data from the database, had a significant overhead compared to other fetching approaches. It is therefore important to investigate how this can be solved, by studying the internal structure of the library to find the bottleneck.

9 Conclusion

Literature on accessibility suggests a divergence in how it should be conceptualized, and how it should be measured. On the one hand, accessibility should account for human perceptions, socio-economic activity and the demand of participants. On the other, it focuses on the importance and supply of amenities, as well as the physical proximity between them.

The development of dynamic accessibility instruments have facilitated urban planning with a variety of analysis layers to overcome these differences. However, identified instruments are commonly limited in their geographical coverage, rigid accessibility metrics or choice of amenities.

The purpose of this project is to explore the possibilities of creating an accessibility instrument which account for these limitations. To achieve this, it should be globally scoped, compute accessibility metrics on-demand, give the user free choice of study area and amenities, and accurately model street-level accessibility indicators.

To create an accessibility instrument with these properties, several architectural structures have been analyzed which eventually led to the implementation of Maption; an API using a spatial database with pre-processed OSM data. The pre-processing phase transforms data such that it complies with a simplified graph structure, yet preserves all geometries of the street network without using additional storage.

Maption does not use pre-computed distance queries, due to the storage limitations it entails. Because of that, distance queries needs to be computed quickly while still contributing analytical detail. Efforts to ensure this is by performing "all-nodes" k-nearest-queries with contraction hierarchies, using the Pandana library.

This query approach creates a detailed aggregated analysis by finding the k-nearest POIs from all nodes in the area of analysis. By utilizing contraction hierarchies the search space is reduced compared to classic Dijkstra, which results in fast enough computation, such that it can be done upon user request.

Maption combines spatial database queries, a geo-formatted extension to Pandana, and a single-page application to showcase how a user can query any combination of amenities in a user-drawn area on-demand.

To evaluate the feasibility of these architectural decisions, different parts

of the request cycles have been tested by several experiments. The experiments indicate that there are inherent limitations in performance for the largest metropolitan areas. They also identified a bottleneck in the GeoPandas library when processing large amounts of data from PostgreSQL/PostGIS. Furthermore, the lacking functionality for serialization of networks using the Pandana library, place memory-constraints on the current setup, and would require an alternative solution for a production-ready application. In spite of these limitations, the architectural design choices can serve as a framework for scaling accessibility instruments with more customizable, precise and dynamic planning support.

References

[1] 15-minutes.city. *15-minutes.city*. Jan. 1, 2022. URL: https://15-minutes.city/ (visited on 05/30/2022).

[2] Accessibilityplanning. *COST-META*. META-Accessibility. URL: https://www.accessibilityplanning.eu//about/ (visited on 05/30/2022).

[3] M Adnan, A Singleton, and P Longley. "Developing efficient web-based GIS applications". In: (), p. 16.

[4] European Environment Agency. *The first and last mile - the key to sustainable urban transport — European Environment Agency*. Feb. 3, 2020. URL: https://www.eea.europa.eu/publications/the-first-and-last-mile (visited on 04/20/2022).

[5] Sonam Agrawal and R. D. Gupta. "Web GIS and its architecture: a review". In: *Arabian Journal of Geosciences* 10.23 (Nov. 29, 2017), p. 518. ISSN: 1866-7538. DOI: 10.1007/s12517-017-3296-2. URL: https://doi.org/10.1007/s12517-017-3296-2 (visited on 04/06/2022).

[6] Overpass API. *Overpass API*. Jan. 1, 2022. URL: http://overpass-api.de/ (visited on 05/30/2022).

[7] *API Reference — Mapbox GL JS*. Mapbox. URL: https://docs.mapbox.com/mapbox-gl-js/api/ (visited on 05/30/2022).

[8] Michael Batty. "Accessibility: In Search of a Unified Theory". In: *Environment and Planning B: Planning and Design* 36.2 (Apr. 1, 2009). Publisher: SAGE Publications Ltd STM, pp. 191–194. ISSN: 0265-8135. DOI: 10.1068/b3602ed. URL: https://doi.org/10.1068/b3602ed (visited on 05/19/2022).

[9] Marco te Brömmelstroet et al. "Strengths and weaknesses of accessibility instruments in planning practice: technological rules based on experiential workshops". In: *European Planning Studies* 24.6 (June 2, 2016), pp. 1175–1196. ISSN: 0965-4313, 1469-5944. DOI: 10.1080/09654313.2015.1135231. URL: https://www.tandfonline.com/doi/full/10.1080/09654313.2015.1135231 (visited on 04/04/2022).

[10] Marco te Brömmelstroet, Cecilia Silva, and Luca Bertolini. *Assessing usability of accessibility instruments*. OCLC: 883953178. Amsterdam: COST office ; 2014. ISBN: 978-90-90-28212-1.

[11] Valentin Buchhold. "Fast Computation of Isochrones in Road Networks". In: (), p. 112.

[12] Rajkumar Buyya, Christian Vecchiola, and S. Thamarai Selvi. "Chapter 2 - Principles of Parallel and Distributed Computing". In: *Mastering Cloud Computing*. Ed. by Rajkumar Buyya, Christian Vecchiola, and S. Thamarai Selvi. Boston: Morgan Kaufmann, Jan. 1, 2013, pp. 29–70. ISBN: 978-0-12-411454-8. DOI: 10.1016/B978-0-12-411454-8.00002-4. URL: https://www.sciencedirect.com/science/article/pii/B9780124114548000024 (visited on 05/04/2022).

[13] C40 Cities. *Green & Just Recovery Agenda*. C40 Cities. Jan. 1, 2020. URL: https://www.c40.org/what-we-do/raising-climate-ambition/green-just-recovery-agenda/ (visited on 04/20/2022).

[14] Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. "Faster Batched Shortest Paths in Road Networks". In: (2011). In collab. with Marc Herbstritt. Artwork Size: 12 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 12 pages. DOI: 10.4230/OASICS.ATMOS.2011.52. URL: http://drops.dagstuhl.de/opus/volltexte/2011/3266/ (visited on 05/06/2022).

[15] Daniel Delling and Renato F Werneck. "Customizable Point-of-Interest Queries in Road Networks". In: (), p. 19.

[16] Daniel Delling et al. "Engineering Route Planning Algorithms". In: *Algorithmics of Large and Complex Networks*. Ed. by Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig. Vol. 5515. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 117–139. ISBN: 978-3-642-02093-3 978-3-642-02094-0. DOI: 10.1007/978-3-642-02094-0_7. URL: http://link.springer.com/10.1007/978-3-642-02094-0_7 (visited on 05/11/2022).

[17] Daniel Delling et al. "PHAST: Hardware-Accelerated Shortest Path Trees". In: (), p. 20.

[18] DEM. *What is a digital elevation model (DEM)? — U.S. Geological Survey*. Jan. 1, 2022. URL: https://www.usgs.gov/faqs/what-digital-elevation-model-dem (visited on 05/30/2022).

[19] *Documentation — GeoPandas 0.10.2+0.g04d377f.dirty documentation*. URL: https://geopandas.org/en/stable/docs.html (visited on 05/30/2022).

[20] *Fig. 1 Simplified OSM conceptual data model. Source: [30]*. ResearchGate. URL: https://www.researchgate.net/figure/Simplified-OSM-conceptual-data-model-Source-30_fig1_335692603 (visited on 05/27/2022).

[21] Fonte. *(PDF) Assessing VGI Data Quality*. Jan. 1, 2017. URL: https://www.researchgate.net/publication/319632519_Assessing_VGI_Data_Quality (visited on 05/30/2022).

[22] Fletcher Foti and Paul Waddell. "A Generalized Computational Framework for Accessibility: From the Pedestrian to the Metropolitan Scale". In: (), p. 14.

[23] Andrew U Frank. "Requirements for a Database Management System for a GIS". In: (), p. 8.

[24] Koos Fransen et al. "A spatio-temporal accessibility measure for modelling activity participation in discretionary activities". In: *Travel Behaviour and Society* 10 (Jan. 1, 2018), pp. 10–20. ISSN: 2214-367X. DOI: 10.1016/j.tbs.2017.09.002. URL: https://www.sciencedirect.com/science/article/pii/S2214367X17300467 (visited on 05/30/2022).

[25] Robert Geisberger. "Advanced Route Planning in Transportation Networks". In: (), p. 227.

[26] Robert Geisberger et al. "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks". In: *Experimental Algorithms*. Ed. by Catherine C. McGeoch. Vol. 5038. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 319–333. ISBN: 978-3-540-68548-7 978-3-540-68552-4. DOI: 10.1007/978-3-540-68552-4_24. URL: http://link.springer.com/10.1007/978-3-540-68552-4_24 (visited on 05/06/2022).

[27] *Getting Started – React*. URL: https://reactjs.org/docs/getting-started.html (visited on 05/30/2022).

[28] K.T Geurs and Ritsema Van Eck. *Accessibility measures: review and applications. Evaluation of accessibility impacts of land-use transportation scenarios, and related social and economic impact*. Report. Accepted: 2012-12-12T19:06:15Z Journal Abbreviation: Bereikbaarheidsmaten: review en case studies. Beoordeling van bereikbaarheidseffecten van ruimtelijk-infrastructurele scenario's, en gerelateerde sociale en economische effecten. Universiteit Utrecht-URU, June 9, 2001. URL: https://rivm.openrepository.com/handle/10029/259808 (visited on 05/12/2022).

[29] Karst T. Geurs and Bert van Wee. "Accessibility evaluation of land-use and transport strategies: review and research directions". In: *Journal of Transport Geography* 12.2 (June 2004), pp. 127–140. ISSN: 09666923. DOI: 10.1016/j.jtrangeo.2003.10.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0966692303000607 (visited on 05/11/2022).

[30] GISGeography. *The Ultimate List of GIS Formats and Geospatial File Extensions*. GIS Geography. July 16, 2019. URL: https://gisgeography.com/gis-formats/ (visited on 05/30/2022).

[31] Eduardo Graells-Garrido et al. *A city of cities: Measuring how 15-minutes urban accessibility shapes human mobility in Barcelona*. Mar. 22, 2021. DOI: 10.1371/journal.pone.0250080. arXiv: 2103.15638[cs]. URL: http://arxiv.org/abs/2103.15638 (visited on 05/30/2022).

[32] Hansen. *How Accessibility Shapes Land Use: Journal of the American Institute of Planners: Vol 25, No 2*. Jan. 1, 1959. URL: https://www.tandfonline.com/doi/abs/10.1080/01944365908978307 (visited on 05/30/2022).

[33] Angela Hull, Cecília Silva, and Luca Bertolini. "Accessibility Instruments for Planning Practice". In: (), p. 409.

[34] Mohammed Islam and Sohel Ahmed. *"Demand of Participants" or "Supply of Opportunities": Measuring Accessibility of Activity Places Based on Time Geographic Approach — Journal of Urban Planning and Development — Vol 134, No 4*. URL: https://ascelibrary.org/

doi/abs/10.1061/(ASCE)0733-9488(2008)134:4(159) (visited on 05/19/2022).

[35] Sebastian Knopp et al. "Computing Many-to-Many Shortest Paths Using Highway Hierarchies". In: *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*. Ed. by David Applegate and Gerth Stølting Brodal. Philadelphia, PA: Society for Industrial and Applied Mathematics, Jan. 6, 2007, pp. 36–45. ISBN: 978-1-61197-287-0. DOI: 10.1137/1.9781611972870.4. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611972870.4 (visited on 05/15/2022).

[36] Jim Lambers. *Three-Dimensional Coordinate Systems*. Oct. 2009. URL: https://www.math.usm.edu/lambers/mat169/fall09/lecture17.pdf.

[37] Katrin Lättman and Margareta Friman. *A new approach to accessibility – Examining perceived accessibility in contrast to objectively measured accessibility in daily travel - ScienceDirect*. URL: https://www.sciencedirect.com/science/article/pii/S0739885917302445 (visited on 05/19/2022).

[38] Katrin Lättman, Lars E. Olsson, and Margareta Friman. "Development and test of the Perceived Accessibility Scale (PAC) in public transport". In: *Journal of Transport Geography* 54 (June 1, 2016), pp. 257–263. ISSN: 0966-6923. DOI: 10.1016/j.jtrangeo.2016.06.015. URL: https://www.sciencedirect.com/science/article/pii/S0966692316303295 (visited on 05/19/2022).

[39] Liu, Shiqin, and Boeing. *(PDF) A Generalized Framework for Measuring Pedestrian Accessibility around the World Using Open Data*. Jan. 1, 2021. URL: https://www.researchgate.net/publication/351701785_A_Generalized_Framework_for_Measuring_Pedestrian_Accessibility_around_the_World_Using_Open_Data (visited on 05/26/2022).

[40] Ying Long and Yao Shen. "Mapping parcel-level urban areas for a large geographical area". In: *Annals of the American Association of Geographers* 106.1 (Jan. 2, 2016), pp. 96–113. ISSN: 2469-4452, 2469-

4460. DOI: 10.1080/00045608.2015.1095062. arXiv: 1403.5864. URL: http://arxiv.org/abs/1403.5864 (visited on 05/09/2022).

[41] Claudia Bauzer Medeiros and Fatima Pires. "Databases for GIS". In: *ACM SIGMOD Record* 23.1 (Mar. 1994), pp. 107–115. ISSN: 0163-5808. DOI: 10.1145/181550.181566. URL: https://dl.acm.org/doi/10.1145/181550.181566 (visited on 05/04/2022).

[42] Eric Miller. "Measuring Accessibility". In: (2020). DOI: https://doi.org/https://doi.org/10.1787/8687d1db-en. URL: https://www.oecd-ilibrary.org/content/paper/8687d1db-en.

[43] Eric J. Miller. *Accessibility: measurement and application in transportation planning*. URL: https://www.tandfonline.com/doi/full/10.1080/01441647.2018.1492778 (visited on 05/19/2022).

[44] NN.Group. *10 Usability Heuristics for User Interface Design*. Nielsen Norman Group. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (visited on 05/29/2022).

[45] Nominatim. *nominatim*. URL: https://nominatim.org.

[46] OpenStreetMap. *Planet.osm - OpenStreetMap Wiki*. URL: https://wiki.openstreetmap.org/wiki/Planet.osm (visited on 02/23/2022).

[47] OSM. *Relation*. URL: https://wiki.openstreetmap.org/wiki/Relation.

[48] OSM. *Way*. URL: https://wiki.openstreetmap.org/wiki/Way.

[49] Overpass. *Overpass Commons*. URL: https://dev.overpass-api.de/overpass-doc/en/preface/commons.html.

[50] *Package - contraction-hierarchy-js*. URL: https://npmmirror.com/package/contraction-hierarchy-js (visited on 05/30/2022).

[51] Antonio Páez, Darren M. Scott, and Catherine Morency. "Measuring accessibility: positive and normative implementations of various accessibility indicators". In: *Journal of Transport Geography* 25 (Nov. 1, 2012). Publisher: Pergamon, pp. 141–153. ISSN: 0966-6923. DOI: 10.1016/j.jtrangeo.2012.03.016. URL: https://www.sciencedirect.com/science/article/pii/S0966692312000798 (visited on 04/26/2022).

[52] Elias Pajares et al. "Accessibility by proximity: Addressing the lack of interactive accessibility instruments for active mobility". In: *Journal of Transport Geography* 93 (May 1, 2021), p. 103080. ISSN: 0966-6923. DOI: 10.1016/j.jtrangeo.2021.103080. URL: https://www.sciencedirect.com/science/article/pii/S0966692321001332 (visited on 04/04/2022).

[53] Manuel Santana Palacios and Ahmed El-geneidy. "Cumulative versus Gravity-based Accessibility Measures: Which One to Use?" In: *Findings* (Feb. 10, 2022). Publisher: Findings Press, p. 32444. DOI: 10.32866/001c.32444. URL: https://findingspress.org/article/32444-cumulative-versus-gravity-based-accessibility-measures-which-one-to-use (visited on 05/30/2022).

[54] *Pandana — pandana 0.6.1 documentation*. URL: https://udst.github.io/pandana/ (visited on 05/30/2022).

[55] ParisCSL. *whatif-machine*. Jan. 1, 2022. URL: http://whatif.cslparis.com/15minCity.html (visited on 05/30/2022).

[56] Plan4Better. *Isochrone as Alphashape — Plan4Better*. Isochrone as Alphashape — Plan4Better. URL: https://plan4better.de/en/docs/alphashape/ (visited on 05/30/2022).

[57] Plan4Better. *Technical Architecture — Plan4Better*. Jan. 1, 2022. URL: https://plan4better.de/en/docs/technicalarchitecture/ (visited on 05/30/2022).

[58] PostGIS. *PostGIS spatial indexes*. URL: https://postgis.net/docs/using_postgis_dbmanagement.html#gist_indexes (visited on 05/29/2022).

[59] PostgreSQL. *PostgreSQL: About*. URL: https://www.postgresql.org/about/ (visited on 04/07/2022).

[60] *Redux - A predictable state container for JavaScript apps. — Redux*. URL: https://redux.js.org/ (visited on 05/30/2022).

[61] Jean Ryan and Rafael H. M. Pereira. "What are we missing when we measure accessibility? Comparing calculated and self-reported accounts among older people". In: *Journal of Transport Geography* 93 (May 1, 2021), p. 103086. ISSN: 0966-6923. DOI: 10.1016/j.jtrangeo.2021.103086. URL: https://www.sciencedirect.com/science/article/pii/S0966692321001393 (visited on 05/30/2022).

[62] David S Vale and Mauro Pereira. *The influence of the impedance function on gravity-based pedestrian accessibility measures: A comparative analysis*. Jan. 1, 2017. URL: https://journals.sagepub.com/doi/10.1177/0265813516641685 (visited on 05/12/2022).

[63] Dominik Schultes. "Route Planning in Road Networks". In: (), p. 235.

[64] Walk Score. *Walk Score Professional - For Your Project, Site or Mobile App. Walk Score Widget and Walk Score API*. URL: https://www.walkscore.com/professional/ (visited on 05/30/2022).

[65] Sedgewick. *Sedgewick & Wayne, Algorithms, 4th Edition — Pearson*. Jan. 1, 2011. URL: https://www.pearson.com/uk/educators/higher-education-educators/program/Sedgewick-Algorithms-4th-Edition/PGM940484.html (visited on 05/30/2022).

[66] I. Sommerville. *Software Engineering*. 9th ed. Addison-Wesley , Harlow, England, 2010.

[67] Steiniger and Hunter. *Free and Open Source GIS Software for Building a Spatial Data Infrastructure — SpringerLink*. Jan. 1, 2012. URL: https://link.springer.com/chapter/10.1007/978-3-642-10595-1_15 (visited on 05/24/2022).

[68] ROBERT STEUTEVILLE. *Walkability indexes are flawed. Let's find a better method*. cnu. Jan. 10, 2019. URL: https://www.cnu.org/publicsquare/2019/01/10/walkability-indexes-are-flawed-lets-find-better-method1.

[69] Nathan R. Swain et al. "A review of open source software solutions for developing water resources web applications". In: *Environmental Modelling & Software* 67 (May 1, 2015), pp. 108–117. ISSN: 1364-8152. DOI: 10.1016/j.envsoft.2015.01.014. URL: https://www.

sciencedirect.com/science/article/pii/S1364815215000353 (visited on 04/07/2022).

[70] TAJC. *15_Minute_Cities___Research_Project*. Jan. 1, 2021.

[71] Here Technologies. *The 15-min city – check your access to essential living needs*. URL: https://app.developer.here.com/15-min-city-map/ (visited on 05/30/2022).

[72] *The insider: A planners' perspective on accessibility - ScienceDirect*. URL: https://www.sciencedirect.com/science/article/pii/S0966692316304240?pes=vor (visited on 05/19/2022).

[73] *Turf.js — Advanced geospatial analysis*. URL: https://turfjs.org/ (visited on 05/30/2022).

[74] UN. *2018 Revision of World Urbanization Prospects — Multimedia Library - United Nations Department of Economic and Social Affairs*. Jan. 1, 2022. URL: https://www.un.org/development/desa/publications/2018-revision-of-world-urbanization-prospects.html (visited on 05/19/2022).

[75] *Welcome to Flask — Flask Documentation (2.1.x)*. URL: https://flask.palletsprojects.com/en/2.1.x/ (visited on 05/30/2022).

10 Appendices

Table 2: Density of areas experiments

km^2	nodes	edges	density (edge/km^2)
8.88	8212	6415	722.02
17.18	15115	11540	671.45
34.69	30704	22902	661.74
69.79	38191	28664	410.71
139.18	61215	45920	329.90
278.27	124022	93436	335.76
415.32	180401	135541	326.35
1244.87	281603	214036	171.93
10721.80	548189	430711	40.17

Table 3: GIST run-time by area (milliseconds)

area (km^2)	mean	std
88.88	38.06	1.52
17.18	62.43	0.709
34.69	118.25	7.003
69.79	145.26	4.109
139.18	228.77	9.976
278.27	512.05	154.748
415.32	780.52	4.972
1244.87	1032.24	15.454
10721.80	2083.59	14.339

Table 4: SP-GIST run-time by area (milliseconds)

area (km^2)	mean	std
88.88	65.10	11.63
17.18	119.60	53.23
34.69	195.5016	42.25
69.79	222.95	96.015
139.18	386.03	75.83
278.27	879.13	93.43
415.32	1703.09	338.55
1244.87	2072.96	74.00
10721.80	4425.17	365.22

Table 5: BRIN run-time by area size (milliseconds)

area (km^2)	mean	std
88.88	844.92	299.03
17.18	723.73	104.19
34.69	692.13	3.43
69.79	713.22	3.28
139.18	796.44	15.54
278.27	984.90	3.07
415.32	1319.76	18.18
1244.87	1601.92	173.73
10721.80	3737.69	1235.94

Table 6: No index run-time by area (milliseconds)

area (km^2)	mean	std
88.88	590.99	8.91
17.18	615.29	13.91
34.69	658.31	6.06
69.79	763.83	179.013
139.18	748.06	3.17
278.27	947.46	4.15
415.32	1283.46	17.71
1244.87	1489.41	9.25
10721.80	2557.63	174.76

Table 7: GIST run-time by density (milliseconds)

area (km^2)	mean	std
548.97	1285.11	197.21
548.97	140.48	53.97
548.97	205.09	26.94
548.97	164.31	46.09
548.97	83.19	41.86
548.97	678.15	45.73
548.97	200.37	27.71

Table 8: SP-GIST run-time by density (milliseconds

area (km^2)	mean	std
548.97	1312.60	167.16
548.97	130.09	48.88
548.97	209.37	34.27
548.97	154.04	56.62
548.97	84.61	41.91
548.97	713.86	35.21
548.97	155.98	56.75

Table 9: k-nearest POIs run-time for different areas (seconds)

count pois	mean	std
271	0.05	0.0064
248	0.09	0.00083
250	0.27	0.20645
259	0.22	0.0031
290	0.32	0.0082
257	0.67	0.1862
270	0.94	0.21
284	1.39	0.23
229	2.53	0.18

Table 10: k-nearest POIs run-time by count POIs (seconds)

count pois	mean	std
2780	1.21	0.06
1469	1.11	0.02
856	1.09	0.04
395	1.04	0.05
214	1.00	0.04
107	1.00	0.05
48	0.97	0.06
21	0.95	0.05

Table 11: Fetch Maption API run-time by area (seconds)

area (km^2)	mean	std
88.88	1.25	0.0156
17.18	1.78	0.043
34.69	3.05	0.074
69.79	3.74	0.075
139.18	7.38	2.324
278.27	10.64	0.056
415.32	15.51	0.439
1244.87	24.28	0.599
10721.80	49.22	2.659

Table 12: Fetch Overpass API run-time by area (seconds)

area (km^2)	mean	std
88.88	2.23	1.194
17.18	3.51	1.760
34.69	3.31	1.264
69.79	4.57	1.182
139.18	5.07	1.661
278.27	0.87	0.275
415.32	11.45	2.742
1244.87	20.43	5.506
10721.80	105.74	40.261

Table 13: Fetch using SQLAlchemy run-time by area (seconds)

area (km^2)	mean	std
8.88	0.65	0.028
17.18	0.71	0.024
34.60	0.89	0.028
69.79	1.09	0.167
139.18	1.49	0.142
278.27	2.33	0.076
415.32	2.95	0.113
1244.87	5.14	0.533
10721.80	9.26	0.449

Table 14: Fetch using GeoDataFrame run-time by area (seconds)

area (km^2)	mean	std
88.88	0.99	0.0814
17.18	1.30	0.0243
34.69	2.14	0.160
69.79	2.55	0.163
139.18	3.63	0.085
278.27	6.79	0.092
415.32	9.68	0.091
1244.87	15.10	0.539
10721.80	29.55	0.273

Table 15: Creating contraction hierarchies run-time by area (seconds)

area (km^2)	mean	std
88.88	0.27	0.022
17.18	0.47	0.006
34.60	0.98	0.006
69.79	1.22	0.022
139.18	1.96	0.025
278.27	4.52	0.204
415.32	6.87	0.0239
1244.87	10.67	0.246
10721.80	19.60	0.350

Table 16: Fetch Network from Maption API run-time by area (seconds)

km^2	mean	std
88.88	1.53	0.046
17.18	2.37	0.067
34.60	4.20	0.088
69.79	5.29	0.402
139.18	7.90	0.322
278.27	15.69	0.262
415.32	22.40	0.518
1244.87	34.50	0.463
10721.80	67.11	0.550

Table 17: Fetch Network from OSM using Pandana run-time by area (seconds)

km^2	mean	std
8.88	4.89	1.323
17.18	7.67	1.077
34.60	14.79	1.408
69.79	16.74	0.916
139.18	26.09	1.852
278.27	0.96	0.593
415.32	78.31	1.141
1244.87	119.54	2.029
10721.80	284.64	41.267