MASTER'S THESIS 2024

# Locally Generated Unique Identifiers for Geospatial Data

Tim Jangenfeldt


ISSN 1650-2884
LU-CS-EX: 2024-11

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-11

# Locally Generated Unique Identifiers for Geospatial Data

## Lokalt Genererade Unika Identifierare för Geospatial Data

**Tim Jangenfeldt**

# Locally Generated Unique Identifiers for Geospatial Data

## (Constructing a Hash Function for Collision Free IDs on Road Segments in OpenStreetMap)

Tim Jangenfeldt
ti8752ja-s@student.lu.se

February 26, 2024

Master's thesis work carried out at Apple Inc..

Supervisors: Patrick Cording, pcording@apple.com
Jonas Skeppstedt, jonas.skeppstedt@cs.lth.se

Examiner: Flavius Gruian, flavius.gruian@cs.lth.se

**Abstract**

This thesis tackles challenges in geospatial data identification within digital mapping applications, focusing on roads in OpenStreetMap. Traditional methods for generating unique IDs face limitations, prompting the exploration of functional ID generation using hashing algorithms. The Geohash portion of the hash, crucial for representing the geospatial position, undergoes extensive optimizations, including an elevation representation with zero bit cost. The resulting hashing function demonstrates zero collisions globally in the OpenStreetMap dataset using 71 bits. Various other optimization attempts are detailed, as well as an expanded 115 bit version exhibiting collision free IDs for all possible future roads. The study contributes insights into geospatial hashing for efficient and collision-free feature identification, offering practical implications for large-scale mapping applications.

**Keywords**: Geospatial Hashing, Collision-Free IDs, Geohash Optimization, Road Representation, OpenStreetMap, Mapping.

# Acknowledgements

I extend my sincere gratitude to my supervisor at Apple, Patrick Cording, and Vladimir Smida for introducing me to this challenging problem and offering unwavering support throughout the entire thesis process. Patrick, your boundless enthusiasm for problem-solving in the geospatial domain has been truly infectious, serving as an engineering beacon to aspire to.

I am also thankful to Jonas Skeppstedt for serving as the university supervisor and to Flavius Gruian for examining the thesis.

Lastly I would like to express my appreciation to Sophia Bladh, who endured my endless ramblings about road IDs in OpenStreetMap.

# Contents

# Chapter 1
# Introduction

This chapter serves as a means to introduce the reader to the subject. A broad background and motivation, as well as previous research, will be presented, as well as the disposition of the thesis. We will also introduce our research questions and the limitations we decided upon.

## 1.1 Background

In today's digital landscape, maps play an essential role in countless applications, from navigation and location-based services to urban planning and environmental monitoring. These maps are composed of various geographic elements, such as landmarks, roads, and buildings, which are collectively referred to as *features*. Unique identification of data is paramount for any database or collection [1].

Traditional methods for generating unique IDs, such as Universally Unique IDs (UUIDs) or a centralized service have been widely used across various domains. However, when it comes to geospatial data within maps, generating unique IDs can present challenges, especially when entities are derived from other entities, leading to requirements for idempotent data processing. UUIDs also take up a large key space (128 bits) [19] and will produce a new ID every time it is used. For geospatial data, functional IDs that are created based on properties of the features in a deterministic way would be beneficial, as such IDs could store information about the feature while also being generated in a local manner without overhead considerations.

To implement geospatial data into an ID, a natural approach would be to use some kind of hashing algorithm. Geohash [12] is one notable technique that combines these areas, designed to encode coordinates into a string of letters and digits. This allows for efficient indexing and retrieval of geospatial data based on location. This technique comes with two main limitations. The first is handling complex geometries. Complex geometries in maps can include irregularly shaped areas, intricate coastlines, and multi-polygon representations of geographic entities. This means that they are too complex to be represented by one Geohash,

as these only represent a fixed spatial bounding box [20]. Another caveat is the high risk of collisions due to Geohash only using coordinates, which are not specific enough for a digital map's features where properties like elevation need to be taken into account.

For all ID-related problems, it is critical to minimize collisions, as they can lead to data integrity issues when making the IDs non-unique [28]. Inaccurate identification can result in incorrect data analysis, misinterpretation of spatial relationships, and hinder core functionality. Therefore, developing ID generation algorithms with minimized collision probabilities is essential to ensure reliability and accuracy.

Apple Maps is one of the world's largest digital mapping applications. The organization has an interest in investigating ways of making ID generation within the map more functional and useful.

## 1.2 Motivation

As outlined in the preceding section, the proposed algorithm serves as a viable substitute for the conventionally centralized ID service or UUID within the context of large-scale mapping applications. The enormity of data inherent in these applications, often collaboratively maintained in a distributed fashion, renders a centralized service susceptible to reliability issues. In a global system, the introduction of potential latency problems to the central server becomes a notable constraint, especially in scenarios like streaming pipelines where requests are sequentially dispatched. A decentralized system, wherein IDs can be generated locally, eliminates this dependency, mitigating the risks associated with downtime and constraining latency based on the algorithm's efficiency.

In contrast, UUIDs, while leveraging their considerable size as a strength, also bear a significant size downside. Their extensive length serves to markedly diminish the likelihood of collisions through techniques like timestamps [19]. Nevertheless, it is crucial to acknowledge that the probability of a collision, while sufficiently low for almost every practical use case, is not absolute zero. The algorithm under investigation in this thesis aims to surpass this collision rate while concurrently reducing the size of each ID, enhancing manageability on a mass storage scale. This endeavor seeks to strike a balance, outperforming established collision rates while optimizing the storage footprint for practical implementation.

## 1.3 Scope & Limitations

Given the expansive scope of possible feature types within a mapping application, it became imperative to narrow our focus to a more manageable subset of data. Consequently, we chose to confine ourselves to a specific dataset extractable from the complete OpenStreetMap (OSM) data by matching the *Highway* tag [35] on OSM ways. This subset encompasses all paths from point A to B, including various road types. These range from major highways like motorways and trunks to local roads such as residential and unclassified streets. The dataset also includes special road types like service roads and tracks, as well as paths dedicated to pedestrians (*footway*) and bicycles (*cycleway*). Public transportation features like dedicated bus lanes (*busway*) are also considered. Designations for specific purposes, such as equestrian paths (*bridleway*) and staircases or steps (*steps*), are incorporated.

For simplicity, we will refer to these ways as *roads*. As of February 2024, there are roughly ten billion features in OSM [27], where one billion of these are ways. There were 224 million ways matching the *Highway* tag, meaning we limit ourselves to 2,2% of the overall data, and 22,4% of the ways.

To create a perfect or near-perfect hash function, the source domain must be known [10], and manipulations of the domain need to be made to reduce the probability of collisions. Consequently, there can be no guarantee that the resulting function will perform equally well in any given future world. Therefore, we predicate that the hash function is considered perfect if it can hash all roads in the world with zero collisions, while also not utilizing any random elements.

## 1.4 Problem Definition

In the context of the background and scope, we have formulated the following research questions:

1. What is the minimum ID bit width computed as perfect hashes in the available data?
2. How is the solution impacted by new data being added in the future?

Due to the limitation of a perfect hash being possible only for the current known dataset, as stated in *Scope & Limitations*, RQ#1 has the constraint that the function will only be perfect for the data extract used in the thesis, a snapshot of all data in OSM, taken at October 30th, 2023. Nevertheless, it is interesting to explore ways to ensure the function performs well as more road data is added, which is what we will investigate when attempting to answer RQ#2.

## 1.5 Previous Work

### 1.5.1 Geohash (2008)

Introduced by Gustavo Niemeyer [12], Geohash has become a fundamental geospatial data hashing algorithm extensively utilized in diverse applications. The Geohash algorithm, which will be described in more detail in the Theoretical Background, Section 2.1.2 operates by iteratively partitioning the global space into smaller bounding boxes, enabling the efficient encoding of two-dimensional coordinates into a single string of characters. This hierarchical structuring of data facilitates varying levels of precision, allowing for the representation of both large and small geographic regions. Notably, the implementation of Z-level curves, also known as Morton curves, enables efficient spatial indexing and proximity-based searches.

An example of an existing implementation of Geohash in OpenStreetMap are the short links [35], providing concise identifiers for specific map locations.

### 1.5.2 Vukovic's Hilbert Geohash (2016)

In 2016, Vukovic proposed an innovative adaptation of the Geohash algorithm, known as *Hilbert Geohash* [34]. This novel approach integrated the Hilbert space-filling curve to enhance the precision and efficiency of spatial indexing. Vukovic's work aimed to address the

limitations of the traditional Geohash algorithm by leveraging the spatial locality-preserving properties of the Hilbert curve. As detailed in the theoretical background (Section 2.1.2), the incorporation of the Hilbert curve ensures that nearby points in the two-dimensional space are accurately represented along the one-dimensional curve. This adaptation not only improves the accuracy of proximity searches and location-based queries but also offers an effective solution for spatial data indexing, particularly in scenarios where the conventional Geohash method falls short.

The property of increased spatial preservation means that Hilbert Geohash will be relevant in our project whenever we investigate tiles in conjunction, to increase accuracy.

### 1.5.3 Guo et al.'s Adapted Hilbert Geohash (2019)

In 2019, Guo, Xiong, Wu, Chen, and Jing introduced an enhanced version of the Hilbert Geohash algorithm, termed the Adapted Hilbert Geohash (AHG) [14]. Their research, as discussed in the theoretical background (Section 2.1.2), focused on refining the representation of complex spatial geometries within the Geohash framework. The AHG method addressed the challenges associated with encoding non-point spatial objects by introducing an adaptive coding level calculation based on the Minimum Bounding Rectangle (MBR) of the objects. The utilization of the Hilbert space-filling curve, as demonstrated by Guo et al., effectively captured the spatial position and size characteristics of 2D spatial objects with enhanced precision. Notably, the AHG algorithm demonstrated improved spatial encoding efficiency, allowing for accurate representation of complex geometries while minimizing data size requirements.

### 1.5.4 Tectonic Plate Motion and Global Datasets

The study on the effect of tectonic plate motion on georeferenced long-term global datasets [23] conducted by Mocnik and Westerholt in 2020, sheds light on the dynamic nature of geographic coordinates in the context of global datasets, such as those utilized in OpenStreetMap and other geographic databases. The research highlights how tectonic plate motion introduces a level of complexity, leading to an increasing mismatch between the actual physical locations and the associated coordinates within these datasets. The manuscript proposes strategies for systematic updates of coordinate values to mitigate the effects of tectonic plate motion on long-term datasets, aiming to preserve the accuracy and relevance of global coordinates over time. While this research presents a valid issue that is very interesting and relevant to the thesis, the solution lies beyond the scope and will be discussed further in the *Future Work* chapter.

### 1.5.5 Locality Sensitive Hashing for Polygons (2017)

Investigations into the usage of Locality Sensitive Hashing (LSH) for polygons were conducted by Gudmundsson and Pagh in 2017 [13]. In their paper, they attempt to utilize LSH, and in particular MinHash, described in detail in the *Theoretical Background* chapter to hash similar polygons to the same hash.

It would be of interest to investigate if this could be extended to roads in the geospatial domain by applying a tiling scheme to the road- and applying MinHash to group similar roads together, as we could then distinguish between these similarities with some delimiter without needing many bits to do so. The investigation of this is described in the *Discussion* chapter.

## 1.6 Disposition

This section outlines the structure of the thesis, presenting key chapters and their respective contributions.

- **Introduction** The introduction sets the stage for the thesis, establishing the problem statement, research questions, and objectives. It provides a brief overview of subsequent chapters and their roles in addressing the project's challenges.
- **Theoretical Background** Gives an introduction to the main fields of study the thesis is based on, primarily hashing and the Geohash algorithm.
- **Methodology** The methodology chapter outlines the design and implementation strategy. It details the rationale behind choosing Scala with Apache Spark, the use of cloud computing for large datasets, and the acquisition and preprocessing of data extracts from various regions.
- **Algorithm Implementation** This chapter delves into the details of the Geohash-based algorithm implementation. It discusses key components, such as considerations for angle, length, and elevation requirements, and provides a thorough explanation of the algorithm's functionality.
- **Discussion** The discussion chapter interprets the results and examines the successes and challenges encountered during the investigation process. It analyzes the implications of the proposed Geohashing algorithm in handling complexities of OpenStreetMap data and provides insights into potential applications and future research directions.

# Chapter 2
# Theoretical Background

This chapter will introduce the relevant technical background in the field. The main relevant theory relates to hashing, with most focus being on the Geohash method.

## 2.1 Hashing

Hashing serves as a fundamental concept in computer science, offering a versatile mechanism for mapping data of varying sizes into a fixed-length representation known as a hash code or hash value. At its core, hashing involves the use of deterministic mathematical functions that consistently produce the same hash value for a given input. This deterministic nature is crucial for ensuring data integrity and consistency [5], as it guarantees that identical inputs will always result in identical hash values. To illustrate a hash function, one could use a concept utilized in this thesis; the modulo operator. An example of such a hash function could be $H(i) = i\%17 + 4, i \subset \mathbb{N}$. No matter which number is entered, a number different from the input is generated, and the result will always be constant for that input. This result is called the *hash code* of the input.

In addition to being deterministic, hash functions also strive for uniformity, aiming to distribute hash values evenly across the potential output space. In an ideal scenario, each unique input should map to a distinct hash value, minimizing collisions where different inputs yield the same hash value. Achieving this property defines a hash function as a *perfect hash function*. In the mentioned simple function, input 1 and 18 would both map to the same output; 1. These inputs cause a *collision*, and the function cannot be considered perfect.

Perfect hash functions are crucial when collision avoidance is vital, such as in our geospatial data mapping application. Unlike general-purpose hash functions, which may introduce collisions due to the arbitrary nature of their outputs, a perfect hash function guarantees no collisions for a specific set of inputs. However, achieving perfection often requires knowledge about the input space [10], and perfect hashing might be challenging for dynamic or unpredictable datasets.

Randomness in hash functions, often discussed in the context of a *random oracle* model, introduces an element of unpredictability. A random oracle is a hypothetical black-box model where the hash function responds to each distinct input with a truly random output. The output is always the same for a unique input, meaning the function is still deterministic. While practical implementations of hash functions cannot be truly random oracles, the concept serves as a useful theoretical framework for discussing the desirable properties of hash functions, particularly their resistance to collision attacks.

Efficiency is a practical concern when working with large datasets. Ideally, hash functions should be computationally efficient to ensure that the hashing process does not introduce significant computational overhead. Striking a balance between uniformity, determinism, and efficiency is essential when designing hashing algorithms, especially in applications where minimizing collisions is a priority.

Hashing finds extensive use in unique identifier generation, with one application being the Universal Unique Identifier.

### 2.1.1 Universally Unique Identifiers

Universally Unique Identifiers, commonly known as UUIDs [19], are standardized identifiers used across all kinds of systems to ensure the uniqueness of entities. A UUID is a 128-bit value, typically represented as a sequence of hexadecimal characters and they offer several key properties that make them invaluable in various applications.

The primary purpose of UUIDs is to guarantee global uniqueness. UUID generation algorithms are designed to minimize the likelihood of collisions, ensuring that each generated UUID is highly unlikely to match any other UUID globally. This property is essential in distributed systems where different nodes need to generate identifiers independently.

Many UUID generation techniques incorporate elements of randomness. Randomness contributes to enhanced uniqueness, as it reduces the risk of predictable UUID patterns that could be exploited by malicious actors. In our ID generating use case, the concern for malicious actors does not exist, and as UUID implementation is based on the timestamp it was produced [19], the same UUID is not produced for a given input. This origins in the fact that the concept of an input is missing from UUID.

UUIDs find applications in a wide range of use cases. Their versatility and uniqueness make them a preferred choice for creating identifiers that need to be both globally unique and resistant to collision. In the context of this thesis, UUIDs represent a significant benchmark for the uniqueness and collision resistance required in geospatial data identification, and will be used as a benchmark for the thesis.

### 2.1.2 Geohash

The Geohash algorithm [12] is a way of hashing geospatial data. It encodes two-dimensional coordinates into a one-dimensional string of characters by continuously bisecting the global space, making each added bit target a smaller bounding box of the space. The initial steps of this bisection of the global space are illustrated in Figure 2.1. This bisection is done on the latitude and longitude separately, and the bits are then interleaved giving the bit sequence result. Figure 2.2 shows this step-by-step process of zooming into the correct point.

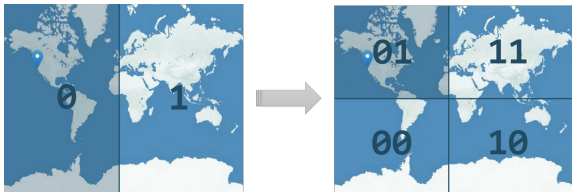



**Figure 2.1:** Illustration of the first two iterations of Geohash on the global space. Image by Ian Rees (2015) [30].

A notable result of this is the property of hierarchical structure, as Geohash provides varying levels of precision based on the length of the generated code. Longer Geohashes represent smaller geographic areas with greater accuracy, while shorter Geohashes cover larger regions with reduced precision. This flexibility allows users to balance accuracy and data size according to their specific needs. This also allows for efficient indexing, enabling systems to quickly locate and retrieve geographic data within a specified range [20].

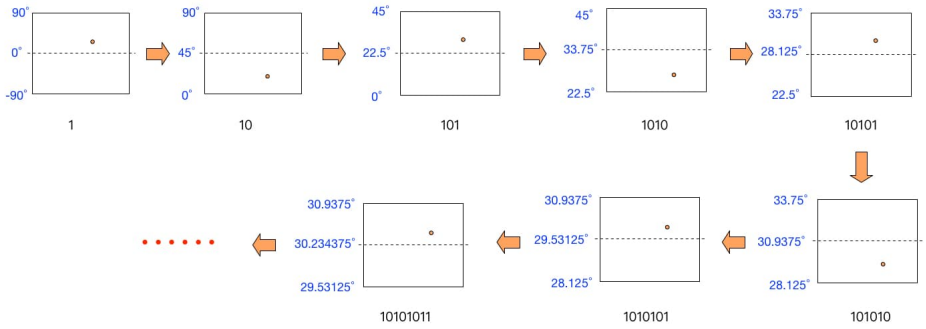



**Figure 2.2:** Bisection of the latitude space for a point located at 30.42°. Image by user Alibabatech (2018) [2].

Thanks to the hierarchical structure, Geohash has the property that two geographically close points will share the same Geohash prefix. For each matching character in the Geohash, the closer the two points are. For this reason, Geohash is commonly used in a spatial indexing context, for example, to do a proximity search. The opposite will however not always be true; two geographically close points will not always share a prefix. Geohash uses Z-level curves (also known as Morton curves) [26]. This means that the grids in the Geohash are related in a Z-like pattern, see Figure 2.3. As can be seen in the figure, points A and B are geographically fairly close, but the same cannot be said for their grid's Geohash representation.

In the context of generating IDs, this is no issue for single-point features and Geohash encoding can be applied directly to the coordinates. For linear or polygonal objects that can be represented by their minimum bounding rectangle (MBR), things become a bit more complicated. A typical approach involves encoding corners of the MBR. However, in scenarios where the MBR of an object intersects a Geohash grid border, the resulting length of the matching prefix might be insufficient for effective object localization. In Figure 2.3 A and B have no common prefixes. A real-world example of this occurs for coordinates that are on either side of the equator and prime meridian. In an attempt to improve upon these issues caused by Z-level curves, one could use another space-filling curve to construct the Geohash algorithm. One of the proposed ways for this is to use Hilbert space-filling curve.

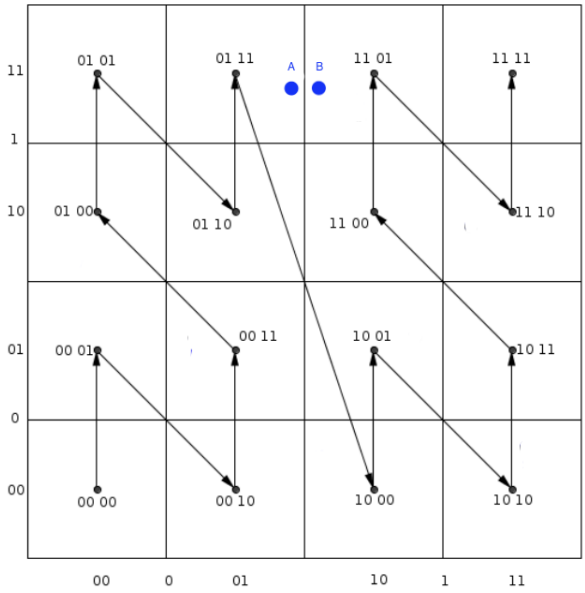



**Figure 2.3:** Second order Z-level curve relationship for 4-bit Geohash. Modified image based on one created by Vukovic [34]

## Hilbert Geohash

Hilbert Geohash is an adaptation of the Geohash algorithm proposed by Vukovic [34] in 2016. Unlike the standard Geohash, the Hilbert Geohash algorithm incorporates the use of the Hilbert space-filling curve, a type of continuous fractal space-filling curve that preserves the spatial locality of the encoded data points. The Hilbert curve improves the probability the two-dimensional space are mapped to nearby points along the one-dimensional curve [18], thus addressing some of the limitations posed by the Z-level curves used in the original Geohash algorithm.

The Hilbert curve is a space-filling curve that recursively traverses a square in a way that preserves locality better than other commonly used alternatives. By following the Hilbert curve, the algorithm effectively reduces the distortion observed in the Z-level curves, ensuring that points nearby are represented by similar codes along the curve.

Through the integration of the Hilbert curve, the Hilbert Geohash algorithm aims to enhance the accuracy of spatial indexing. The algorithm has shown to be more efficient in proximity searches and location-based queries [24], particularly in the previously mentioned scenarios, where traditional Geohash methods may fall short. Figure 2.4 illustrates the application of the Hilbert curve in the Hilbert Geohash algorithm.

Hilbert Geohash still experiences issues with representing complex geometries in the same way the traditional Geohash does. To account for this an adaptation was researched by Ning Guo, Wei Xiong, Ye Wu, Luo Chen, and Ning Jing in 2019 [14], where they proposed the Adapted Hilbert Geohash (AHG). It begins by calculating the encoding precision level based on the size of the MBR of the spatial object. This adaptation ensures that the grid size at this level is not less than the spatial extent of the MBR. Then Hilbert Geohashing is used on the centroid of the MBR. For a more detailed explanation see the flowchart Guo et al. provided in their paper. The resulting AHG code accurately reflects the spatial position of

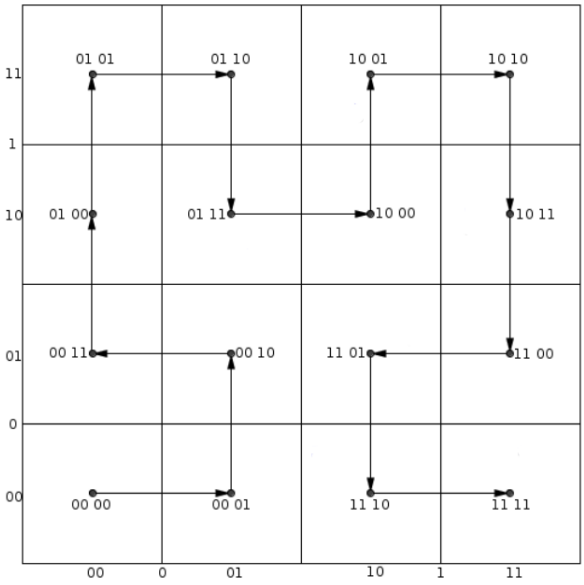



**Figure 2.4:** Third order Hilbert curve relationship for 4-bit Geohash. Modified image based on one created by Vukovic [34]

the object with adaptive precision, where the code length indicates the approximate size of the object.

In the paper, Guo et al. demonstrate the algorithm using the case of Lake Superior, the world's largest freshwater lake. The AHG method calculates the adaptive coding level to be five based on the size of the MBR. This corresponds to a binary code length of 10 bits. Utilizing the centroid point of the MBR as the encoding input and employing the Hilbert space-filling curve for traversing the 5-level grids, the resulting AHG code for Lake Superior is *0110000000*. Notably, the precision of this code level corresponds to an approximate location error of 625 kilometers, aligning with the size of the lake. This example highlights the AHG method's ability to effectively capture both the location and size characteristics of 2D spatial objects, with the very interesting side effect of using a smaller data size to do so. If we are able to encode a rough estimation of the road using AHG, it would be interesting to investigate if the remaining bits can be used to resolve potential collisions, totaling to less overall bits.

### 2.1.3 Locality Sensitive Hashing (LSH)

Locality Sensitive Hashing (LSH) [29] is a technique used in computer science and data mining for approximate similarity search in high-dimensional spaces. The main idea behind LSH is to hash similar items to the same "buckets" with high probability, enabling the identification of potentially similar items efficiently. This approach is particularly useful when traditional exact search methods become computationally expensive or infeasible due to the dimensionality of the data.

### Minhash and Jaccard Similarity

One of the foundational concepts in LSH is a min-wise independent permutation locality-sensitive hashing scheme (Minhash), which is commonly applied to estimate Jaccard similarity between sets. The Jaccard similarity coefficient measures the similarity between two sets by calculating the ratio of the size of their intersection to the size of their union. It is defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

For our thesis, Minhash was investigated as a means to identify similar roads using few bits. This proved to not be fruitful, with implementation attempts and reasons why being discussed in Section 5.1.1.

## 2.2 Haversine Formula

A widely used technique in Geographical Information Systems (GIS), particularly in navigation, is the Haversine formula, designed to compute great-circle distances on spheres such as our planet [7]. The formula is expressed in Equation 2.1.

$$\begin{aligned} a &= \sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right) \\ c &= 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \\ d &= R \cdot c \end{aligned} \tag{2.1}$$

where:

- $d$ is the distance between the two points,
- $R$ is the radius of the sphere (e.g., Earth's radius),
- $\phi_1, \phi_2$ are the latitudes of the two points in radians,
- $\lambda_1, \lambda_2$ are the longitudes of the two points in radians,
- atan2 is the arctangent function with two arguments, yielding the angle whose tangent is the quotient of the two specified numbers.

In this thesis, we utilized the Haversine formula to compute accurate length analysis on the roads in our dataset.

# Chapter 3
# Methodology

This chapter serves as a means to describe tools and the workflow for the thesis. We will also describe the relevant dataset and its structure.

## 3.1 Tools

### 3.1.1 OpenStreetMap

OpenStreetMap is an open-source mapping platform, offering a vast repository of geographic data contributed by a global community of mappers [15]. OSM's collaborative model allows individuals worldwide to contribute geographical information, making it a comprehensive and continuously evolving dataset [16].

A downside to projects with collaborative freedom is that the data cannot always be relied upon, as it is subject to bad data coming from user error, intentional or not. To prevent this, a requirement would be some kind of governing body enforcing data correctness. There is research into the detection of such errors, with an example being the work done by Svenonius of *Detecting Anomalies in OpenStreetMap Changesets using Machine Learning* [33]. There is no implementation of this built into OSM itself, but there exists different user directed data cleanup and verification projects.

OSM's comprehensive dataset and open-access nature have made it a valuable resource for various research domains [25]. The features in OSM belong to one of three buckets; nodes, ways, and relations.

#### Nodes

Nodes in OSM represent individual point features on a map as a specific latitude and longitude coordinate pair with seven digits of decimal precision. These nodes serve as the basic building blocks for defining more complex geographical structures and are often used to

represent landmarks, businesses, or specific locations of interest. They can also contain additional metadata, such as the name of a location or its descriptive tags containing contextual information associated with the geographic points.

### Ways

Ways are sequences of nodes that form linear features, such as roads, rivers, and boundaries. Ways can be either open, forming a polyline, or closed, defining an area or a polygon. This flexibility enables the representation of diverse geographical entities, ranging from highways and footpaths to administrative boundaries and water bodies.

### Relations

Relations in OSM are utilized to describe the connections and associations between multiple nodes and ways. They are handy for representing complex and non-hierarchical relationships, such as routes and multi-polygon features. Relations enable the creation of more intricate and sophisticated geographical structures within the OSM data model. For instance, they can define the spatial arrangement of multiple interconnected features, like a network of interconnected roads forming a specific transportation route [16].

## 3.1.2 OSM Tags

Apart from its fundamental elements, OSM data is enriched with an extensive tagging system that provides a detailed and comprehensive description of geographic features. The tagging system in OSM entails key-value pairs that are assigned to specific map elements, providing additional context and descriptive information about the features. These tags can range from simple attributes like names, addresses, and classifications to more intricate characteristics such as specific amenities, geographical properties, or functional attributes of the mapped entities.

## 3.1.3 Scala

Scala [32] is a statically typed programming language that combines object-oriented and functional programming concepts. It was designed by Martin Odersky to be concise, elegant, and compatible with Java, Scala has gained widespread adoption in the industry, especially in the context of big data processing [22]. Its key features include strong static typing, type inference, pattern matching, and a strong focus on immutability.

Scala's functional programming capabilities enable developers to write highly modular, reusable, and composable code [6]. Its support for higher-order functions, immutable data structures, and powerful collection libraries aid in the creation of complex data pipelines and processing workflows. Furthermore, Scala's seamless integration with Java libraries allows developers to leverage the vast ecosystem of existing Java tools and frameworks.

For the thesis, Scala was chosen for the implementation. The reason behind this is that we can easily leverage the capabilities of Apache Spark for distributed processing.

### 3.1.4 Apache Spark

Apache Spark is an open-source, distributed computing system that provides an efficient and flexible framework for large-scale data processing. Offering in-memory data processing capabilities, Spark enables the rapid execution of complex analytic tasks on large data sets [31].

Key components of Apache Spark include its resilient distributed data set (RDD) abstraction, which allows data to be stored in memory across a cluster of machines, and its high-level APIs in languages such as Scala, Java, Python, and R [11]. Spark's rich set of libraries for SQL, streaming data, machine learning, and graph processing makes it a comprehensive solution for a wide range of big data use cases.

#### Cloud Computing for Data Processing

The integration of Apache Spark with cloud computing services has revolutionized the processing of vast volumes of data. With the capability to distribute processing across a cluster of machines in the cloud, Spark can efficiently handle the scale and complexity of datasets, enabling organizations to perform data-intensive operations [3]. By leveraging cloud computing resources, organizations can harness the scalability and elasticity of cloud infrastructure to execute complex data analysis, including tasks like data cleaning, transformation, and machine learning at scale.

## 3.2 Work Process

Scala, coupled with Apache Spark, was selected as the programming language for the implementation phase. This choice was driven by the need for cloud computing support, especially crucial for handling the extensive datasets provided by OSM. We will need to process all 224 million roads in the world, and keep their hashed ID in memory in order to compute collisions, something that is a difficult task without utilizing distributed processing. The reason behind choosing the Scala version of Apache Spark over python, java and R was that we were most comfortable and experienced with it. The focus could thus be shifted to learning Spark-specific implementation details, without getting caught up on syntax and conventions.

We opted to collect smaller data extracts for individual countries, regions, and cities. This way we could run tests faster for purposes where a global run was not needed. That way, we could target specific areas displaying weak points of our algorithm, and iterate in a faster manner. Data extracts were obtained from `https://download.bbbike.org/osm/` and `http://download.geofabrik.de/`. Refer to Table 3.1 for an overview of the downloaded extracts, all in the *.osm.pbf* (protocol buffer binary format).

The implementation of the hashing function was motivated by addressing the research questions. To answer these questions, we divided this implementation into two steps. Step one aims to create a function with zero global collisions, while step two optimizes this result to achieve the smallest size possible. After these two steps, we should be able to address RQ#1 and provide us with the tools to analyze the result and its impact on handling new features added in the future, to address RQ#2.

In order to apply to the limitations discussed in Section 1.3, we also conducted a preprocessing step, where these limitations were applied to extract the 224 million roads from the

larger 10 billion feature dataset.

In constructing the hashing function, the overarching approach was to begin with adding a property associated with the road, compute the hash, and visualize the roads whose hash code collide in OSM's web API. We could then decide upon what caused the collision, modify our property, or add a new one, and keep iterating. This was repeated until we reached zero collisions on the global dataset, and the process was then restarted with the goal of optimizing the size of our properties. Our metric of success was that that no more optimization was possible, while still having zero global collisions.

**Table 3.1:** Dataset size and region for the OSM extracts.

| Region | Size |
|---|---|
| Malmö | 20MB |
| San Francisco | 22MB |
| Stockholm | 33MB |
| Copenhagen | 35MB |
| Luxembourg | 39MB |
| Madrid | 42MB |
| Switzerland | 423MB |
| Sweden | 654MB |
| Spain | 1.1GB |
| Japan | 1.8GB |
| Germany | 3.9GB |
| USA | 9.3GB |
| Asia | 12.3GB |
| Europe | 27.5GB |
| Global | 71GB |

# Chapter 4
# Algorithm Implementation

This chapter focuses on the implementation related to the road dataset. We present our method of preprocessing and go through our implementation and optimization details. See Appendix A for the source code corresponding to the sections.

## 4.1 Preprocessing

In the preprocessing step, roads were extracted from the raw data using the *Highway* tag on the OSM type way subset. Highways containing the area tag were filtered out, as represent an area surrounded by a road, such as a square [35]. We also filtered out roads that exist but only have one node, as the osm wiki documentation [35] state that these roads are corrupted data. The node references of these ways were then inner-joined from the Node subset to obtain their coordinates, with the order they appeared in the node references preserved. This preservation is necessary, as a road is a composition of road segments between two nodes. If this order is not preserved, the road is transformed into a different shape. Lastly, we also included the tags of the ways.

The processing of this data was carried out using the *osm4scala* library [4]. This library was specifically designed to parse *.osm.pbf* files into a Spark data frame, simplifying the complexity of this process into the succinct code extract $.read.format(\text{"}file.osm.pbf\text{"})$, while harnessing Spark's parallelization capabilities.

## 4.2 Creation of the Core Hashing Function

### 4.2.1 Geohash

Given the prevalent focus on Geohash in previous research, it serves as a natural baseline for our hashing function. The Geohash implementation employs a recursive binary search

method, with the coordinate range for the entire world: longitude $\in (-180, 180)$ and latitude $\in (-90, 90)$.

As mentioned in Section 2.1.2, the input for Geohash is a coordinate, meaning one latitude and one longitude value. Due to the definition of a *way* in OSM being that it is a composition of many such *nodes*, a decision needed to be made on which coordinate point should be used for the Geohash. We decided to use the middle point of the road, meaning the average latitude and longitude values of all nodes. As coordinate points in OSM are represented as 32-bit floating point numbers with seven-digit precision, we cannot describe the coordinate in more detail than exists. Thus, the maximum precision for our Geohash becomes 64 bits.

When testing this on our small datasets, it could be observed that employing Geohash at max precision, we as expected only get collisions that are nearby spatially. These collisions generally fell into one of three buckets; an intersection case (Figure 4.1), a length case (Figure 4.2) as well as an elevation case (Figure 4.3). These collisions served as the direction of what needed to be implemented next.

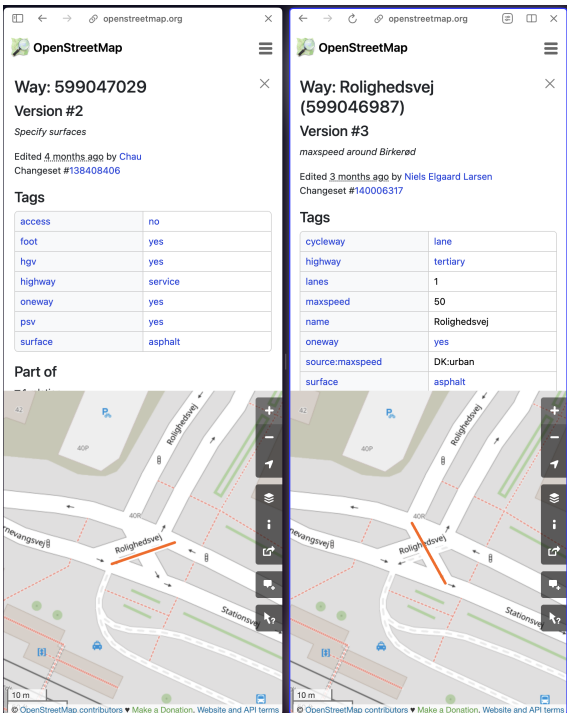



**Figure 4.1:** Observed intersection collision for high precision Geohash.

### 4.2.2 Intersection Collisions

In cases where roads intersect in an X-pattern such as in Figure 4.1, the middle points are often the same coordinate, leading to collisions for the Geohash. This was observed to occur when the only distinguishable difference between the two roads was the angle of the vector. To account for these collisions, the angle of the road vector is thus needed in the hash function. This angle was calculated between the start- and end-point of the vector, using the

Scala math library function *atan2*. The resulting angle was then encoded into a binary representation using a modulo operation with the number of desired bits B on the form $2^B$. The reason behind this choice was that if these collisions can occur for any two angles, we cannot perform better than random. As we do not want random elements in our hash function we investigated optimizations for this encoding, described in Section 4.3.1, but utilizing modulo with ten bits was seen as sufficient for step one. At ten bits no collisions remained globally, so we chose this number to represent angle.

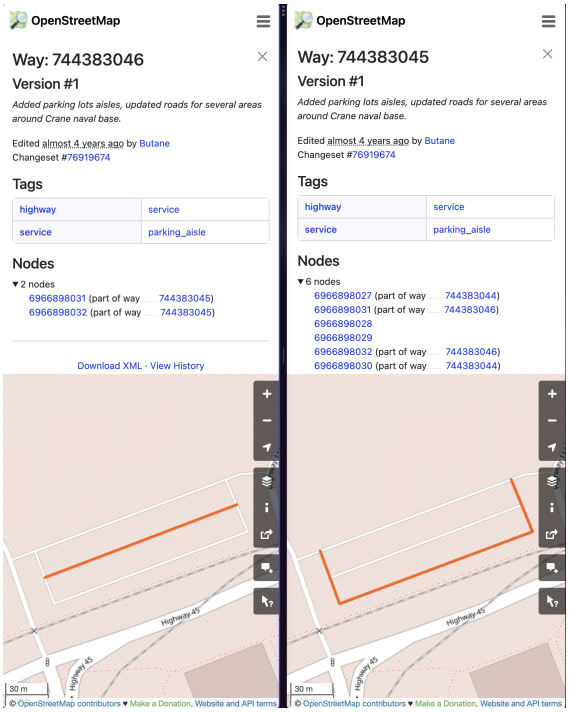



**Figure 4.2:** Observed length collision for high precision Geohash.

## 4.2.3 Length Collisions

Figure 4.2 highlights instances in the data where the Geohashed midpoint and the previously mentioned angle addition result in collisions. To address such cases, another metric becomes necessary. The observed discrepancy in the lengths of colliding roads necessitates the inclusion of length information in our hash. To incorporate length into the hash, the road's nodes are divided into distinct road segments. The start and end nodes of these segments are then utilized in the Haversine function presented in Section 2.2. The resulting lengths are summed up, and this total length is encoded into the hash using a modulo operation on the form $2^B$, akin to the procedure described in the preceding section. Once again, replacing this modulo operation was left for an optimization once we reached zero collisions globally, described in Section 4.3.2. As in the angle case, ten bits was sufficient to resolve all global collisions.

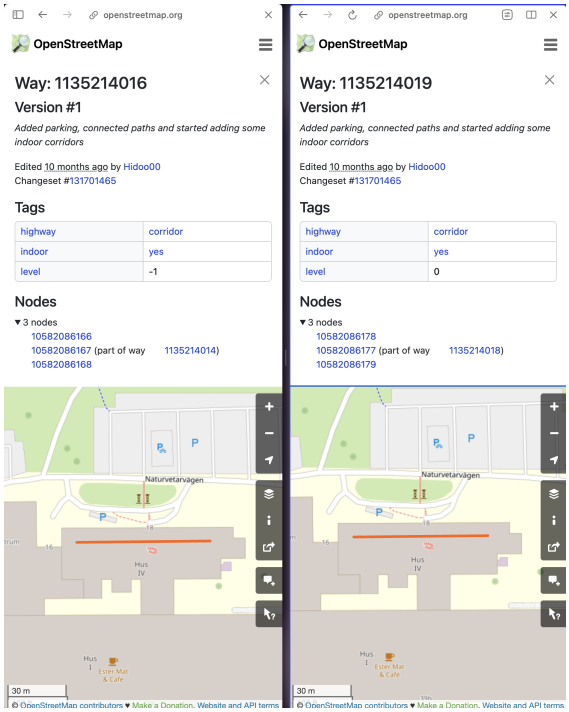



**Figure 4.3:** Observed elevation collision for high precision Geohash.

### 4.2.4 Elevation Collisions

Another collision category that emerged was elevation collisions, exemplified in Figure 4.3. This scenario depicts two roads at Lund University, sharing identical hash values for midpoint, angle, and length. The sole distinguishing factor between these roads lies in their elevation, represented by the OSM tags *layer* and *level*.

#### Layer Tag

The *layer* tag signifies elevation concerning other ways. If the tag is absent, the implicit value is zero [35]. The typical range of layer values spans from -5 to 5, represented by whole integers. For instance, a bridge positioned above a road would possess a tag with a value of one, while the road itself would have no such tag. Conversely, a tunnel situated below a road might carry a value of -1, with a subsequent tunnel underneath bearing the value -2.

#### Level Tag

Primarily utilized for indoor features [35], the *level* tag designates the floor level within a building. In the context of way features, this tag finds application in stairs or indoor corridors, similar to those depicted in Figure 4.3. A corridor in the basement of a building, for instance, could have a *level* tag with a value of -1, along with the ground floor marked as zero. The *level* tag may also denote an elevation span, using the syntax $a;b$, indicative of an ascent from level $a$ to level $b$. While fractional values such as 0.5 or 1.5 are subject to ongoing discussions [35], the permissible range for this tag remains unclear.

### Encoding Elevation

Considering the *layer* tag, a four-bit representation effectively encompasses the range $-5 \leq \text{layer} \leq 5$, justified by $2^4 > (5 - (-5) + 1)$. Conversely, determining an optimal range for the *level* tag presented challenges due to ambiguous documentation. Notably, discussions regarding the utilization of fractional values further complicated this matter [35]. In the absence of precise guidelines, empirical testing on the entire global dataset was conducted, assuming the correctness of the available data. The decision was made to hash only the "from" value in cases where the *level* tag denoted a range. Moreover, a prioritization strategy favored *level* over *layer* when both tags coexisted.

Empirical results from testing revealed that encoding values within the range $-3.0 \leq \text{level} \leq 3.0$ with a step of 0.1, along with $3.5 \leq \text{level} \leq 10$ with a step of 0.5, encompassed the existing collisions where the *level* tag was decisive. This range amounted to 72 distinct values, necessitating seven bits for their comprehensive representation.

## 4.2.5 Composition of Geohash, Angle, Length, and Elevation

Combining the four previously introduced hash components—Geohash, Angle, Length, and Elevation—we conducted a performance benchmark using the *planet.osm.pbf* dataset extract, containing 224 million roads. Utilizing maximal precision, with 64 bits for Geohash, ten bits for Angle, ten bits for Length, and seven bits for Elevation, initial observations revealed numerous collisions. In-depth analysis, facilitated by OpenStreetMap's web API, confirmed that these collisions were exclusively attributed to inaccuracies or errors in the dataset. This pervasive issue, stemming from the open-source nature of the data, will be elaborated upon in Chapter 5.

Upon filtering out collisions stemming from erroneous data, the revised hash—referred to as the *Collision-Free Complete Hash (CFCH)* achieved zero collisions globally using 91 bits. This achievement sets the stage for further enhancements, aiming to answer the research questions posed and refine CFCH into a definitive perfect hash function.

# 4.3 Optimizations for Collision-Free Complete Hash (CFCH)

This section is dedicated to exploring strategies aimed at reducing the size of CFCH while maintaining optimal collision resistance.

## 4.3.1 Angle Optimisations

The initial step in optimizing the size of any hash is to explore the removal of bits and analyze its impact on collision resistance. In the angle implementation detailed in Section 4.2.2, collisions were essentially random as long as the angles were not the exact same, resulting in collision probability performing as well as random chance. The performance for random

collisions is described as *given* $\boldsymbol{k}$ *randomly generated hashes, with* $\boldsymbol{N}$ *number of ways of describing each hash, what is the probability that two hashes will collide.*

This problem is well-documented in literature, more commonly known as the Birthday Problem [21], where $\mathbf{N}$ is equal to the days in a year (commonly set to 365), and $\mathbf{k}$ is the number of people in your dataset. The formula to compute this collision probability can be seen in Equation 4.1. For cases where $\mathbf{N}$ and $\mathbf{k}$ are large (which is often the case in hash collision scenarios), and using the fact that $1 - x \approx e^{-x}$ as $x \to 0$, the formula can be approximated to $\frac{k^2}{2N}$.

$$P(\text{collision}) = 1 - \prod_{i=0}^{k-1}\left(1 - \frac{i}{N}\right) \tag{4.1}$$

To surpass random performance, assumptions about the dataset the function will run on would need to be made and utilized in some manner. The resulting hash function would then perform better, as long as these trends are present in the dataset. As stated in Section 1.3, we have to make the assumption that these trends will remain in order to consider our function perfect or near-perfect.

Examining the intersection cases that necessitate an angle hash revealed a key characteristic; the angle between vectors in an intersection has a lower bound, denoted as *angle_lower*. If we split the angle space $0 \leq \text{angle} \leq 2 \cdot \pi$ into buckets and hash the bucket index, then two roads can only collide if their angle is sufficiently small. If bucket size $<$ angle_lower, we will not get any collisions for the entire dataset.

For the scenario in Figure 4.4, the angle between the two roads is 0.10575 radians. $\frac{2 \cdot \pi}{2^6} \approx 0.0982$, meaning that six bits would be sufficient for the two vectors to be placed into different buckets, even if one of the vectors coincides exactly with the start of a bucket interval.

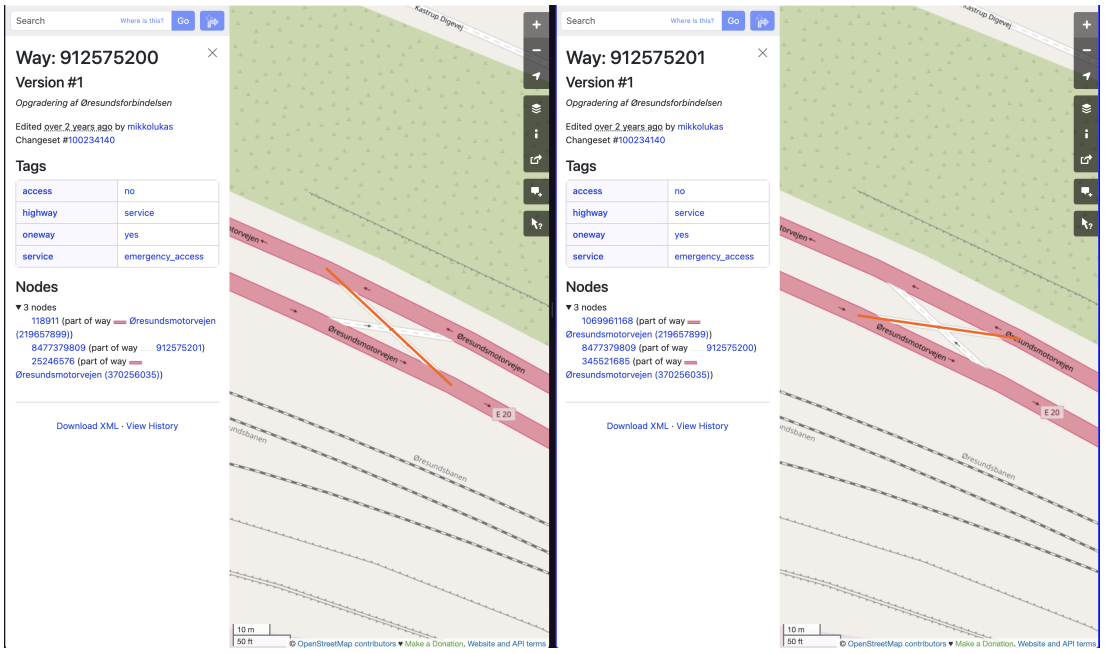



**Figure 4.4:** Intersection collision with a small angle between the road vectors.

To determine the global lower bound, we executed our hash function on the global dataset

using this new bucket method. This could be implemented using the base simple binary search function, utilizing $0 \leq \text{angle} \leq 2 \cdot \pi$ as the range. Running the job using 64 bits for Geohash, 10 bits for length, 7 bits for elevation and varying the value for angle, we managed to reach zero collisions caused by intersections at six bits.

Thus, we successfully removed four bits from the angle portion of the hash, simultaneously eliminating the random elements associated with the modulo operator and satisfying RQ#1, as we now fulfill the requirements for a perfect hash as described in the *Problem Definition* Section 1.4.

### 4.3.2 Length Optimizations

In optimizing for road lengths, our approach mirrored the one used for angle. The goal was to eliminate the random elements introduced by the modulo operation, striving to achieve a perfect hash state for the known dataset.

Implementing a bucket solution for lengths posed a more complex challenge compared to angles. In the angle scenario, the assumption that all angles were equally likely to occur, coupled with the limited range of $0 \leq \text{angle} \leq 2 \cdot \pi$, facilitated a straightforward approach. However, for road lengths, no clear upper bound existed, as roads could be arbitrarily long. The likely non-uniform distribution of lengths would result in a highly imbalanced bucket distribution based solely on length. To address this, we sought to understand the length distribution in the current dataset, recognizing the need to make reasonable assumptions to create a hash function close to perfection.

To determine the bucket distribution, we analyzed the lengths of the approximately 224 million roads in the global dataset. The resulting rough distribution is depicted in Figure 4.5. Notably, longer roads were less frequent in the dataset. To implement the same bucket (binary search) methodology employed for angles, a pre-processing step was necessary to determine which length values should belong to each bucket.

In the pre-processing step, roads were sorted based on their lengths. For $b$ bits, the sorted list was divided into $b$ equally large segments, and the head value for each sub-sequence was extracted. The resulting list's initial element was substituted with zero, and $\infty$ was appended as an upper bound beyond the last value. These intervals then constituted the range. The list was cached, and in the hashing pipeline, the bucket index could be determined by searching for the index satisfying list[index] $\leq$ length $<$ list[index+1], a process implemented using binary search.

Executing this pre-processing job with various bit counts revealed that at six bits we had resolved most collisions caused by length. The remaining collisions shared a common characteristic—they all involved circular roads where the sole distinguishable difference was the circumference. An illustration of one such collision is presented in Figure 4.6. Distinguishing between these cases necessitated additional bits in the length hash. However, given that all these collisions involved circular roads with identical start and end points, the angle hash contributed no additional information. This insight led to the optimization that when the road's start and end points coincide, all angle bits could be utilized for extra detail in length. With six angle bits and six length bits, this translated to an increase from $2^6 = 64$ buckets to $2^{12} = 4096$ buckets. This adjustment proved more than adequate to address circumference collisions without requiring extra bits.

When reducing the size to five bits, a pattern emerged among the introduced collisions—they

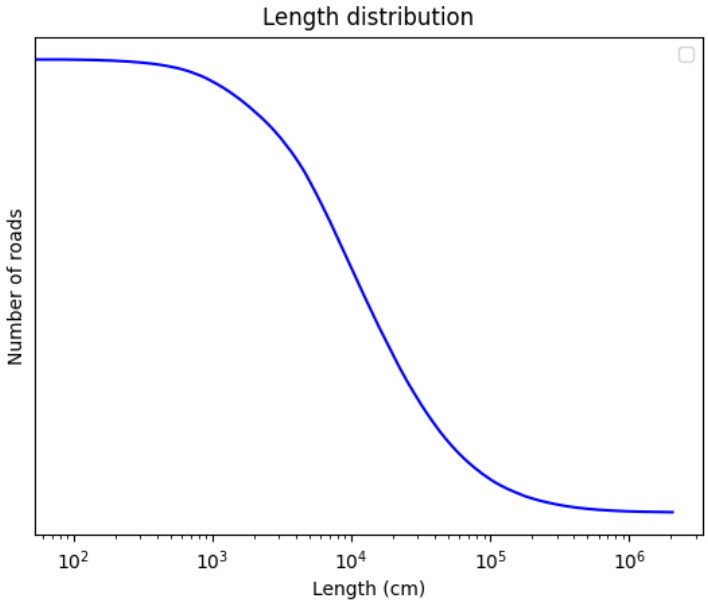



**Figure 4.5:** Rough distribution of road lengths in the global data set in cm, as the shortest roads were only centimeters long.

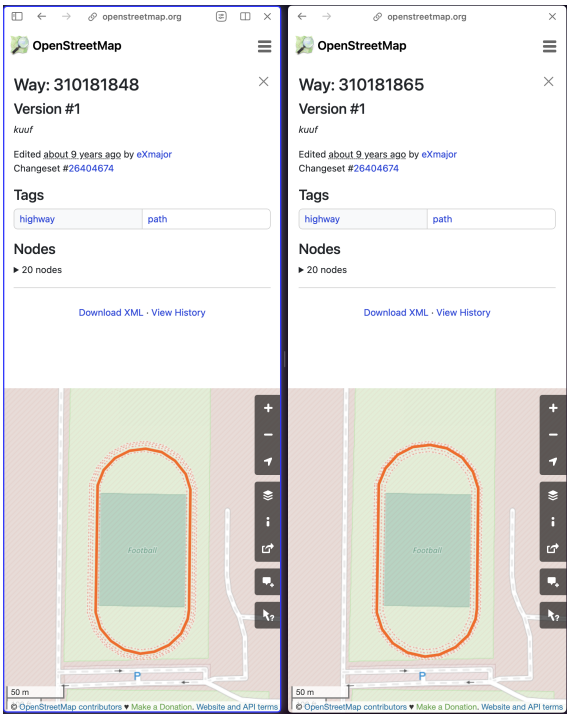



**Figure 4.6:** Circumference collision for six bit length optimisation

formed V-shaped structures, differing in the length between the start and end points, as visualized in Figure 4.7. Similar to the case with the circumference, where the angle had no impact, we leveraged this observation to re-purpose angle bits for encoding length. The condition was set that the length from the start to the end node needed to be shorter than the

length from the start to the middle-most node. This optimization proved successful, and we managed to resolve all length collisions using five bits.

As was the case in angle, this optimization was based upon the assumption that the length distribution trend stays relevant in the data. If this distribution changes in the future, our function is less accurate and becomes more prone to collisions.

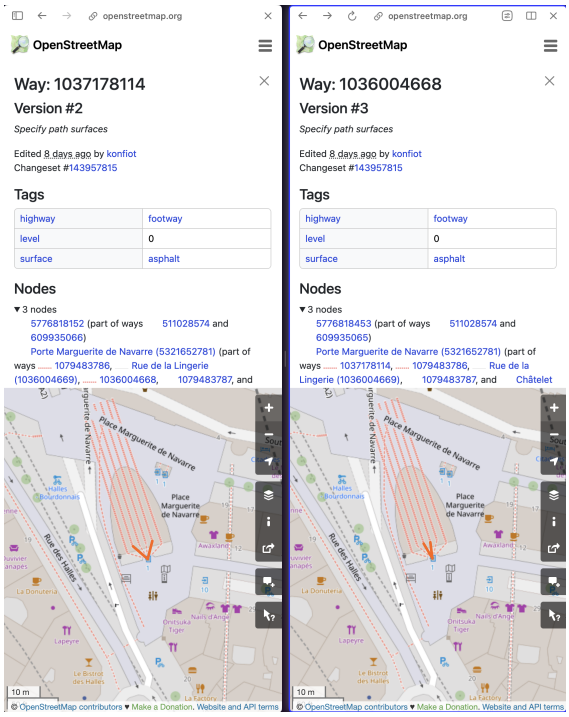



**Figure 4.7:** Very similar roads, sharing the same middle point

### 4.3.3 Elevation Optimisations

The elevation hash presented a unique characteristic compared to other components: only roads containing elevation information utilized this portion of the hash. Given that this subset of roads was relatively small within the entire dataset, the seven bits allocated for elevation were largely unused by the majority of roads. Consequently, the investigation aimed to integrate these elevation bits into the broader hash without significant loss of accuracy.

A key insight emerged from the fact that, in cases involving elevation, collisions occurred only among features that were identical except for elevation elements, thanks to the optimizations for angle and length. When constructing the Geohash portion, it was decided to run Geohash on the midpoint of the road, representing the average latitude and longitude values of all nodes. Realizing that the point where Geohash is executed can be shifted, the method of shifting can convey information. This shift, if unique for a given elevation value, eliminates collisions for features with identical geospatial information, except for elevation.

The initial concept involved using seven fewer bits for Geohash when a feature contained elevation data. This would result in a Geohash tile $2^7$ times larger, where each small tile, originally used for Geohash precision, could be assigned a static value mapped to an elevation value. One value was reserved to handle collisions between the Geohash point of the

elevation-lacking feature, which still used full Geohash precision. This collision possibility was mitigated by checking if the reserved elevation tile matched what would be computed without elevation. If so, the reserved tile was used. Visualization of this concept is presented in Figure 4.8, where four bits are used for clarity. The red tile represents the full precision Geohash tile if the feature lacked elevation. All tiles except for the reserved green tile are mapped to an elevation value. For the elevation value mapped to 1001, a collision is found when compared to the full precision tile. In this case, the green tile is chosen to resolve the collision.

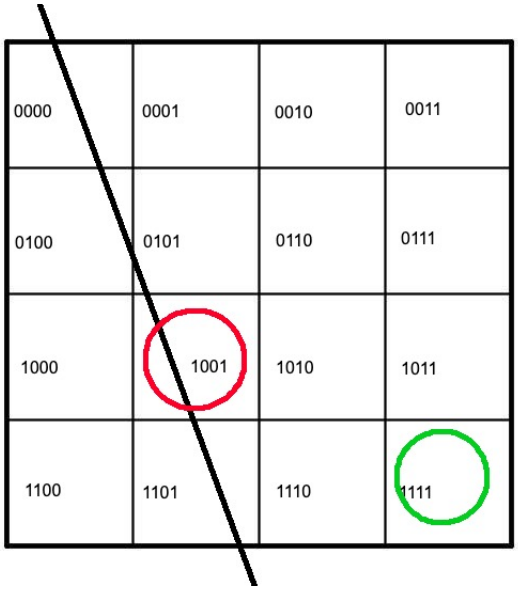



**Figure 4.8:** 4-bit elevation-Geohash replacement

Implementation attempts revealed that seven bits introduced a slightly higher error, causing collisions with nearby features. Figure 4.8 provides an intuitive understanding of this issue when looking at the edge tile 0011 which is located fairly far away from the road vector. To address this, the values were encoded into different four bit geohash tiles, shifted along the road vector. Elevation collision candidates, having the same node coordinates but different elevation values, allowed for shifting the midpoint along the polyline into the next Geohash tile. Multiple four-bit tiles shifted k-tiles from the midpoint along the vector were employed to cover all required elevation values. To fill the 72 necessary values, a total of five four-bit tiles were needed: one at the actual midpoint and two shifted ones in both directions along the vector. This process is visualized in Figure 4.9, where the original midpoint is represented in red, along with the first two shifted four-bit Geohash grids based on the yellow tiles, located one Geohash diagonal away along the road vector.

Performance measurements employing these optimizations demonstrated the elimination of all elevation-related collisions without adding any extra bits to the composite hash. Once again this optmization heavily relies on the trend for how elevated roads appear in the current data, and is prone to collisions if this trend changes.

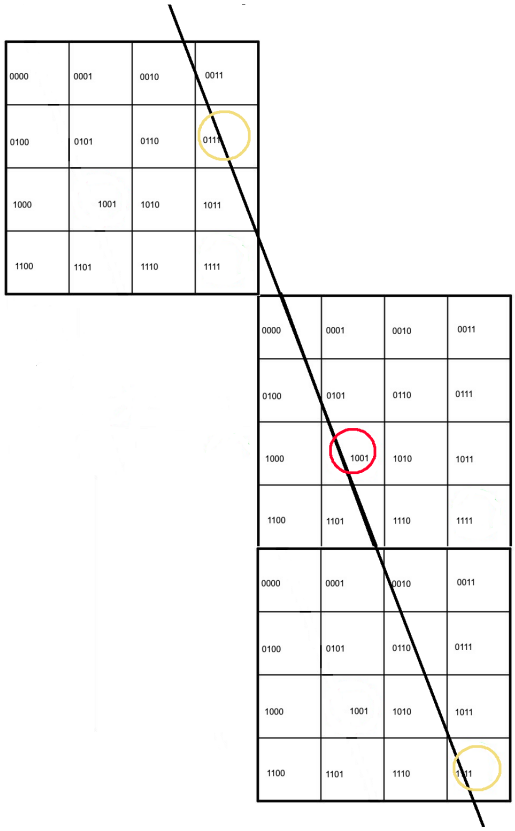



**Figure 4.9:** Midpoint shifted 4-bit elevation-Geohash replacements

### 4.3.4 Geohash Improvements

The Geohash component of the hash structure holds paramount importance due to its substantial size, making it crucial to explore optimization avenues. As discussed earlier, reducing precision in Geohash may lead to collisions, especially in scenarios where features demand higher precision for accurate differentiation. An illustrative collision example is presented in Figure 4.10, depicting a bus stop where the sidewalk and platform closely align with identical angle, length, and elevation.

Given these challenges, the focus shifted towards optimizing the leading portion of the Geohash. Considering the geographical distribution of roads, particularly in urban areas, roads are not uniformly distributed across the globe. For instance, large portions of the Pacific Ocean have minimal road presence, while cities exhibit dense road networks. Consequently, the initial 64 bits of our hash are sparsely distributed in most regions but highly clustered in specific areas.

As previously mentioned, while the global road distribution might evolve, creating a perfect hash for every conceivable scenario remained an insurmountable task. Leveraging this uneven distribution to our advantage, we had the idea of implementing hash tables. The idea involved dynamically allocating hash bits based on the local feature density. For instance, in sparsely populated regions like the Pacific Ocean, we could allocate a smaller percentage of the hash space, optimizing the representation for the actual feature distribution rather than adhering to a fixed global allocation. In practice, a manual determination of coordinate ranges with sparse road distribution was impractical and challenging to define

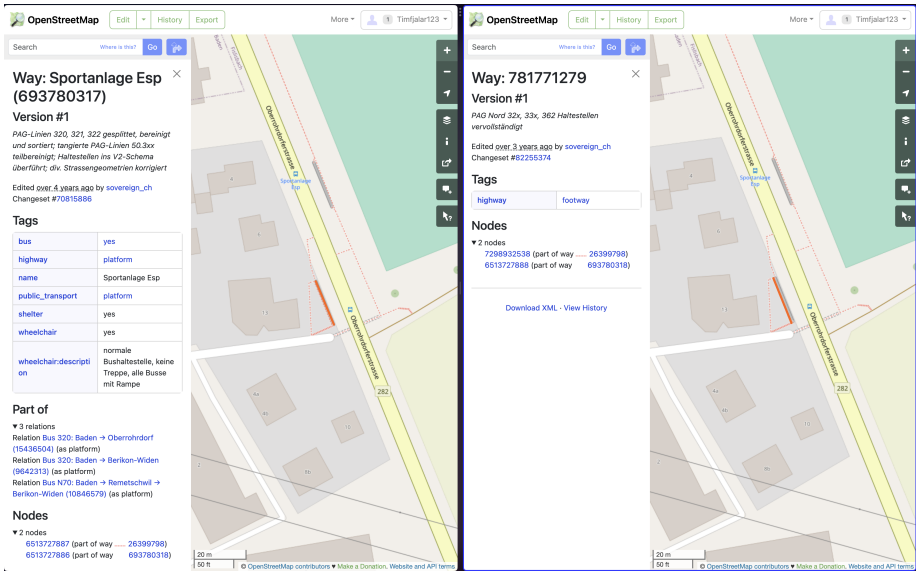



**Figure 4.10:** Example highlighting the need for high-precision Geohash to distinguish features

comprehensively.

To gauge the density-entropy of sparse road distributions, we devised a strategy by hashing all roads to a lower precision and measuring the density-entropy of each corresponding tile. In alignment with RQ#2, we also extended this to consider nearby tiles likely to experience increased density in the future, especially those adjacent to dense tiles. To measure the density of tiles at an even lower Geohash precision, we again ran the same job assigning each road to its Geohash tile at the lower precision. This essentially meant zooming out to ascertain if larger tiles exhibited higher density proportions. We utilized Geohash Hilbert for this, as it has the property of preserving a better locality than regular Geohash, something we discussed in Section 2.1.2.

This pre-processing job involved encoding indices for tiles with entropy above a defined threshold. Sparse non-empty tiles below the threshold are placed in a separate sparse bucket, and each is assigned a specific hash value. Subsequently, these indices are encoded using the minimum required bits. For new hashes with a distribution of zero during pre-processing, it will be assigned the first available index. In the case that the world grows beyond the spaces that fit within the size we have chosen, a random assignment of one of the values within the sparse distribution is made. Although this introduces a collision risk for the first part of the Geohash until the pre-processing is rerun, the shared tile's sparsity significantly mitigates the collision risk.

We chose a precision of 28 bits for the small tile measurement, providing a maximum latitude/longitude error of 1221 m at the equator [17]. For larger tiles, a precision of 24 bits, equivalent to an error of 4886.496 m at the equator, was chosen. We set the density thresholds at 10 for small tiles and 320 for larger tiles. The reason behind these choices was that any tile below 10 (small) or 320 (large) at 1221 m or 4886 m respectively we determined to be urban. We used an OR check on whether the tile's density exceeds these values to determine whether a tile should be marked as sparse or dense. The results are presented in Table 4.1. Utilizing 24 bits to assign indices to all 12615718 populated tiles yields an additional $2^{24} - 12615718 =$

4161498 unused reserved tiles for future pre-processing runs. Once these are filled, a small risk of a collision is introduced.

**Table 4.1:** Number of 20-bit tiles matching the predicate.

| Predicate | Number of matching tiles |
|---|---|
| Global tiles | 268.435.456 |
| Populated tiles | 12.615.718 |
| Sparse tiles | 8.945.804 |
| Dense tiles | 3.669.914 |

## 4.4 Runtime Analysis

As described in Section 1.2, a key advantage of avoiding a centralized service is the elimination of latency associated with service requests. The extent of this benefit, however, hinges on the efficiency of the proposed replacement algorithm. Therefore, it was imperative to assess the efficiency through a runtime analysis.

Firstly, the time complexity of the four hash properties was analyzed to identify its upper bound. While generic binary search algorithms exhibit a logarithmic time complexity proportional to the size of the array [9], our algorithm's recursive depth is limited to the number of bits used to represent the hash. Consequently, the time complexity for the binary searches is O(precision), where precision is a constant. The Hilbert Geohash function also only depends on the static precision parameter, and thus also performs in constant time.

Although the binary search for length index is constant, the calculation of this length is not. The haversine function runs at constant runtime, but as we calculate the length as the composite of the length between nodes, we need to run haversine $N - 1$ times, where N is the number of nodes in the road segment. Thus the time complexity becomes O(N) for this part.

Since time complexity is decided by the worst case, our composite algorithm runs at O(N), where N is the number of nodes in the road. Experiments conducted on an Apple M2 Max with 64 GB RAM, validated this observation. The results are illustrated in Table 4.2 For a road having two nodes, the time complexity reduces down to constant, and our algorithm could compute its hash in a mere 0.3 ms. For the upper bound, a test using a road with 270 nodes yielded a runtime of 7.3 ms.

**Table 4.2:** Runtime for roads of differing lengths.

| Number of nodes | Runtime (ms) |
|---|---|
| 2 | 0.3 |
| 270 | 7.3 |

# Chapter 5
# Discussion

This chapter will analyse and discuss the results from the previous section, aiming to address RQ#2. Additional implementation ideas that were unfruitful will be discussed, and we will also present ideas for future research directions on the subject.

## 5.1 Evaluation of Results

In Chapter Four, we successfully developed a hashing function that achieved zero collisions in the global OpenStreetMap data set. The initial iteration utilized 91 bits, and through subsequent improvements, we managed to reduce this to 71 bits without incorporating any random elements. A summary of these optimizations can be seen in Table 5.1 While this reduction is noteworthy, common data type representations often align with powers of two. Therefore, to represent an ID exceeding 64 bits, a 128-bit alternative is typically employed. We propose a way to work around this would be to store the ID as a combination of one 64-bit part, and one eight-bit part totalling 72 bits. Nevertheless, an exploration of possibilities to further reduce the size below 64 bits was seen as necessary, which will be detailed in the following section.

**Table 5.1:** Summary of results from the optimizations to the core algorithm.

| Subset of hash | CFCH | Optimized CFCH |
|---|---|---|
| Geohash | 64 | 60 |
| Length | 10 | 5 |
| Angle | 10 | 6 |
| Elevation | 7 | 0 |
| Total | 91 | 71 |

In Section 4.4 we analysed the time complexity of our algorithm- and ran experiments to verify our hypothesis regarding the runtime. In Figure 4.5 we could see that long roads are uncommon in the data, and as this is the slowest part of our algorithm, most roads will be computed fast. A side-note that needs to be taken into account is that our experiments were made on a powerful machine, and may thus not give an accurate representation of all potential users. A centralized service also has the benefit of very easily handling batch requests. The algorithm we proposed was implemented in a sequential fashion, so a concurrent alternative may be of interest to investigate if a user struggles with performance.

The obtained results are compared with the alternatives discussed in Section 1.2, illustrated in Figure 5.1. This also includes a depiction of the original CFCH without optimizations, alongside an extended CFCH designed to almost guarantee uniqueness for all conceivable future roads. The upper bound for the latter, totaling 115 bits, was estimated by considering various factors:

- Length range: $0 \leq \text{length} \leq 2.043.071$ (max length found in data), with a step of $0.1$ cm, requiring $25$ bits.
- Angle range: $0 \leq \text{angle} \leq 2 \cdot \pi$ radians, with a step of $0.0001$, demanding $16$ bits.
- Geohash set to $64$ bits, representing maximum precision.
- Elevation range: $-100 \leq \text{elevation} \leq 100$, with a step of $0.1$, necessitating $10$ bits.

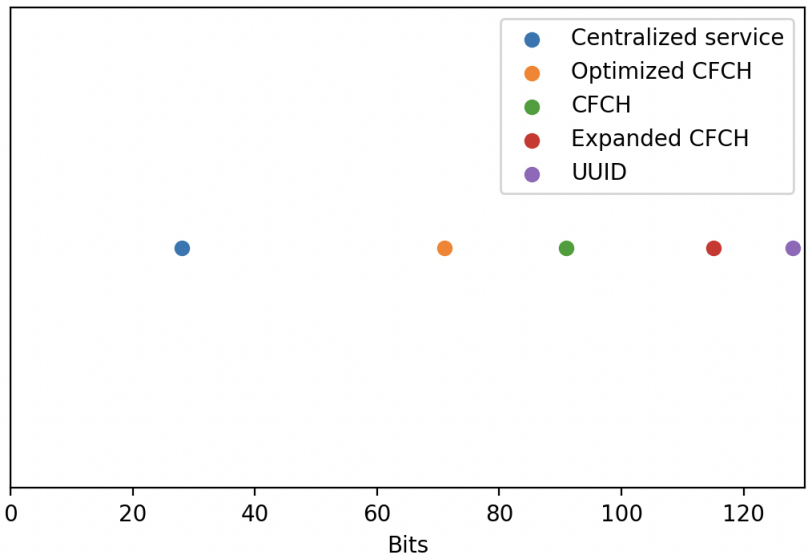



**Figure 5.1:** Number of bits required for each ID representation convention.

In contrast, a centralized service only needs to convey the cardinality of the set, equivalent to the current number of elements in the data. For the utilized dataset, encompassing 224 million features, a mere 28 bits suffice today. The graphical representation underscores that our proposed solution achieves a smaller size than UUID while avoiding a dependency on a centralized service, as detailed in Section 1.2. Notably, it also possesses the unique capability to detect duplicates at creation, further explored in subsequent sections.

An intriguing revelation from the graph is the size of the expanded CFCH, approaching a near guarantee of uniqueness for all future roads, while not exceeding the size of the 128 bit alternative, UUID.

### 5.1.1 Explorations of Further Improvements

This section serves as a means to describe attempts to further optimize the function, mainly through the exploration of the technologies described in *Previous Work* (Section 1.5).

#### Locality-Sensitive Hashing (LSH) Approach

As discussed in the *Previous Work* chapter, an intriguing approach involves leveraging the clustering properties of Locality-Sensitive Hashing (LSH). The proposed method integrates MinHash with $k$ different randomly chosen hash functions applied to road features. The minimum value across these hash functions is compiled into a $k$-length hash. Thanks to the characteristics of MinHash, the resulting hash exhibits the property that more similar roads yield more similar hash values. To represent roads, a Geohash tiling approach with lower precision was employed. This involved interpolating between each pair of nodes within a road to ensure that even short roads with two nodes received multiple tiles.

Each hash function was executed on each Geohash tile associated with a road, and the minimum value was extracted. Utilizing a Java library based on the fast non-cryptographic XXHash algorithm [8], a seed parameter allowed using the same $k$-number hash functions for every road. The code implementation can be found in Appendix A.

To resolve collisions introduced by the LSH approach, the same resolution strategy used in the original implementation was applied: appending the tail bits from the Geo-elevation-hash. With this strategy, all collisions resulting from LSH, with parameters $k = 48$ and Geo-elevation-hash precision 8, were successfully resolved, achieving 0 collisions globally using 56 bits.

Despite this achievement, the optimization was deemed unsuitable due to an increase in false positives when reducing precision further. False positives refer to roads that, by chance, map to the same hash value but are not similar in any capacity. Drawing parallels to the birthday problem described in Section 4.3.1, the formula for computing the collision probability given $k$ and $N$ was adapted to estimate the number of collisions for the same parameters. The Scala code implementation for this calculation can be found in Listing 5.1. This same formula applies to the randomness introduced with false positives in LSH.

This highlights that using less bits than required to accurately distinguish between collision candidates introduces a random risk of a collision. As these false positive collisions are purely random, they do not perform better than a random oracle and because of this we discarded the strategy.

**Listing 5.1:** Expected number of collisions for a random hash function

```
def expectedCollisions(k: Int, N: Int): BigDecimal = {
  val denominator = BigDecimal(2) * BigDecimal(2).pow(N)
  BigDecimal(k) * BigDecimal(k) / denominator
}
```

### Large Drop of Head in GeoHash

Another approach considered was to drop a significant prefix of the GeoHash. This would result in all roads sharing a broader location on Earth. The assumption here was that collisions would be exceedingly rare since coinciding factors, such as location within this tile, angle, and length, would all need to align for a collision to occur. However, upon implementation, it became evident that, once again, the random nature of these parameters would lead to a probability akin to the Locality-Sensitive Hashing (LSH) attempts—essentially resembling a perfectly random function.

It is worth noting that roads within a tile still cannot collide. Thus, while there is a marginal improvement over random hashing, the difference is not significant given the volume of available data.

### Application of Guo et al.'s Adapted Hilbert Geohash (AHG)

Attempts were made to apply the findings by Guo et al. The idea was that while the application of AHG (Angle Histogram Grid) might introduce more collisions for roads sharing the same Minimum Bounding Rectangle (MBR), it could potentially allow us to use fewer bits than those stripped off during the process, thus resolving collisions more efficiently than increasing Geohash precision. However, this approach encountered two main challenges. Firstly, some road segments were exceptionally short, necessitating a relatively high GeoHash precision to capture them accurately. Consequently, the MBR of these short segments did not significantly exceed the maximum precision Geohash tile, limiting the number of bits that could be saved for this edge case. Suppose bits were used to resolve collisions for roads with a large MBR. In that case, it might inadvertently introduce collisions with shorter roads that coincidentally share the resulting hash as a Geohash prefix. Consequently, we opted not to further explore this approach in our algorithm.

## 5.1.2 Geohash Optimization Challenges

The most intricate optimization lies within the Geohash component of the hash. To leverage the inherent likelihood that certain regions of the world contain fewer roads and to allocate less space for these areas, we had to make a decision regarding a cutoff precision. Higher precision allowed for more space reduction but risked overfitting the current dataset. We settled on 28 bits, as the error at that precision is at the scale of a city. Additionally, we introduced the larger tile at 24 bits to check if our tile was close to a populated one, anticipating that cities tend to grow, and most, if not all, new roads would connect to existing ones. While it is possible to increase this precision much further and create a nearly perfect hash mapping with each 64-precision Geohash tile having a reserved index, doing so would guarantee collisions with new additions to the map. This extreme scenario would significantly reduce the size of the Geohash portion of our composite hash but could introduce issues with new map additions.

Another consideration when increasing precision too much is the storage space required for this Geohash map. Creating an artifact hundreds of gigabytes large for this purpose may not be desirable.

Determining the size of the Geohash prefix from 28 to 24 bits is straightforward in C

using the built-in function **sizeof()**. Scala lacks an equivalent function due to the properties of the JVM and object references in Java-based languages, making it challenging to obtain an exact answer. However, an estimation for code without references can be obtained using the code shown in Listing 5.2. Running this on the Global dataset, we estimate the mapping artifact to be five GB for the current extract, with an average of 344 bytes per key-value pair. Due to the large size, it would be beneficial to investigate the compression of this artifact in a manner where you could still obtain the value for a key without decompressing the entire artifact. This is left as future work potential.

**Listing 5.2:** Measurement of mapping size

```
val denseSparseToIndexRDD = denseRoadMappingsRDD ++ sparseRoadMappingsRDD
val baos = new ByteArrayOutputStream()
val oos = new ObjectOutputStream(baos)
oos.writeObject(denseSparseToIndexRDD.collect().toMap)
oos.close();
val size = baos.size()
```

Another concern arises when adding roads to a previously unused tile after exhausting all indices in 24 bits. For instance, in the scenario where a new city is built in the middle of a desert, our implementation randomly assigns a sparse tile. While the collision risk within this tile is minimal, it does introduce potential collision risks in the future. In cases where such occurrences are unacceptable, adjustments may be necessary, such as increasing the precision to 25 bits or reconsidering this optimization altogether.

### 5.1.3 Churn Issues

A noteworthy consideration when employing this hashing function is the potential churn issues it may introduce. Given that the IDs are intricately tied to the road layout, modifying the structure of a road will result in an ID change. This behavior is generally undesirable for stable IDs. To address this, modifications might be necessary, indicating the use of the old version's ID when, for instance, extending a road by a few meters. However, as the ID is assigned at creation, it could be argued that any change warrants the consideration of a new road altogether. The resolution of this issue depends on one's definition of what constitutes a road or road segment.

### 5.1.4 Usage in Validation of Duplicate Roads

As outlined in Section 3.1.1, OpenStreetMap lacks proper validation and detection mechanisms for incorrect data. An intriguing aspect of our algorithm is its potential to identify actual data discrepancies. During our verification process, we observed that a significant portion of the detected collisions corresponded to genuine instances of incorrect duplicates in the data, an example of which can be seen in Figure 5.2. These were roads that were identical in geospatial location, and elevation, and were candidates for removal upon discovery. A majority of the duplicate ways identified early in the project have since been deleted by users attempting to rectify the map, such as the one in Figure 5.3. We were also able to catch a lot of cases where the roads were not duplicates, but the data did not follow the correct conventions, especially in the tags where there is no restriction on the text you can enter.

Executing the batch job developed in this thesis would provide an efficient means to uncover existing incorrect data in the dataset. Furthermore, it could serve as a future verification tool, notifying editors when a newly created road receives the same ID as an existing one, prompting them to discard the change.

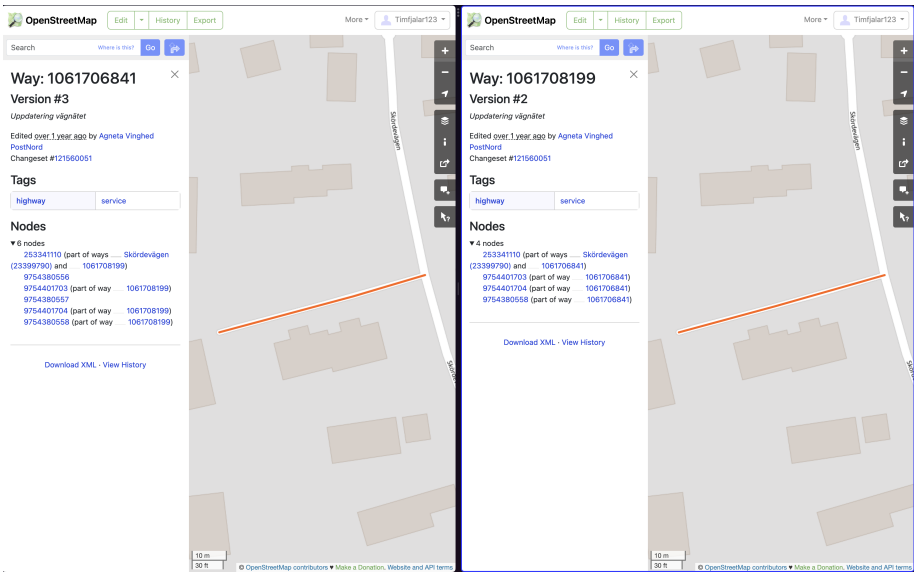



**Figure 5.2:** Duplicate road example

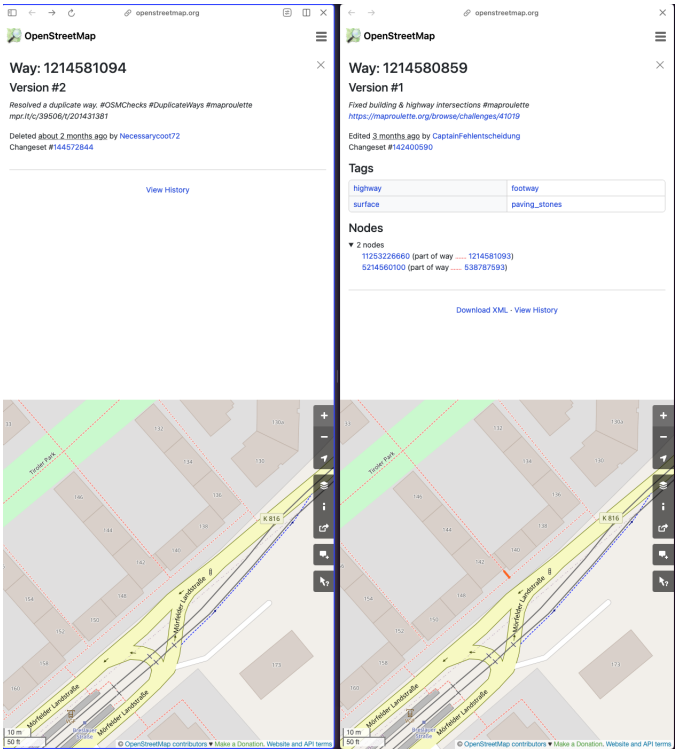



**Figure 5.3:** Deleted duplicate road as part of cleanup project

## 5.2 Answering the Problem Statement

### 5.2.1 Research Question One

Our first research question, *What is the minimum ID bit width computed as perfect hashes in the available data?* was the primary focus of Chapter 4. We successfully reduced the size to 71 bits, an improvement of $2^{20}$ times from the initial 91 bit implementation inhibiting the zero-collision property. This accomplishment was achieved by eliminating all random elements, effectively satisfying RQ#1. In this *Discussion* chapter, we also deliberated on additional attempts to further minimize the size and provided reasons for discarding certain approaches.

### 5.2.2 Research Question Two

Research question two, *How is the solution impacted by new data being added in the future?*, was primarily tackled by avoiding a significant performance decrease for new data under the assumption that it follows the existing data trends, a large focus in Section 4.3.1 and 4.3.2. As outlined in Section 1.3, predicting future data patterns is inherently uncertain. Nevertheless, by constraining optimizations that overly tailor to the current dataset, we aim to maintain comparable performance with future data. We also discussed an expanded CFCH in Section 5.1, which provides immense collision resistance while still not exceeding UUID in size.

## 5.3 Future Work

### 5.3.1 Expansion to Other Feature Types

An evident area for extending this research involves adapting the concept of ID generation based on feature properties to different feature types. To narrow the thesis scope, our focus was exclusively on roads, serving as a sort of "proof of concept." However, exploring the extension of this approach to all features in OSM is of interest. Nodes should be straightforward, and represented by a single point, while other types may pose more challenges.

Concerning ways, as mentioned in the *Preprocessing* chapter, we didn't analyze roads with the area tag. The reason is that they resemble buildings, also represented in ways, more than roads do. This is because they describe the area within the polyline, not the polyline itself. Extending the algorithm to include buildings should address these cases, and matching on feature type could determine the appropriate algorithm for each feature.

Relations, being the most complex structure in OSM, may prove to be the most challenging for adaptation. However, since OSM uses different ID namespaces for nodes, ways, and relations (meaning the same ID could represent one way and a completely different node), it would be possible to implement this hashing technique on a subset, such as only nodes and ways.

### 5.3.2 Choice of Point to Run Geohash On

As detailed in Section 4.2.1, the decision to use the midpoint defined by the average latitude/longitude values for running Geohash was not arbitrary. Several other options were considered before settling on the midpoint. Initial attempts involved using either the start or end nodes, but this proved problematic in scenarios such as the one illustrated in Figure 5.4, where roads with very similar characteristics shared the same start and endpoint. Another option, the middle node, also encountered issues, as demonstrated in Figure 4.7. Despite observing the most success using the average latitude/longitude point, determining the optimality of this choice remains challenging and is a subject for potential exploration in future work.

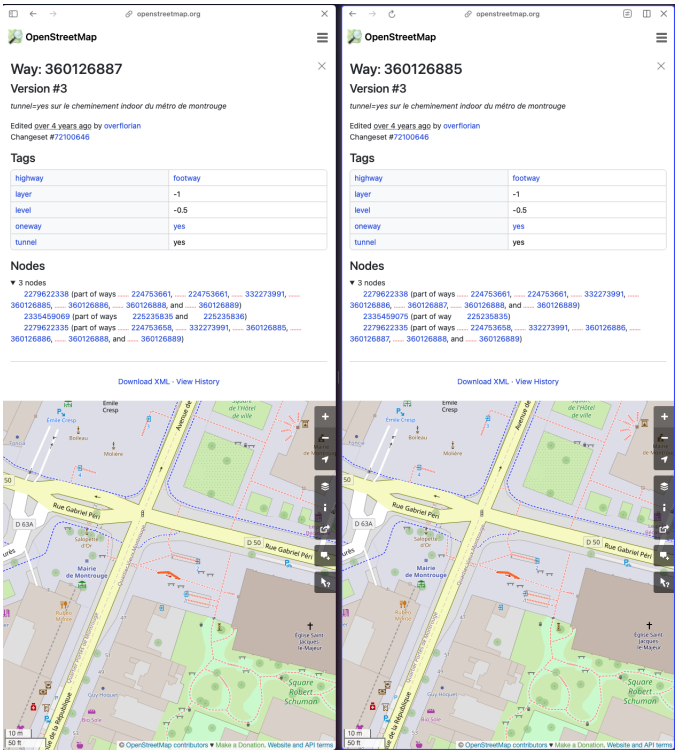



**Figure 5.4:** Very similar roads, sharing the same start and endpoint

### 5.3.3 Resolution of Small Changes in Feature Properties

As mentioned earlier, addressing the churn in ID generation, where the ID is closely tied to the object itself, is essential. As discussed in Section 1.5, a noteworthy problem arises from the change in coordinates of all OSM features worldwide due to tectonic plate motion, as outlined by Mocnik and Westerholt. Earthquakes pose an even more significant challenge, as coordinates can shift by whole meters from one day to the next. Resolution of this issue is deferred to future work, as OSM currently lacks support for batch updates to features to accommodate such changes, a topic also acknowledged by Mocnik and Westerholt in their paper.

### 5.3.4 Handling of Full Dataspace in Geohash Optimisation

As discussed earlier, further work is needed to refine the handling of Geohash optimization, specifically the mapping of 28 precision tiles to indices. Key limitations include determining a suitable bit cutoff to prevent overflow in the future or establishing a robust strategy for handling such overflow. The precision itself warrants additional analysis, as it hinges on the desire to align with the current world, where full precision would achieve a perfect fit (perfect hash) for the existing data set. Additionally, the storage of the mapping table should be compressed effectively to ensure that the artifact remains of manageable size for users.

# Chapter 6
# Conclusion

In this study, we successfully addressed the challenges of generating unique identifiers for geospatial features in OpenStreetMap. Our hashing algorithm, designed to minimize identifier size while ensuring zero collisions, achieved a significant improvement by reducing the size from the initial 91 bits to 71 bits, which can be stored in one 64-bit datatype and one 8-bit datatype. The elimination of random elements and incorporation of geospatial features, including angles, lengths, and elevations, played a pivotal role in achieving uniqueness.

Our deterministic hashing function ensures consistent hash generation, and the use of a perfect hash function enhances collision avoidance, a crucial feature in geospatial data mapping applications. Our optimized 71 bit function operated under assumptions of existing trends in the available current data, but we also presented a 115 bit extended version that would work for all possible future worlds as well.

Future work should explore the extension of the algorithm to other OSM feature types, refine the handling of small changes in feature properties, and address potential churn issues due to coordinate shifts. Further optimization of Geohash mapping, including strategies for overflow scenarios, can enhance the algorithm's scalability.

# References


[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[2] Alibabatech. Putting China's Second-Hand Economy on the Map with GeoHash Matching. `https://hackernoon.com/putting-chinas-second-hand-economy-on-the-map-with-geohash-matching-f6eb7626ff96`, 2018. Accessed: January 2024.

[3] Ameen Alkasem, Hongwei Liu, Decheng Zuo, and Basheer Algarash. Cloud computing: A model construct of real-time monitoring for big dataset analytics using apache spark. In *Journal of Physics: Conference Series*, volume 933. IOP Publishing, 2017.

[4] Angel Cervera Claudio. osm4scala. `https://simplexspatial.github.io/osm4scala/`, Accessed: October 2023.

[5] Muhammad Rehan Anwar, Desy Apriani, and Irsa Rizkita Adianita. Hash algorithm in verification of certificate data integrity and security. *Aptisi Transactions on Technopreneurship (ATT)*, 3(2):181–188, 2021.

[6] Paul Chiusano and Runar Bjarnason. *Functional programming in Scala*. Simon and Schuster, 2014.

[7] Nitin R Chopde and Mangesh Nichat. Landmark based shortest path detection by using a* and haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering*, 1(2):298–302, 2013.

[8] Yann Collet. xxHash. `https://xxhash.com/`. Accessed: January 2024.

[9] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[10] Zbigniew J Czech, George Havas, and Bohdan S Majewski. Perfect hashing. *Theoretical Computer Science*, 182(1-2):1–143, 1997.

[11] The Apache Software Foundation. Apache Spark. `https://spark.apache.org/`. Accessed: October 2023.

[12] geohash.org. Geohash.org. `http://geohash.org`, Accessed October 2023.

[13] Joachim Gudmundsson and Rasmus Pagh. Range-efficient consistent sampling and locality-sensitive hashing for polygons. In *Proceedings of 28th International Symposium on Algorithms and Computation (ISAAC 2017)*, Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, 2017.

[14] Ning Guo, Wei Xiong, Ye Wu, Luo Chen, and Ning Jing. A geographic meshing and coding method based on adaptive hilbert-geohash. *IEEE Access*, 7:39815–39825, 2019.

[15] Muki Haklay. Openstreetmap: User-generated street maps. *PPGIS*, 2008, 2008.

[16] Muki Haklay and Patrick Weber. How good is volunteered geographical information? a comparative study of openstreetmap and ordnance survey datasets. *Environment and Planning B: Planning and Design*, 37(4):682–703, 2010.

[17] Tammo Ippen. geohash-hilbert. `https://pypi.org/project/geohash-hilbert/`, 2020. Accessed: January 2024.

[18] Jonathan K. Lawder and Peter J. H. King. Querying multi-dimensional data indexed using the hilbert space-filling curve. *ACM Sigmod Record*, 30(1):19–24, 2001.

[19] P. Leach, M. Mealling, and R. Salz. A universally unique identifier (uuid) urn namespace. Technical Report RFC 4122, Network Working Group, July 2005. Request for Comments.

[20] Jiajun Liu, Haoran Li, Yong Gao, Hao Yu, and Dan Jiang. A geohash-based index for spatial data management in distributed memory. In *2014 22nd International Conference on Geoinformatics*, pages 1–4, 2014.

[21] Earl H McKinney. Generalized birthday problem. *The American Mathematical Monthly*, 73(4):385–387, 1966.

[22] John A Miller, Casey Bowman, Vishnu Gowda Harish, and Shannon Quinn. Open source big data analytics frameworks written in scala. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 389–393. IEEE, 2016.

[23] Franz-Benjamin Mocnik and René Westerholt. The effect of tectonic plate motion on georeferenced long-term global datasets. *International Journal of Applied Earth Observation and Geoinformation*, 94:102183, 2021.

[24] B. Moon, H.V. Jagadish, C. Faloutsos, and J.H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[25] Peter Mooney, Marco Minghini, et al. A review of openstreetmap data. *Mapping and the citizen sensor*, pages 37–59, 2017.

[26] G. M. Morton. A computer oriented geodetic data base; and a new technique in file sequencing. *Phys. Plasmas*, 24(7):159–173, 1966.

[27] OpenStreetMap Foundation. OpenStreetMap Taginfo - Database statistics. `https://taginfo.openstreetmap.org/reports/database_statistics`, ongoing. Accessed: February 2024.

[28] N. Paskin. Toward unique identifiers. *Proceedings of the IEEE*, 87(7):1208–1227, 1999.

[29] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern recognition letters*, 31(11):1348–1358, 2010.

[30] Ian Rees. Geohashes and You. `https://www.mapzen.com/blog/geohashes-and-you/`, 2015. Accessed: January 2024.

[31] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1:145–164, 2016.

[32] Scala Contributors. Scala Programming Language. `https://www.scala-lang.org/`. Accessed: February 2024.

[33] Svenonius, Dan. Detecting Anomalies in OpenStreetMap Changesets using Machine Learning. Master's thesis, Lund University, 2023.

[34] Tibor Vukovic. Hilbert-geohash-hashing geographical point data using the hilbert space-filling curve. Master's thesis, NTNU, 2016.

[35] OpenStreetMap Wiki. Openstreetmap wiki,. `https://wiki.openstreetmap.org/`, 2023. Accessed October 2023 Pages: [Key:layer, Key:level, Key:area, Key:highway, Way, Shortlink].

# Appendices

# Appendix A
# Source code

Github repository documenting the source code can be found at `https://github.com/Timfjalar/thesis_tim/tree/main`

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2024-02-22

**EXAMENSARBETE** Locally Generated Unique Identifiers for Geospatial Data
**STUDENT** Tim Jangenfeldt
**HANDLEDARE** Patrick Cording (Apple Inc.) Jonas Skeppstedt (LTH)
**EXAMINATOR** Flavius Gruian (LTH)

# Hur mycket information behövs för att unikt identifiera alla världens vägar?

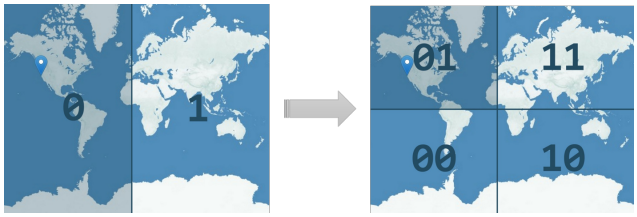
POPULÄRVETENSKAPLIG SAMMANFATTNING **Tim Jangenfeldt**

I vårt arbete presenterar vi en geografisk algoritm som genererar IDn på vägar i OpenStreetMap utifrån vägens utformning. Med en kombination av geografiskt läge, längd och vägvinkel lyckades vi unikt identifiera 224 miljoner vägar med 71 bitar.

Ponera att du vill komma på ett sätt att en människa ett helt unikt förnamn, så när du ser en ny person vet du direkt vad den heter. Du har ej vetskap om vad andra människor heter, och du vill använda så få bokstäver som möjligt för att uppnå detta. Din uppgift blir att komma på en optimal kombination av egenskaper hos personen som tillsammans är helt säregen. Eftersom alfabetet har 29 bokstäver och jordens befolkning ligger på ca åtta miljarder kan du uppnå detta med minimalt sju bokstäver teoretiskt ($29^7 > 8md$), men i verkligheten blir det mer komplicerat. Du kan exempelvis välja att representera personens ögonfärg med första positionen, längd med nästa, vikt med en tredje och så vidare. Du kanske också upptäcker regler runt namngivningen såsom att fler än två av samma bokstav inte får förekomma i följd. Problemet blir snabbt komplext.

I detta examensarbete försöker vi uppnå detta på alla världens vägsegment i karttjänsten OpenStreetMap, och vårt resulterande ID består av ettor och nollor (bitar) istället för bokstäver. På så sätt slipper man beroende av en central enhet som håller koll på vilka ID som använts hittils. Andra metoder metoder såsom Universally Unique Identifier (UUID) förlitar sig på sannolikhet, där ID är unika till följd av storleken. Vårt ID beror inte på slumpmässiga element på samma sätt, och vi föreslår två storlekar; 71 samt 115 bitar, där den större hanterar möjliga framtida scenarion bättre. Vi fann även att det kan användas för att snabbt verifiera om en kopia till en väg som läggs till i kartan redan existerar, då dessa får samma ID.

Egenskapen där mest antal bitar behövdes var geografiskt läge. Här användes en befintlig algoritm Geohash, där man kan beskriva koordinater med rutor vars storlek beror på hur många bitar som används. Eftersom världen är ofantligt stor behövs mycket information för att beskriva en 2D punkt ner till centimeternivå.




Vi lyckades namnge varenda väg med 71 bitar, utan att inkludera slumpmässiga element. Det visar sig även att vår metod är snabb på att generera enskilda ID, vilket är det primära användningsområdet. Vi föreslår därmed en lösning som är effektiv och kan användas för att upptäcka felaktig data, samtidigt som vi slipper beroende av en central enhet som håller koll på vilka ID som använts.