# Putting the car on the map: A scalable map matching system for the Open Source Community

Sebastian Mattheis,
Kazi Khaled Al-Zahid, Birgit Engelmann, Andreas Hildisch,
Stefan Holder, Olexiy Lazarevych, Daniel Mohr, Felix Sedlmeier,
Richard Zinck

BMW Car IT GmbH - Munich Lab
Petuelring 116
80809 Munich
sebastian.mattheis@bmw-carit.de

**Abstract:** Recent years have seen a proliferation of mobile devices connected to the internet, including connected cars. These systems send a stream of position annotated messages when requesting location based services. The position information in turn can be used to improve those services. Here, we focus on online map matching of the most recent position of a connected vehicle on a road map. This information, aggregated and privacy aware, can serve as a basis e.g. for machine learning algorithms used to improve traffic prediction. We describe a system for online map matching in the backend that implements a state of the art algorithm based on a Hidden Markov Model. This system uses only open source software and open data. The development of the map matcher was motivated by a perceived lack of a scalable system in the open source realm. We discuss its role as part of a scalable backend system designed to provide spatially aware services.

## 1 Introduction

Connected mobile devices have become commonplace, reaching from smartphones and tablets to cars and watches. These devices use their connection to the internet to provide various services such as navigation, traffic alerts, shopping, restaurant reviews and other points of interest (POI) information. Many of these applications need to know the location of the user in to provide the best service. Hence, several studies have been published that evaluate movement patterns in cities from pedestrians to cyclists and taxis ([RWFP06, LLC+14, KSR13, LZL+12]). Here, our focus is on cars as connected devices. In contrast to pedestrians and cyclists, cars are bound in their movement to road networks. In many cases, it is not necessary to know the exact location of a car in order to provide a service, e.g. pharmacies close by can be listed with only a rough position estimate of the car. Nevertheless, in order to receive relevant traffic information, it is essential to know a drivers exact location including the position on a road segment. Crucially, such information can be used to assess the current traffic situation at a specific location. In

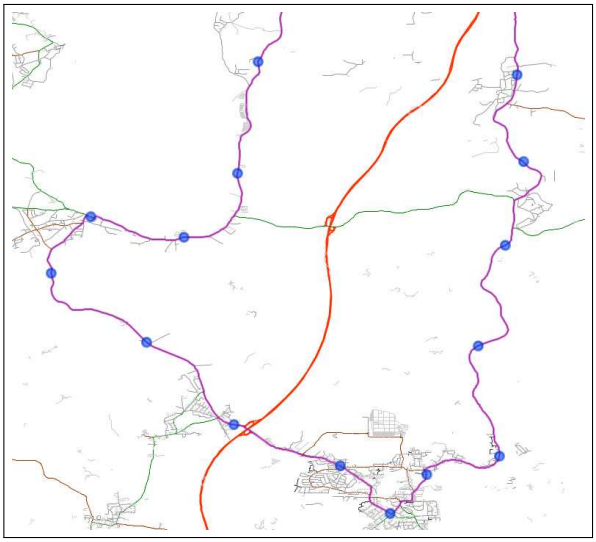

Figure 1: Shows the map matched path of a vehicle (blue dots connected by purple line). The road map is compiled from OSM data. Red color indicates a primary road, green color secondary roads.

such examples, noisy location information, provided e.g. by GPS measurement, must be mapped onto the most likely position on a road network. This problem is referred to as map matching.

Here, we describe a system for online map matching in the backend. It is part of a scalable backend system designed to provide spatially aware online services. The development of this system was motivated by a perceived lack of a scalable map matcher in the Open Source realm. The main contribution is the map matcher[1] that is to be released to the Open Source community. Section 2 gives an introduction to the algorithm we used for map matching. Section 3 presents the architecture of the map matcher and discusses options for scaling. Finally, we discuss future steps.

## 2 Foundations

### 2.1 Map matching

A trajectory is defined here as the path of a vehicle on a road network. The objective of trajectory map matching is to estimate this path from noisy position data (e.g. from GPS). The road map represents the topology of the road network and, since every point is localized, it also provides a geometric representation. The vertices of the road topology correspond to intersections. Edges represent end-to-end connections between intersections, also referred

[1] https://github.com/bmwcarit/barefoot

to as road segments. Since we define edges to be unidirectional, a bidirectional road segment is represented by two edges with opposite directions. The vehicle's path is sampled in a sequence of position measurements $(z_0, \ldots, z_T)$ during a time interval $[0, 1, \ldots, T]$. Each position measurement $z_t$ with $t \in [0, T]$ corresponds to a position $s_t$ on the map, i.e. a position on an edge of the road topology. Since position measurements $z_t$ are subject to measurement errors, they can be matched to more than one map position. The set of possible map positions for each measurement $z_t$ is denoted as the set of position candidates $\mathcal{S}_t$.

Map matching can be modeled as a probabilistic problem. It corresponds to finding the most likely sequence of system states in a Markov chain [RN99].

**Definition 1. Trajectory map matching** is defined as finding the most likely sequence of position candidates $\tilde{P} = (s_0, \ldots, s_T)$ with

$$\tilde{\mathcal{P}} = \underset{s_0,\ldots,s_T}{\arg\max}\, p(s_0, \ldots, s_T | z_0, \ldots, z_T), \tag{1}$$

where $s_t \in \mathcal{S}_t$ with $t \in [0, T]$. The probability that the sequence $(s_0, \ldots, s_T)$ corresponds to the actual trajectory of the vehicle, given the measurements $(z_0, \ldots, z_T)$, is denoted as $p(s_0, \ldots, s_T | z_0, \ldots, z_T)$.

Definition 1 restricts trajectory map matching to finding a sequence of map positions only. It ignores the vehicle's path between map positions. Nevertheless, the path gives evidence on the likeliness of a position sequence e.g. a shorter path is more likely than a longer one. This is considered in the probability $p(s_0, \ldots, s_T | z_0, \ldots, z_T)$. However, identifying a path requires route assumptions for interpolating the vehicle's trajectory. This is left to a router that can be configured to use an arbitrary routing cost function. The router takes a tuple of map positions $(s_{t-1}, s_t)$ with $t \in [1, T]$ and returns the path from position $s_{t-1}$ to position $s_t$, denoted as $\langle s_{t-1}, s_t \rangle$, with minimum costs according to the chosen cost function. A path is a sequence of edges of the road topology. The obtained paths are then assessed with a probability and can be considered in the overall solution of the map matching problem.

Trajectory map matching is computationally expensive, especially if the number of position measurements is large. However, many online services require a position estimate $s_t$ only for the most recent measurement $z_t$. In a Markov chain, this corresponds to system state filtering [RN99].

**Definition 2. Online map matching** is defined as finding the most likely map position $\tilde{s}_t$ with

$$\tilde{s}_t = \underset{s_t}{\arg\max}\, p(s_t | z_0, \ldots, z_t), \tag{2}$$

where $s_t \in \mathcal{S}_t$ and $t \in [0, T]$. The probability that the position candidate $s_t$ corresponds to the actual position of the vehicle on the road network, provided the measurements up to the most recent $(z_0, \ldots, z_t)$, is denoted as $p(s_t | z_0, \ldots, z_t)$.

Our focus lies on online map matching as it is required by many online services.

### 2.2 Hidden Markov Model map matching

Hidden Markov Model map matching has been established as state of the art method for both offline [NK09] and online map matching [GDM+12]. The first-order Hidden Markov Model (HMM) defines a system's behavior over time as a sequence of system states $(s_0, \ldots, s_T)$. They are referred to as hidden since the actual system states can only be observed by a sequence of emissions $(z_0, \ldots, z_T)$ [RN99]. As illustrated in Figure 2, each emission $z_t$ is the result of an observation (wavy arrows) of the system's state $s_t$ at time $t \in [0, T]$. The state $s_t$ at time $t$ is one of many possible states in a finite set of state candidates $\mathcal{S}_t$. Since observations are subject to measurement noise, their uncertainty can be modeled by an emission probability $p(z_t|s_t)$. The state transitions, depicted as straight arrows, are part of a stochastic process with transition probability $p(s_t|s_{t-1})$.

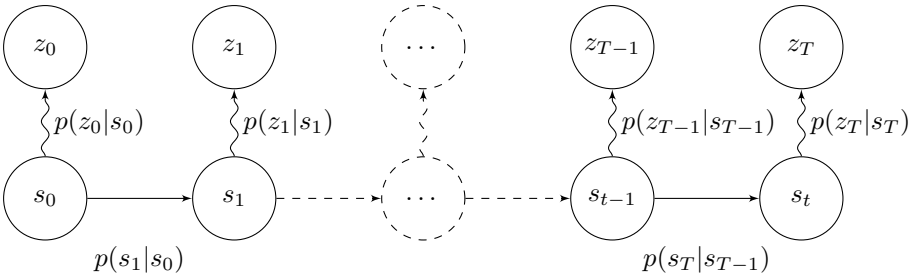


Figure 2: First-order Hidden Markov Model for trajectory map matching. Each emission $z_t$ is the result of an observation (wavy arrows) of the system's state $s_t$ at time $t \in [0, T]$. The state transitions (straight arrows) are part of a stochastic process with transition probability $p(s_t|s_{t-1})$. A state refers to a map position of a vehicle and emissions refer to position measurements. The transition between system states corresponds to the path between map positions.

The first-order HMM makes the following assumptions about state transitions and emissions [RN99]:

- Markov state assumption: The system state $s_t$ at time $t \in [1, T]$ depends only on the previous state $s_{t-1}$ such that $p(s_t|s_0, \ldots, s_{t-1}) = p(s_t|s_{t-1})$.
- Markov sensor assumption: The probability of emission $z_t$ at time $t \in [0, T]$ depends only on the current state $s_t$ such that $p(z_t|z_0, \ldots, z_{t-1}, s_0, \ldots, s_t) = p(z_t|s_t)$.

In the context of map matching, a state refers to a map position of a vehicle and emissions refer to position measurements. The transition between system states corresponds to the path between map positions. The solution to the online map matching problem follows a recursive approach of state filtering [RN99]:

**Theorem 1.** The most likely map position $\tilde{s}_t$ corresponding to a position measurement $z_t$ can be deduced from Definition 2 given that

$$p(s_t|z_0 \ldots z_t) = \alpha \cdot p(z_t|s_t) \cdot \sum_{s_{t-1}}^{\mathcal{S}_{t-1}} p(s_t|s_{t-1}) \cdot p(s_{t-1}|z_0 \ldots z_{t-1}), \tag{3}$$

where $p(s_t|z_0 \dots z_t)$ is the probability of position candidate $s_t \in \mathcal{S}_t$, given the position measurements $(z_0, \dots, z_t)$. Further, $p(s_{t-1}|z_0, \dots, z_{t-1})$ is the probability of position candidate $s_{t-1} \in \mathcal{S}_{t-1}$, given the position measurements $(z_0, \dots, z_{t-1})$. The emission probability of position candidate $s_t$ is $p(z_t|s_t)$ and transition probability from position $s_{t-1}$ to position $s_t$ is $p(s_t|s_{t-1})$, and $\alpha$ is a normalizing constant. The probability of initial position candidates $s_0 \in \mathcal{S}_0$ is

$$p(s_0|z_0) = \alpha \cdot p(z_0|s_0). \tag{4}$$

Both equations (3 and 4) are derived using Bayes' rule.

### 2.3 Online map matching algorithm

Theorem 1 provides a recursive solution to the online map matching problem (Definition 2). It can be implemented as an online algorithm that iteratively determines the best estimate of the vehicle's map position $\tilde{s}_t$. Each position measurement $z_t$ triggers an iteration of the algorithm and updates the position estimate. The implementation of the algorithm is straight-forward. It remains to provide robust strategies for the selection of position candidates and the determination of their emission and transition probabilities. The strategies of our implementation are mainly adopted from [NK09] which are similar to those used by [LZZ$^+$09, GDM$^+$12]. It identifies map position candidates $\mathcal{S}_t$ as points on a road segment that have minimum geodesic distance to the measurement $z_t$. The road segments must overlap with a geodesic circle around the position measurement $z_t$. The emission probabilities $p(z_t|s_t)$ are defined as a Gaussian distribution

$$p(z_t|s_t) \sim \frac{1}{\sqrt{2\pi\sigma_z^2}} \exp\left\{-\frac{||z_t - s_t||^2}{2\sigma_z^2}\right\}, \tag{5}$$

where $\sigma_z$ is the standard deviation of GPS measurements and $||z_t - s_t||$ is the geodesic distance between map location $s_t$ and measurement $z_t$. The determination of transition probabilities requires routing from position candidate $s_{t-1}$ to $s_t$. The obtained path is denoted as $\langle s_{t-1}, s_t \rangle$ and has length $|\langle s_{t-1}, s_t \rangle|$. In [NK09], transition probabilities have been experimentally determined to fit a negative exponential distribution

$$p(s_t|s_{t-1}) \sim \lambda \exp\Big\{\lambda \left(||z_t - z_{t-1}|| - |\langle s_t - s_{t-1} \rangle|\right)\Big\}, \tag{6}$$

where it remains to find the best parameterization (estimate of $\lambda$) for a specific sampling set, i.e. the sequence of position measurements.

# 3 The system architecture

## 3.1 The online map-matching system

The online map-matching system processes a stream of position measurements $(z_0, \ldots, z_t)$ in order to determine a vehicle's most likely position $\tilde{s}_t$ on the map that corresponds to the most recent position measurement $z_t$. Figure 3 depicts the high level architecture of our system. It consists of an online map matching component (Filter), a geometrical road map (Map), a routing unit with a separate road map topology (Router) and a memory unit that provides access to state information of the tracked vehicle (State).

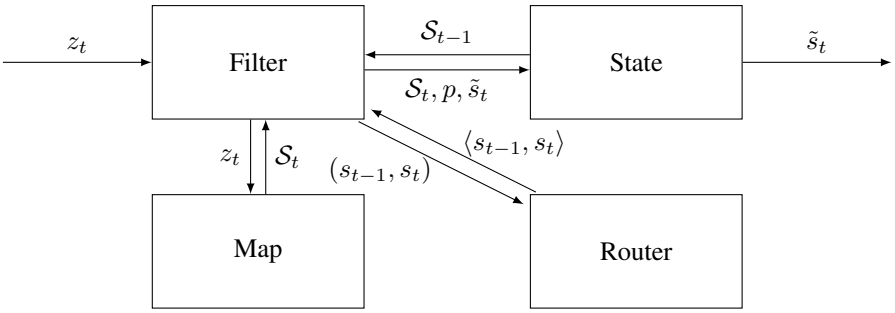


Figure 3: Architecture of a HMM map matching system consisting of an online map matching component (Filter), a geometrical road map (Map), a routing unit with a separate road map topology (Router) and a memory unit that provides access to state information of tracked vehicle (State).

The filter implements the online map matching algorithm as described in Section 2.3 and is executed when it receives a position measurement $z_t$. In the first step, the filter selects map position candidates $\mathcal{S}_t$ from the map, i.e. road segments near measurement $z_t$. The map is compiled from OpenStreetMap[2] data and represents road geometries using geometry data structures provided with ESRI's Java Geometry API[3]. This library comes with a spatial index data structure (Quadtree) that enables efficient spatial range search for nearby road segments [Sam90]. Spatial operations such as geodesic distance calculations or point-to-line projections in WGS-84 map projection are only partially supported. To provide this functionality, we used and extended the Java implementation of GeographicLib[4] maintained by Charles Karney which provides even more exact geodesic distance calculations [Kar13].

In the second step, the filter requests the path $\langle s_{t-1}, s_t \rangle$ for each pair of position candidates $(s_{t-1}, s_t)$ with $s_{t-1} \in \mathcal{S}_{t-1}$ and $s_t \in \mathcal{S}_t$. The path is provided by the router component. We evaluated several Open Source routers that rely on OpenStreetMap data. The pgRouting[5] project provides routing extensions for the PostgreSQL database management system

[2] http://openstreetmap.org
[3] http://github.com/Esri/geometry-api-java
[4] http://geographiclib.sourceforge.net
[5] http://pgrouting.org

that can be executed within a SQL query. It has a significant overhead since the routing algorithms are implemented in C functions that load data from database into memory on each execution. GraphHopper[6] is a Java-based router that is highly optimized for long distance routing, e.g. using graph contraction hierarchies. However, its programming interface does not support routing between pre-defined road segments. Routino[7] is another router implementation for OpenStreetMap data and was discarded because it is licensed under the AGPL (GNU Affero General Public License) that we assessed as too restrictive.

In the final step, the filter determines emission probabilities $p(z_t|s_t)$, transition probabilities $p(s_t|s_{t-1})$ and posterior probabilities $p(s_t|z_0, \ldots, z_t)$ for each pair of position candidates $(s_{t-1}, s_t)$ with $s_{t-1} \in \mathcal{S}_{t-1}$ and $s_t \in \mathcal{S}_t$. The vehicle's state information, i.e. map position candidates $\mathcal{S}_t$, probabilities $p(s_t|z_0, \ldots, z_t)$ for each $s_t \in \mathcal{S}_t$ (in Figure 3 denoted as $p$) and the most likely map position $\tilde{s}_t$, is saved to memory. To enable online map matching of multiple vehicles, it is necessary to associate incoming position measurements $z_t$ and state information saved in memory with object identifiers. To provide efficient memory access, it is necessary to maintain a search index on those identifiers. Furthermore, services typically query for all objects that are within a certain range. Thus, we also use a spatial search index on position estimates $\tilde{s}_t$ to increase search performance [Sam90].

### 3.2 Strategies for scalability

Our online map-matching system must process and organize information timely without being affected in its performance by the number of mobile objects that it tracks. We refer to the number of messages per second as the system's load. A system is scalable if it guarantees a fixed response time independently of its load by increasing computing, storage and communication elements. There are two approaches to system scaling: The first is to scale up a computer system by increasing the capacity of existing resources, such as by using faster processors or bigger memory. This is referred to as vertical scaling which is the focus of research in multicore systems and storage media. The second approach is referred to as horizontal scaling in which a computer system is scaled out by adding more machines (cluster nodes). Here, we focus on horizontal scaling.

Figure 4 shows a generic architecture for a scalable online map-matching system. Components can be implemented using different Open Source software solutions. The architecture scales up the online map-matching system of Figure 3. It uses two major clusters, i.e compute cluster (C), with $n$ nodes, and storage cluster (S), with $m$ nodes. Each node of the compute cluster runs an instance of the filter, map and router. The state memory is distributed over nodes $s_1, \ldots, s_m$ of the storage cluster (S). A load-balancer (L) distributes incoming messages over compute nodes. Each compute node performs map matching as described for a single-node system, while it reads and writes state information from or to the storage cluster. The filters of compute nodes access state information directly by object identifiers. In contrast, service applications (A) query state information mostly by spatial search through a query layer (Q) provided by the storage cluster.

[6] http://graphhopper.com
[7] http://routino.org

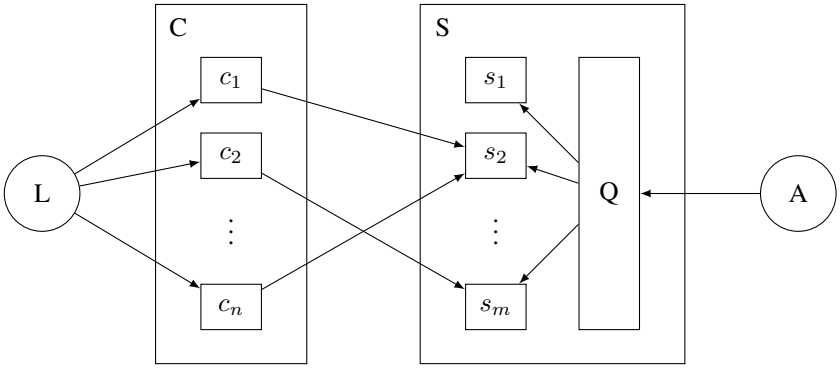


Figure 4: Scalable system architecture for online map matching. A load balancer (L) distributes message streams to a compute cluster (C). The compute cluster uses the storage cluster (S) that itself is queried from applications (A) via a query layer (Q). We refer to two physically separated clusters; however, advanced approaches may separate these clusters only logically to enable physical locality of computing and storage resources.

The major design decisions made with this architecture are:

- The compute and storage cluster can be physically separated. Hence, the size of each cluster ($m$ and $n$) can be scaled independently such that only those resources have to be added that are running out, i.e. either storage or computing resources.
- The compute cluster consists of interchangeable nodes meaning that each message can be processed by any of those nodes. This simplifies load-balancing.
- The storage cluster is accessed by compute nodes and services in a different manner, allowing for the independent optimization of each type of state access, i.e. by identifier search or spatial search.

The components of this architecture (Figure 4) can be implemented completely with Open Source software. Apache Kafka[8] is a distributed messaging system that can be used as multi-node load-balancer. A broker distributes incoming messages over a set of queues, referred to as partitions, randomly. Each compute node can fetch messages from a dedicated partition exclusively in a work-stealing fashion. In each partition, ordering of messages is strictly conserved; however, distribution over partitions violates strict ordering of messages among partitions. This is a relaxation of fairness that is necessary to gain scalability. [HKLP12] In our system, this is acceptable if the ordering of messages sent from a participant is conserved with high probability, which is satisfied if no partition has significantly higher throughput than any other. Messages that are processed out-of-order are discarded. This is necessary to prevent overwriting of state information with older information. Hence, the service quality corresponds to that of a soft real-time system.

The compute cluster requires a runtime environment for the map matcher components, i.e. filter, map and router. This also requires capabilities for scalable real-time stream

[8] http://kafka.apache.org

processing as it is available with Apache Storm[9]. It provides a framework for implementing directed acyclic computation graphs which are automatically deployed to compute nodes. Message streams are processed under soft real-time constraints. It is fail-safe and scalable to an arbitrary number of nodes. Apache Spark Streaming[10], a recent extension of Apache Spark for stream processing, provides an alternative with similar capabilities. Synchronization between compute nodes can be implemented with Apache Zookeeper[11], a scalable distributed directory service for maintaining configuration information and providing distributed synchronization. It is also used by various other software components as e.g. Apache Kafka and Apache Storm.

The storage cluster can be realized by Apache Cassandra[12], a database management system that provides key-value data access and linear scalability. During message processing, compute nodes can access state information by object identifiers with low latency by using respective index structures. The index structure is organized in such a way that the range of identifiers is partitioned over storage nodes where each storage node holds state information for a well-defined range of identifiers. This way, accessing storage by object identifiers can be forwarded directly to the respective storage node. (This is also referred to as direct mapping.)

Apache Cassandra supports map reduce queries with Apache Hadoop[13] by implementing the interface of the Hadoop Distributed File System (HDFS). This is sufficient to implement a query layer e.g. by using the following approach: To enable efficient search for objects by spatial properties, i.e. the object's last-known position, one must define a secondary index. This index can be created by hashing the position, e.g. using a geohash as implemented in one of various Open Source libraries[14]. A general problem with secondary indices, however, is that the values associated with a key can be located on any storage node. This is referred to as fully-associativity which is contrary to direct mapping. As a consequence, secondary index keys must be queried on each storage node which decreases performance. Nevertheless, map reduce with Apache Hadoop can query each storage node in parallel avoiding high latencies. Another performance problem arises with geohashes that require a range query to be split to all possible geohashes within this range. Querying larger areas may then be subject to massive performance drops. The problem is similar with querying nearest neighbors and could be solved in future with spatial index data structures such as Quadtree [Sam90], R-Tree [Gut84] or R*-Tree [BHPSS90].

## 4 Discussion

In this paper we discussed the role of map matching as a first and central step in providing location based services. We describe a map matcher that is both scalable and uses a state

[9] http://storm.incubator.apache.org
[10] http://spark.incubator.apache.org/docs/0.9.0/streaming-programming-guide.html
[11] http://zookeeper.apache.org
[12] http://cassandra.apache.org
[13] http://hadoop.apache.org
[14] https://github.com/davidmoten/geo

of the art map matching algorithm (Hidden Markov Model) that we believe is missing in the Open Source domain. This software is important yet non-differentiating from the OEM (Original Equipment Manufacturer) point of view and can hence be published and developed in a cooperative Open Source community.

The quality of trajectory map matching depends on the selection of state candidates and determining emission and transition probabilities. These problems are tightly coupled since candidates can be selected by maximizing the observation and transition probabilities. This is not yet fully utilized as the transition probability depends on local characteristics of the road network that are ignored. Further, in-vehicle systems could exploit information from the car (vehicle turns, velocity, computer vision, etc.). Backend systems, in turn, can exploit statistical information including:

- Turn probabilities at intersections can serve as base rate if other measurements are inclusive or contradictory.
- The velocity distribution of road segments can provide information for map matching, e.g. if parallel roads are close to each other and one of the two is a secondary road with usually lower speeds.

As coverage of technologies like LTE increases we anticipate a steady rise in communication between connected devices and backend systems, including cars. Nonetheless, since many services require the use of multiple data sources, we expect that the backend will continue to need the ability to localize all information on a common map and hence will require a scalable map matcher component.

We envision our map matcher as a component of a scalable, Open Source based backend ecosystem optimized for spatial-temporal data and service management. Such an ecosystem should enable scalable stream processing and data management with low latency service access and the ability to process large data sets for statistical information that can be used to optimize services. The quality of these services, in turn, is a differentiating factor where competition between market participants will take place. We believe that a tight integration to projects from established players in the Open Source community, such as the OpenStreetMap and the Apache Software Foundation, will be the key to success.

## References


[BHPSS90] N. Beckmann, Kriegel H.-P, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *International conference on management of data*, 1990.

[GDM+12] C.Y. Goh, J. Dauwels, N. Mitrovic, M.T. Asif, A. Oran, and P. Jaillet. Online map-matching based on Hidden Markov model for real-time traffic sensing applications. In *International IEEE Conference on Intelligent Transportation Systems*, 2012.

[Gut84] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *International conference on management of data*, 1984.

[HKLP12] A. Haas, C. Kirsch, M. Lippautz, and H. Payer. How FIFO is your concurrent FIFO queue? In *Workshop on Relaxing synchronization for multicore and manycore scalability*, 2012.

[Kar13] C. Karney. Algorithms for geodesics. *Journal of Geodesy*, 87(1):43–55, 2013.

[KSR13] Kevin S. Kung, Stanislav Sobolevsky, and Carlo Ratti. Exploring universal patterns in human home/work commuting from mobile phone data. *CoRR*, abs/1311.2911, 2013.

[LLC+14] Thomas Louail, Maxime Lenormand, Oliva García Cantú, Miguel Picornell, Ricardo Herranz, Enrique Frias-Martinez, José J Ramasco, and Marc Barthelemy. From mobile phone data to the spatial structure of cities. *arXiv preprint arXiv:1401.4540*, 2014.

[LZL+12] Xiao Liang, Xudong Zheng, Weifeng Lv, Tongyu Zhu, and Ke Xu. The scaling of human mobility by taxis is exponential. *Physica A: Statistical Mechanics and its Applications*, 391(5):2135–2144, 2012.

[LZZ+09] Y. Lou, C. Zhang, Y. Zheng, X . Xie, W. Wang, and Y. Huang. Map-matching for Low-sampling-rate GPS Trajectories. In *Proceedings of the International Conference on Advances in Geographic Information Systems*, 2009.

[NK09] P. Newson and J. Krumm. Hidden Markov Map Matching Through Noise and Sparseness. In *Proceedings of International Conference on Advances in Geographic Information Systems*, 2009.

[RN99] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1999.

[RWFP06] Carlo Ratti, S Williams, D Frenchman, and RM Pulselli. Mobile landscapes: using location data from cell phones for urban analysis. *Environment and Planning B Planning and Design*, 33(5):727, 2006.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.