In-Memory Trajectory Indexing for On-The-Fly Travel-Time Estimation

Waury, Robert

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

IN-MEMORY TRAJECTORY INDEXING FOR ON-THE-FLY TRAVEL-TIME ESTIMATION

BY
ROBERT WAURY

DISSERTATION SUBMITTED 2019

AALBORG UNIVERSITY
DENMARK

In-Memory Trajectory Indexing for On-The-Fly Travel-Time Estimation

Ph.D. Dissertation

Robert Waury

Dissertation submitted August, 2019

Dissertation submitted: August, 2019

PhD supervisor: Prof. Christian S. Jensen
Aalborg University

Assistant PhD supervisor: Prof. Kristian Torp
Aalborg University

PhD committee: Professor Bent Thomsen (chairman)
Aalborg University

Associate Professor Alberto Belussi
Università degli Studi di Verona

Professor Karine Bennis Zeitouni
University of Versailles Saint-Quentin

PhD Series: Technical Faculty of IT and Design, Aalborg University

Department: Department of Computer Science

ISSN (online): 2446-1628
ISBN (online): 978-87-7210-493-5

Published by:
Aalborg University Press
Langagervej 2
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2019

Abstract

This thesis presents a new method that is able to efficiently select subsets of sub-trajectories from a set of network-constrained vehicle trajectories. The method can be used to improve the quality of histogram-based travel-time estimates for paths in the underlying road network by identifying the trajectories that are most relevant for a given road network path and time. The thesis shows this with a series of detailed qualitative studies based on real-life trajectory data sets capturing several years of vehicular travel in Northern Denmark. We observe that the quality of travel-time estimates depends heavily on the type of road and whether the road is located in a rural or an urban area. After demonstrating the quality improvements provided by our method over previous approaches, we design a network-constrained trajectory index to efficiently apply our proposed collection method to compute travel-time histograms. The proposed in-memory index utilizes methods from string processing as a spatial index to identify trajectories in the indexed data that traverse a given path. The majority of its memory footprint is in the temporal index that comprises a collection of B+-trees or CSS-trees. To further increases performance, we integrate the new index with a very accurate cardinality estimator that allows us to minimize the accesses to the considerably larger temporal index. To augment our collection method, we propose a greedy algorithm that allows our index to provide a best-effort travel-time estimate even if few trajectories matching a path are available. This is achieved by allowing the index to rewrite a path-based travel-time query as a series of sub-queries that each cover only a sub-path of the original query path by partitioning the path in multiple iterations until a viable sub-query is found. We perform an extensive performance study and qualitative analysis of our index structure and partitioning strategies. We show that our method for travel-time estimation exhibits performance suitable for real-time applications when used in conjunction with our in-memory index. To show the scalability of the index, we integrate it into a multi-threaded trajectory store that allows to concurrently update and query the indexed trajectory set. To gain the full advantages of a multi-threaded application, we test it on a modern multi-core system. Modern multi-core systems in-

creasingly adopt a non-uniform memory access (NUMA) architecture, which necessitates different software design approaches compared to those for systems providing uniform memory access. We propose several optimizations to make the trajectory store NUMA-aware and provide an extensive experimental study of the system that shows considerable performance improvements over the non-optimized version. Further, we propose a path-based API for trajectory analysis that allows the aforementioned systems to be integrated easily into trajectory analysis applications. We present an example trajectory analysis application that allows users to not only compute travel-time estimates, but also allows to see the impact of trajectory anonymization on the result.

Resumé

Denne afhandling præsenterer en metode der kan bruges til at analysere turdata (trajectory data), der er map-matchet til et vejnetværk. Metoden forbedrer estimeringen af rejsetider for ruter i et vejenetværk ved at beregnet disse tider ud fra histogrammer på de enkelte vejsegmenter eller et samlet histogram for en sekvens af vejsegmenter. Afhandlingen viser forbedringen i estimeringen af rejse ved en række detaljerede studier baseret på virkelige data bestående af ture fra personbiler indsamlet over flere år i den nordlige del af Danmark. Vi observere, at kvaliteten af de estimerede rejsetider afhænger kraftigt af de vejtyper (f.eks. motorvej) der indgår i ruten og om ruten går igennem landområder eller byområder. Den præsenterede metoden kan forbedre kvaliteten af de estimerede rejsetider sammenlignet med eksisterende metoder. For at gøre dette effektivt har vi designet et hukommelsen-baseret (main-memory) indeks, som effektivt kan anvendes til at finde og lave beregninger på udelukkende de relevante histogrammer fra vejnetværket. Indekset anvender metoder fra tekststreng processering som et spatielt indeks til at identificerer ruten effektivt. Det meste af hukommelsen der bruges af indekset anvendes til et temporale sub-indeks. Dette sub-indeks er implementeret som en samling af B+-træer eller CSS-træer. Vi tilføjer en meget præcis kardinalitets estimering til indekset for at opnå hurtigere svartider. Forbedringen sker ved at minimerer mængden af data, der skal læses. Herudover tilføjer vi en grådig algoritme til vores indsamlingsmetode, der gør det muligt for indekset altid at returnere det bedste resultat, selvom der er meget få ture på en rute. Dette sker ved at omskrive rutebasserede forespørgsler for rejsetid på en lange rute til en serie af forespørgsler på kortere ruter, der dækker samlet dækker den oprindelig lange rute. Vi viser, at vores metode til estimering af rejsetid er så effektiv, at den kan anvendes i realtids programmer når indekset er i hukommelsen. For at vise skalerbarheden af indekset til store datasæt anvendes mange tråde i implementering af indekset. Disse tråde kan både opdatere (skrive) og lave forespørgsler (læse) det indekserede datasæt. For at opnå maksimal fordel ved flertrådet software evaluerer vi metoden på en moderne computer med flere kerner. Moderne computere med flere kerner er typisk opbygget uden ensartet svartider i tilgangen til hele hukom-

melsen fra alle kerner. I stedet anvendes en såkaldt Non-Uniform Memory Access (NUMA) arkitektur. En NUMA arkitektur kræver, at software designes anderledes for at være mest effektiv. Vi foreslå flere optimeringer som gør hukommelsen med turdata bevidst om, hvordan en computer med en NUMA arkitektur kan udnyttes bedst mht. svartider. Vi udfører et større eksperimentel studie af NUMA systemet. Studiet viser, at de tilføjede optimeringer giver betydelige forbedringer sammenlignet med en tilsvarende ikke-NUMA udgave af systemet. Herudover foreslår vi et rute-baseret API til tur-baseret analyse, der tillader, at systemet kan integreres i andre applikationer. Som et eksempel præsenterer vi en applikation til tur-baseret analyse, der gør det muligt både at estimere rejsetider og evaluere effekten af at anonymiserer turdata.

Acknowledgments

I would like to express my sincerest gratitude to everyone who made this thesis possible and who supported me during my studies.

First of all, I would like to thank my supervisors Christian S. Jensen and Kristian Torp for their patience, support, and constructive feedback during my PhD studies. Their doors have always been open throughout the years and I could approach them with questions as well as doubts about my plans to which they always provided helpful insights and suggestions.

I would also like to thank everybody from the Database, Programming and Web Technologies Group at Aalborg University for making the department such an interesting and pleasant place to work at. Special thanks goes to my co-authors from the department, Peter Dolog, Jilin Hu, and Bin Yang, to Johannes Borresen for helping me settle into Aalborg, and Søren Kejser Jensen and Ilkcan Keles for giving me helpful advice on the thesis and for being great office mates. My thanks also go to Helle Schroll and Helle Westmark for their advice and guidance in administrative matters.

I would also like to thank the staff and students in the Database Laboratory at Nagoya University for being such great hosts during my external stay. Especially, I would like to thank Yoshiharu Ishikawa, Chuan Xiao, and Satoshi Koide for the fruitful collaboration and their support during my research visit.

Moreover, I would like to thank the Obel Family Foundation for funding my PhD studies and the Otto Mønsteds Fond for providing travel grants that allowed me to attend several international conferences.

Last, but by no means least, I would like to thank my family for providing the crucial logistical and moral support during my stay in Aalborg, that made this PhD possible.

Contents

Thesis Details

Thesis Title:	In-Memory Trajectory Indexing for On-The-Fly Travel-Time Estimation
Ph.D. Student:	Robert Waury
Supervisors:	Prof. Christian S. Jensen, Aalborg University Prof. Kristian Torp, Aalborg University

The main body of this thesis consists of the following papers.

[A] Robert Waury, Jilin Hu, Bin Yang, Christian S. Jensen, "Assessing the Accuracy Benefits of On-The-Fly Trajectory Selection in Fine-Grained Travel-Time Estimation." *In Proceedings of the 18th IEEE International Conference on Mobile Data Management, Daejeon, South Korea, pages 240-245, 2017*.

[B] Robert Waury, Christian S. Jensen, Kristian Torp, "Adaptive Travel-Time Estimation: A Case for Custom Predicate Selection." *In Proceedings of the 19th IEEE International Conference on Mobile Data Management, Aalborg, Denmark, pages 96-105, 2018*.

[C] Robert Waury, Christian S. Jensen, Satoshi Koide, Yoshiharu Ishikawa, Chuan Xiao, "Indexing Trajectories for Travel-Time Histogram Retrieval." *In Proceedings of the 22nd International Conference on Extending Database Technology, Lisboa, Portugal, pages 157-168, 2019*.

[D] Robert Waury, Christian S. Jensen, Kristian Torp, "A NUMA-aware Trajectory Store for Travel-Time Estimation." *Accepted at 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Chicago, Illinois, USA, 2019*.

[E] Robert Waury, Christian S. Jensen, Peter Dolog, Kristian Torp, "Analyzing Trajectories Using a Path-based API." *In Proceedings of the 16th International Symposium on Spatial and Temporal Databases, Vienna, Austria, pages 198-201, 2019*.

This thesis has been submitted for assessment in partial fulfillment of the PhD degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The permission for using the published and accepted articles in the thesis has been obtained from the corresponding publishers with the conditions that they are cited and DOI pointers and/or copyright/credits are placed prominently in the references.

Part I

Thesis Summary

Thesis Summary

1 Introduction

1.1 Background and Motivation

Vehicular transportation is an important global phenomenon that impacts the lives of virtually all of us: We rely on it for mobility, and we are affected by congestion, accidents, and air and noise pollution. Trailing only the energy sector, the transportation sector is responsible for enormous greenhouse gas (GHG) emissions and thus contributes to global warming [33]. Increasing the efficiency of transportation networks is therefore a key approach to reducing overall GHG emissions.

We are witnessing two trends that are relevant to our problem:

Big Trajectory Data We are seeing a rapid accumulation of vehicle trajectory data, typically obtained from GPS data consisting of timestamped latitude-longitude pairs. Trajectory data can be used for a variety of applications, e.g., travel-time estimation.

Power Wall vs. Moore's Law Since processors are hitting the power wall, increasing the number of processing units remains the only efficient way to increase processing power as opposed to increasing the performance of single processing units, e.g., by increasing clock speeds [9, 24].

Not only is GPS data now available in larger quantities, but also at high enough sampling rates [2] at which it can be reliably map-matched by online [10] as well as off-line [25, 29] algorithms to publicly available map data, e.g., OpenStreetMap [26].

We show that personalized travel-time estimations can improve transportation efficiency by providing more accurate estimations of travel-time. Personalized travel-time estimation is, however, more compute- and data-intensive than previous approaches and therefore needs to be optimized for modern hardware to achieve the short response times required for real-time applications. When targeting modern hardware, we need to eschew data

structures that perform poorly in in-memory settings, e.g., the R-tree [12]. This is due to the availability of ever cheaper main memory, which has made it feasible to keep even big trajectory data sets memory resident. Furthermore, we need to optimize our architecture for modern multi-core systems, which rarely provide uniform memory access (UMA), further complicating the design of efficient multi-threaded systems.

We demonstrate that through careful optimization of the underlying implementation, existing applications like, e.g., on-line routing, can be improved by "on-the-fly" analysis of large trajectory data sets [39–42]. We also show how this new type of analysis can be integrated into new applications with a novel path-based API [38].

1.2 Trajectory-Based Travel-Time Estimation

We model travel-time estimation as a query $Q = (P, t, f)$, consisting of a path P, a starting timestamp t from which we obtain a time interval I, and an optional filter predicate f that returns a travel-time histogram H for the path P, based on trajectories within I that satisfy predicate f. We decide to compute histograms because this also allows us to compute expected travel-time ranges and because travel times often do not follow parametric distributions [5]. Trajectory data can be used in different ways to obtain travel-time estimates:

Segment-based estimates, that are based on aggregates for every single single road segment of a given path P and can be easily augmented by traffic data obtained from other sources, e.g., Bluetooth sensors or induction loops [4].

Path-based estimates, that are based on traversal times for a full path P.

Obtaining path-based estimates is considerably more complex than obtaining segment-based estimations, but we argue that doing this is worth the effort due to the higher accuracy of this approach [39]. A motivating example for the advantage of path-based estimates is shown in Figure 1, where multiple trajectories traverse an intersection. If one wants to obtain a travel-time histogram for the path $P = \langle A, D \rangle$ based on the given trajectory set, a segment-based approach would collect a travel-time histogram for each of segments A and D and then return the convolution of the histograms $(H(A) * H(D))$ as the estimate for the full path. The convolution $f * g$ creates a mixture of the functions f and g and for discrete functions, e.g., histograms, it is defined as $(f * g)(x) = \sum_{y=-\infty}^{\infty} f(y)g(x-y)$.

This approach creates several problems in our example scenario:

- Histogram convolution assumes statistical independence between the distributions of the segments, which is often an unrealistic assumption.

- The travel-time histogram for the full path exhibits an artificially high variance [41].

This can be avoided with path-based estimation that is only based on trajectories that fully traverse the given path $P = \langle A, D \rangle$, shown in red in the example. This approach takes into account turn costs and does not require to model these costs separately.

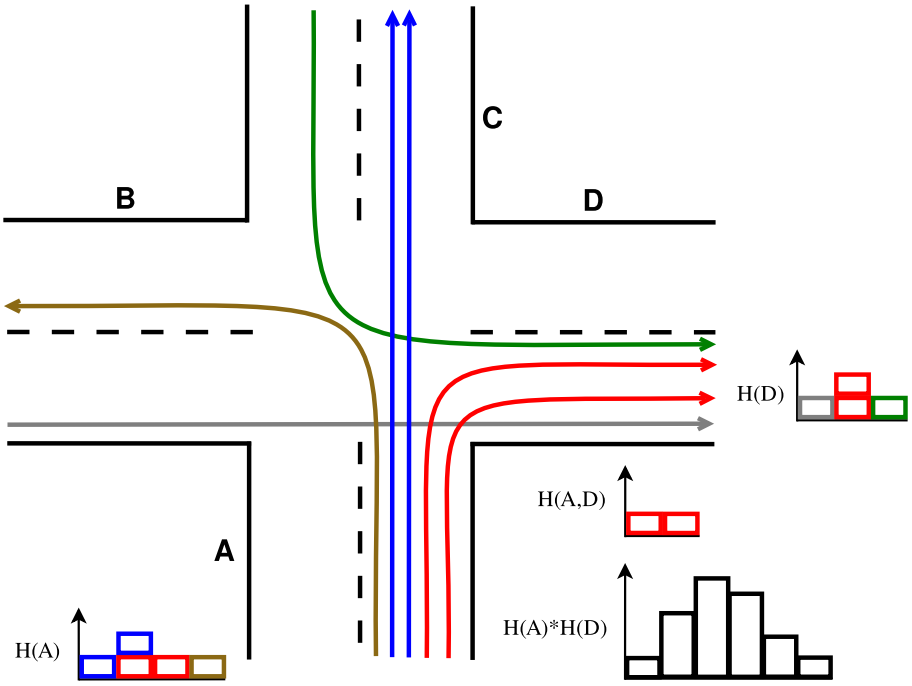


Fig. 1: Motivating Example [38]

Furthermore, path-based queries can be modeled as strict path queries (SPQ), first proposed by Krogh et al. [18], which only select trajectories from the set which follow a given path exactly. Several index structures that efficiently process SPQs have been proposed. In addition to being able to retrieving travel-times of any path in the network, the "on-the-fly" collection also allows much finer-grained filter predicates that would not be feasible in a pre-computed approach.

1.3 NCT Indexing on Modern Hardware

The type of query required to facilitate path-based estimates poses several challenges compared to a segment-based approach. Where a segment-based approach can be implemented as a convolution of pre-computed segment

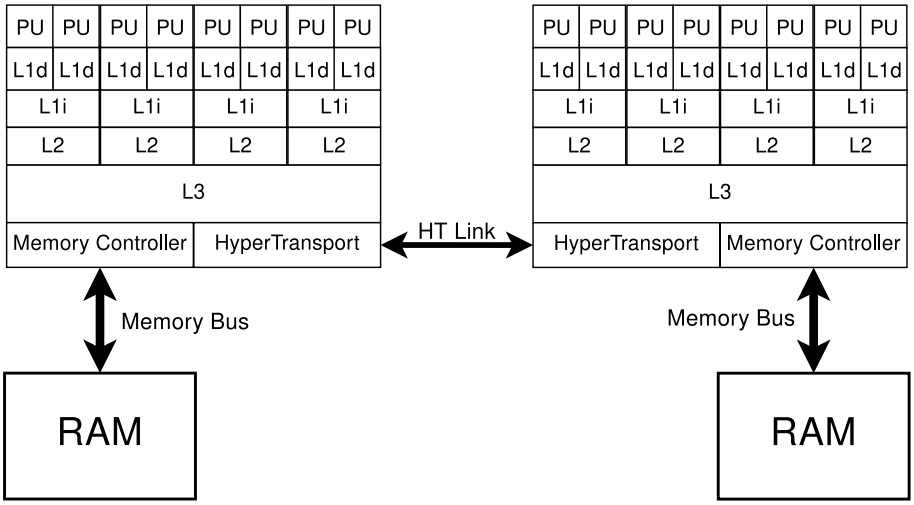


Fig. 2: Example NUMA Topology [42]

histograms, a path-based approach relies on identifying all trajectories that follow a path exactly, i.e., an SPQ, and cannot be pre-computed for most paths. Since this type of query relies on map-matched trajectory data, indexing them using information of the graph structure of the road network is more promising than using geo-spatial indexing methods, e.g., the R-tree [8].

Several such indexing methods have been described in the literature and were considered for adaptation to travel-time estimation. Krogh et al. propose the disk-based NETTRA index, which is designed to facilitate integration into already existing database sytems [18]. Since we focus on real-time query performance and since the main memories of modern systems continue to grow, we design an in-memory index.

The challenge is to identify an approach that scales well on modern hardware. For this thesis, we understand modern hardware to mean:

- Large main memory.
- A non-uniform memory access (NUMA) architecture [19].

Figure 2 shows the memory hierarchy of an example NUMA system used in our test system that consists of AMD Opteron 6376 processors, where L1i is the instruction cache and L1d is the Level 1 data cache [42]. It shows that each processing unit (PU) is connected to the main memory by a hierarchy of caches that are connected directly to only a portion of the main memory. A memory region which can only be accessed by a subset of the PUs directly is called a NUMA region. If a PU accesses memory in another NUMA region, it needs to use the processor interconnect, e.g., HyperTransport for

AMD processors or QPI for Intel processors [13]. While those access methods guarantee cache-coherence between different PUs, they do not provide uniform access times or bandwidths, which can differ by more than an order of magnitude between PUs depending on their NUMA regions and where the data is located.

1.4 Open Trajectory Data

Just indexing trajectory data efficiently is not enough, the data must also be made available to users. We suggest an API for path-based queries that can be integrated with a RESTful service to provide travel-time estimation services and other trajectory-based analyses to users. To demonstrate the applicability of our API, we use it to implement an example trajectory analytics web application. The web application provides data privacy features that would allow trajectory data sets to be made available to a wider range of users without infringing upon drivers' privacy.

1.5 Related Work

This section gives an overview of the most relevant related work in the three areas that intersect with the topic of this thesis.

Travel-Time Estimation

Due to the importance of traffic management, a large number of approaches for travel-time estimation have been proposed. Our overview is limited to estimation methods based on trajectory data. Different approaches based on pre-computed histograms have been proposed [5, 22]. These approaches lack the flexibility of travel-time histograms computed at runtime, which can adapt filter predicates at query time to achieve a considerably finer granularity of the histograms than is feasible in the case of pre-computation approaches.

An approach based on tensor decomposition that also addresses the problem of data sparseness has also been proposed by Wang, Zheng, and Xue [37]. Another approach using deep neural networks has also been proposed by Wang et al. [36]. These approaches, however, does not provide travel-time distributions but only scalar estimates.

Trajectory Indexing

Ding's R-tree-based UTR-tree [6] and the B+-tree-based PARINET from Popa et al. [30] are network-constrained trajectory indexes optimized for nearest neighbor queries and spatial range queries. Both indexes are, however, ill-suited for strict path queries required for path-based estimation.

These shortcomings are addressed by Krogh et al.'s NETTRA index [18] that can be integrated with an existing RDBMS and by Koide et al.'s in-memory SNT-index [17] that both support efficient processing of strict path queries.

NUMA-Aware Query Processing

The increasing number of cores in modern computers and the accompanying change from UMA architectures to NUMA architectures posed the problem of how to best adapt database systems to exploit NUMA architectures.

The DORA system proposed by Pandis et al. [27] changes the "thread-to-query" processing model, until then prevalent in OLTP, to a "thread-to-data" model, which assigns partitions of a database to pinned threads with a local lock manager to limit inter-core communication with the goal of increasing transaction throughput on multi-core systems.

This data-oriented architecture has also been extended to OLAP workloads in the ERIS system by Kissinger et al. [15], which employs specialized worker threads to process analytical queries on a partitioned database, or in the extension of the HyPer database system proposed by Leis et al. [21] that utilizes "work-stealing" to achieve optimal resource utilization on NUMA systems.

1.6 Organization

The remainder of this summary is structured as follows. Section 2 summarizes Paper A that provides a preliminary study of time-travel estimation based on "on-the-fly" trajectory selection. Section 3 summarizes Paper B that conducts a much more detailed study based on a large real-world data set. The study shows which type of predicates applied to a trajectory data set produce the most accurate travel-time estimates. Section 4 summarizes Paper C that introduces a new in-memory trajectory index that facilitates efficient processing of strict path queries (SPQ) that can be adapted for travel-time estimation. Section 5 summarizes Paper D that adapts the in-memory index to be used with real-time data and optimizes it for non-uniform memory access (NUMA) systems. Section 6 summarizes Paper E that introduces an API for path-based trajectory analytics.

Section 7 gives a summary of the contributions in the thesis, and Section 8 concludes.

2 On-the-fly Trajectory Selection

This section gives an overview of Paper A [39].

2.1 Problem Motivation and Statement

In our initial experiments, we aim to assess whether previously proposed "one-size-fits-all" approaches to travel-time estimation can be improved upon. The "one-size-fits-all" approaches use the same data points for all estimates for different drivers if their trip is on the same road segments and in the same time window. It allows to, e.g., detect heavy congestion, but does not consider personal driving behavior or the influence of weather or other filter predicates on travel-time. The "one-size-fits-all" approaches have the advantage that they lend themselves well to pre-computation, which reduces processing time considerably. These approaches, however, become infeasible when the number of predicates to be pre-computed becomes too large, negating the approaches' main benefit.

We contrast these approaches with our "on-the-fly" approach, in which all predicates are evaluated at runtime, allowing for a much finer grained selection of trajectories to base an estimate on. For an "on-the-fly" query, the whole set of trajectories traversing the chosen segments is considered and filtered at query time.

To identify whether "on-the-fly" trajectory selection can provide more accurate travel-time estimations, we perform an analysis of traffic in Northern Denmark based on a detailed trajectory data set from private car owners that allows personalized, i.e., based on a single driver's trips, estimation. The analysis is based on the map-matched "Young Drivers" data set, which contains around $670,000$ trajectories collected over two years at a 1 Hertz sampling rate [43]. We use a histogram-based approach to estimate the traversal times for single segments and short paths (two and three segments) to perform a preliminary evaluation of our approach.

The histograms obtained are compared to actual traversal times from the trajectory data set in order to assess their accuracy. The analysis shows that driver-based trajectory selection can improve travel-time histogram accuracy with a query time of only a few milliseconds.

2.2 Histogram-Based Travel-Time Estimation

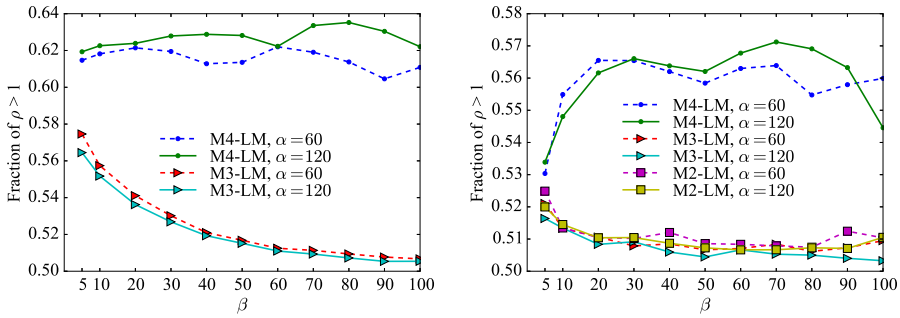
To evaluate our two "on-the-fly" approaches (Methods 3 and 4) we compare them to "one-size-fits-all" histogram-based approaches for single segments and short paths (the Legacy Method and Method 2), whose details are shown in Table 1. As a baseline, we use the pre-aggregated Legacy Method (LM) that constructs travel-time histograms for each static time interval of a fixed size

(α) and computes path histograms by convolving the segment histograms. Method 2 (M2) is identical, but also provides pre-aggregated histograms for path traversals up to a length of 3. The "on-the-fly" Method 3 (M3) selects the trajectories for a path histogram with a dynamic time interval that is centered around the query timestamp t, whereas Method 4 (M4) also applies a filter predicate f. In our study, we filter by drivers, but other filter predicates can also be used.

Method	Temporal Features	Path Cost	Predicate
LM	Static, Off-line	No	No
M2	Static, Off-line	Yes	No
M3	Dynamic, On-line	Yes	No
M4	Dynamic, On-line	Yes	Yes

Table 1: A Summary of the four Methods [39]

Figure 3 shows how often the other methods improve over the baseline (LM), i.e., their likelihood ratio ρ is larger than 1. The figure shows that only the method using additional filter predicates, i.e., filtering by driver, (M4) provides a consistent quality improvement for single segments and short paths with different sample sizes (β) . Simply applying dynamic fine-grained temporal filtering to the trajectory set provides little benefit. This can be seen in both Figures 3a and 3b where the share of improvement of Method 3 and Method 4 decreases to only about 50% for higher values of β.



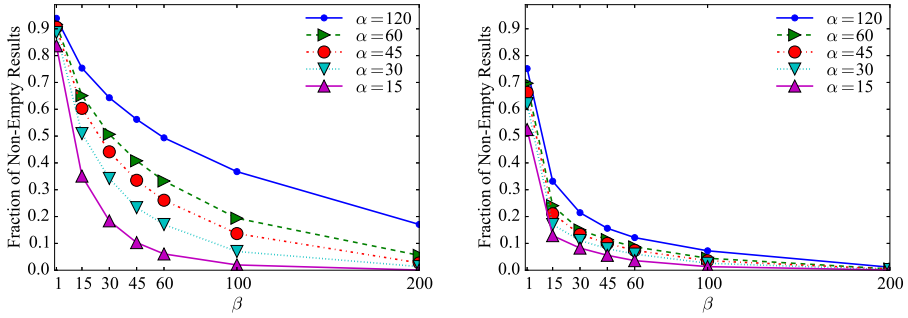
(a) Single Segment, Daily

(b) Two Segments, Daily

Fig. 3: Histogram Accuracy [39]

When comparing Figures 4a and 4b, which show the fraction of non-empty result histograms with respect to the sample size β, we can see the biggest drawback of M4. The trajectory set becomes a lot sparser when ap-

plying more selective predicates. If M4 is to be adapted for a universal travel-time estimation system that can answer travel-time queries about arbitrary paths, it needs to be combined with less restrictive methods.



(a) LM/M2 Non-Empty Result Fraction

(b) M4 Non-Empty Result Fraction

Fig. 4: Fraction of Non-Empty Results [39]

2.3 Discussion

Our experiments show that applying additional filter predicates to a trajectory data set when computing travel-time histograms for which pre-computation is not practical can provide travel-time estimates with an up to 20% higher accuracy. The study concludes with a performance study of the "on-the-fly" approach implemented with a simple immutable in-memory index based on Rao and Ross's CSS-tree [32], which shows that queries with fine-grained filter predicates can be processed in less than 3 milliseconds on average, a speed suitable for real-time applications. Our results are encouraging and we perform a more detailed analysis with another data set from the same area which is more than twice as large.

3 Custom Predicate Selection

This section gives an overview of Paper B [41].

3.1 Problem Motivation and Statement

For our extended study, we not only analyze single segments and short paths, but pick the most often traversed paths in a network and see by how much a purely path-based "on-the-fly" approach could improve estimation. Initial

results show that when just choosing the most traveled paths without taking into account the structure of the network, i.e., the types of roads covered by the path, no consistent improvement can be achieved. However, if we focus our study on only paths consisting of a single segment category, noticeably improvements can be observed. For each one of four different segment categories (motorway, primary, secondary, or tertiary), we study three to four different paths. For the study, we use the larger ITSP trajectory set, which consists of over 1.4 million trajectories and has also been enriched with weather category data [1] so that we can also evaluate the impact of other filter predicates on the quality of estimation.

To provide a more in-depth study, we also devise a new two-part metric, evaluating the accuracy and precision of travel-time histograms. Accuracy measures how close the mean of a travel-time histogram is to the actual result using the symmetric absolute mean percentage error (sMAPE) [3]. Precision measures the "uncertainty" of an estimate based on the size of the distribution-free prediction interval which is in turn based on the sample variance of the collected values [35]. This metric can be adapted to not only optimize routing towards the fastest travel-time, but also the most predictable travel-time.

3.2 Full Path Histograms

Figure 5 shows the results for four paths on motorways in Northern Denmark compared to the "one-size-fits-all" approach based on single segment histograms, which we use as a baseline.

We also evaluate the effects predicates filtering by weather and user have on the quality of the results. While the "on-the-fly" approaches improve on the baseline in most cases, the weather-based predicates provide only little if any improvement over purely temporal filtering. User predicates provides the best results on motorways and primary roads, especially for traffic outside cities.

Since real-life travel-time estimation will rarely require estimates for paths consisting of only a single road category, this approach needs to be integrated into a more extensive system. Figure 6 shows the architecture of a system that provides a travel-time histogram for a full path by partitioning a query into multiple non-overlapping sub-queries. The boxes represent software components, and the arrows show the data flow between them and the user. To obtain a travel-time histogram for a full path, the histograms obtained from the sub-queries are convolved into a full path histogram, which is returned to the user. If insufficient trajectories for a given sub-path are available, the sub-query predicates are relaxed or the sub-path is repeatedly split until the desired sample size is reached for constructing the corresponding partial result histogram.

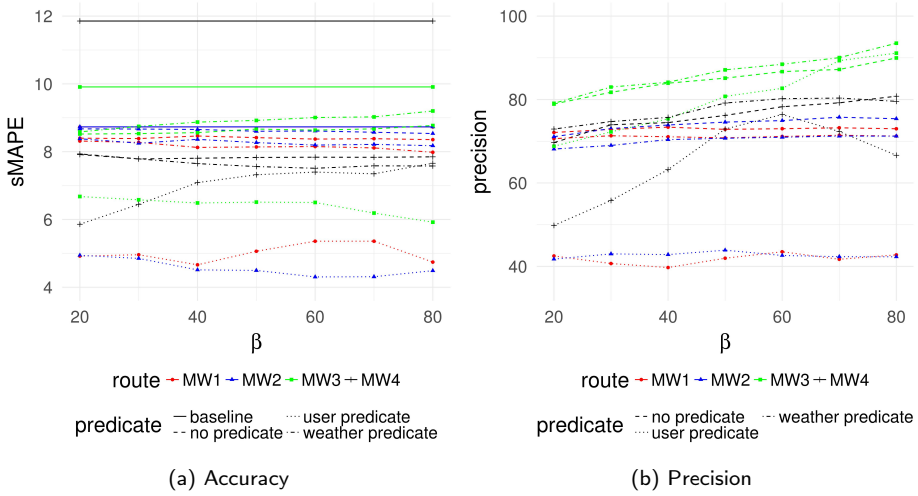


Fig. 5: Motorway Histogram Quality [41]

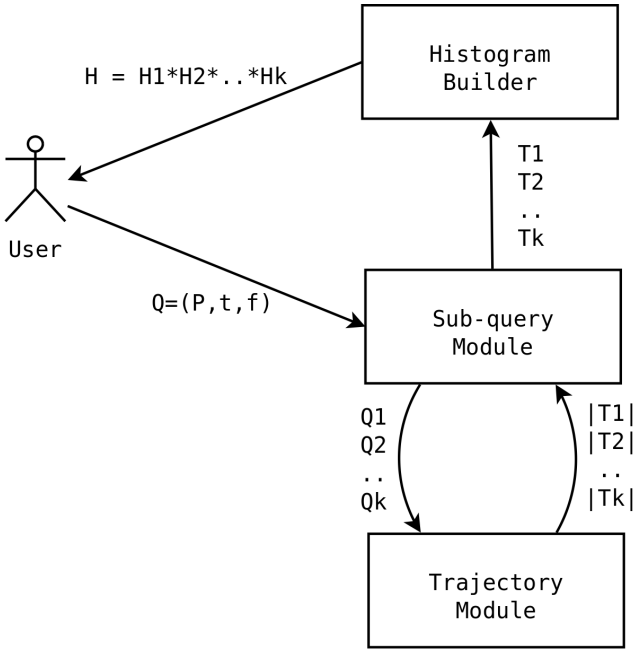


Fig. 6: Architecture of a System Computing Full Path Histograms [41]

3.3 Discussion

The results suggest how to best partition a query into sub-queries and what types of filter predicates improve travel-time estimation. We can also see a significant difference in the quality of travel-time estimates depending on the road location and category. Travel-time estimates for roads leading through rural areas are on average more accurate than those for urban roads, especially primary roads and motorways. The estimation error for, e.g., motorways through rural areas, can be halved with our personalized path-based approach. There are also considerable differences between primary and secondary/tertiary roads, with the latter allowing only considerably lower quality estimates.

In the experiments, we use the same immutable index as in Paper A. However, this initial index design exhibits performance degradation when queried with longer paths due to random memory accesses becoming a bottleneck. Our next goal is to efficiently process such sub-queries in a integrated system with an architecture similar to the system shown in Figure 6. This requires identifying an index structure that is able to efficiently handle queries for any kind of path while still being memory resident.

4 In-Memory Index

This section gives an overview of Paper C [40].

4.1 Problem Motivation and Statement

Given the results from Papers A and B, we set out to design an in-memory index to facilitate the travel-time histogram retrieval for full trip paths. The index is integrated into a query processing system that, even if not enough trajectory data is available, always makes a best effort estimate based on other data, e.g., speed limits, if necessary.

We adapt a network-constrained trajectory index based on data structures for which efficient in-memory implementations already existed. Specifically, we build on Koide et al.'s SNT-index that was designed to efficiently retrieve trajectory IDs from strict path queries [17] and also provides better scalability in respect to path length compared to our initial approach to indexing. The index consists of two components:

A spatial index that allows fast look-ups of paths in the trajectory set and that is implemented with an FM-index [7]. The FM-index is a text index that facilitates fast substring searches while at the same time compressing the indexed string. To find out whether trajectories exist that follow a given path P exist in the trajectory set, at most $|P|$ rank queries

with a complexity of $\mathcal{O}(log\,\Sigma)$, where Σ is the size of the network, are required. In addition to the number of matching trajectories, the FM-index also provides a spatial filter predicate which can then be applied to the temporal indexes.

A temporal index that enables temporal filtering of the trajectories to be retrieved and that is implemented as a forest of B+-trees. The leaf nodes of these indexes also store the spatial information of each data point, which can be filtered with two integer comparisons with the information obtained from the spatial index.

No matter how long a query path is, the SNT-index only needs to scan the temporal index of the last segment of the path to apply temporal filters. We extended the SNT-index so that we could obtain the traversal time of any path by only scanning the first and last segments' temporal indexes.

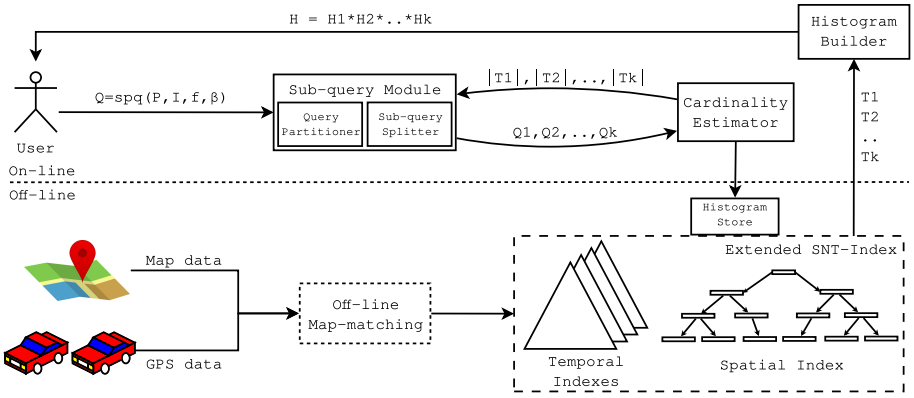


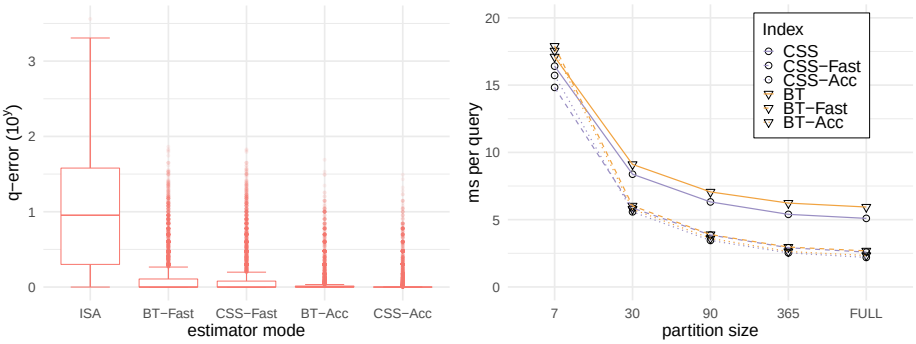
Fig. 7: Overall Architecture [40]

Figure 7 shows the architecture of our indexes which is based on the system architecture introduced in Paper B. A user dispatches a full path query $Q = (P, I, f, \beta)$, with β being the required sample size, to the `Sub-query Module`, which partitions the query into sub-queries, that are then checked by the `Cardinality Estimator`. The cardinality estimator checks whether the returned trajectory set of each sub-query returns at least β trajectories by querying the `Spatial Index` and the `Histogram Store`. The `Histogram Store` contains the time-of-day histograms for every segment and is used to estimate the selectivity of temporal predicates, i.e., the interval I. If the selectivity is too high, the predicates of the sub-query are relaxed, or the sub-query path is split by the `Sub-query Splitter` into new sub-queries until enough matching trajectories are found. For our evaluation, we use the ITSP data set introduced in Paper B. The map-matched trajectory data is batch loaded into the index but how real-time ingestion can be handled is discussed in Paper D.

We also add a cardinality estimator that bases its results on the spatial index and a time-of-day histogram of each segment to avoid unnecessary scans of the temporal indexes. This increases performance since scans of the temporal indexes are several orders of magnitude slower than querying the spatial index.

4.2 Indexing for Strict Path Queries

The cardinality estimator accuracy is shown in Figure 8a, where the cardinality estimates based purely on the spatial index (ISA) are compared to estimates based on the spatial index and selectivity estimations of the temporal filters. The Q-error [23] (lower is better) is computed by comparing the cardinality estimate to the actual cardinality obtained by the query. The selectivity estimate of the time window (α) is based on either a simple formula (BT-Fast, CSS-Fast) or on the time-of-day histogram of the first segment in the sub-query path (BT-Acc, CSS-Acc). We can see that the combined cardinality estimation outperforms the one based exclusively on the spatial index by an order of magnitude on average.



(a) Q-Error (b) Runtime

Fig. 8: Cardinality Estimator [40]

As the different cardinality estimator modes show, the paper also evaluates two different data structures for use as temporal indexes:

B+-trees (BT) originally a disk-based index that supports out-of-order updates and can also be used in-memory [11].

CSS-trees a compact index on sorted arrays that has been extended to support appends [32].

We can also observe that the CSS-tree-based implementation has a small accuracy advantage over the B+-tree-based version. This is because the CSS-

tree can efficiently obtain the exact selectivity of a time range, which can only be estimated with a B+-tree.

The impact of different temporal index types and temporal partitioning on performance is shown in Figure 8b. Overall, we see that with the exception of very fine-grained temporal partitioning, the cardinality estimator cuts processing times in half on average. We can also see that CSS-trees provide a slight performance advantage over B+-trees, which, however, becomes barely noticeable when using the cardinality estimator.

The paper also provides a comprehensive qualitative analysis of the collection methods. The parameters considered are:

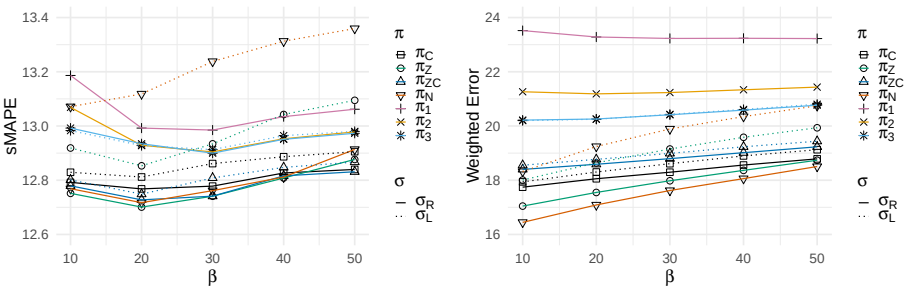
Sample Size (β) The required number of trajectories for each sub-query from 10 to 50 in increments of 10.

Predicate (f) Filter trajectory set by time and/or user, or only by path.

Partitioning Method (π) Initial partitioning of Q is performed by road category (π_C), zone (rural/urban) (π_Z), the combination of category and zone (π_{ZC}), or no pre-partitioning (π_N). This is compared against a fixed-size partitioning of length 1, 2, or 3 ($\pi_{1/2/3}$).

Splitting Method (σ) If a sub-query does not return the required number of trajectories, we provide two methods for splitting the sub-query into two sub-queries: (1) *regular splitting* that cuts the query path exactly into two halves (σ_R) and (2) *longest prefix splitting* that splits the path so that the first sub-query consists of the longest possible path for which the trajectory set is greater and or equal to β (σ_L).

Figure 9a shows the estimation error of a full path histogram for different collection methods. Figure 9b shows the same error weighted by the length of the paths of the sub-queries.



(a) sMAPE (b) Weighted Error

Fig. 9: Qualitative Analysis [40]

4.3 Discussion

Our Extended SNT-index allows spatial queries to be performed in $\mathcal{O}(|P|log\,\Sigma)$ time, i.e., it does not depend on the size of the indexed data set. The empirical analysis shows that the simpler splitting method performs considerably better in terms of quality as well as runtime and that applying non-temporal filter predicates worsens processing time considerably while only providing qualitative improvements for certain types of sub-queries.

The Extended SNT-index scales considerably better than our earlier approaches, and the processing time can be cut in half on top of that by reducing unnecessary scans of the temporal indexes by using a cardinality estimator.

5 Optimizing for Modern Hardware

This section gives an overview of Paper D [42].

5.1 Problem Motivation and Statement

The index structure proposed in Paper C is only implemented to run in a single thread. To evaluate the scalability of our approach, we implement a parallelized version of the index. In addition to the parallelization, we extended the index to also allow for real-time data being ingested. Modern multi-core systems provide two main avenues for parallelization:

Vectorization, which uses the CPU's SIMD registers to achieve data parallelism.

Multi-threading, which schedules threads on different cores to achieve task parallelism.

Multi-threading requires coordination between the threads running on different cores. This is done via shared memory, whose latency and bandwidth can differ by more than an order of magnitude between different core pairs on NUMA systems.

Figure 5 shows the latency of different types and sizes of memory accesses on our test system collected by the BenchIT tool [14]. For systems with a larger number of cores or a more complex processor topology, those differences can be even more pronounced.

These constraints require us to minimize the coordination required between remote cores because if cores are not located in the same region they have to use the processor interconnect to access main memory and cannot use their local memory bus or their shared lower level caches. In our test system all processing units of a NUMA region share the L3 data cache.

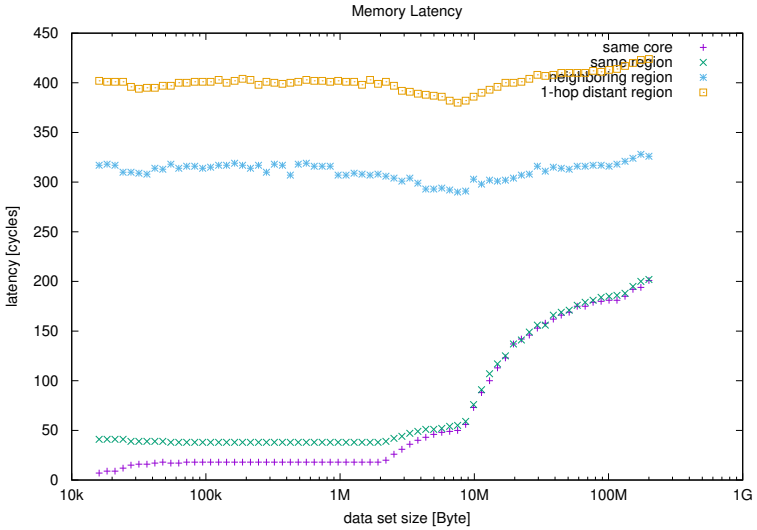


Fig. 10: Memory Access Latencies in a NUMA System [42]

5.2 Indexing on Modern Hardware

The index consists of two main data structures:

The FM-index for computing the spatial filter predicates and the cardinality of the trajectory set for a path.

B+-trees/CSS-trees for temporal filtering and storing segment traversal information.

These two components exhibit very different runtime behavior and also have very different storage requirements, so we assign them to different types of threads as shown in Figure 11.

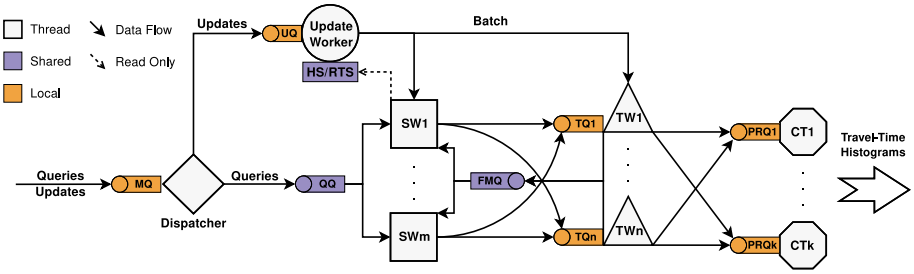


Fig. 11: NUMA-Aware Architecture [42]

A query is processed by five types of worker threads that communicate via message queues:

Dispatcher distributes incoming messages from the message queue (MQ) among worker queues (UQ and QQ) that are consumed by the Update Worker and the Spatial Workers, respectively.

Update Worker handles trajectory updates that are regularly merged into the index. In-between batch merges, the new trajectory data can be queried at the segment-level only. The Update Worker also maintains the time-of-day histograms (HS) that are used by the cardinality estimator. It also updates the real-time store (RTS) that is used for outlier detection on a segment level to be able to recognize congestion caused by unforeseen events, e.g., traffic accidents.

Spatial Worker (SW) performs the initial query partitioning based on metadata and the spatial index and also handles spatial queries and cardinality estimation. After a query has been partitioned, its sub-queries are enqueued into their respective Temporal Queues (TQ) that are consumed by the Temporal Workers.

Temporal Worker (TW) manages a subset of the temporal indexes, performs scans to answer sub-queries, and enqueues their results into the Partial Result Queues (PRQ) that are consumed by the Convolution Threads. If insufficient trajectories are found, the sub-query is split and enqueued into the FM queue (FMQ) to recompute the spatial filter predicate for the new sub-queries.

Convolution Thread (CT) is a parallelized version of the `HistogramBuilder` component that acts similar to a reducer. A CT combines sub-query results into travel-time histograms covering the full query path.

Since the FM-index only takes up a fraction of the overall index size, it can be replicated to decrease shared memory accesses and increase parallelism. While this design supports a data-oriented architecture, we also apply two additional optimizations to make it NUMA-aware:

Thread pinning assigns a worker thread to a PU and ensures that it is not rescheduled on another PU.

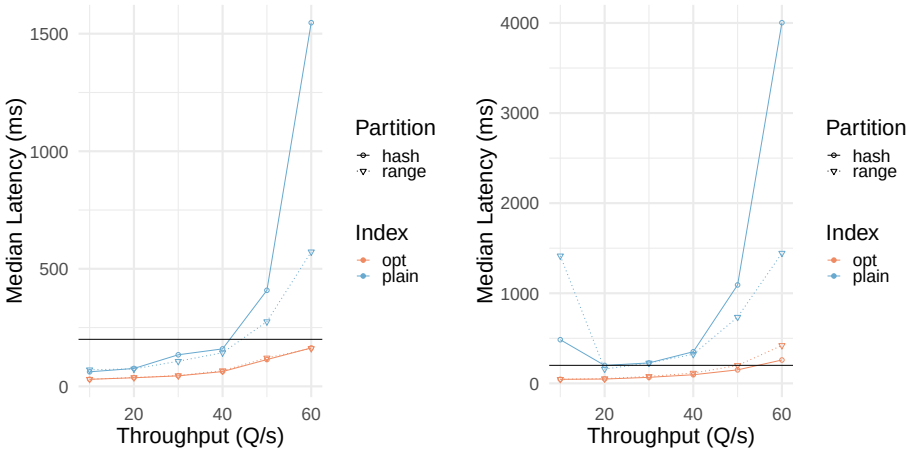
NUMA allocation ensures that memory is allocated in the same NUMA region on which the thread that is accessing it is running. This was implemented as a C++ allocator that uses the `libnuma` library [16].

For our evaluation, we use an "upsampled" version of the ITSP data set we used in Papers B and C. Figure 10 shows how the NUMA-aware implementation compares to the naive implementation, i.e., the same code without the aforementioned optimizations, on the same hardware. The figure compares the median time from dispatching a travel-time query to the computation of

the full path histogram (latency) relative to throughput (queries per second). Since each Temporal Worker processes scans of a different partition of the network, we also evaluate different partitioning methods for the network:

Hash partitions by the hash of the segment ID in a round-robin fashion.

Range partitions the segment IDs by range, which due to the way OpenStreetMap assigns the IDs results in a simple form of spatial clustering.



(a) Socket (16 threads)

(b) System (58 threads)

Fig. 12: Latency of Queries

Figure 12a shows the latencies when running with 16 threads, which on our test system is identical with one socket that contains two NUMA regions. Figure 12b shows the latencies when running with 58 out of 64 available cores.

As we can see from both figures, the NUMA-aware implementation exhibits at most half the latency of the naive implementation. This difference becomes even more pronounced when increasing the throughput. In the baseline version, we can also observe a noticeable difference between the different partitioning methods at high throughput. In the baseline setting, range partitioning reduces latency by over 60% compared to hash partitioning because the simple spatial clustering increases memory locality for many sub-queries. In the NUMA-aware version, we can see an inverse effect, since here the better load balancing between partitions through hash partitioning becomes the dominant factor.

5.3 Discussion

We show that a multi-threaded variant of our index remains scalable and can be adapted to work in a data-oriented architecture suitable for NUMA systems. Especially under high throughput, its NUMA-aware optimizations give it superior performance over the baseline implementation. We also show that the choice of partitioning method can have considerable impact on performance. Furthermore, the NUMA-aware optimizations have a comparatively small code footprint of only about 3.5% of the full trajectory store source code and therefore lend themselves well to optimizing existing systems and not just systems implemented from scratch.

6 An API for Trajectory Analytics

This section gives an overview of Paper E [38].

6.1 Problem Motivation and Statement

After devising a trajectory store for modern hardware, we evaluate how to best provide its processing capabilities to users. In parallel to developing the new trajectory store, we thus design an API that does not rely on a specific model of the trajectory data. We integrate the new API with our existing PostGIS [31] trajectory data warehouse that uses the NETTRA index [18] instead of the new trajectory store. The API can, however, be used with either system as a storage layer. To demonstrate the applicability of the API, we provide a RESTful [28] trajectory anonymization service whose architecture is shown in Figure 13.

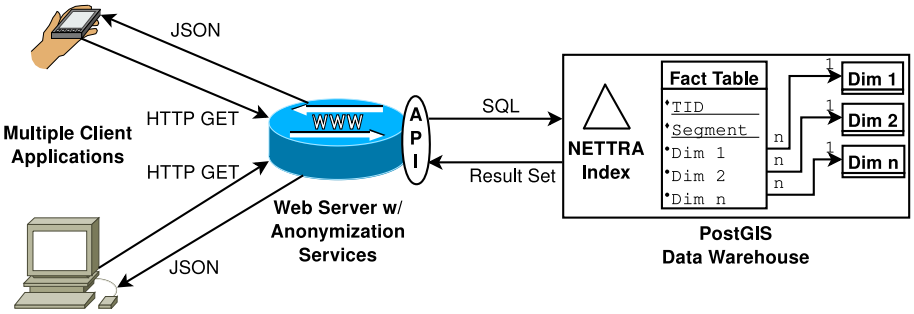


Fig. 13: Architecture of the API Demonstration [38]

6.2 Open Data for Trajectories

Even if user-specific identifiers are removed from trajectory data, drivers can often still be identified by trajectory pattern matching, creating data privacy concerns when making trajectory data sets available. We therefore apply a method called *k*-anonymization that was first proposed by Samarati et al. [34] to our travel-time query results before releasing them. This method guarantees that data from a user cannot be distinguished from the data from at least $k-1$ other users.

The service implements a travel-time analysis service with our path-based API that translates service calls into SQL queries against the trajectory data warehouse. Table 2 describes the subset of the functions used to implement the anonymized travel-time service. Figure 14 shows how a RESTful service that retrieves travel-times of an (origin,destination) pair filtered by time (`tr`) and date range (`dr`) is implemented with the API. First, the two chosen coordinates are matched to their closest road segment. Then the most traveled path between those two segments is identified, and if there are enough trajectories, their travel-time data is retrieved and then anonymized [20] and provided to the user. If insufficient data is available to guarantee *k*-anonymity, no data is reported, to ensure user privacy.

Function	Arguments	Behavior
`NearestSegment`	`lon, lat`	Maps coordinates to the nearest road segment ID (`sid`)
`MostUsedRouteTrips`	`sid1, sid2, dr, tr`	Provides the number of trips on the most used path between two segments and its path identifier (`pid`) filtered by date (`dr`) and time range (`tr`)
`TripInfo`	`sid1, sid2, dr, tr`	Retrieves all trip data on the most used path between two segments filtered by date and time range

Table 2: Subset of the Path-based API Functions

6.3 Discussion

We demonstrate how the path-based API can be used to provide trajectory analysis applications to users while still ensuring the data privacy of the drivers. We do this by showcasing an intuitive map-based graphical user interface that allows to choose two points on the map and specify additional filters, e.g., weekdays, weather, and time of day. The relevant trajectories are

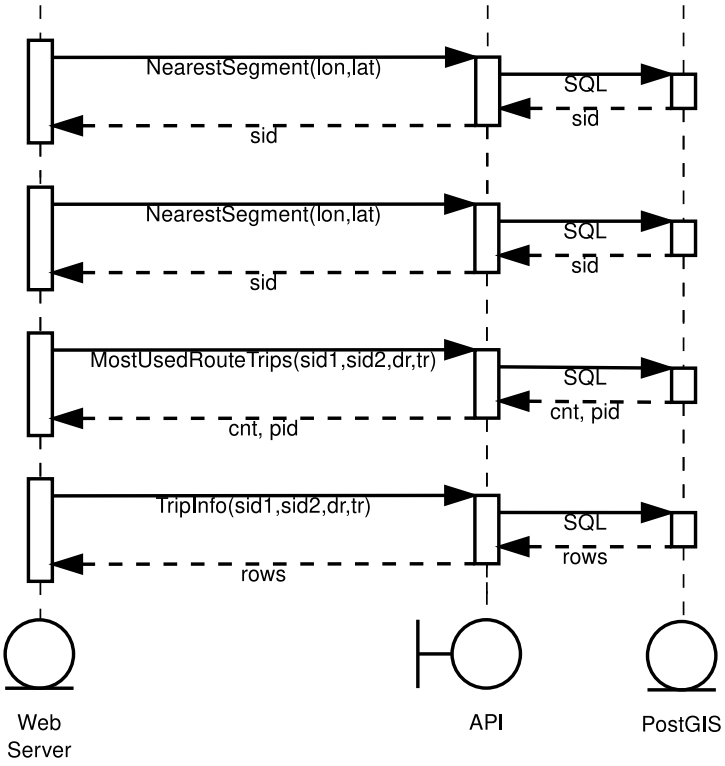


Fig. 14: Anatomy of the RESTful Service [38]

then retrieved from the database with the API, and the path and the trajectory data are then displayed alongside each other. The user can also modify the anonymization level to see the effect of parameter k.

7 Contributions

This thesis makes the case for custom selection of trajectories to achieve higher quality path-based travel-time estimates. Furthermore, it proposes a novel in-memory trajectory index to facilitate efficient fine-grained selection from big trajectory data sets, and provides detailed qualitative and quantitative experimental evaluations. Finally, it proposes a novel path-based API that allows these new selection methods for trajectory data to be integrated into user applications.

- Paper A [39] proposes the new "on-the-fly" collection approach for trajectory-based travel-time estimation and provides a preliminary ex-

perimental study of the new approach. The study shows that user-specific trajectory collection can improve travel-time estimation accuracy by up to 20% while providing real-time performance for single road segments and short paths.

- Paper B [41] extends on the previous study by providing a two-part metric to evaluate the quality of travel-time histograms and by conducting a study of different types of collection methods. The study evaluates the impact of the temporal, user, and weather predicates on the quality of travel-time estimation and shows an improvement in accuracy of up to 50%. The paper also proposes an architecture for a system processing "on-the-fly" queries.

- Paper C [40] proposes an new in-memory trajectory index structure integrated into the query processing system proposed in the previous paper. The system can process travel-time queries for any path in only a few milliseconds by implementing a best effort algorithm that always provides an estimate based on trajectory and map data. The paper presents a comprehensive runtime and qualitative analysis of the new index structure based on a real world data set.

- Paper D [42] proposes a parallelized version of the trajectory index structure proposed in the previous paper that is integrated into a trajectory store optimized for non-uniform memory access platforms. The paper discusses implementation details of the data-oriented architecture and provides a detailed runtime and scalability analysis of the trajectory store. We show that query latency is reduced by at least 50% with low load scenarios when compared to our baseline. The difference in latency between our NUMA-aware implementation and the baseline grows to over one order of magnitude in high load scenarios.

- Paper E [38] proposes a path-based API to perform travel-time queries on a trajectory data set. The usability of the API is demonstrated with a trajectory analysis use case implemented as a RESTful service. The implementation also allows users to evaluate the effect of anonymization on the quality of the analysis.

8 Conclusion

This thesis shows that travel-time estimation for road segments and paths can be improved by using on-the-fly methods that allow for a much more fine-grained trajectory selection than approaches relying on pre-computation. We verified our "on-the-fly" approach with one preliminary and two extensive qualitative studies. The studies also reveal that increasing the sample size

does not necessarily increase the quality of estimates. This is mostly due to more stale data being included into the result set when requiring higher sample sizes, especially with queries containing very selective filter predicates.

To efficiently process complex path-based queries, we propose a novel network-constrained trajectory index structure that allows efficient in-memory processing of strict path queries. The index consists of two main data structures: (1) a space and runtime efficient spatial index and (2) a collection of temporal indexes necessary for temporal filtering. The new index structure is integrated into a travel-time estimation system that answers queries by rewriting them into a series of smaller strict path sub-queries using a greedy algorithm. This allows us to return a travel-time histogram covering the full path of a query even with sparse trajectory data sets. We find that applying complex filter predicates selectively, e.g., only using user filters on motorway paths, yields the most accurate estimates while also considerably reducing the runtime of sub-queries. The runtime is reduced by a further 50% by a novel cardinality estimator that uses the spatial index and segment-level histograms to allow the system to avoid costly scans of the temporal indexes.

To show that this system can be scaled up, we integrate our new index structure into a scalable trajectory store optimized for modern NUMA systems. The NUMA-aware system allows travel-time queries and trajectory inserts to be processed concurrently. We perform an extensive performance study of the system and compare it to a version that is not NUMA-aware. The NUMA-aware system provides 50% lower query latency in a low load setting. This difference in latency grows to over 90% under high load. This is particularly interesting since the code footprint of the NUMA-aware features is comparatively small.

Finally, we suggest a path-based API that makes path-based trajectories more accessible to users. We demonstrate the applicability of the API by means of a service that allows anonymized travel-time analyses on a map-based user interface.

Our work leaves multiple avenues of future work. The suggested travel-time estimation method could be extended to include an adaptive component that throughout a trip continuously updates the estimate based on real-time data and current driving behavior. It could also be augmented further by methods addressing data sparseness to not only base estimates for segments without any trajectory data on the speed limit exclusively, e.g., the approach suggested by Zheng and Van Zuylen [44]. The proposed trajectory store could be improved upon by more advanced partitioning of the temporal workers, e.g., load-based re-partitioning which can also utilize information about previous queries to place frequent build and probe segment pairs in the same NUMA region to further increase memory locality.

References

[1] 14th Weather Squadron Fleet Numerical Meteorology and Oceanography Detachment, "Federal Climate Complex Data Documentation For Integrated Surface Data," National Climatic Data Center, Asheville, NC 28801-5001 USA, Tech. Rep., 2016.

[2] O. Andersen and K. Torp, "Sampling Frequency Effects on Trajectory Routes and Road Network Travel Time," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, no. 30. ACM, 2017.

[3] J. S. Armstrong, *Long-Range Forecasting: From Crystal Ball to Computer*, 2nd ed. John Wiley & Sons, Inc., 1985.

[4] J. L. Borresen, C. S. Jensen, and K. Torp, "FoGBAT: Combining Bluetooth and GPS Data for Better Traffic Analytics," in *Proceedings of the 17th IEEE International Conference on Mobile Data Management*. IEEE, 2016, pp. 325–328.

[5] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu, "Path Cost Distribution Estimation Using Trajectory Data," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 85–96, 2016.

[6] Z. Ding, "UTR-tree: An Index Structure for the Full Uncertain Trajectories of Network-Constrained Moving Objects," in *Proceedings of the 9th IEEE International Conference on Mobile Data Management*. IEEE, 2008, pp. 33–40.

[7] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro, "An Alphabet-Friendly FM-index," in *Proceedings of the International Symposium on String Processing and Information Retrieval*. Springer, 2004, pp. 150–160.

[8] E. Frentzos, "Indexing Objects Moving on Fixed Networks," in *Proceedings of the 8th International Symposium on Spatial and Temporal Databases*. Springer, 2003, pp. 289–305.

[9] P. P. Gelsinger, "Microprocessors for the new millennium: Challenges, opportunities, and new frontiers," in *Proceedings of the 2001 IEEE International Solid-State Circuits Conference. Digest of Technical Papers. ISSCC (Cat. No. 01CH37177)*. IEEE, 2001, pp. 22–25.

[10] 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 *Proceedings of the 15th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2012, pp. 776–781.

[11] Google Inc., "cpp-btree," https://code.google.com/archive/p/cpp-btree/, 2011.

[12] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*. ACM, 1984, pp. 47–57.

[13] D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing Cache Architectures and Coherency Protocols on x86-64 Multicore SMP Systems," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2009, pp. 413–422.

[14] G. Juckeland, S. Börner, M. Kluge, S. Kölling, W. E. Nagel, S. Pflüger, H. Röding, S. Seidl, T. William, and R. Wloch, "BenchIT - Performance Measurements and Comparison for Scientific Applications," *Advances in Parallel Computing*, vol. 13, pp. 501–508, 2004.

[15] T. Kissinger, T. Kiefer, B. Schlegel, D. Habich, D. Molka, and W. Lehner, "ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workloads," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, pp. 1–12, 2014.

[16] A. Kleen, "An NUMA API for Linux," SUSE Labs, Tech. Rep., 2004.

[17] S. Koide, Y. Tadokoro, and T. Yoshimura, "SNT-index: Spatio-temporal index for vehicular trajectories on a road network based on substring matching," in *Proceedings of the 1st International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. ACM, 2015, pp. 1–8.

[18] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp, "Path-based Queries on Trajectory Data," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2014, pp. 341–350.

[19] C. Lameter *et al.*, "NUMA (Non-Uniform Memory Access): An Overview." *ACM Queue*, vol. 11, no. 7, p. 40, 2013.

[20] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Mondrian Multidimensional K-Anonymity," in *Proceedings of the 22nd IEEE International Conference on Data Engineering*. IEEE, 2006, pp. 25–35.

[21] V. Leis, P. Boncz, A. Kemper, and T. Neumann, "Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 2014, pp. 743–754.

[22] Y. Ma, B. Yang, and C. S. Jensen, "Enabling Time-Dependent Uncertain Eco-Weights For Road Networks," in *Proceedings of the Workshop on Managing and Mining Enriched Geo-Spatial Data*. ACM, 2014, pp. 1–6.

[23] G. Moerkotte, T. Neumann, and G. Steidl, "Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 982–993, 2009.

[24] G. E. Moore, ""Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[25] P. Newson and J. Krumm, "Hidden Markov Map Matching Through Noise and Sparseness," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2009, pp. 336–343.

[26] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org," https://www.openstreetmap.org, 2014.

[27] I. Pandis, R. Johnson, N. Hardavellas, and A. Ailamaki, "Data-Oriented Transaction Execution," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 928–939, 2010.

[28] C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision," in *Proceedings of the 17th ACM International World Wide Web Conference*, 2008, pp. 805–814.

[29] F. C. Pereira, H. Costa, and N. M. Pereira, "An off-line map-matching algorithm for incomplete map databases," *European Transport Research Review*, vol. 1, no. 3, pp. 107–124, 2009.

[30] I. S. Popa, K. Zeitouni, V. Oria, D. Barth, and S. Vial, "PARINET: A Tunable Access Method for in-Network Trajectories," in *Proceedings of the 26th IEEE International Conference on Data Engineering*. IEEE, 2010, pp. 177–188.

[31] PostGIS contributors, "PostGIS: Spatial and Geographic objects for PostgreSQL," https://postgis.net/, 2019.

[32] J. Rao and K. A. Ross, "Cache Conscious Indexing for Decision-Support in Main Memory," in *Proceedings of the 25th International Conference on Very Large Databases*, 1999, pp. 78–89.

[33] Ritchie, Hannah and Roser, Max, "CO2 and Greenhouse Gas Emissions," https://ourworldindata.org/co2-and-other-greenhouse-gas-emissions, 2018.

[34] P. Samarati and L. Sweeney, "Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression," SRI International, Tech. Rep., 1998.

[35] J. G. Saw, M. C. Yang, and T. C. Mo, "Chebyshev Inequality With Estimated Mean and Variance," *The American Statistician*, vol. 38, no. 2, pp. 130–132, 1984.

[36] D. Wang, J. Zhang, W. Cao, J. Li, and Y. Zheng, "When Will You Arrive? Estimating Travel Time Based on Deep Neural Networks," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. AAAI Press, 2018, pp. 2500–2507.

[37] Y. Wang, Y. Zheng, and Y. Xue, "Travel Time Estimation of a Path using Sparse Trajectories," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2014, pp. 25–34.

[38] R. Waury, P. Dolog, C. S. Jensen, and K. Torp, "Analyzing Trajectories Using a Path-based API," in *Proceedings of the 16th International Symposium on Spatial and Temporal Databases*, 2019, pp. 198–201.

[39] R. Waury, J. Hu, B. Yang, and C. S. Jensen, "Assessing the Accuracy Benefits of On-the-Fly Trajectory Selection in Fine-Grained Travel-Time Estimation," in *Proceedings of the 18th IEEE International Conference on Mobile Data Management*. IEEE, 2017, pp. 240–245.

[40] R. Waury, C. S. Jensen, S. Koide, Y. Ishikawa, and C. Xiao, "Indexing Trajectories for Travel-Time Histogram Retrieval," in *Proceedings of the 22nd International Conference on Extending Database Technology*, 2019, pp. 157–168.

[41] R. Waury, C. S. Jensen, and K. Torp, "Adaptive Travel-Time Estimation: A Case for Custom Predicate Selection," in *Proceedings of the 19th IEEE International Conference on Mobile Data Management*. IEEE, 2018, pp. 96–105.

[42] ——, "A NUMA-aware Trajectory Store for Travel-Time Estimation," in *Accepted in ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2019.

[43] B. Yang, M. Kaul, and C. S. Jensen, "Using Incomplete Information for Complete Weight Annotation of Road Networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 5, pp. 1267–1279, 2014.

[44] F. Zheng and H. Van Zuylen, "Urban link travel time estimation based on sparse probe vehicle data," *Transportation Research Part C: Emerging Technologies*, vol. 31, pp. 145–157, 2013.

Part II

Papers

Paper A

Assessing the Accuracy Benefits of On-The-Fly Trajectory Selection in Fine-Grained Travel-Time Estimation

Robert Waury, Jilin Hu, Bin Yang, Christian S. Jensen

The paper has been published in
Proceedings of the 18th IEEE International Conference on Mobile Data Management, pp. 240–245, 2017.
DOI: 10.1109/MDM.2017.40

Abstract

Today's one-size-fits-all approach to travel-time computation in spatial networks proceeds in two steps. In a preparatory off-line step, a set of distributions, e.g., one per hour of the day, is computed for each network segment. Then, when a path and a departure time are provided, a distribution for the path is computed on-line from pertinent pre-computed distributions. Motivated by the availability of massive trajectory data from vehicles, we propose a completely on-line approach, where distributions are computed from trajectories on-the-fly, i.e., when a query arrives. This new approach makes it possible to use arbitrary sets of underlying trajectories for a query. Specifically, we study the potential for accuracy improvements over the one-size-fits-all approach that can be obtained using the on-the-fly approach and report findings from an empirical study that suggest that the on-the-fly approach is able to improve accuracy significantly and has the potential to replace the current one-size-fits-all approach.

1 Introduction

A range of services related to road networks rely on the computation of travel-times of paths [1]. For example, this applies to vehicle routing services that compute fastest paths and to services that pre-compute payments for transportation based on estimated travel-times.

The travel-time of a path is time-varying due to congestion and external conditions such as precipitation and road-surface conditions; and travel-time also varies across drivers.

Due to the proliferation of GPS data from vehicles, travel-time estimation is increasingly being based on vehicle trajectory data. As more and more such data becomes available, we believe that it is becoming feasible to estimate travel-times in a much more fine-grained manner than what is possible with today's state-of-the-art approach, which is intuitively a one-size-fits-all approach.

The one-size-fits-all approach pre-computes histograms off-line for each road network segment based on travel-times extracted from trajectories that traversed the segment. Specifically, this approach typically partitions a day into intervals, e.g., 15-minute, 30-minute, or 1-hour intervals, and computes a histogram for each interval based on trajectories that occurred during that interval. It is also possible to pre-compute histograms for consecutive segments, i.e., paths [2, 3]. When a query in the form of a path and a departure time are provided, this approach produces a travel-time histogram for the traversal by combining pertinent pre-computed histograms. The off-line computation of histograms offers efficiency, but it also restricts flexibility, as all queries must use the pre-computed histograms.

Instead, we propose an on-line approach, where histograms are computed on-the-fly at query time. While this may reduce running time efficiency, it is much more flexible: depending on the query, any subset of relevant trajectories can be used to form a histogram for segments and paths. We hypothesize that this approach has the potential to improve the accuracy of travel-time distribution estimates substantially over the one-size-fits-all approach. Put differently, we believe that if we choose the "right" trajectories from which to compute histograms, we get much more accurate histograms than is possible with the one-size-fits-all approach.

To motivate this study, consider the road network of Denmark, which consists of around 1.6 million edges. Pre-computing histograms for all potentially interesting subsets of trajectories, e.g., one set for every driver in personalized routing [4, 5], is infeasible as exponentially many such subsets may exist.

The paper aims to provide insight into the potential of on-the-fly travel-time histogram computation. To do this, we define several specific proce-

dures for histogram computation that exemplify both the one-size-fits-all approach and the on-the-fly approach. We then detail a methodology for comparing the different procedures empirically using GPS data. In particular, we build histograms for a road network using some of the data. Then we use trajectories from the remaining data for testing. Intuitively, we provide means of determining which histograms best predict the travel-times seen in the test trajectories. Finally, we apply the methodology to a substantial GPS dataset and report findings that suggest that on-the-fly histogram computation indeed has potential to improve the accuracy of travel-time distributions in road networks.

The paper's contributions are as follows: (i) we motivate and identify a new paradigm for travel-time distribution computation; (ii) we define specific procedures for travel-time histogram computation; (iii) we identify a methodology for empirical evaluation of the accuracy of histograms; and (iv) we report findings from an empirical study with a substantial GPS dataset.

The remainder of the paper is structured as follows. Section 2 describes the paper's setting and trajectory selection criteria and defines the histogram computation techniques to be compared, Section 3 details the empirical evaluation methodology, and Section 4 reports on the empirical study. Finally, Section 5 concludes.

2 Travel-Time Estimation Methods

This section provides the setting of our study and describes four histogram computation methods.

2.1 Trajectories in Spatial Networks

A spatial network, as shown in Figure A.1, is a weighted directed graph $G = (V, E, l)$, where V is a vertex set, $E \subseteq V \times V$ is an edge set, and $l : E \to \mathbb{R}$ is a length function. Every edge $e \in E$ represents a road segment.

A trajectory tr in a spatial network is represented as a tuple.

$$tr = (a, s),$$

where $a = (a_1, .., a_k)$ is a tuple of trajectory-specific information like driver and vehicle ID and s is a sequence of triples (t, e, p), where t is a timestamp, $e \in E$ is a segment in the spatial network G, and p is the vehicle's position on segment e, given as a distance from the start of the segment, meaning that $\forall tr\ \forall (t, e, p) \in tr.s\ (p \leq l(e))$. Table A.1 provides five example trajectories.

A traversable sequence of neighboring segments is called a path. Examples of paths include $\langle e_1, e_5, e_8 \rangle$ and $\langle e_1, e_2, e_4 \rangle$. Path-based traffic information can provide more detailed insights into travel-time, e.g., if a path contains

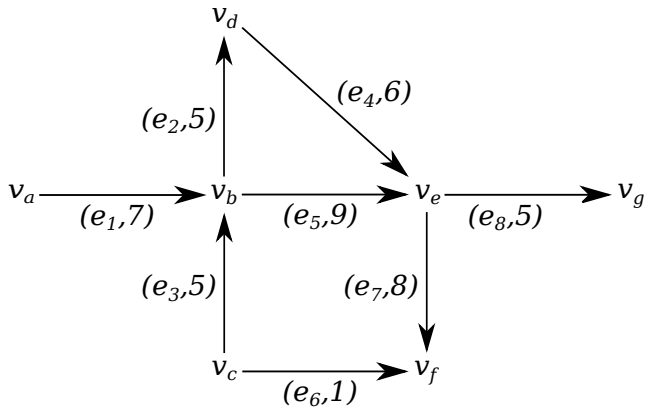


Fig. A.1: Example Road Network Graph

ID	user	s
tr_1	A	$\langle(1,e_1,3),(2,e_1,6),(3,e_5,4),(4,e_8,5)\rangle$
tr_2	A	$\langle(21,e_1,4),(22,e_5,3),(23,e_8,4)\rangle$
tr_3	A	$\langle(36,e_1,4),(37,e_2,3),(38,e_4,3),(39,e_4,6)\rangle$
tr_4	B	$\langle(3,e_1,2),(4,e_1,4),(5,e_1,6),(6,e_5,3),(7,e_8,3)\rangle$
tr_5	B	$\langle(16,e_1,1),(17,e_1,4),(18,e_1,7),(19,e_3,3),(20,e_6,1)\rangle$

Table A.1: Example Trajectories

Method	Temporal Features	Path Cost	Predicate
LM	Static, Off-line	No	No
M2	Static, Off-line	Yes	No
M3	Dynamic, On-line	Yes	No
M4	Dynamic, On-line	Yes	Yes

Table A.2: A Summary of the four Methods

left turns where right of way has to be observed, path-based histograms can pick up those path-specific delays, whereas single-segment histograms would only give an average based on all directions.

2.2 Histogram Computation Techniques

Given a path $P = \langle e_1, e_2, \ldots, e_X \rangle$ and a departure time t, we apply four different methods to compute a travel-time distribution of using path P at t. We represent a distribution as a histogram and use $H_{P,t}$ to denote the travel-time histogram of traversing path P at t. A summary of the four methods is given in Table A.2.

Legacy Method (LM) [4, 6, 7]: In LM we partition every day into multiple equal-sized intervals, e.g., 96 15-min intervals or 24 1-hour intervals. For each segment e in the road network and for each interval I, we identify a set of trajectories $T_{e,I}$ that occurred on segment e during interval I. If the cardinality of set $T_{e,I}$ exceeds threshold β, we derive a histogram $H^{LM}_{e,I}$ from the trajectories in $T_{e,I}$, which represents the travel-time distribution of traversing e during interval I.

This procedure is conducted *off-line*, meaning that all histograms are precomputed once and for all.

To compute the travel-time distribution of P at t using LM, we first identify the interval I such that $t \in I$. Next, we are able to obtain relevant histograms for all segments in P, i.e., $H^{LM}_{e_1,I}$, $H^{LM}_{e_2,I}$, ..., $H^{LM}_{e_X,I}$. Finally, we convolve the segments' histograms to derive the histogram for path P at t:

$$H^{LM}_{P,t} = \odot^{X}_{i=1} H^{LM}_{e_i,I},$$

where $\odot$ is the convolution operator.

For example, for $P = \langle e_1, e_2, e_3, e_4 \rangle$, the travel-time distribution at t is computed as follows.

$$H^{LM}_{P,t} = H^{LM}_{e_1,I} \odot H^{LM}_{e_2,I} \odot H^{LM}_{e_3,I} \odot H^{LM}_{e_4,I}$$

Convolution of histograms has been used in multiple earlier approaches [8–10] and assumes independence between different segments, which often does not hold for all segments [6].

In LM, the intervals of the histograms that are use in convolution are pre-defined and not dependent on the departure time t. Thus LM is a *static* approach. Next, LM considers all trajectories and does not apply any predicates to further filter trajectories. Thus, LM is a *non-specific* approach. Finally, LM only assigns histograms to *segments*, but not to paths. This explains the "Path Cost" column in Table A.2.

Method 2 (M2) [2, 3]: We consider a static, ***path-based***, and non-specific method, M2.

M2 differs from LM in that it computes histograms not only for all segments, but also for some paths.

Specifically, if more than β trajectories occurred on path P_k during interval I, we maintain a histogram $H^{M2}_{P_k,I}$ for path P_k. The cardinality of any path P_k is typically small. The larger the cardinality of a path P_k, the less likely the path is to have more than β trajectories.

Similar to LM, the procedure is also conducted *off-line*.

To estimate the travel cost distribution of P at t, we first identify the interval I such that $t \in I$. Next, we choose the coarsest combination of available histograms of segments and subsequences of P and convolve them to obtain $H^{M2}_{P,t}$.

$$H^{M2}_{P,t} = \odot_{P_k \in CoarsestSet} H^{M2}_{P_k,I},$$

For example, let $P = \langle e_1, e_2, e_3, e_4 \rangle$. Assume that we have, in addition to the histograms of all segments, histograms $H^{M2}_{\langle e_1,e_2\rangle,I'}$, $H^{M2}_{\langle e_2,e_3\rangle,I'}$, $H^{M2}_{\langle e_1,e_2,e_3\rangle,I}$. Then, we have $CoarsestSet = \{\langle e_1, e_2, e_3\rangle, e_4\}$ since they together cover path P and $\langle e_1, e_2, e_3\rangle$ is coarser than paths $\langle e_1, e_2\rangle$ and $\langle e_2, e_3\rangle$. Thus, we have the following:

$$H^{M2}_{P,t} = H^{M2}_{\langle e_1,e_2,e_3\rangle,I} \odot H^{M2}_{e_4,I}.$$

Consider another scenario where we also have histogram $H^{M2}_{\langle e_1,e_2,e_3,e_4\rangle,I}$. Then $CoarsestSet = \{\langle e_1, e_2, e_3, e_4\rangle\}$, and thus the histogram can be returned directly, and no convolution is needed.

Method 3 (M3): We consider a ***dynamic***, path-based, and non-specific method, M3.

In M3, we consider dynamically constructed intervals that are dependent on the given departure time t. Specifically, we construct interval $I' = [t - \frac{\alpha}{2}, t + \frac{\alpha}{2},]$, where α is an interval parameter. For example, if $\alpha = 60$ minutes, we construct interval $I' = [9{:}06, 10{:}06]$ if the departure time t is 9:36. We only consider the trajectories that occurred during I' to derive histograms for segments or paths.

In addition, when we consider building histograms for the segments and paths that do not contain the first segment in the given path P, we use a *shift-and-enlarge* procedure [2] that shifts interval I' by the minimum traversal time and enlarges I' by the maximum traversal time of the proceeding segments. For instance, assume that traversing $\langle e_1, e_2, e_3\rangle$ takes at least 2 minutes and at most 5 minutes. When retrieving the trajectories for building a histogram for segment e_4, we should then use a shifted-and-enlarged interval $I' = [9{:}08, 10{:}11]$.

This procedure is conducted *on-line*, since the dynamically constructed interval I' is based on the departure time t, meaning that the histograms cannot be pre-computed.

Finally, we compute the path cost distribution for path P at t as follows.

$$H^{M3}_{P,t} = \odot^{X}_{P_k \in CoarsestSet} H^{M3}_{P_k,I'}.$$

Method 4 (M4): We consider a dynamic, path-based, and ***specific*** method, M4.

In M4, we only consider the trajectories further that satisfy particular predicates during the dynamically constructed intervals. Such predicates help us choose trajectories under specific conditions and may enable more accurate travel-time estimation when such specific conditions apply.

For example, a predicate defined on drivers enables us to consider trajectories from only a specific driver. Alternatively, a predicate defined on weather enables us to only consider trajectories from different weather conditions, e.g., rainy and icy conditions. This procedure is also on-line. We use

$H^{M4}_{P_k,I',f}$ to denote the histogram of path P_k that is derived from trajectories that satisfy predicate f during interval I'. Then, we have the following:

$$H^{M4}_{P,t,f} = \odot^{X}_{P_k \in CoarsestSet} H^{M4}_{P_k,I',f}$$

For our example in Figure A.1 and Table A.1, the histogram $H^{M4}_{P,t,f}$ with $P = \langle e_1, e_5, e_8 \rangle$, $t = 4$, $\alpha = 6$, and $f =$ "$user = B$" would be based on the trajectory set $\{tr_4\}$, and $H^{M3}_{P,t}$ would be based on $\{tr_1, tr_4\}$.

3 Experimental Design

This section explains how we evaluate the accuracy of the histograms computation techniques from Section 2.2. The objective of the study is to assess the potential for higher travel-time distribution prediction accuracy of histograms constructed using the on-the-fly approach. In order to assess the accuracy benefits of on-the-fly approach, we utilize the concept of likelihood.

3.1 Experimental Setup

Since our method is expected to increase predictive performance, we consider trajectories from a real-world dataset (cf. Section 4.1). To evaluate our results, we first store all trajectories from the first year (until December 2007) in a temporal index. All trajectories for the following year are initially held back and are then inserted in order. From those, k trajectories tr_q with a duration of at, least 120 seconds are randomly selected, and a query $Q = (P, t, \alpha, R, f)$ is generated from them before they are inserted. P is the path for which the histograms are computed, t is the center of an interval I, α is the interval parameter, R is the recurrence of the interval, and f is a predicate. The query is run with $t = t_1$, where t_1 is the first timestamp of each tr_q, P is the sequence of segments traversed in $tr_q.s$, and f is the predicate used for M4 $user \in tr_q.a$. For α and R, multiple different values are considered.

We compare all methods introduced in Section 2.2. In addition to the above parameters, the parameters $d \geq 1$ and $\beta \geq 1$ are specified. Parameter d denotes the maximum cardinality of paths for which traversal times are collected and β denotes the minimum sample size. The histogram collection therefore returns up to d histograms for every segment in S. If the number of trajectories traversing them is below β, they yield empty histograms.

3.2 Likelihood Ratio

Since our method aims to improve travel-time predictions, we evaluate the method introduced in Section 2.2 by considering the likelihood the retrieved

histograms for our sample trajectory tr_q would have predicted an actual traversal time TT_e of each segment or path in $tr_q.s$.

To achieve this, we first define a discrete probability density function p_H derived from a histogram H:

$$p_H(TT_e) = \frac{m(TT_e)}{n},$$

where $m : \mathbb{R} \to \mathbb{N}$ is a function that maps a traversal time to the number of trajectories that traversed the segment or path of the histogram within the time range of the bin of TT_e. Further, $n = \sum m_i$ is the number of trajectories sampled in all bins of the histogram.

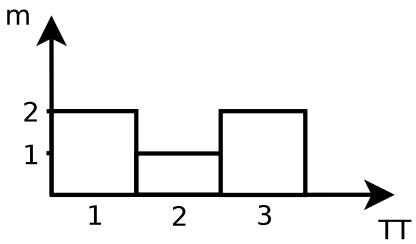


Fig. A.2: Histogram for e_1 in G

The histogram for e_1 from our example trajectories in Table A.1 is shown in Figure A.2, where $m(1) = 2$, $m(2) = 1$, $m(3) = 2$, and $n = 5$. It follows that $p_H(1) = \frac{2}{5}$, $p_H(2) = \frac{1}{5}$, and $p_H(3) = \frac{2}{5}$.

Next, from the probability density function p_H, the likelihood $\mathcal{L}(H|TT_e) = p_H(TT_e)$ of the traversal time TT_e is computed. If likelihood for the M3 dynamic histograms $\mathcal{L}_{M3} = \mathcal{L}(H^{M3}|TT_{e_i})$ consistently improves compared to the LM static approaches $\mathcal{L}_{LM} = \mathcal{L}(H^{LM}|TT_{e_i})$, the on-the-fly approach can provide more accurate travel-time estimations.

The likelihood ratio $\varkappa = \frac{\mathcal{L}_{M3}}{\mathcal{L}_{LM}}$, where $\mathcal{L}_{LM}$ is the likelihood derived from the convolved static histogram, and $\mathcal{L}_{M3}$ is the likelihood derived from the dynamic one, is then computed from the respective histograms.

If $\frac{\mathcal{L}_{M3}}{\mathcal{L}_{LM}} > 1$ then the likelihood for the dynamic approach is higher and therefore is a better predictor of travel-time. The same comparison is performed for the likelihood of the M4 dynamic user histogram $\mathcal{L}_{M4}$, and for the M2 non-convolved static histogram $\mathcal{L}_{M2}$.

4 Empirical Study

We evaluate our results by examining the accuracy of travel-time estimation based on a real-life trajectory dataset. In the study, we exclude traversal times

for the first and last segments of trajectories to avoid any outliers that may exist for such segments. In the experiments, we only consider paths with cardinalities up to 3 since the share of empty results for M4 exceeds 30% in our dataset after that. We use a sample size of $k = 1,000$ trajectories, and we use equi-width histograms with bin width $h = 1s$.

4.1 Young Drivers Dataset

The Young Drivers dataset covers ca. 670,000 trajectories within Aalborg and the surrounding area during the period from December 2006 to December 2008 and comprises over 100 million map-matched GPS records [11] sampled at 1 Hertz [12]. The data was collected from private cars, so all trajectories with the same vehicle ID are assumed to be generated by the same driver. This allows us to use the vehicle ID as a predicate in Method 4. Apart from timestamp, trajectory ID, and vehicle ID, the records also contain a segment ID and the vehicle's position on the segment. These records are grouped by trajectory ID, sorted by timestamp, and then aggregated to form a trajectory set TR as defined in Section 2.1.

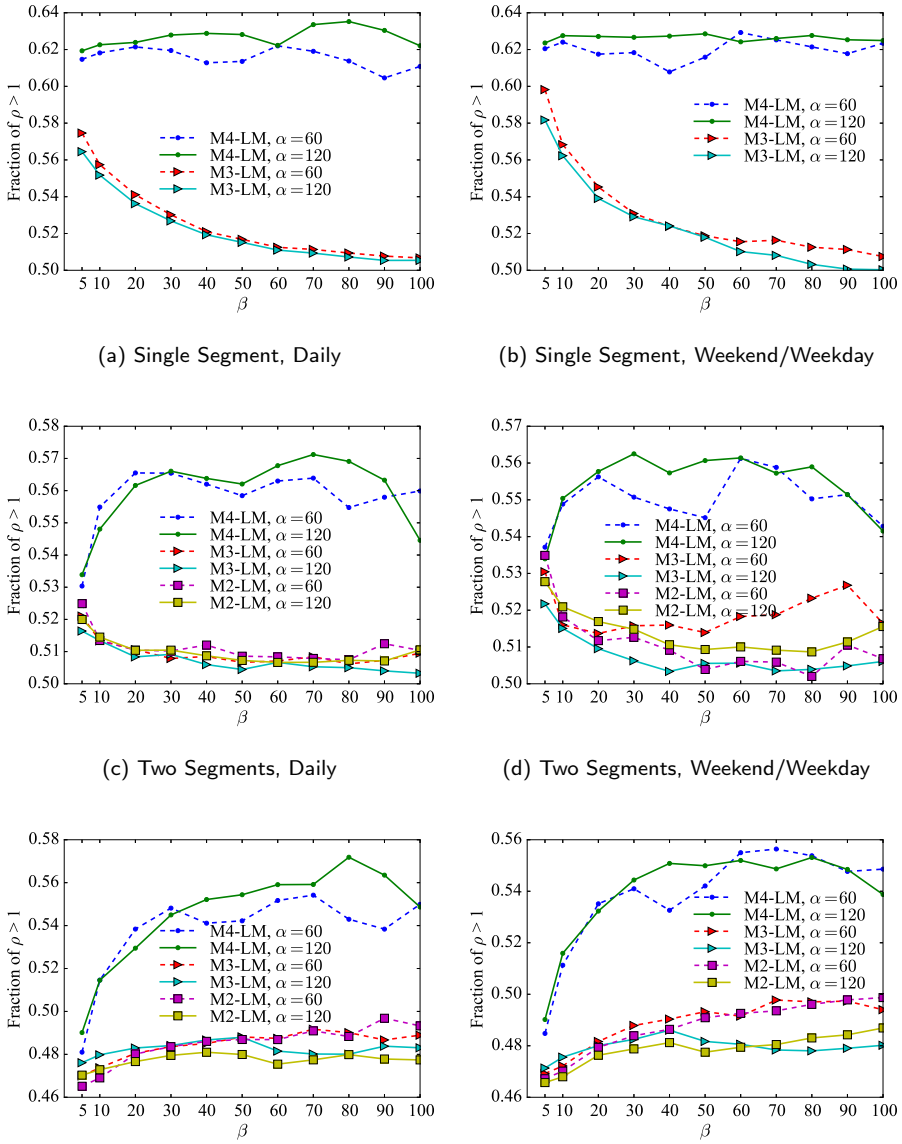
4.2 Histogram Accuracy

We first consider the accuracy of the advanced methods when compared to the baseline LM. Specifically, we measure the likelihood ratios $æ_{M2} = \frac{\mathcal{L}_{M2}}{\mathcal{L}_{LM}}$, $æ_{M3} = \frac{\mathcal{L}_{M3}}{\mathcal{L}_{LM}}$, and $æ_{M4} = \frac{\mathcal{L}_{M4}}{\mathcal{L}_{LM}}$ of the histograms computed by the different methods. When the fraction of likelihood ratios above 1 exceeds 0.5 substantially, this suggests that the method represented by the nominator is an improvement over the method represented by the denominator.

Figures A.3a, A.3c, and A.3e show how often the likelihood of M2, M3, and M4 histograms improves upon the likelihood of the LM histograms for single segments and paths of length 2 and 3 when selecting trajectories in I for every day, with α set to either 60 or 120 minutes and when varying β from 5 to 100 trajectories.

Similarly, Figures A.3b, A.3d, and A.3f show the fraction of likelihood ratios above 1 when using trajectories from all weekdays or all weekends. Specifically, if departure time t falls on a weekday, all trajectories within I on weekdays are considered, and if it falls on a Saturday or Sunday, all trajectories within I on weekends are considered. For the single segment results reported in Figures A.3a and A.3b, the difference between convolved (LM) and non-convolved (M2) static is omitted since no convolution occurs.

From the figures, we see that the M3 histograms, despite considering a larger time range due to the *shift-and-enlarge* procedure, provide no noticeable improvement on the LM histograms. The same observation holds for the



(a) Single Segment, Daily

(b) Single Segment, Weekend/Weekday

(c) Two Segments, Daily

(d) Two Segments, Weekend/Weekday

(e) Three Segments, Daily

(f) Three Segments, Weekend/Weekday

Fig. A.3: Histogram Accuracy

M2 histograms. However, the M4 histograms exhibit a consistent improvement, especially for $\beta \geq 10$. We also see that for the non-specific histograms,

$\alpha = 60$ performs better than the larger time range in most cases. For the M4 histograms, no such pattern can be observed. While outperforming the LM baseline for single segment histograms, the other two non-specific methods are only slightly better than the baseline method for two-segment histograms (cf. Figures A.3c and A.3d), and both methods are slightly worse than the baseline for three-segment histograms (cf. Figures A.3e and A.3f). With single segments as well as paths, the M4 histograms consistently outperform their non-specific counterparts; and for paths, they are the only methods that manage to improve significantly on the baseline. The improvement remains consistent even for larger sample sizes for the single segment histogram methods as well as for the path-based methods.

4.3 Fractions of Non-Empty Results

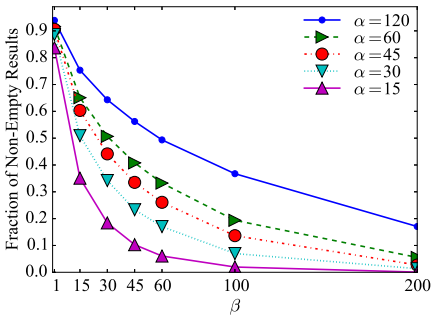
Next, we evaluate how significantly filtering by user reduces the sample that is considered for histogram construction. This is to understand the trade-off between higher sample size and "fit" of the trajectories that is made when using the different methods. To evaluate the impact, we look at how often queries return non-empty histograms for segments in S, i.e., when the number of trajectories $n \geq \beta$. Figures A.4a–A.4c show how often queries with $d = 1$ and a daily interval return non-empty results with our dataset for different segments given different time ranges α and minimum sample sizes β. The fractions of non-empty returns of the M3 method (cf. Figure A.4b) are slightly higher than those of the static methods (cf. Figure A.4a), which is to be expected due to the *shift-and-enlarge* procedure that increases the time range for every subsequent segment.

The fraction of non-empty returns for M4, shown in Figure A.4c, is significantly lower than for the two other histogram types, since single drivers conduct considerably fewer trips than the whole population of drivers. Datasets as sparse as the Young Drivers dataset would therefore require the histograms computed with the other methods as fall-back from M4 to provide sufficient coverage of segments and paths.

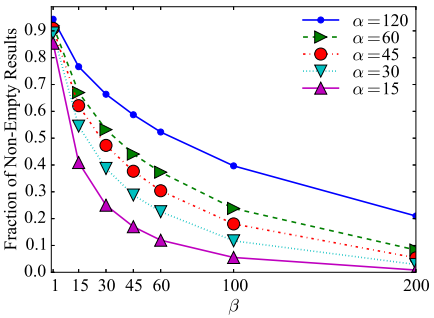
4.4 Computational Efficiency

While computational efficiency is not the focus of this paper, but is rather a different subject to be considered after the improved accuracy of the on-the-fly approach is established, current results are encouraging.

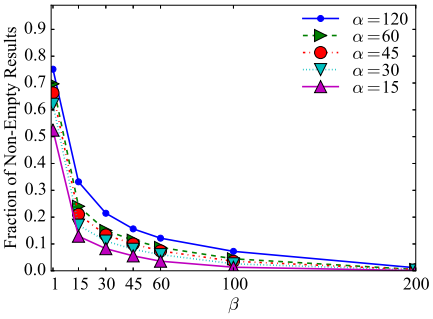
With our in-memory temporal index, the retrieval of all M3 histograms over a period between one and two years, with $d = 3$ took between 1.4 and 2.5 ms per trajectory for the daily and weekend/weekday interval, and between 0.2 and 0.5 ms for the weekly interval. For the M4 histograms, the retrieval times fell from 0.9 to 1.5 ms and from 0.15 to 0.25 ms, respectively.



(a) LM/M2 Non-Empty Result Fraction



(b) M3 Non-Empty Result Fraction



(c) M4 Non-Empty Result Fraction

Fig. A.4: Fraction of Non-Empty Results

5 Conclusion

The empirical study indicates that on-the-fly histogram construction is able to provide considerable accuracy improvements over the state-of-the-art, even for sparser datasets. The evaluation also shows that the improvements are consistent across smaller as well as larger sample sizes. With GPS data becoming more and more abundant, on-the-fly histogram construction will become even more attractive in the future.

Acknowledgments

This research was supported in part by a grant from the Obel Family Foundation and by the DiCyPS project.

References

[1] C. Guo, C. S. Jensen, and B. Yang, "Towards Total Traffic Awareness," *SIGMOD Record*, vol. 43, no. 3, pp. 18–23, 2014.

[2] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu, "Path Cost Distribution Estimation Using Trajectory Data," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 85–96, 2016.

[3] J. Hu, B. Yang, C. S. Jensen, and Y. Ma, "Enabling time-dependent uncertain eco-weights for road networks," *GeoInformatica*, vol. 21, no. 1, pp. 57–88, 2017.

[4] B. Yang, C. Guo, Y. Ma, and C. S. Jensen, "Toward personalized, context-aware routing," *The VLDB Journal*, vol. 24, no. 2, pp. 297–318, 2015.

[5] J. Dai, B. Yang, C. Guo, and Z. Ding, "Personalized route recommendation using big trajectory data," in *Proceedings of the 31st IEEE International Conference on Data Engineering*, 2015, pp. 543–554.

[6] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang, "Stochastic Skyline Route Planning Under Time-Varying Uncertainty," in *Proceedings of the 30th IEEE International Conference on Data Engineering*, 2014, pp. 136–147.

[7] C. Guo, B. Yang, O. Andersen, C. S. Jensen, and K. Torp, "EcoSky: Reducing Vehicular Environmental Impact Through Eco-Routing," in *Proceedings of the 31st IEEE International Conference on Data Engineering*, 2015, pp. 1412–1415.

[8] A. Chen and Z. Ji, "Path Finding Under Uncertainty," *Journal of Advanced Transportation*, vol. 39, no. 1, pp. 19–37, 2005.

[9] E. Nikolova, M. Brand, and D. R. Karger, "Optimal Route Planning under Uncertainty," in *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*. AAAI Press, 2006, pp. 131–140.

[10] S. Lim, C. Sommer, E. Nikolova, and D. Rus, "Practical Route Planning Under Delay Uncertainty: Stochastic Shortest Path Queries," in *Robotics: Science and Systems*, 2013, pp. 249–256.

[11] F. C. Pereira, H. Costa, and N. M. Pereira, "An off-line map-matching algorithm for incomplete map databases," *European Transport Research Review*, vol. 1, no. 3, pp. 107–124, 2009.

[12] B. Yang, M. Kaul, and C. S. Jensen, "Using Incomplete Information for Complete Weight Annotation of Road Networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 5, pp. 1267–1279, 2014.

Paper B

Adaptive Travel-Time Estimation: A Case for Custom Predicate Selection

Robert Waury, Christian S. Jensen, Kristian Torp

The paper has been published in
Proceedings of the 19th IEEE International Conference on Mobile Data Management, pp. 96–105, 2018.
DOI: 10.1109/MDM.2018.00026

Abstract

Travel-time estimation for paths in a road network often relies on pre-computed histograms that are usually available on a road segment level. Then the pre-computed histograms of the segments of a path are convolved to obtain a histogram that estimates the travel time. With the growing sizes of trajectory datasets, it becomes possible to compute histograms for increasingly longer sub-paths. Since pre-computation is infeasible for all sub-paths in a road network, we propose computing histograms on-the-fly, i.e., during routing. Such an on-the-fly method must filter the underlying trajectory dataset by spatio-temporal predicates to obtain the relevant trajectories and offers the opportunity to apply additional filtering predicates to the trajectories with little overhead. We report on a study showing that considerable improvements in accuracy of the histograms obtained for paths can be obtained by choosing filtering predicates that not only adapt to the intended start of a trip, but also to the driver and the weather. We also make the cases for a sub-path partitioning based on segment categories since there are significant differences between road types when applying our on-the-fly method.

1 Introduction

Travel times in road networks depend to a large degree on speed limits and congestion, but are also influenced by the vehicle type, weather conditions, and the driver's behavior.

Figure B.1 illustrates the main challenge addressed in this paper. A driver wants to go from Point A to Point B during non-rush hours and dry weather conditions and issues a query to a trajectory database. This database contains six trajectories labeled tr_1 to tr_6. For each of the trajectories, the weather condition, rush/non-rush, and the travel time from A to B is shown. As an example, tr_1 uses 200 seconds to go from A to B during rush hour and dry weather conditions.

The database contains no trajectories for the query that match exactly the route and the dry weather conditions and that occurred during non-rush hour. The challenge is then which trajectories to use for the travel-time estimation. Should the travel-time be estimated using tr_1 and tr_2, as they match the route and the weather conditions, but do not occur during rush hour? Or should tr_3 and tr_4 be used as they match the route, the rush-hour condition, but not the weather condition? Or should tr_5 and tr_6 be used as they match the rush hour and weather condition, but not the complete path? Or should all trajectories tr_1 to tr_6 be used?

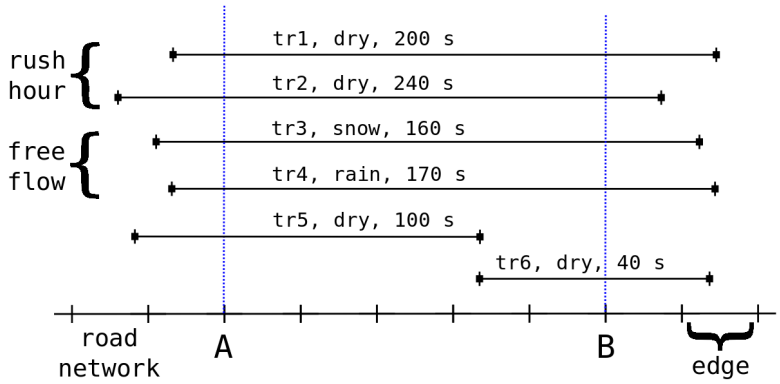


Fig. B.1: Trajectories on a Path

Current approaches to travel-time estimation base their estimates on all recorded trajectories crossing a path in a fixed time window. However, this arrangement only estimates the travel time of an average vehicle and fails to account for behavior related to specific circumstances and the driver.

The goal of this study is to extend an approach presented previously [1] to longer paths within the road network based on more detailed data and to focus on how segment categories and congestion influence travel time. With

this extended on-the-fly approach to trajectory selection, the considered data can be limited to the most relevant trajectories so that estimates based on unrelated or less relevant trajectories is avoided. Furthermore, we show that our on-the-fly approach can consider travel patterns specific to certain sub-path types, thereby increasing the overall accuracy of travel-time estimates. The amount of personalized GPS data can be expected to grow in the coming years, making our approach even more viable.

In an earlier study, we showed that the predictive performance of travel-time histograms can be improved with an on-the-fly approach as opposed to simply using pre-computed histograms [1]. This study, however, was limited to short paths only and used a small dataset. In this study, we consider how histogram quality is affected when our method is applied to longer paths and when categories of segments are accounted for. Since we are unable to make assumptions about the underlying distributions of the travel time, we cannot rely on proven methods like Student's one sample t-test to evaluate our results [2]. To address this limitation, we suggest a two-factor quality measurement, encompassing definitions of accuracy and precision, to evaluate histograms.

Section 2 reviews related work, provides an overview of core concepts, and defines the problem. Section 3 provides details about our collection method and gives a detailed example. Next, Section 4 describes the experimental setting and our baseline, and Section 5 reports our results. Section 6 concludes and provides an outlook for future research.

2 Preliminaries and Problem Formulation

This section introduces basic concepts related to travel-time estimation in spatial networks and formulates the problem we are addressing.

2.1 Related Work

Previous studies on travel-time estimation either estimate segment traversal times and turn costs [3] or estimate histograms for single segments or short pre-defined paths with considerable traffic [4, 5]. In our approach, travel-times are computed for paths instead of only for individual segments, which eliminates the need to estimate turn costs. To our knowledge, no previous work has considered different segment categories or suggested two-factor quality metrics that can be applied to travel-time estimations of single trajectories.

2.2 Trajectories in Spatial Networks

A spatial network is modeled as a directed graph $G = (V, E)$, where V is a vertex set and $E \subseteq V \times V$ is a set of edges that represent road segments. Every edge $e \in E$ has a category that captures the road type of the segment it represents. A traversable sequence of segments $P = \langle e_1, e_2, ..., e_l \rangle$ is called a path, with $|P| = l$. Its sub-paths $\langle e_i, ..., e_j \rangle$, with $1 \leq i \leq j \leq l$, are denoted as $P[i, j]$. The set of trajectories we call T. A trajectory $tr \in T$ of a user u in a spatial network is denoted as $(tr_x, u) \rightarrow s$, where tr_x is the trajectory identifier and s is a sequence of 4-tuples:

$$s = \langle (e_1, t_1, d_1, w_1), (e_2, t_2, d_2, w_2), ..., (e_l, t_l, d_l, w_l) \rangle,$$

where $e_i = (v_x, v_y) \in E$, $t_1, .., t_l$ are the timestamps when a segment was entered with $\forall i \forall j : i < j \Rightarrow t_i < t_j$, $d_i > 0$ is the duration of the traversal, w_i is the weather during t_i, and l is the number of segments traversed. The path of trajectory tr is called P_{tr}, and its starting time is $tr.t_1$. The duration function $D(tr, P) = d_i + d_{i+1} + ... + d_j$, with $P_{tr}[i, j] = P$, returns the traversal time TT_{tr}^P of a path P by a trajectory if tr does not fully traverse P, $D(tr, P)$ is undefined.

2.3 Problem Formulation

Our goal is to provide a histogram that estimates the travel-time of a trip on a pre-defined path dependent on the day of the week, the time of day, the driver, and environmental factors, e.g., the weather. To achieve that, we answer the path query $Q = (P, t, u, v)$. This query on a trajectory set takes as parameters a path P, a starting time t, user information u, and environmental factors v at t. It returns a trajectory set T^P of trajectories that best capture the travel-time of user with u on P during t given v. From this set, travel times $X^P = \{x_1, ..., x_\beta\}$ are extracted from which a histogram can be computed that approximates the travel-time distribution of a trip.

As the example in Figure B.1 shows, there are several possibilities to select a set of relevant trajectories. The easiest way would be to collect all trajectories that traverse any of the segments $e \in P$ and compute a travel-time histogram H_i for each segment. Then these can be convolved to obtain a travel-time histogram H for the complete path: $H = H_1 \odot H_2 \odot ... \odot H_l$. This approach has the advantage that the segment histograms can be pre-computed and updated easily. The histograms can also be pre-computed for discrete time intervals to be able to distinguish between rush hour and free-flow times. This approach, however, fails to consider factors such as turn costs, weather conditions, or user-specific behavior.

It is therefore advantageous to base histograms on trajectories that not only closely match user and environmental factors, but also whose paths

most closely resemble P. As mentioned in Section 2.2, segments have different properties depending on their category. The study presented in this paper aims to show which predicates result in the most relevant trajectory sets for travel-time estimates based on the different segment categories in a road network, such that the highest-quality travel-time histograms are computed.

3 Collection Method

To address the shortcomings of the segment level approach, we employ the strict path query $Q = (P, t, f)$ that returns a set of trajectories $T^P \subseteq T$ that follow all segments on the path P without stops or detours:

$$T^P = \{tr \in T | \exists i, j[P_{tr}[i, j] = P \wedge tr.t_i \in I_t \wedge f(u, w_i)]\},$$

where I_t is a temporal filter predicate derived from the starting time t and f is a set of filter predicates that trajectories in T^P have to fulfill.

Using such a query Q for an average trip path, which can consist of dozens of segments, is unlikely to return a sufficient number of trajectories. Instead, we need to split Q into k sub-queries $\langle Q_1, Q_2, ..., Q_k \rangle = \langle (P_1, t_1, f_1), (P_2, t_2, f_2), ..., (P_k, t_k, f_k) \rangle$ that return the trajectory sets $\langle T_1, T_2, ..., T_k \rangle$, where P_i are non-overlapping sub-paths of P, and $t_1 \leq t_2 \leq ... \leq t_k$.

3.1 Architecture

Figure B.2 shows the overall architecture of the implementation of our approach. After a user dispatches a query Q to the `Sub-query Module`, the query is initially partitioned according to a simple heuristic, e.g., sub-paths of a fixed length, or sub-paths that have the same segment category. Each of the k sub-queries is then assigned temporal and trajectory filter predicates. The `Trajectory Module` then runs each sub-query and returns the cardinalities of their trajectory sets $\langle |T_1|, |T_2|, ..., |T_k| \rangle$ to the `Sub-query Module`. The module then checks whether pre-defined sample size requirements are met (or significantly exceeded) in each sub-query result. If the sample size requirement is not met by a sub-query, the query can be modified in three ways:

- The trajectory filter predicates $\langle f_1, f_2, ..., f_k \rangle$ are relaxed.
- The temporal filter predicates $\langle I_1, I_2, ..., I_k \rangle$ are relaxed.
- The partitioning of the paths $\langle P_1, P_2, .., P_k \rangle$ is changed by splitting it into smaller sub-paths.

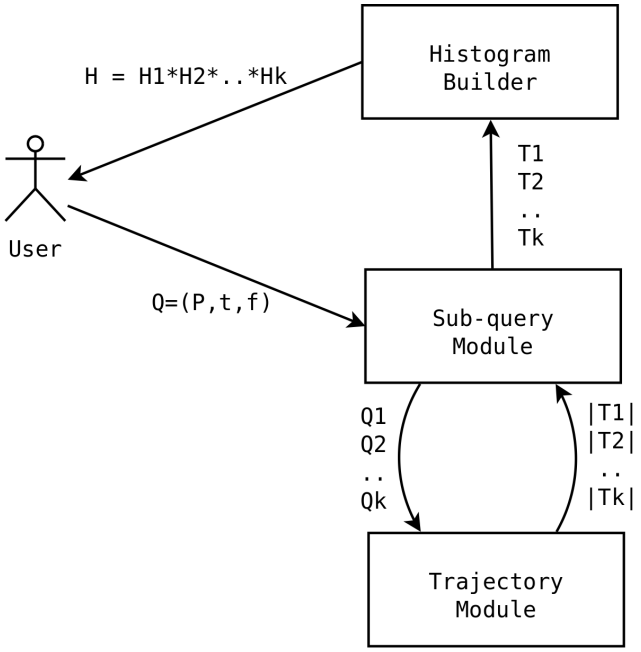


Fig. B.2: Overall Architecture

When the sample size requirements are satisfied for every trajectory set in $\langle T_1, T_2, ..., T_k \rangle$ travel-time histograms of the sub-paths $\langle H_1, H_2, ..., H_k \rangle$ are computed in the `Histogram Builder` and are convolved into a single histogram $H = H_1 \odot H_2 \odot ... \odot H_k$ that aims to approximate the travel-time distribution for the complete path P [6].

If the sample size requirements are significantly exceeded, it may also be prudent to increase the selectivity of the predicates to obtain more relevant trajectory sets.

3.2 Parameters

In our study, we perform a strict path query $Q = (P, t, f)$, described above with three additional parameters: $\alpha \in (0, 24\,hours]$, $R \in \{daily, weekly, ...\}$, and $\beta \geq 1$. For each of our queries $Q = (P, t, f, \alpha, R, \beta)$, two types of filter predicates are supported in f:

- User predicates u
- Weather predicates w

This means that only trajectories of user u or with weather condition w qualify for T^P. If $f = \emptyset$ then only the temporal filter predicates are checked.

The parameter α is the time window size, which means that with the starting time t, only trajectories in the interval $I_t = [t - \frac{\alpha}{2}, t + \frac{\alpha}{2})$ are considered. R is the recurrence parameter for interval I_t. It describes which days are considered, e.g., if $R = daily$ then the trajectories on all days that fall into I_t are considered. The parameter $\beta \geq 1$ is the sample size requirement. Since not all eligible trajectories might be required for an accurate estimate, only the first β trajectories are collected. They are collected from the closest eligible interval until the sample size requirement is met or the eligible intervals are exhausted. This parameter is included to reduce the likelihood of obtaining stale results and to reduce query processing time.

3.3 Example

We proceed to describe four queries on the trajectory set T from Table B.1 in the example road network G in Figure B.3. In all examples, the query path is $P = \langle A, B, E \rangle$, the starting time is $t = [\texttt{Mon}, 9{:}40]$, and the sample size parameter β is omitted. All durations d are in seconds. If the filter parameter $f = \emptyset$, only temporal filters are applied.

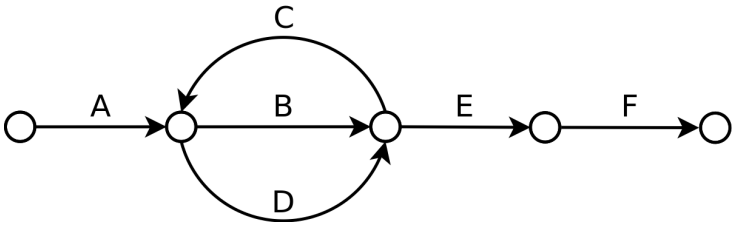


Fig. B.3: Example Road Network Graph G

1. $Q_1 = (P, t, 30, daily, \emptyset)$. Q_1 includes every trajectory in the interval $I = [9{:}25, 9{:}55)$ on every day of the week and returns $T_1 = \{tr_1, tr_2, tr_3, tr_4, tr_6, tr_7\}$.

2. $Q_2 = (P, t, 20, daily, \emptyset)$. Q_2 includes every trajectory in the interval $I = [9{:}30, 9{:}50)$ on every day of the week and returns $T_2 = \{tr_1, tr_2, tr_3, tr_7\}$.

3. $Q_3 = (P, t, 30, weekly, \emptyset)$. Q_3 includes every trajectory in the interval $I = [9{:}25, 9{:}55)$ on every Monday and returns $T_3 = \{tr_1, tr_2, tr_7\}$.

4. $Q_4 = (P, t, 30, daily, \{\text{"}\texttt{u} = \texttt{a}\text{"}\})$. Q_4 includes every trajectory in the interval $I = [9{:}25, 9{:}55)$ on every day of the week and of user a and returns $T_4 = \{tr_1, tr_3\}$.

Trajectory tr_5 is never selected because it does not meet the strict path requirement.

tr	u	s
tr_1	a	$\langle(A, [\texttt{Mon}, 9{:}35], 57, \texttt{fog}), (B, [\texttt{Mon}, 9{:}36], 36, \texttt{fog}),$ $(E, [\texttt{Mon}, 9{:}36], 47, \texttt{fog}), (F, [\texttt{Mon}, 9{:}37], 13, \texttt{fog})\rangle$
tr_2	b	$\langle(A, [\texttt{Mon}, 9{:}42], 52, \texttt{fog}), (B, [\texttt{Mon}, 9{:}43], 35, \texttt{fog}),$ $(E, [\texttt{Mon}, 9{:}43], 40, \texttt{fog})\rangle$
tr_3	a	$\langle(A, [\texttt{Tue}, 9{:}41], 50, \texttt{wet}), (B, [\texttt{Tue}, 9{:}42], 35, \texttt{wet}),$ $(E, [\texttt{Tue}, 9{:}43], 41, \texttt{wet}), (F, [\texttt{Tue}, 9{:}44], 11, \texttt{wet})\rangle$
tr_4	c	$\langle(A, [\texttt{Tue}, 9{:}51], 63, \texttt{wet}), (B, [\texttt{Tue}, 9{:}52], 39, \texttt{wet}),$ $(E, [\texttt{Tue}, 9{:}53], 51, \texttt{wet}), (F, [\texttt{Tue}, 9{:}54], 13, \texttt{wet})\rangle$
tr_5	a	$\langle(A, [\texttt{Wed}, 9{:}30], 56, \texttt{wet}), (D, [\texttt{Wed}, 9{:}31], 12, \texttt{wet}),$ $(C, [\texttt{Wed}, 9{:}31], 25, \texttt{wet}), (B, [\texttt{Wed}, 9{:}32], 40, \texttt{wet}),$ $(E, [\texttt{Wed}, 9{:}33], 40, \texttt{wet})\rangle$
tr_6	b	$\langle(A, [\texttt{Wed}, 9{:}29], 47, \texttt{wet}), (B, [\texttt{Wed}, 9{:}30], 38, \texttt{wet}),$ $(E, [\texttt{Wed}, 9{:}31], 35, \texttt{wet}), (F, [\texttt{Wed}, 9{:}31], 13, \texttt{wet})\rangle$
tr_7	c	$\langle(A, [\texttt{Mon}, 9{:}38], 56, \texttt{dry}), (B, [\texttt{Mon}, 9{:}39], 39, \texttt{dry}),$ $(E, [\texttt{Mon}, 9{:}40], 50, \texttt{dry}), (F, [\texttt{Mon}, 9{:}41], 10, \texttt{dry})\rangle$

Table B.1: Example Trajectory Set T

3.4 Congestion-dependent Window Size

In addition to the above method we define the query $Q = (P, t, f, \alpha, R, \beta, sim_{CI})$, where $sim_{CI} \in [0, 1)$ is the similarity of the congestion index. The congestion index CI_I is a measure of the congestion on a path in a time interval I. It is defined in terms of the average travel time TT_I in the interval and the free-flow travel-time TT_{ff}, which is the average travel time between 22:00 and 6:00. The congestion index for interval I is defined as follows: $CI_I = max(1 - (TT_{ff}/TT_I), 0) \in (0, 1)$.

The congestion-dependent window size is computed by first dividing the day (or week) into $s = \frac{24\,hours}{\alpha}$ ($s = \frac{168\,hours}{\alpha}$) intervals $\langle I_1, I_2, ..., I_s\rangle = \langle I_{[0,\alpha)}, I_{[\alpha,2\alpha)}, ..., I_{[(s-1)\alpha,s\alpha)}\rangle$, e.g., if $\alpha = 15$ minutes then the intervals are $\langle[0{:}00, 0{:}15), [0{:}15, 0{:}30), ..., [23{:}45, 0{:}00)\rangle$. Second, the congestion index of each interval is computed. A range of intervals $\langle I_i, ..., I_j\rangle$ is denoted as $I_{[i,j]}$. We call the initial starting interval of the starting time t, $I_t = I_i = I_{[i,i]}$, so that $(i-1)\alpha \leq t < i\alpha$. Then, for the adjacent intervals, it is checked whether the difference of their congestion indexes exceeds the maximum allowed difference $\delta_{max} = sim_{CI}\,|max_{CI} - CI_{I_t}|$, where max_{CI} is the maximum congestion index of the path. If not, they are merged into a larger interval. The metric ensures that the most congested intervals are never merged with other inter-

$$m(I_{[i,j]}) = \begin{cases} m(I_{[i,(j+1) \bmod s]}) & \text{if } |CI_{I_{(j+1) \bmod s}} - CI_{I_t}| \leq \delta_{max} \\ m(I_{[(i-1) \bmod s,j]}) & \text{if } |CI_{I_{(i-1) \bmod s}} - CI_{I_t}| \leq \delta_{max} \\ I_{[i,j]} & \text{otherwise} \end{cases} \quad \text{(B.1)}$$

vals; and if $sim_{CI} = 0$, only intervals with an identical congestion index are merged, i.e., the lower sim_{CI} the closer the congestion indexes of two adjacent intervals have to be to be merged. This is repeated until the first interval with a difference larger than the maximum difference is found. This procedure is defined by the recursive function $m(I_{[i,i]})$ in Equation B.1.

4 Experimental Setup

This section describes the six hypotheses we aim to study, the datasets and the sample used in our experiments, their evaluation criteria, and the baseline.

4.1 Hypotheses

This section describes the hypotheses we study in our experiments and offers some explanation of their relevance.

- $\mathcal{H}_A$: **Impact of User Predicate** We assume that user predicates improve the predictive performance of our method since we expect driving behavior to be more predictive of travel-time than the time of day (and day of the week) alone.
- $\mathcal{H}_B$: **Impact of Weather Predicate** We assume that weather predicates improve the predictive performance of our method because we expect driving behavior to change with the weather.
- $\mathcal{H}_C$: **Impact of Temporal Filters** We assume that smaller time windows improve the predictive performance of our method since they are more sensitive to changes in congestion.
- $\mathcal{H}_D$: **Impact of Sample Size** We assume that larger sample sizes improve the quality of our estimates since they would be less sensitive to outliers.
- $\mathcal{H}_E$: **Impact of Segment Category** We assume that there will be a difference in predictive performance between the different segment categories since there is less interference with the traffic, e.g., by roundabouts or traffic lights on segment categories like motorways.

- $\mathcal{H}_F$: **Impact of Congestion** We assume that the queries using congestion-based window sizes provide improved predictive performance because it allows them to be sensitive to congestion while also being less affected by data staleness since the windows sizes can be larger than for the methods using fixed size time windows.

4.2 Datasets

This section describes the map and trajectory datasets used.

OpenStreetMap

Our network graph is based on the OpenStreetMap data [7] of the road network of Northern Denmark, which contains around 750,000 road segments. When converted to a spatial network graph, this graph has around 1.46 million directed edges. Each edge represents a direction on a segment and has one of 17 different segment categories. Our study focuses on four major segment categories in particular:

- *motorways*: major divided roads that have at least two lanes
- *primary roads*: important roads that link larger towns
- *secondary roads*: roads that link towns
- *tertiary roads*: roads connecting villages or that are urban main streets

This categorization is available for all OpenStreetMap maps and makes segment category-based partitioning possible for other map-matched trajectory datasets as well [8].

ITSP Dataset

The "ITS Platform" dataset contains over 1.1 billion GPS points sampled at 1 Hertz collected from 458 distinct vehicles in Aalborg and the surrounding region during the period from May 2012 to December 2014 [9].

In a preprocessing step, the GPS points are map-matched to obtain in excess of 79 million segment traversals that form around 1.4 million trajectories, where a new trajectory is created if more than a 180 seconds have elapsed since the last GPS point. The map-matching algorithm also discards GPS points at the beginning and end of a trip if too few points are matched to the start and end segments of the trajectory [10]. This is done so the durations of the segment traversals are meaningful. During the pre-processing, the trajectories are also annotated with weather categories.

Each GPS record contains the following information:

- trajectory ID
- vehicle ID
- segment ID
- time and date the segment was entered (minute resolution)
- time on segment (second resolution)
- weather type

Since the cars in our dataset are privately owned, we treat the vehicle ID as the user ID. The segment IDs are derived from the unique mapping of OpenStreetMap segment key and the driving direction. The time on a segment is also computed during the preprocessing step. The following weather categories are derived from the NOAA weather data [11] for Denmark: fog, thunder, drifting, wet, dry, snow, and freezing. An additional category "none" is included, which happens to be most prevalent in our dataset due to missing data from the weather stations.

4.3 Trajectory Sample

For our study, we pick a point in time covered by the ITSP dataset and designate it the "cutoff timestamp" t_c. In our case, it is a timestamp on September 8th 2013, the median of the starting timestamps $tr.t_1$. This is to ensure that subsequent queries have at least a year of trajectory data available. We then identify 14 paths that could be viable sub-paths in a partitioned query as described in Section 3 and whose trajectory sets T^P satisfy the following criteria:

- They have a trajectory set before the "cutoff timestamp" with $|T^P_{\leq t_c}| \geq 5000$.
- $|P| \geq 6$
- They have a length of at least 500 meters.
- They do not overlap with any of the other chosen paths.
- They consist of a single segment category.
- If multiple overlapping paths meet the requirements, the longest one is chosen.

name	category	$\|P\|$	length	$\bar{TT}/TT_{ff}$	max_{CI}	$\|T^P_{\leq t_c/>t_c}\|$	$tl/r/y/o$
MW1	motor-way	11	7836	241/241	0.095	10611/ 9066	0/0/0/0
MW2	motor-way	12	17565	531/538	0.087	8157/ 6623	0/0/0/0
MW3	motor-way	11	16486	517/516	0.137	7015/ 5056	0/0/0/0
MW4	motor-way	17	9064	321/327	0.154	6932/ 5705	0/0/0/0
P1	primary	28	7022	357/343	0.081	5104/ 4479	0/3/0/1
P2	primary	18	5543	253/243	0.076	5504/ 4349	0/2/0/0
P3	primary	20	7960	352/340	0.061	7324/ 5793	0/1/0/0
S1	secon-dary	13	1539	135/120	0.187	5192/ 4176	5/0/0/0
S2	secon-dary	17	1503	155/135	0.282	5133/ 3807	4/0/0/2
S3	secon-dary	11	1355	73/70	0.136	5213/ 4204	0/1/0/1
S4	secon-dary	12	2973	131/124	0.353	5109/ 4263	0/0/0/1
T1	tertiary	11	1038	67/62	0.240	5095/ 4535	0/1/0/3
T2	tertiary	11	720	22/22	0.335	5096/ 6462	0/0/0/8
T3	tertiary/ unclas-sified	8	884	90/71	0.495	5479/ 4421	0/1/1/2

Table B.2: Paths chosen for the study

Details pertaining to the chosen paths can be found in Table B.2 that contains the name, segment category, number of segments, length in meters, average travel-time in seconds, free-flow travel-time, maximum congestion index, number of full traversals before and after t_c, and number of obstacles. The table also shows that most paths are considerably longer than the short minimum length (500 m). Obstacles are separated into four groups: signaled intersections (tl), roundabouts (r), stop and yield signs (y), and other obstacles (o), i.e., speed bumps, pedestrian crossings, and bus stops at the same lane.

The obstacles are also counted if they immediately precede or follow the path. Link type road segments were ignored since they do not form longer paths in the road network. Segments categorized as *trunk*, *residential*, *living-street*, *service*, *road*, *track*, and *unpaved* were excluded since the ITSP dataset does not contain sufficient numbers of trajectories traversing them.

The paths MW1–3 are motorway paths outside of cities and MW4 is within city limits. P1–3 are primary roads outside cities. S1 and S2 are on secondary roads within city limits and S3 and S4 are outside cities. The tertiary paths T1–3 are all within city limits. Path T3 also contains unclassified segments and is included because no other paths of sufficient length and sample size are in the dataset.

From each of those paths, we sample a set $\{tr_1, .., tr_n\} \subset T^P_{>t_c}$ of trajectories that started after t_c and fully traverse them. For our study, we choose $n = 1000$ for each path.

4.4 Sample Types

For each sampled trajectory tr, three different sets of trajectories are retrieved:

- T^P_p, based on all preceding traversing trajectories
- T^P_u, based on all preceding traversing trajectories of the same user (cf. Section 3.3, example 4)
- T^P_w, based all preceding traversing trajectories during the same weather condition w

Each of these sets is a subset of $T^P_{<tr.t_1}$. The same temporal filter predicates are applied to the three sets $T^P_{p/u/w}$. The combination of a user and a weather predicate is supported but is not included in the study because our dataset is too sparse for such a selective predicate.

We evaluate seven different sample sizes β ($20, 30, 40, 50, 60, 70$, and 80), six different time window sizes α ($15, 20, 30, 45, 60$, and 90 minutes), and four recurrence patterns R:

- daily (D)

- weekly (W)
- Monday to Friday (F)
- Monday to Thursday (T)

The recurrence from Monday to Thursday is included because studies of driving behavior show that driving behavior on Fridays often differs significantly from those of the preceding week days [12].

4.5 Qualitative Assessment

The predictive performance for each trajectory sample $T_\beta^P = \langle tr_1, ..., tr_\beta \rangle$ is evaluated by its set of travel times $X^P = \langle d(tr_1, P), ..., d(tr_\beta, P) \rangle = \langle x_1, ..., x_\beta \rangle$. For each travel-time sample X^P retrieved for a sample trajectory tr_i we evaluate the accuracy, which we define as distance to the sample mean, and its precision, which we define as the width of its prediction interval. For both values, we consider lower values as better since they allow better estimates of travel time.

Accuracy

The accuracy of the retrieved samples for a path P is defined as the symmetric mean absolute percentage error [13] between the mean of the sample $\bar{X_i^P}$ and the travel-time of the trajectory tr_i on P, $TT_{tr_i}^P = D(tr_i, P)$:

$$\text{sMAPE}_P = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|\bar{X_i^P} - TT_{tr_i}^P|}{\frac{1}{2}(\bar{X_i^P} + TT_{tr_i}^P)}$$

where n is the size of the trajectory sample mentioned in Section 4.3.

The sMAPE was chosen because it allows the comparison of results from different paths and is not as biased against overestimations as its non-symmetric variant [14].

Precision

The precision of the distribution of the sample $\langle x_1, ..., x_\beta \rangle$ is based on the width of its prediction interval.

Since earlier studies show that travel-time distributions cannot be expected to follow a parametric distribution [5], we use the distribution-free prediction interval suggested by Saw et al. [15]:

$$[\bar{X}^P - A^P, \bar{X}^P + A^P],$$

where $\bar{X}^P$ is the sample mean, $A^P = \lambda(1 + \frac{1}{\beta})^{\frac{1}{2}} S^P$, S^P is the square root of the sample variance, and β is the sample size.

We choose $\lambda \geq 1$ so that we obtain the 90% prediction interval $\mathcal{P}(|\bar{X}^P - x_{\beta+1}| \leq A^P) \geq 0.9$, where $TT^P_{tr_i}$ is considered the $(\beta+1)$-th sample $x_{\beta+1}$. For computing λ, the method suggested by Konijn [16] is used, i.e., for $\beta = 20$, we choose $\lambda = 3.154$.

For the precision to be comparable between different paths, we normalize it with the travel time and average it for each path and method:

$$precision_P = \frac{100\%}{n} \sum_{i=1}^{n} \frac{2A^P_i}{TT^P_{tr_i}}$$

Since the distribution-free prediction interval makes very little assumptions about the underlying distribution, it provides only a very conservative estimate. It is therefore of little use in predicting future values but allows us to compare the ranges of the underlying distributions of different paths and collection methods.

It should be noted that good precision is subordinate to accuracy since a method with good precision but poor accuracy will be less likely to include the actual travel time than a method with poor accuracy and poor precision. The precision is therefore useful when comparing methods with similar accuracy or when trying to show that one methods is outperforming another in both regards. Our precision metric could also be used when making routing decision in a setting where time constraints, e.g., a trip might not take longer than 45 minutes, have to be observed. Sub-paths with large prediction intervals could then be avoided by the routing algorithm since they introduce more uncertainty to the travel time.

4.6 Baseline

As a baseline for accuracy, we use the sum of the average travel times of each segment of path $P = \langle e_1, ..., e_l \rangle$, $\sum_{i=1}^{l} \bar{X}^{\langle e_i \rangle}$, which is equivalent to the average of the travel-time samples from the trajectory sets returned for the sub-queries $Q_i = (\langle e_i \rangle, 0, 24\ hours, daily, \emptyset, \infty)$, with $1 \leq i \leq l$.

We do not provide a baseline for precision. Our metric requires a single trajectory set to be comparable, so we only compare the three different filter methods with each other.

5 Results

This section describes the effect sample size, window size, recurrence, and different collection methods have on accuracy and precision.

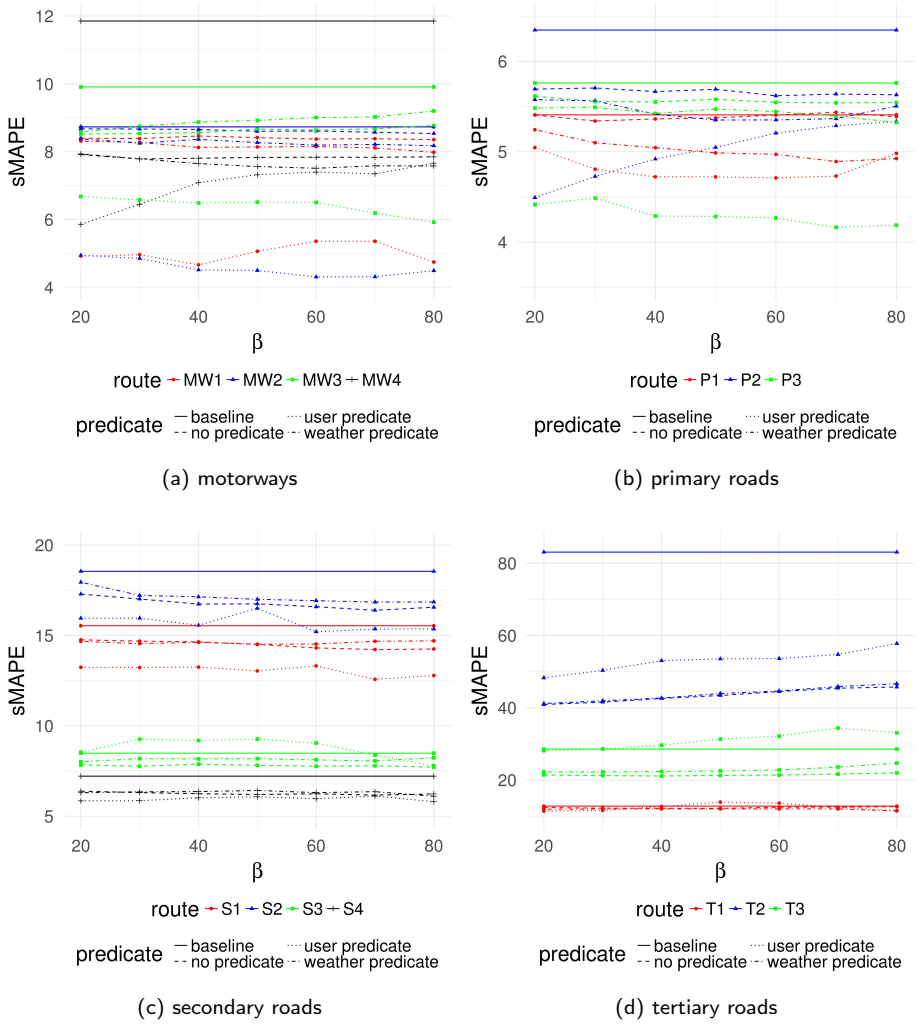


Fig. B.4: Accuracy w/ different sample sizes

5.1 Effect of Sample Size

Figures B.4a to B.5d show the results for a daily sampling with a window size of $\alpha = 20$ minutes with seven different sample size requirements β. Samples where the requirement is not met are ignored. The y-axis in Figures B.4a to B.4d shows the average symmetric mean absolute percentage error of the samples, and the y-axis in Figures B.5a to B.5d shows the average width of the prediction interval in percent of travel-time.

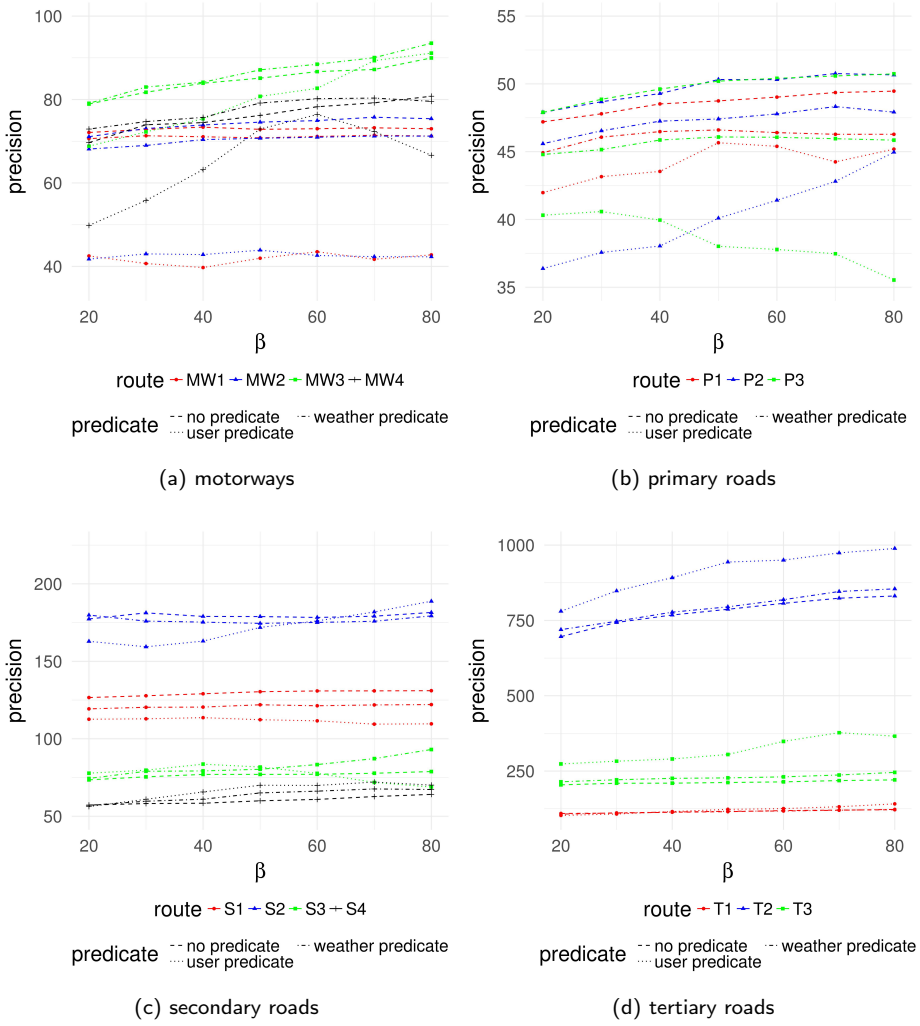


Fig. B.5: Precision w/ different sample sizes

Figures B.4a and B.5a show that for the four motorway paths, the user-based method outperforms the other two in accuracy as well as precision, except for MW3 that produces slightly worse precision with larger sample sizes. The accuracy baseline for MW1 is not visible in Figure B.4a (and B.6a) because it is nearly identical to the baseline of MW2. Use of the weather predicate shows only little improvement compared to method without trajectory predicates. Increasing the sample size has little effect on accuracy and generally makes the precision worse. This is most likely due to the higher

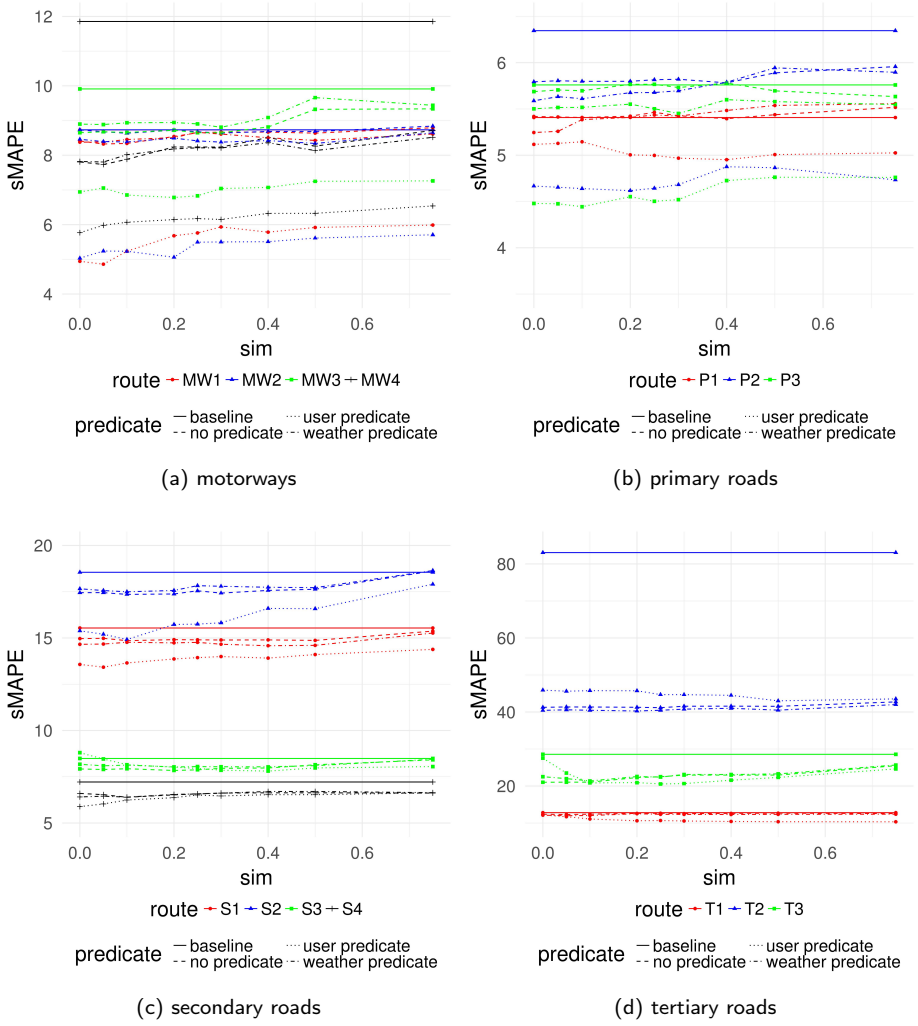


Fig. B.6: Accuracy w/ congestion-dependent window sizes

probability of outliers in the larger samples and because of data staleness. Figure B.4a also shows that MW4 exhibits the worst accuracy and precision for all methods. This most likely is because it is within the city limits and therefore has a much higher density of motorway accesses and exits, which make driving behavior less predictable. This is also consistent with its higher congestion index.

A similar pattern can be seen for the three primary paths in Figures B.4b and B.5b but the user-based method gets worse with higher sample sizes.

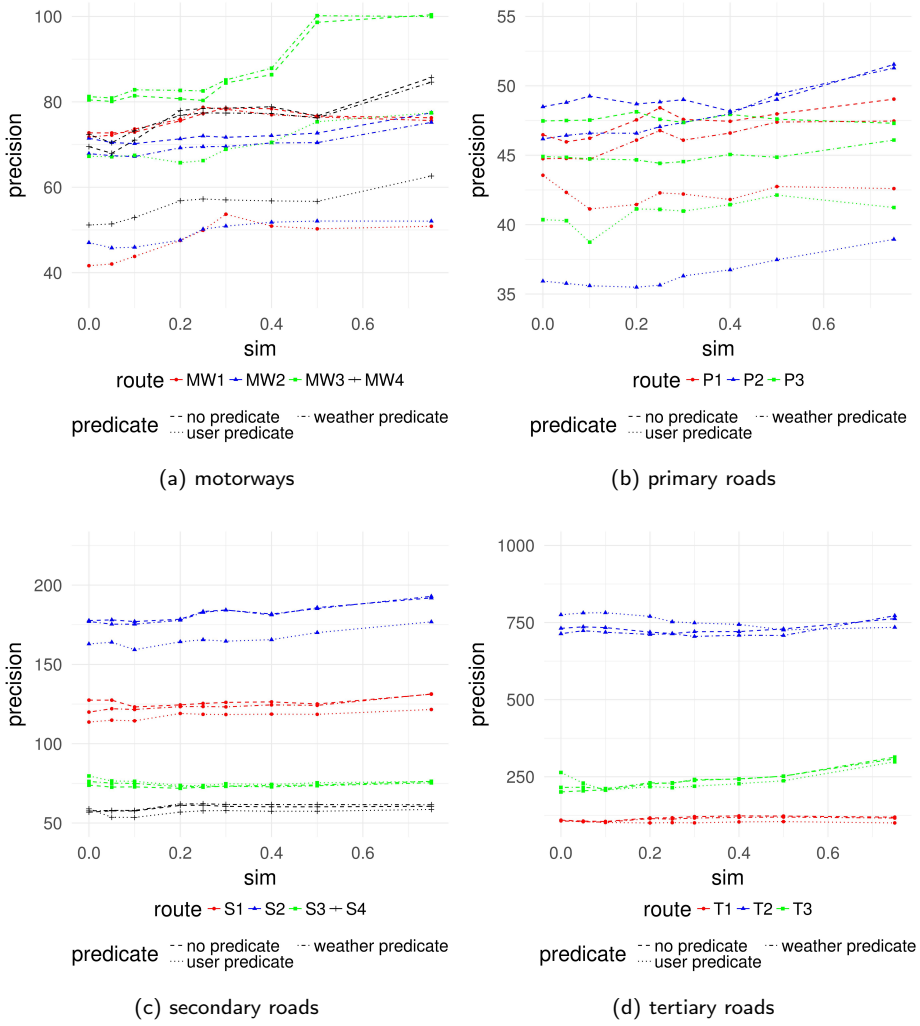


Fig. B.7: Accuracy w/ congestion-dependent window sizes

This may be explained by the high selectivity of the user-based method, which requires the method to collect trajectories further back in time than the other methods to meet the sample size requirements. When data staleness becomes a problem, this may suggest that those paths are especially affected by long-term cycles, e.g., seasons. This is consistent with the developments for P2 where the degradation in accuracy is the most pronounced for the user-based method but also shows some improvements with increasing sample sizes for the weather-based method. Among all segment categories,

the three primary paths are closest to each other in accuracy and precision, which could in part be due to them being the least affected by congestion.

For the secondary paths in Figures B.4c and B.5c, the user-based method manages to improve accuracy, except for S3, which never manages to exceed the results of the method without predicates. This may be because of the relative fast average traversal time of S3, which suggests that the user-based method is best for secondary paths with longer average travel-times. The two inner city paths S1 and S2 exhibit the same behavior as MW4 insofar as they also have much lower accuracy and precision than their counterparts outside city limits.

The tertiary paths in Figures B.4d and B.5d exhibit very different behaviors from each other. T1 has the most accurate and precise results but the user-based method only provides better results with lower sample sizes. This is most likely again due to having to rely on more stale data than the other methods due its high selectivity. For T1, the weather-based method also shows the best results for larger sample sizes. Path T2 performs the worst, which is most likely due to it being short and due it being in a residential area, which can cause considerably more interference. The predicate-based methods provide no improvements over the plain on-the-fly method. For path T3, the user-based method exhibits decreasing accuracy and precision with increasing sample size and does not improve upon the baseline. All three paths share the characteristic that the user-based method provides the worst precision for higher sample sizes.

5.2 Congestion-dependent Window Size

Figures B.6a to B.7d show accuracy and precision when using the congestion-based window size with a daily sampling, a sample size of $\beta = 20$, and an initial time window of $\alpha = 15$ minutes. For each segment category, nine window sizes based on the congestion similarities sim_{CI}, $(0, 0.05, 0.1, 0.2, 0.25, 0.3, 0.4, 0.5, 0.75)$ are shown in the figures.

The accuracy for motorways shown in Figure B.6a exhibits no significant improvements. The precision shown in Figure B.7a is slightly improved for low values of sim_{CI} for MW4 and the user-based method of MW3. Figures B.6b and B.7b show similar results for the primary roads with no improvements in accuracy and small improvements in precision for smaller similarity values. The accuracy of the user-based method can be improved with similarity values between 0.05 and 0.2, which also provides small improvements in precision, as shown in Figures B.6c and B.7c.

Figure B.6d shows that congestion-based window sizes can significantly improve the accuracy of the user-based methods on tertiary roads. So much in fact that it now manages to outperform the other methods on paths T1 and T3. The increased window sizes also increase precision for larger similarity

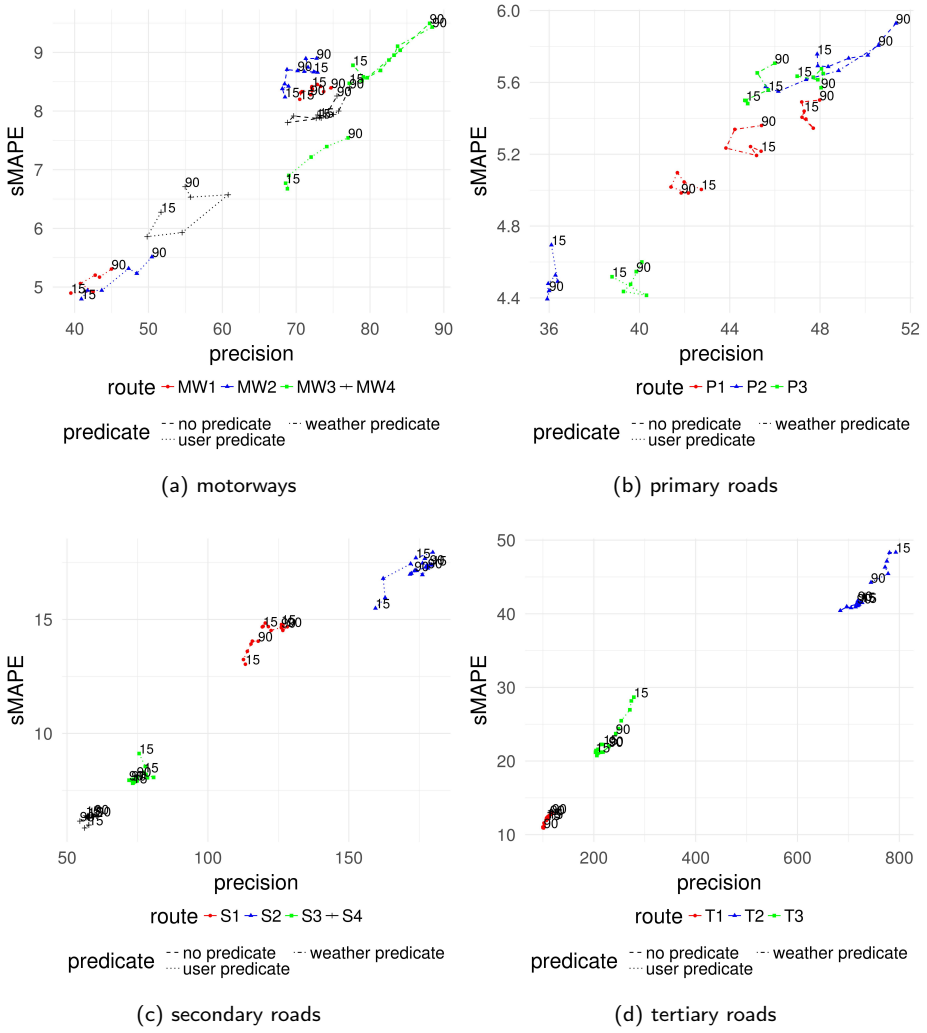


Fig. B.8: Accuracy & Precision w/ different window sizes

values, which can be seen in Figure B.7d.

5.3 Effect of Window Size

Figures B.8a to B.8d show the results for a daily sampling, a sample size of $\beta = 20$, with six different window sizes α (15, 20, 30, 45, 60, and 90 minutes).

Figure B.8a shows that smaller window sizes improve accuracy as well as performance for the motorway paths. A similar pattern can be seen in

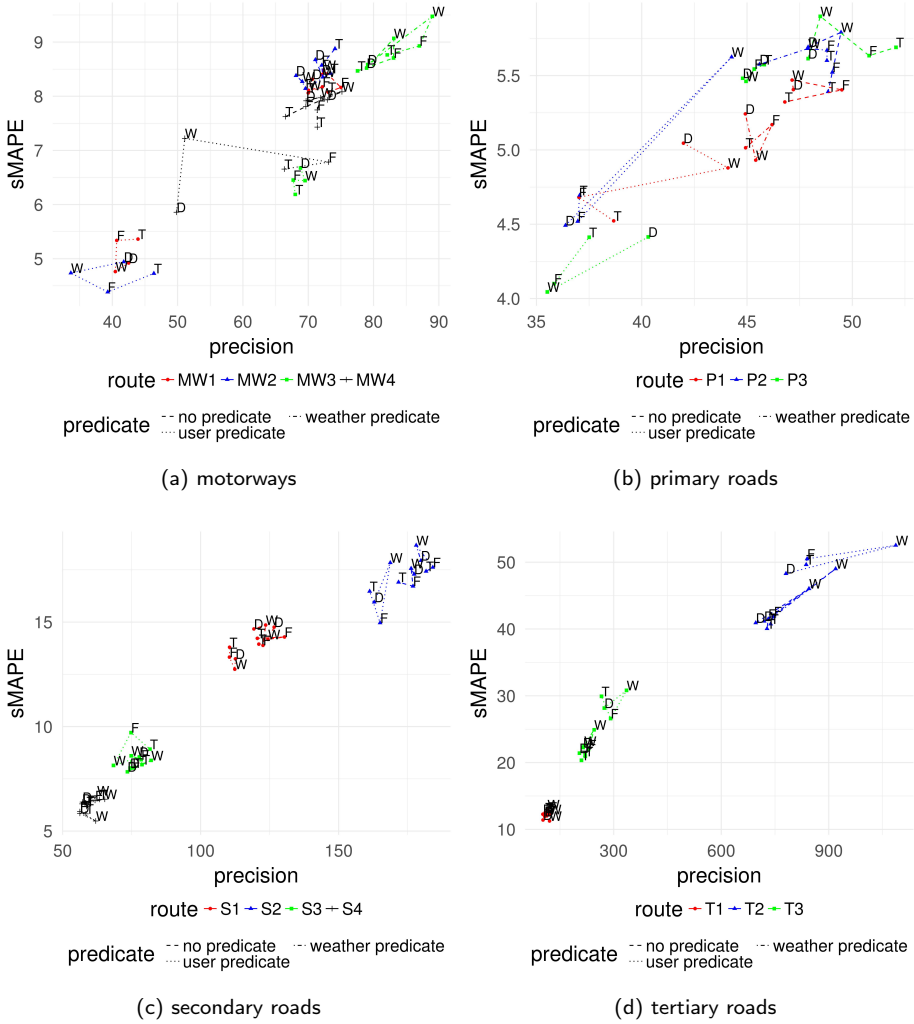


Fig. B.9: Accuracy & Precision w/ different recurrences

Figure B.8b for primary paths, but for slightly larger window sizes (20 to 30 minutes). The secondary roads in Figure B.8c show similar behavior and perform best with smaller window sizes with the exception of S3. The user-based method performs best for the tertiary paths in Figure B.8d with larger window sizes, which suggests that more recent data is more important to the quality of the estimates than the closeness to the time of day. This is also consistent with the results for the congestion-based window size where a large tolerance for merging adjacent intervals also increases accuracy and

precision.

5.4 Effect of Recurrence

Figures B.9a to B.9d show the results for a sample size of 20, a window size of 20 minutes, with four different recurrences (weekly (W), daily (D), Monday to Friday (F), and Monday to Thursday (T)).

Figure B.9a shows that the user-based method performs best with a weekly recurrence except for the inner city motorway MW4, which has best the precision and accuracy with a daily recurrence and Monday to Thursday. The other methods perform best with a workday recurrence. The results for the primary roads in Figure B.9b are consistent with earlier observations for P2 since the recurrence with the highest selectivity again yields the worst results for the user-based method. The same is true for the method without predicates, but the effect is far less pronounced. For nearly all paths and methods, the recurrence from Monday to Thursday or Friday yields the best accuracy. Figure B.9c shows that the recurrence of Monday to Thursday or Friday exhibits the best accuracy for the inner city paths, except for the user-based method where a weekly recurrence does best. For the other two paths, the weekly recurrence is the most inaccurate, except for the user-based method for S3. In all other cases for S3 and S4, one or both workday recurrences perform the best. The tertiary paths in Figure B.9d show that recurrences with higher selectivity perform worse, which is consistent with our results for the different window sizes.

6 Conclusion and Outlook

Our study shows that the quality of travel-time estimates to a large extent depends on the segment types. With errors as low as 4% for motorway paths and as high as at least 40% for some tertiary paths, the diverging results for different road types suggest that there is no single best way to identify the most relevant trajectories for travel-time estimates.

Hypothesis $\mathcal{H}_A$ turned out to hold in most cases, as user-specific data can significantly improve the accuracy and precision of estimates on longer paths. Hypothesis $\mathcal{H}_B$, on the other hand, proved to be less applicable since weather-based predicates do not provide consistent improvements. The impact of temporal filters ($\mathcal{H}_C$) is very segment-category dependent, where motorways and primary and secondary roads are very congestion-sensitive, i.e., they yield better results with smaller window sizes. In contrast, tertiary roads seem to be more sensitive to data staleness, i.e., they get worse with smaller window sizes (as well as smaller sample sizes).

Hypothesis $\mathcal{H}_D$ turned out to not hold in most cases. First, only the user-based method is considerably affected by the sample size; furthermore, the precision deteriorates in nearly all cases when the sample size is increased. Hypothesis $\mathcal{H}_E$ turned out to generally hold since the four different segment categories exhibit very different predictive performance. But the varying performance of the secondary and tertiary paths within the same category suggests that other factors need to be considered as well, e.g., the number of traffic lights, or whether they are inside or outside cities, to meaningfully distinguish sub-paths. Hypothesis $\mathcal{H}_F$ proved to hold, but the study also showed that the different paths exhibit very different sensitivity to congestion.

Our results show that sub-path estimates hold the potential to yield considerably better results than segment-based methods and also show that the segment category has a large influence on their quality. These results suggest that to achieve the best possible estimations, different collection methods should be employed for different sub-paths of a trip. This could be implemented as a hybrid approach of pre-computed histograms as well as histograms computed on the fly with different collection methods where pre-computation is not feasible. New index structures that facilitate these hybrid estimation queries are also of interest for further research. Several different indexes that support strict path queries on trajectory datasets have been proposed [17–19]. They can be extended in future research to allow the efficient execution of the collection methods studied in this paper.

Acknowledgments

This research was supported in part by a grant from the Obel Family Foundation and by the DiCyPS project.

References

[1] R. Waury, J. Hu, B. Yang, and C. S. Jensen, "Assessing the Accuracy Benefits of On-the-Fly Trajectory Selection in Fine-Grained Travel-Time Estimation," in *Proceedings of the 18th IEEE International Conference on Mobile Data Management*. IEEE, 2017, pp. 240–245.

[2] Student, "The Probable Error of a Mean," *Biometrika*, vol. 6, no. 1, pp. 1–25, 1908.

[3] S. Winter, "Modeling Costs of Turns in Route Planning," *GeoInformatica*, vol. 6, no. 4, pp. 345–361, 2002.

[4] J. L. Borresen, O. Andersen, C. S. Jensen, and K. Torp, "Interactive Intersection Analysis using Trajectory Data," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2017, pp. 87:1–87:4.

[5] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu, "Path Cost Distribution Estimation Using Trajectory Data," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 85–96, 2016.

[6] E. Nikolova, M. Brand, and D. R. Karger, "Optimal Route Planning under Uncertainty," in *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*. AAAI Press, 2006, pp. 131–140.

[7] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org," https://www.openstreetmap.org, 2014.

[8] ——, "Map Features," https://wiki.openstreetmap.org/wiki/Map_Features#Highway, 2018, accessed: 2018-04-25.

[9] Aalborg University, "ITS Platform," http://www.itsplatform.dk/, 2018.

[10] F. C. Pereira, H. Costa, and N. M. Pereira, "An off-line map-matching algorithm for incomplete map databases," *European Transport Research Review*, vol. 1, no. 3, pp. 107–124, 2009.

[11] 14th Weather Squadron Fleet Numerical Meteorology and Oceanography Detachment, "Federal Climate Complex Data Documentation For Integrated Surface Data," National Climatic Data Center, Asheville, NC 28801-5001 USA, Tech. Rep., 2016.

[12] S. Schönfelder, "Some notes on space, location and travel behaviour," in *Proceedings of the 1st Swiss Transport Research Conference (STRC)*, 2001.

[13] J. S. Armstrong, *Long-Range Forecasting: From Crystal Ball to Computer*, 2nd ed. John Wiley & Sons, Inc., 1985.

[14] J. S. Armstrong and F. Collopy, "Error Measures for Generalizing About Forecasting Methods: Empirical Comparisons," *International Journal of Forecasting*, vol. 8, no. 1, pp. 69–80, 1992.

[15] J. G. Saw, M. C. Yang, and T. C. Mo, "Chebyshev Inequality With Estimated Mean and Variance," *The American Statistician*, vol. 38, no. 2, pp. 130–132, 1984.

[16] H. S. Konijn, "Distribution-Free and Other Prediction Intervals," *The American Statistician*, vol. 41, no. 1, pp. 11–15, 1987.

[17] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp, "Path-based Queries on Trajectory Data," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2014, pp. 341–350.

[18] S. Koide, Y. Tadokoro, and T. Yoshimura, "SNT-index: Spatio-temporal index for vehicular trajectories on a road network based on substring matching," in *Proceedings of the 1st International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. ACM, 2015, pp. 1–8.

[19] M. Fouladgar and R. Elmasri, "Temporally Enhanced Network-Constrained (TENC) R-tree," in *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*. ACM, 2016, pp. 27–36.

Paper C

Indexing Trajectories for Travel-Time Histogram Retrieval

Robert Waury, Christian S. Jensen, Satoshi Koide, Yoshiharu Ishikawa, Chuan Xiao

The paper has been published in
Proceedings of the 22nd International Conference on Extending Database Technology, pp. 157–168, 2019.
DOI: 10.5441/002/edbt.2019.15

Abstract

A key service in vehicular transportation is routing according to estimated travel times. With the availability of massive volumes of vehicle trajectory data, it has become increasingly feasible to estimate travel times, which are typically modeled as probability distributions in the form of histograms. An earlier study shows that use of a carefully selected, context-dependent subset of available trajectories when estimating a travel-time histogram along a user-specified path can significantly improve the accuracy of the estimates. This selection of trajectories cannot occur in a preprocessing step, but must occur online—it must be integrated into the routing itself. It is then a key challenge to be able to select very efficiently the "right" subset of trajectories that offer the best accuracy when the cost of a route is to be assessed. To address this challenge, we propose a solution that applies novel indexing to all available trajectories and that then is capable of selecting the most relevant trajectories and of computing a travel-time distribution based on these trajectories. Specifically, the solution utilizes an in-memory trajectory index and a greedy algorithm to identify and retrieve the relevant trajectories. The paper reports on an extensive empirical study with a large real-world GPS data set that offers insight into the accuracy and efficiency of the proposed solution. The study shows that the proposed online selection of trajectories can be performed efficiently and is able to provide highly accurate travel-time distributions.

1 Introduction

Vehicular transportation is an important global phenomenon that impacts the lives of virtually all of us. We rely on it for mobility, and we are affected by congestion, accidents, and air and noise pollution. Its influence can be expected to continue into the foreseeable future. For example, in the European Union alone, more than 75% of all freight transport and more than 80% of passenger transport rely on the road networks [1]. The availability of high-resolution GPS trajectories allows for reliable map-matching to a road network. The resulting trajectories are called *network-constrained trajectories* (NCT) and can be used to obtain travel-time estimates for paths in the network, thus making transportation more predictable, safe, and environmentally friendly.

When using such a data set, the most straightforward approach to computing a travel-time estimate for a path is to compute a real-valued estimate for each segment in the path and then sum up these to obtain an estimate for the full path. This approach can be refined by collecting travel-time histograms for each segment and then combine them by means of convolution to obtain a travel-time histogram for the full path. This improves the accuracy of estimates since travel times are better modeled as distributions than real valued. Further, the distributions often do not follow a parameterized distribution, e.g., normal or uniform, and are therefore better estimated with histograms. This segment level approach can also be extended to computing different histograms for different times of day, e.g., the 96 15-minute intervals of the day, to account for changing congestion throughout the day. These histograms can be used as edge weights by routing algorithms to compute better results. All of the above approaches, however, only consider travel-time estimates at the segment level. These approaches fail to take into account factors like the times it takes to pass through intersections, going straight or turning left or right, which are hard to model accurately. An earlier study [2] shows that travel-time estimates for a given path can be improved considerably when they are computed from trajectories that strictly follow the path, as opposed to computing them from segment-level estimates. This type of path-based estimate relies on efficiently processing *strict path queries* (SPQ) as proposed by Krogh et al. [3], which is a query on a trajectory set that only returns trajectories which traversed a given path without detours.

We propose a system that can compute time-varying and personal travel-time histograms for any path in a network based on a large trajectory set. It would be infeasible and impractical to pre-compute and store these time-varying and personal weights for any path in a network before routing occurs. For example, given even a moderately sized road network of a million segments, for all 15-minute windows, nearly a 100 million histograms would

be needed to just cover every single segment, with the storage requirements increasing dramatically when considering larger path lengths. We therefore obtain the weights for a path on-the-fly by expressing them as a series of SPQs, which we can efficiently process using our in-memory NCT index. If any of these sub-queries fails to retrieve a sufficient number of matching trajectories, we apply a greedy algorithm that relaxes the SPQ's predicates until the retrieved trajectory set has a specified cardinality. Since performance is crucial in our setting, we also implement a cardinality estimator for SPQs to prevent unnecessary index traversals. We also show that carefully choosing the initial set of SPQs increases the accuracy of the path weights and increases the performance of the query. We perform extensive experiments using a real-world trajectory data set containing 1.4 million trajectories from Northern Denmark, which shows that our approach is suitable for real-time applications.

The main contributions of this paper are the following:

- An adapted NCT index that supports efficient computation of travel-time histograms for SPQs.
- A greedy algorithm that enables efficient processing of any SPQ in periodic time intervals.
- A cardinality estimator for SPQs.
- A detailed analysis of the accuracy and performance of the solution and its components.

The rest of the paper is structured as follows. Section 2 provides an overview of prior work, preliminaries, and a detailed problem description. Section 3 describes the query processing method, while Section 4 details the construction and use of the NCT index. Section 5 outlines the experimental setup and the evaluation metrics. Section 6 reports on the results of the experiments, and Section 7 concludes.

2 Problem Formulation

This section provides an overview of prior work, preliminaries, and a problem definition.

2.1 Related Work

We review approaches to travel-time estimation and then, we review network-constrained trajectory indexing with a focus on indexes supporting SPQs.

Travel-Time Estimation

Earlier studies on travel-time estimation compute histograms for single segments [4], which still requires to model turn costs [5], or for short pre-defined paths with considerable traffic [6], which are then convolved at query time. In our approach, travel-times are computed for sub-paths instead of only for individual segments. This approach implicitly handles turn costs within sub-paths, and turn costs only need to be modeled explicitly in-between sub-paths if applicable. Other approaches based on tensor decomposition [7], support vector regression [8], variance-entropy-based clustering [9], or deep neural networks [10] have also been proposed. But they either do not provide travel-time distributions or do not provide estimates for specific paths but only for origin-destination pairs.

NCT Indexing

Several indexes for network-constrained trajectories based on R-trees [11–13] or B+-trees [14] have been proposed, but they are often only optimized for range queries or nearest neighbor queries.

Two indexes have been proposed to support strict path queries, NETTRA [3] and the SNT-index [15]. NETTRA is a disk-based index designed to answer SPQs with minimal I/O and also supports efficient updates of the index, but may return false positives due to hash collisions. The SNT-index uses data structures adapted from string matching to efficiently identify matching trajectories. This index was originally designed to retrieve all matching trajectory IDs in a given time interval that fulfill the SPQ requirement. We extend it to accommodate travel-time retrieval as well.

2.2 Network Graph & Trajectories

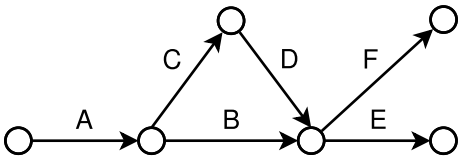


Fig. C.1: Example Road Network

A spatial network is modeled as a directed graph $G = (V, E, \mathcal{F})$, where V is a vertex set, $E \subseteq V \times V$ is a set of edges that represent road segments, and $\mathcal{F} : E \to Cat \times Z \times SL \times L$ is a set of functions, where Cat is the set of road categories, Z is the set of different types of zones the segments are located in, SL is the set of speed limits in kilometers per hour (or $\frac{1000}{3600}$ meters per second), and L is the set of segment lengths in meters. From this we

e	c	z	sl	l	$estimateTT$
A	*motorway*	*rural*	110	900	29.5 s
B	*primary*	*city*	50	120	8.6 s
C	*secondary*	*city*	30	40	4.8 s
D	*secondary*	*city*	30	80	9.6 s
E	*primary*	*city*	50	100	7.2 s
F	*primary*	*rural*	80	800	36.0 s

Table C.1: Example of $\mathcal{F}$ and Function *estimateTT*

can derive the function $estimateTT(e_i) = 3.6\frac{\mathcal{F}(e_i).l}{\mathcal{F}(e_i).sl}$ that returns the traversal time in seconds if the segment is traversed at the speed limit. This function is used as a fallback so that we can return a result even if no data is available for a segment. Every edge $e \in E$ has a category that captures the road type of the segment it represents and a zone type describing its location. Figure C.1 shows the graph representation of the road network we use in examples. Table C.1 shows the mapping of each segment to categories $c \in Cat = \{motorway, primary, secondary\}$ and zones $z \in Z = \{city, rural\}$.

A traversable sequence of segments $P = \langle e_0, e_1, \ldots, e_{l-1} \rangle$ is called a path, with $|P| = l$. A sub-path $\langle e_i, \ldots, e_{j-1} \rangle$, with $0 \leq i < j \leq l$, of P is denoted as $P[i, j)$. The set of trajectories is given as $\mathcal{T} \subseteq \mathcal{D} \times \mathcal{U} \times \mathcal{S}$, where $\mathcal{D}$ is the set of all trajectory ids, $\mathcal{U}$ is the set of all drivers. Further, $\mathcal{S} : \mathbb{N}_l \to E \times \mathcal{TS} \times \mathcal{C}$ is the domain of functions from the set consisting of the first l natural numbers to the range of triples consisting of an edge $e \in E$, a timestamp $t \in \mathcal{TS}$, and a time duration $TT \in \mathcal{C}$. This domain of functions encodes finite sequences of length l.

A trajectory $tr \in \mathcal{T}$ of a user u with the id d is therefore denoted as (d, u, s), where $s \in \mathcal{S}$ is a sequence of 3-tuples:

$$s = \langle (e_0, t_0, TT_0), (e_1, t_1, TT_1), \ldots, (e_{l-1}, t_{l-1}, TT_{l-1}) \rangle,$$

where $t_0, .., t_{l-1}$ are the timestamps when a segment was entered with $\forall i \forall j (i < j \Rightarrow t_i < t_j)$, $TT_i > 0$ is the duration of the traversal of e_i, and l is the number of segments traversed.

The path of trajectory tr is called P_{tr}, and its starting time is $tr.t_0$. The duration function $Dur(tr, P) = TT_0 + TT_1 + \ldots + TT_{l-1}$ returns the sum of all segment traversal times a^P_{tr} of a path P by a trajectory. If a trajectory path P_{tr} does not contain P as a sub-path , $Dur(tr, P)$ is undefined. A trajectory

set in our example road network from Figure C.1 is shown below:

$$tr_0\colon (0,u_1) \rightarrow \langle (A,0,3),(B,3,4),(E,7,4)\rangle$$
$$tr_1\colon (1,u_2) \rightarrow \langle (A,2,4),(C,6,2),(D,8,4),(E,12,5)\rangle$$
$$tr_2\colon (2,u_2) \rightarrow \langle (A,4,3),(B,7,3),(F,10,6)\rangle$$
$$tr_3\colon (3,u_1) \rightarrow \langle (A,6,3),(B,9,3),(E,12,4)\rangle$$

2.3 Travel-Time Query

To address the shortcomings of the segment-level approach, we employ the strict path query $Q = spq(P, I, f, \beta)$ that returns a travel-time histogram H. The histogram can be derived from the traversal times of the set of trajectories $\mathcal{T}^P \subseteq \mathcal{T}$ that traverse path P without stops or detours in the time interval I, and fulfill additional filter predicates f:

$$\mathcal{T}^P = \{tr \in \mathcal{T} | \exists i,j\,(P_{tr}[i,j) = P \wedge tr.s.t_i \in I \wedge f(tr))\},$$

where $I = [t_s, t_e)$ denotes a temporal predicate with a size $\alpha = t_e - t_s$ and β is a cardinality requirement for $\mathcal{T}^P$, i.e., we only proceed if $|\mathcal{T}^P| \geq \beta$. If β is omitted all eligible trajectories are retrieved. The temporal predicate can either cover a fixed time interval, e.g., all trajectories from December 1st 2017 until May 1st 2018, or a periodic time-of-day interval denoted as $I^R = \langle \ldots, \big[t_s - 24\,hours, t_e - 24\,hours\big), \big[t_s, t_e\big), \big[t_s + 24\,hours, t_e + 24\,hours\big), \ldots, \big[t_s + n\,(24\ hours), t_e + n\,(24\ hours)\big)\rangle$, e.g., all trajectories from 8:00 until 8:30 on every day. Parameter f is an additional non-temporal filter predicate that trajectories in $\mathcal{T}^P$ have to fulfill, e.g., being from a specified driver.

Using such a query Q for a typical trip path, which can consist of dozens of segments, may not return a sufficient number of trajectories to derive accurate travel-time estimations. To address this problem, we split Q into k sub-queries $\langle Q_1, Q_2, \ldots, Q_k\rangle = \langle spq(P_1, I_1, f_1, \beta), spq(P_2, I_2, f_2, \beta), \ldots, spq(P_k, I_k, f_k, \beta)\rangle$ that return the trajectory sets $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$, where P_i are sub-paths that partition P. These can then be used to compute a set of k histograms $\{H_1, H_2, \ldots, H_k\}$ if $\forall i\,|\mathcal{T}_i| \geq \beta$. Their convolution we call $H = H_1 * H_2 * \ldots * H_k$, where $*$ is the discrete convolution operator and H is a travel-time histogram that covers the full path P. The intuition behind partitioning into k sub-queries is, that different sub-paths often provide better estimates with different predicates, e.g., user predicates mainly improve accuracy outside of cities [2]. Another advantage of partitioning the query is the increased number of eligible trajectories. How this partitioning into sub-queries is performed and how the sub-queries are processed is discussed in Sections 3 and 4.

An example query for our example trajectory set could be $Q = spq(\langle A, B, E\rangle, [0, 15), u = u_1, 2)$. This would return $\mathcal{T}^P = \{tr_0, tr_3\}$ yielding a histogram with $H = \{[10, 11)\colon 1; [11, 12)\colon 1\}$ since $Dur(tr_0, \langle A, B, E\rangle) =$

11 and $Dur(tr_3, \langle A, B, E \rangle) = 10$. But if a larger cardinality is required, Q could be split into two queries $Q_1 = spq(\langle A, B \rangle, [0, 15), \emptyset, 3)$ and $Q_2 = spq(\langle E \rangle, [0, 15), \emptyset, 3)$ that yield the histograms $H_1 = \{[6, 7) : 2; [7, 8) : 1\}$ and $H_2 = \{[4, 5) : 2; [5, 6) : 1\}$, from which the convolution $H = \{[10, 11) : 4; [11, 12) : 4; [12, 13) : 1\}$ can be obtained.

3 Query Processing

This section describes the architecture of the system, the processing of travel-time queries, and the greedy algorithm used for relaxing sub-query predicates.

3.1 Architecture

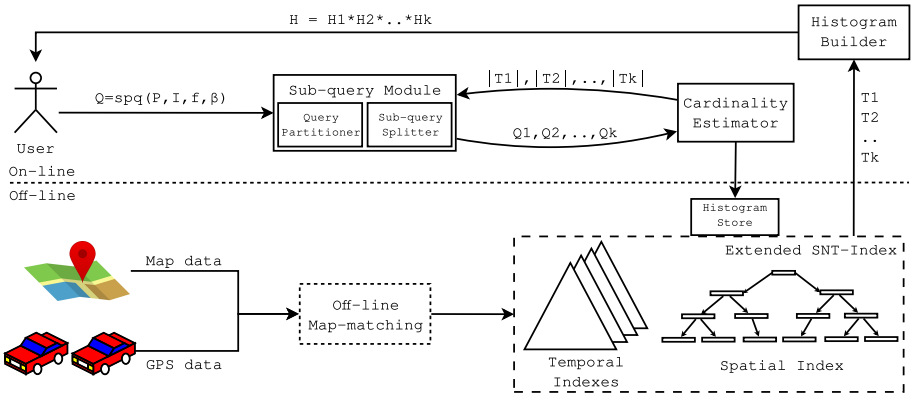


Fig. C.2: Overall Architecture

Figure C.2 shows the overall system architecture, where boxes with dotted lines indicate pre-existing components, dashed lines indicate modified components, and solid lines indicate new components. At first, a GPS data set is map-matched off-line to trajectories and loaded into the modified SNT-index consisting of a collection of temporal indexes and a spatial index.

Once the trajectory set is loaded, a user is able to dispatch a strict path query Q to the `Sub-query Module` where the query is initially partitioned into k sub-queries by the `Query Partitioner` according to a simple heuristic called π, e.g., sub-paths of a fixed length, or sub-paths that have the same segment category. Each sub-query is then assigned temporal and trajectory filter predicates. Next, the `Cardinality Estimator` uses the `Histogram Store` and the `SNT-Index` to estimate the cardinality $\hat{\beta}$ of the trajectory set $\mathcal{T}_i$ returned by the sub-query $spq(P_i, I_i, f_i, \beta)$. If $\hat{\beta}$ is smaller than the desired

cardinality β, the sub-query is modified by the `Sub-query Splitter` using a splitting function σ that relaxes the predicates. If the sub-query's cardinality estimate meets the requirement, it is dispatched to the index, and a trajectory set $\mathcal{T}_i$ is obtained. If $|\mathcal{T}_i| \geq \beta$, it is forwarded to the `Histogram Builder`. If the cardinality is still below the threshold, it is modified again by the `Sub-query Splitter`.

Once all trajectory sets $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$ are obtained, their travel-time sets $\{X_1, X_2, \ldots, X_k\}$ are extracted, with $X_i = \{Dur(tr, P_i) | tr \in \mathcal{T}_i\}$. From those, a set of histograms $\mathcal{H} = \{H_1, H_2, \ldots, H_k\}$ is computed, and they are convolved into a single histogram $H = H_1 * H_2 * \ldots * H_k$ that estimates the travel-time distribution for the complete path P.

3.2 Partitioning Methods

For the initial partitioning of queries, we propose five different methods. We use the query $Q = spq(P, I, f, \beta)$ with path $P = \langle A, C, D, E \rangle$ from the network in Figure C.1 as example. The initial periodic time interval I_i^R is identical for all sub-queries and is always chosen so that $t_e - t_s = \alpha_{min}$, where α_{min} is the minimum time interval size, which is chosen by the system. The predicate f is also initially identical for all sub-queries but may be modified by the splitting method (cf. Section 3.3).

Regular (π_p)

The regular partitioning creates sub-queries for paths of length p, i.e., every query is partitioned into $k = \lceil \frac{l}{p} \rceil$ sub-queries, i.e., the sub-queries $\pi_p(Q) = \langle spq(P[0,p), I_1^R, f_1, \beta), spq(P[p,2p), I_2^R, f_2, \beta), \ldots, spq(P[p\lfloor \frac{l}{p} \rfloor, l), I_k^R, f_k, \beta) \rangle$ are created. In our experiments we chose π_1, π_2 and π_3, which for our example path yield the paths $\langle \langle A \rangle, \langle C \rangle, \langle D \rangle, \langle E \rangle \rangle$, $\langle \langle A, C \rangle, \langle D, E \rangle \rangle$, and $\langle \langle A, C, D \rangle, \langle E \rangle \rangle$, respectively.

Segment Category (π_C)

The segment type partitioning creates partitions of sub-paths with identical segment categories, i.e., two neighboring segments e_i and e_{i+1} are split unless $\mathcal{F}(e_i).c = \mathcal{F}(e_{i+1}).c$. For our example query, this results in the sub-paths $\langle \langle A \rangle, \langle C, D \rangle, \langle E \rangle \rangle$.

Zone Type (π_Z)

The zone type partitioning creates partitions of sub-paths within the same zone type, i.e., two neighbouring segments e_i and e_{i+1} are split unless $\mathcal{F}(e_i).z = \mathcal{F}(e_{i+1}).z$. For our example query, this results in the sub-paths $\langle \langle A \rangle, \langle C, D, E \rangle \rangle$.

Zone Type & Segment Category (π_{ZC})

The zone type and segment category partitioning creates partitions of sub-paths within the same zone type and segment category combination, i.e., two neighboring segments e_i and e_{i+1} are split unless $\mathcal{F}(e_i).z = \mathcal{F}(e_{i+1}).z \wedge \mathcal{F}(e_i).c = \mathcal{F}(e_{i+1}).c$. For our example query, this results in the sub-paths $\langle\langle A\rangle, \langle C, D\rangle, \langle E\rangle\rangle$.

None (π_N)

No initial partitioning is attempted, and the query is processed according to one of the splitting strategies described below. For our example query, this results in the single sub-path $\langle\langle A, C, D, E\rangle\rangle$.

3.3 Splitting Methods

If a sub-query $spq(P, I, f, \beta)$ does not return the desired cardinality, it is modified by a splitting function σ described in Procedure C.1 that takes a query and the list of time interval sizes $A = \langle \alpha_1, \ldots, \alpha_n \rangle$, with $\forall i \forall j \, (i < j \Rightarrow \alpha_i < \alpha_j)$, and $\alpha_1 = \alpha_{min}$ and $\alpha_n = \alpha_{max}$ as arguments.

At first the procedure tries to increase the sample size by increasing the size of the time interval for the path by choosing the next largest size from the list A and widening the periodic interval with $widen([t_s, t_e)^R, \alpha_{i+1}) = [t_s - \frac{\alpha_{i+1}-\alpha_i}{2}, t_e + \frac{\alpha_{i+1}-\alpha_i}{2})^R$. After A has been exhausted, the path is split, and two new sub-queries with the smallest allowed time interval size α_{min} are created.

We propose two types of splitting and again use the path $P = \langle A, C, D, E\rangle$ in examples.

- σ_R Regular splitting cuts the path in half, i.e., $P_1 = P[0, \lfloor \frac{l}{2} \rfloor)$ and $P_2 = P[\lfloor \frac{l}{2} \rfloor, l)$, so splitting the example path P results in $P_1 = \langle A, C\rangle$ and $P_2 = \langle D, E\rangle$.

- σ_L Longest prefix splitting creates two sub-paths $P_1 = P[0, m)$ and $P_2 = P[m, l)$, with $1 \leq m < l$, where the maximum value for m for which $|\mathcal{T}^{P_1}| \geq \beta$ holds is chosen.

If a sub-path cannot be split further, any non-temporal filter predicates are dropped (Line 10). As a fallback, all temporal filters and the β parameter are dropped as well, i.e., for a single segment, all available trajectories are considered in the fixed time interval $[0, t_{max})$ (Line 12).

Algorithm C.1 Modify a sub-query spq to increase sample size (σ):

Require: Sub-query $spq(P, I, f, \beta)$, time interval sizes A
Ensure: a sequence of sub-queries $\langle Q_1, \ldots, Q_k \rangle$
1: $\alpha_i \leftarrow t_e - t_s$
2: **if** $\alpha_i < \alpha_{max}$ **then**
3: $I'^R \leftarrow widen(I^R, \alpha_{i+1})$
4: **return** $\langle spq(P, I'^R, f, \beta) \rangle$
5: **else if** $|P| > 1$ **then**
6: $m \leftarrow split(P)$
7: $I'^R \leftarrow shrink(I^R, \alpha_{min})$
8: **return** $\langle spq(P[0, m), I'^R, f, \beta), spq(P[m, l), I'^R, f, \beta) \rangle$
9: **else if** $f \neq \emptyset$ **then**
10: **return** $\langle spq(P, I^R, \emptyset, \beta) \rangle$
11: **else**
12: **return** $\langle spq(P, [0, t_{max}), \emptyset) \rangle$
13: **end if**

4 The Index

This section describes the SNT-index and how we adapt and optimize it to support travel-time queries using an example trajectory set.

4.1 SNT-Index

Koide et al. [15] proposed the SNT-index for strict path queries using the FM-index as a spatial index and a forest of B+-trees as a temporal index. The advantage of the FM-index over R-tree-based methods is that by representing the trajectory set $\mathcal{T}$ as a string T and adapting a method from substring matching, evaluating spatial queries is only dependent on the size of the spatial network ($|E|$) and not on the size of the trajectory set ($|\mathcal{T}|$). In addition, it can be established from just the FM-index whether a given path is traversed at all, often saving a costly temporal index traversal. While the original index returns a set of trajectory ids given the query $spq(P, I)$, where P is the path and I is a time interval, our index returns the traversal times of the trajectories for P, which can be stored in a histogram. Sections 4.1 and 4.1 recap the previously described SNT-index and the remaining section describes our modifications to it to facilitate the efficient retrieval of travel-times.

The Spatial FM-Index

For our example we are indexing the trajectory set $\mathcal{T} = \{tr_0, tr_1, tr_2, tr_3\}$ introduced earlier.

To index the trajectories, we first need to compute the trajectory string T from the alphabet $\Sigma = E \cup \{\$\}$ where the symbol \$ denotes the end of a trajectory and where $\forall e \in E\,(e > \$)$ and $T = P_{tr_0}\$P_{tr_1}\$ \ldots \$P_{tr_{n-1}}\$$, $\forall tr \in \mathcal{T}$. With our example trajectory set, this yields the trajectory string $T = ABE\$ACDE\$ABF\$ABE\$$.

Algorithm C.2 Calculate ISA range $[st, ed)$ for a path P of length l (*getISARange*):

Require: Burrows-Wheeler transform $T_{bwt}$of the trajectory string T, symbol counts C, path $P = p_0 ... p_{l-1}$
Ensure: ISA range $[st, ed)$ that matches P

```
c ← p_{l−1}
st ← C[c]
ed ← C[c + 1]
for i ← 2 to l do
    c ← p_{l−i}
    st ← C[c] + rank_c(T_bwt, st)
    ed ← C[c] + rank_c(T_bwt, ed)
    if st ≥ ed then
        return [0, 0)
    end if
end for
return [st, ed)
```

From this trajectory string, we compute an array S of all suffixes of T, where $S[i] = T[i, n)$, where $0 \leq i < n = |T|$. These suffixes are then sorted lexicographically to obtain the *suffix array SA* as shown in Figure C.3, where $SA[j]$ contains the index of the j-th smallest suffix. From SA, we can then compute the *inverse suffix array ISA* where $SA[j] = i$ and $ISA[i] = j$ [16]. Every substring (or in our case, subpath) P of length l therefore has a range of ISA values $R(P) = [st, ed)$ that is defined as $R(P) = \{i \mid S[SA[i]]\,[0, l) = P\}$, e.g., the *ISA* range of the path $\langle A \rangle$ is $R(\langle A \rangle) = [4, 8)$ since four trajectories contain A and they appear at the start of the suffixes $S[SA[st]]$ to $S[SA[ed - 1]]$, and the range for $R(\langle A, B \rangle)$ is $[4, 7)$ as only three trajectories traverse this path.

The ISA range of a path can be obtained efficiently from two data structures that comprise the FM-index:

C an array that stores the number of lexicographically smaller characters in the trajectory string for every member of the alphabet Σ, e.g., $C['B'] = 8$ since there exist 8 characters in T that are lexicographically before $'B'$.

T_{bwt} the Burrows-Wheeler transform [17] of the trajectory string T that is

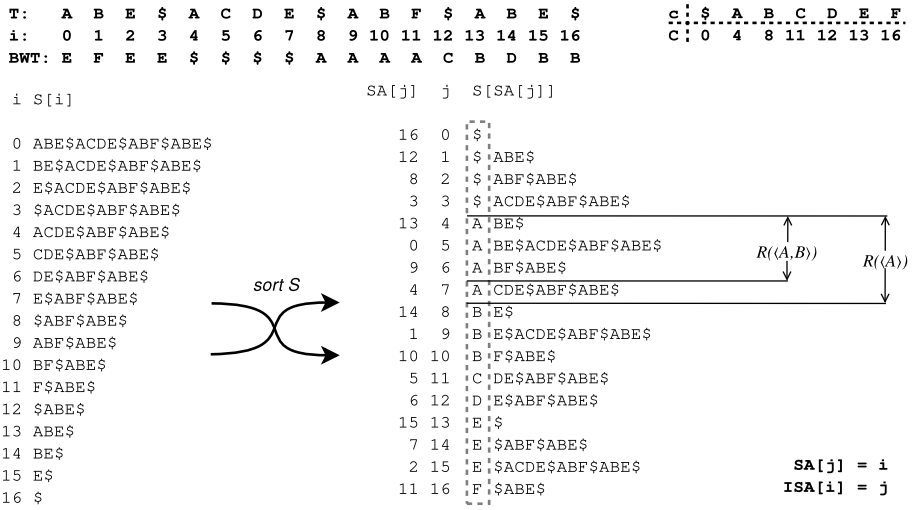


Fig. C.3: The Suffix Array and Burrows-Wheeler-Transform

defined as $T_{bwt}[i] = T[SA[i]-1]$, with $0 \leq i < |T|$. For our example, this yields the string $EFEE\$\$\$\$AAAACBDBB$.

We define the rank operation $rank_c(T_{bwt}, i)$ that counts the occurrences of the character c in $T_{bwt}[0, i)$. As an example of computing the ISA range, we compute the range for the path $P = \langle A, B \rangle$ as described in Procedure C.2. At first, the segment $c \leftarrow B$ is set in Line 1, and $st \leftarrow 8$ and $ed \leftarrow 11$ are initialized in Lines 2 and 3. For the first (and in this case the only) iteration of the loop from Line 4 to 11, $c \leftarrow A$, $st \leftarrow 4 + rank_A(T_{bwt}, 8)$, and $ed \leftarrow 4 + rank_A(T_{bwt}, 11)$, which yields the ISA range $[4, 7)$, since the ranks are 0 and 3, respectively.

The Burrows-Wheeler transform is stored in a wavelet tree to enable rank queries in $O(log|\Sigma|)$ time [18]. Therefore, obtaining the ISA range $[st, ed)$ of any path P can be performed in $O(|P|\ log\ |\Sigma|)$ time, which does not depend on the size of T.

The Temporal Indexes

The temporal indexes $F = \{\Phi_e |\ e \in E\}$ contain a B+-tree for every segment in the network. Each tree indexes the records $r \in \Phi$ by the timestamp t when a trajectory entered the segment. A leaf node entry r for a timestamp t contains the ISA index (*isa*) and the trajectory identifier (*d*).

The original SNT-index is only capable of retrieving the trajectory ids, which would then have to be processed in turn to obtain the traversal times of the query path.

Extensions to the SNT-Index

To support travel-time histogram construction directly using the SNT-index, we add the following information to each leaf node in a temporal index:

- The traversal time TT of the segment in seconds.
- The sequence number seq of the segment in the trajectory.
- The sum of the travel-times $a_{seq} = \sum_{i=0}^{seq} TT_i$ from the start of the trajectory and up to and including the segment.

Figure C.4 shows the contents of the temporal index Φ_A of segment A for our example trajectory set where each leaf is a record r, mapping a timestamp t to a tuple (isa, d, TT, a, seq). Furthermore, we add an associative container U that maps every trajectory id d to its respective user id u to check the filter predicate f. With those fields, we can build a hash table during the scan of the index of the first segment with the trajectory id and sequence number as the key (d, seq) and the aggregate of the preceding segment of the trajectory $(a_0 - TT_0)$, as value as described in Procedure C.3. The sequence number is included to guard against trajectories with circular paths. The spatial filtering is performed with the ISA range $[st, ed)$ obtained from Procedure C.2 during the index scan in Line 3. The filter predicate f can be evaluated in constant time with the associative container U. The cardinality parameter β is used to reduce the processing time since not all eligible trajectories are necessary to obtain a good estimate, and the *buildMap* procedure terminates as soon as β trajectories are found (Line 6). When scanning the temporal index of the last segment in the query, we can obtain the traversal time of the query path by $a_{l-1} - (a_0 - TT_0)$ as described in Procedure C.4.

4.2 Travel-Time Query

When used together, the previous three procedures make it possible to obtain the set of travel times for any path, as shown in Procedure C.5, to answer the sub-query $spq(P, I, f, \beta)$. To obtain all trajectories that traversed a path P during a given time interval I, an ISA range is first obtained from the FM-index in Line 1. If a non-empty range is returned, a range scan on the index of the first (Line 6) and last segment (Line 11) of the path are performed for I and filtered by the ISA index in the leafs. If no matching trajectories exist or no periodic time interval with more than β trajectories is found (Line 7) the query returns the empty set. If the sub-query provided by Procedure C.1 has a fixed time interval, the query is processed regardless of β. If that still yields no trajectories, an estimate based on the speed limit of the segment is provided (Line 13).

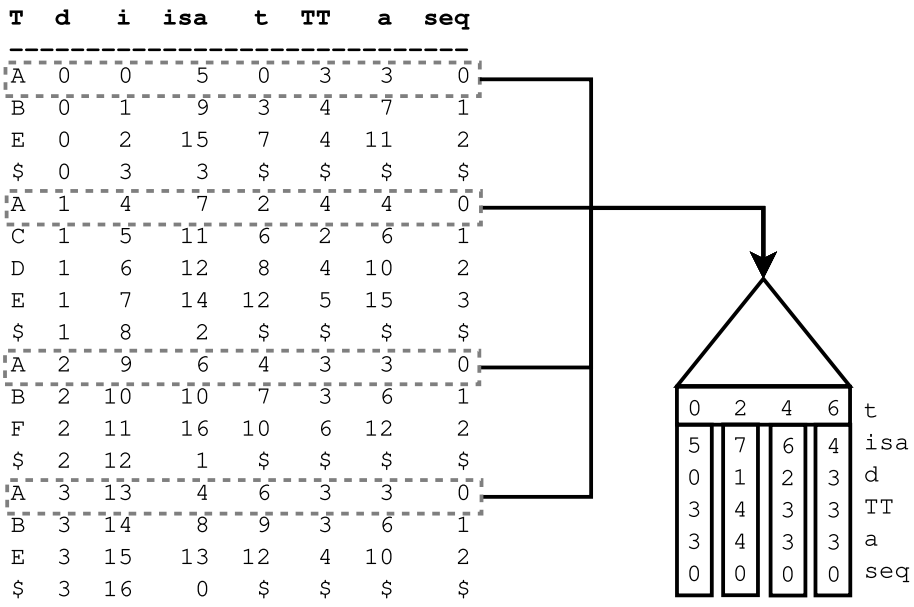


Fig. C.4: Extended Temporal Index

Procedure C.6 shows how a full query is partitioned and processed. For longer trips, the periodic interval I_i^R is adapted with the shift-and-enlarge procedure (Line 4) suggested by Dai et al. [6], that shifts the beginning of the interval t_s by the sum of all previous minimums $S_i = \sum_{j=1}^{j=i-1} H_j^{min}$ and enlarges it by the sum of all previous ranges $R_i = \sum_{j=1}^{j=i-1} (H_j^{max} - H_j^{min})$.

4.3 Optimizations

CSS-Trees

The cache sensitive search tree (CSS-tree) proposed by Rao and Ross is a low memory overhead pointer-less index that speeds up searches in sorted arrays [19]. In our system, we use it as an append-only replacement for the temporal B+-tree forest (cf. Section 4.1) to speed up Procedures C.3 and C.4 and to reduce memory consumption. Furthermore, its ability to efficiently compute the size of a key range in logarithmic time is used to improve the accuracy of the cardinality estimator (cf. Section 4.4). The CSS-tree is optimized to reduce the number of cache misses during a search by using the processor's cache line size as its node size. Since it indexes sorted arrays, only

Algorithm C.3 Create a mapping of trajectory identifier and sequence number (d, seq) to an antecedent travel time *diff* for the first β trajectories matching the predicates (*buildMap*):

Require: Temporal index Φ of the first segment of the query path, ISA range $[st, ed)$, time interval I, predicate f, and cardinality parameter β
Ensure: a mapping of (d, seq) to $(a - TT)$

$M \leftarrow \emptyset$
for all $r \in \Phi$ **do**
  if $r.t \in I \wedge st \leq r.isa < ed \wedge f(r.d)$ **then**
    $diff \leftarrow r.a - r.TT$
    $M \leftarrow M \cup \{(r.d, r.seq) \rightarrow diff\}$
    if $|M| \geq \beta$ **then**
      return M
    end if
  end if
end for
return M

Algorithm C.4 Compute the travel times for all eligible trajectories over the path identified in the *buildMap* function (*probeMap*):

Require: Temporal index Φ of the last segment of the query path, path length l, and probe table M
Ensure: a list of travel times X

$X \leftarrow \emptyset$
for all $r \in \Phi$ **do**
  $b \leftarrow M[(r.d, r.seq + 1 - l)]$
  if $b \neq \emptyset$ **then**
    $X \leftarrow X \cup \{r.a - b.diff\}$
  end if
end for
return X

appends can be performed efficiently. We deem this an acceptable trade off because inserting additional trajectories would also require a re-computation of the entire FM-index, making the index mostly suited for batch updates.

Temporal Partitioning

Temporal partitioning of the SNT-index was originally proposed here [20], but not evaluated. It allows more efficient updates to the index without necessitating a complete re-computation of the FM-index which does not efficiently support updates or appends. Partitioning requires to split the trajec-

Algorithm C.5 Retrieve all travel-times $X = \langle x_0, ..., x_{\beta-1} \rangle$ of trajectories in I that meet predicate f for a path P (*getTravelTimes*):

Require: Burrows-Wheeler transform of the trajectory string T_{bwt}, temporal indexes F, symbol counts C, path $P = p_0...p_{l-1}$, time interval I, predicate f, and cardinality parameter β
Ensure: a set of travel-times X
1: $[st, ed) \leftarrow getISARange(T_{bwt}, C, P)$
2: **if** $st \geq ed$ **then**
3: **return** $\emptyset$
4: **end if**
5: $\Phi_0 \leftarrow F[p_0]$
6: $M \leftarrow buildMap(\Phi_0, [st, ed), I, f, \beta)$
7: **if** $|M| < \beta$ **and** $isPeriodic(I)$ **then**
8: **return** $\emptyset$
9: **end if**
10: $\Phi_{l-1} \leftarrow F[p_{l-1}]$
11: $X \leftarrow probeMap(\Phi_{l-1}, l, M)$
12: **if** $X = \emptyset$ **and** $|P| = 1$ **then**
13: **return** $\{estimateTT(p_0)\}$
14: **end if**
15: **return** X

tory set $\mathcal{T}$ into $\mathcal{T}_1, \ldots, \mathcal{T}_W$, where $\forall i < j \left(\nexists tr_i \in \mathcal{T}_i \left(\forall tr_j \in \mathcal{T}_j \left(tr_i.t_0 \geq tr_j.t_0\right)\right)\right)$. From those trajectory sets, W trajectory strings $T^1, \ldots, T^W$ are then computed, and Procedure C.2 is modified return a collection of ISA ranges from the Burrows-Wheeler transforms $T^1_{bwt}, \ldots, T^W_{bwt}$ using separate segment counters $C^1, \ldots, C^W$. Temporal partitioning also requires adding the partition identifier w to every leaf in the temporal indexes since every partition's FM-index can return a different ISA range for the same path.

4.4 Cardinality Estimator

Cardinality estimators are widely used in DBMSs to improve query plans. In our case, we want to avoid costly scans of our temporal indexes if the required sample size β cannot be met. We require a function $card(Q)$ that returns an estimate $\hat{\beta}$ for the cardinality of the return trajectory set $\mathcal{T}$ and if $\hat{\beta} < \beta$, we apply the split function σ to Q without running a costly query. The cardinality estimator relies on a time-of-day histogram for every segment and fast computation of the ISA range $[st, ed)$, which is enabled by Procedure C.2. The exact count of all trajectories traversing a path $c_P = ed - st$ is efficiently retrieved. After that, the selectivity of the temporal filters needs to be estimated. The easiest way is to assume a uniform distribution throughout the

Algorithm C.6 Compute a histogram H for the query $spq(P, I, f, \beta)$ (*tripQuery*):

Require: Burrows-Wheeler transform of the trajectory string T_{bwt}, temporal indexes F, symbol counts C, query $spq(P, I, f, \beta)$, time interval sizes A, partitioning method π, and splitting method σ

Ensure: a histogram H

1: $\langle Q_1, \ldots, Q_k \rangle \leftarrow \pi(Q); \mathcal{H} \leftarrow \emptyset$
2: **for all** $Q_i \in \langle Q_1, \ldots, Q_k \rangle$ **do**
3: **if** *isPeriodic*(I_i) **and** $i > 1$ **then**
4: $I_i \leftarrow [t_s + S_i, t_e + R_i)^R$
5: **end if**
6: $X_i \leftarrow$ *getTravelTimes*(P_i, I_i, f_i, β)
7: **if** $X_i \neq \emptyset$ **then**
8: $\mathcal{H} \leftarrow \mathcal{H} \cup$ *createHistogram*(X_i)
9: **else**
10: $\langle Q_{i+1}, \ldots, Q_k \rangle \leftarrow \langle Q_{i+1}, \ldots, Q_k \rangle \cup \sigma(Q_i)$
11: **end if**
12: **end for**
13: $H \leftarrow H_1$
14: **for all** $i > 1 \wedge H_i \in \mathcal{H}$ **do**
15: $H \leftarrow H * H_i$
16: **end for**
17: **return** H

day and to divide the size of a periodic interval by the length of the day, which yields the time-of-day selectivity:

$$sel_{tod} = sel(P, I^R = [t_s, t_e)^R) = \frac{t_e - t_s}{24\,hours} \tag{C.1}$$

The uniformity assumption, however, usually does not hold so, the selectivity estimate can be improved by maintaining a time-of-day histogram H_e for every segment e. Then the selectivity can be estimated using the following formula:

$$sel(P, I^R = [t_s, t_e)^R) = \frac{B\big(H_{e_0}, [t_s, t_e)\big)}{B\big(H_{e_0}, [0, 24\,hours)\big)}, \tag{C.2}$$

where $B\big(H, [t_s, t_e)\big)$ counts the elements of all buckets in H in the range $[t_s, t_e)$. In addition to being constrained by the time-of-day a user might wish to limit the query to a certain time frame, e.g., only considering trajectories within the past year. The selectivity can be estimated naively with the following formula:

$$sel_{tf} = sel(P, I = [t_s, t_e)) = \frac{t_e - t_s}{F[e_0]_{max} - F[e_0]_{min}}, \tag{C.3}$$

where $F[e_0]_{min}$ and $F[e_0]_{max}$ are the earliest and latest traversal times of segment e_0. When using the CSS-tree, the number of entries for which $t_s \leq t < t_e$ can be obtained exactly in logarithmic time and sel_{tf} can be computed exactly. To compute the selectivity of user predicates sel_u, we use the default of $\frac{1}{10}$ suggested by Selinger et al. [21]. To obtain the estimate for a query, we combine these selectivity factors to obtain our estimate $\hat{\beta} = sel_{tod} * sel_{tf} * sel_u * c_P$.

We define five different modes for the cardinality estimator:

ISA only uses the size of the ISA range c_P as estimate $\hat{\beta}$

BT-Fast uses formulas C.1 and C.3 to estimate the selectivity

BT-Acc uses formulas C.2 and C.3 to estimate the selectivity

CSS-Fast uses formula C.1 and a fast lookup in the CSS-tree to estimate the selectivity

CSS-Acc uses formula C.2 and a fast lookup in the CSS-tree to estimate the selectivity

5 Experimental Setup

This section describes the data set and quality metrics we use to evaluate our system.

5.1 Datasets

OpenStreetMap

Our network graph is based on the OpenStreetMap data of the road network of Northern Denmark, which contains around 750,000 road segments. When converted to a spatial network graph, this graph has around 1.46 million directed edges [22]. Each edge represents a direction on a segment and has one of 17 different segment categories. This categorization is available for all OpenStreetMap maps and makes segment category-based partitioning possible for other map-matched trajectory datasets as well. The OpenStreetMap data also includes the speed limits for many segments, which we use as a fallback if no trajectory data is available. If the speed limit is not known, we use the median of all known speed limits of its segment category.

Zone Dataset

To distinguish rural and urban areas, we use the zoning map published by the Danish Business Authority [23] that consists of 4,259 zone geometries, each of which assigns one of three categories to an area:

- *city*: segments within city limits
- *rural*: segments in rural areas
- *summer house*: segments in areas zoned for summer house usage

A spatial join is used to assign a zone type to every segment in the map. A fourth category that we call *ambiguous* is assigned to segments located in more than one zone type.

ITSP Dataset

The "ITS Platform" dataset contains over 1.1 billion GPS points sampled at 1 Hertz collected from 458 vehicles in Aalborg and the surrounding region during the period from May 2012 to December 2014 [24].

In a preprocessing step, the GPS points are map-matched [25] to obtain in excess of 79 million segment traversals that form around 1.4 million trajectories, where a new trajectory is created if more than a 180 seconds have elapsed since the last GPS point. The map-matching algorithm also discards GPS points at the beginning and end of a trip if too few points are matched to the start and end segments of the trajectory. This is done so the durations of the segment traversals are meaningful. Each GPS record contains the trajectory ID, the vehicle ID, a segment ID, the time and date the segment was entered (minute resolution), and the time on segment (second resolution). Since the cars in our dataset are privately owned, we treat the vehicle ID as the user ID. The segment IDs are derived from the unique mapping of OpenStreetMap segment key and the driving direction. The time on a segment is also computed during the preprocessing step.

5.2 Query

We derive our query set $\mathcal{Q}$ from a random sample $\mathcal{T}_S \subset \mathcal{T}$ from our trajectory set:

$$\mathcal{Q} = \{spq(P_{tr}, I_{tr}, f, \beta) | tr \in \mathcal{T}_S\},$$

with either $f = \{u = tr.u\}$ or $f = \emptyset$ if no user filters are used and different values of β being used in the experiments. For the time interval I_{tr}, either the periodic time interval $I_{tr}^R = [tr.s.t_0 - \frac{\alpha_{min}}{2}, tr.s.t_0 + \frac{\alpha_{min}}{2})^R$ or or the fixed time interval $I_{tr} = [0, tr.s.t_0)$ is used.

For the interval size we use the values $15\,min$, $30\,min$, $45\,min$, $60\,min$, $90\,min$, and $120\,min$.

5.3 Accuracy Metric

sMAPE

To evaluate the accuracy of the retrieved traversal times, we use the symmetric mean absolute percentage error [26] of the sum of the means of all sub-paths.

$$\text{sMAPE} = \frac{100\%}{|\mathcal{Q}|} \sum_{i=1}^{|\mathcal{Q}|} \frac{|\sum_{j=1}^{k} \bar{X}_j - a_{tr_i}|}{\frac{1}{2}(\sum_{j=1}^{k} \bar{X}_j + a_{tr_i})},$$

where k is the respective number of sub-queries of each query $Q \in \mathcal{Q}$ and $\bar{X}_j$ is the travel-time mean retrieved with the sub-query.

Weighted Error

The weighted error, which we derive from sMAPE, considers the accuracy of the sub-query results and weighs them according to their fraction of the path length.

$$wE = \frac{100\%}{|\mathcal{Q}|} \sum_{i=1}^{|\mathcal{Q}|} \sum_{j=1}^{k} w_j \frac{|\bar{X}_j - a_{tr_i}^{P_j}|}{\frac{1}{2}(\bar{X}_j + a_{tr_i}^{P_j})},$$

with $w_j = \frac{\sum^{e \in P_j} \mathcal{F}(e).l}{\sum^{e \in P} \mathcal{F}(e).l}$, where P is the query path and P_j is the sub-query path.

Log-Likelihood

To evaluate the quality of the histograms, we compute the average log-likelihood of the travel-times a_{tr_i} with a discrete probability density function derived from the result histogram H_i.

For each trajectory with a result histogram H with a bucket width h, we compute the average log-likelihood $log\mathcal{L}$:

$$\frac{1}{|\mathcal{Q}|} \sum_{i=1}^{|\mathcal{Q}|} log\mathcal{L}(a_{tr_i}, H_i),$$

where the likelihood $\mathcal{L}(x, H)$ is defined by the discrete probability density function

$$p_H(x) = \gamma f(x, H) + (1 - \gamma)U(x),$$

where $U(x)$ is a uniform distribution defined for $[t_{min}, t_{max})$, $0 < \gamma < 1$ and

$$f(x, H) = \frac{B\big(H, [\lfloor \frac{x}{h} \rfloor, \lfloor \frac{x}{h} \rfloor + h)\big)}{B(H, [t_{min}, t_{max}))}.$$

The smoothing with the uniform distribution $U(x)$ is performed so that $p_H(x)\ \forall x \in [t_{min}, t_{max})$ never reaches zero.

Q-Error

To evaluate the accuracy of the cardinality estimator, we use the q-error proposed by Moerkotte et al. [27]. To estimate the quality of our cardinality estimate $\hat{\beta}$, we compare it to the actual cardinality of the retrieved trajectory set $n = |\mathcal{T}|$. For every estimate, we obtain the q-error $q = max(\hat{\beta}'/n', n'/\hat{\beta}')$ with $n' = max(n, 1)$ and $\hat{\beta}' = max(\hat{\beta}, 1)$. This is done to handle estimations for empty sets as proposed by Stefanoni et al. [28]. The q-error shows the difference in orders of magnitude between the real cardinality and the estimate.

6 Evaluation

This section reports on the experimental results. For all experiments, a query set $\mathcal{Q}$ is generated from the trajectory set $|\mathcal{T}_S| = 6{,}942$, which is a random 1% sample of all trajectories in $\mathcal{T}$ that occur after the 8th of September 2013, the median of the timestamps in the ITSP data set. This is to ensure that every query has more than a year of trajectory data available. On average, the paths of the query set have a length of 13.7 kilometers, consist of 55 segments, and last 800 seconds.

In our study we evaluate three types of queries:

Temporal Filters that use a periodic time interval and no user filter
$(spq(P_{tr}, I^R_{tr}, \emptyset, \beta))$

User Filters that use a periodic time interval and a user filter
$(spq(P_{tr}, I^R_{tr}, \{u = tr.u\}, \beta))$

SPQ Only that use a fixed time interval and no user filter
$(spq(P_{tr}, [0, t_{max}), \emptyset, \beta))$

6.1 Qualitative Assessment

Figures C.5 to C.8 show the results of accuracy measured with sMAPE, the weighted error, and the log-likelihood and the average sub-query length. The figures show the results for different types of partitioning and splitting methods and filter predicates.

The regular partitioning method π_p (cf. Section 3.2) is used as a baseline with $p = 1, 2$, and 3 because they are the sub-path lengths for which histograms can still be pre-computed at a reasonable overhead and because no

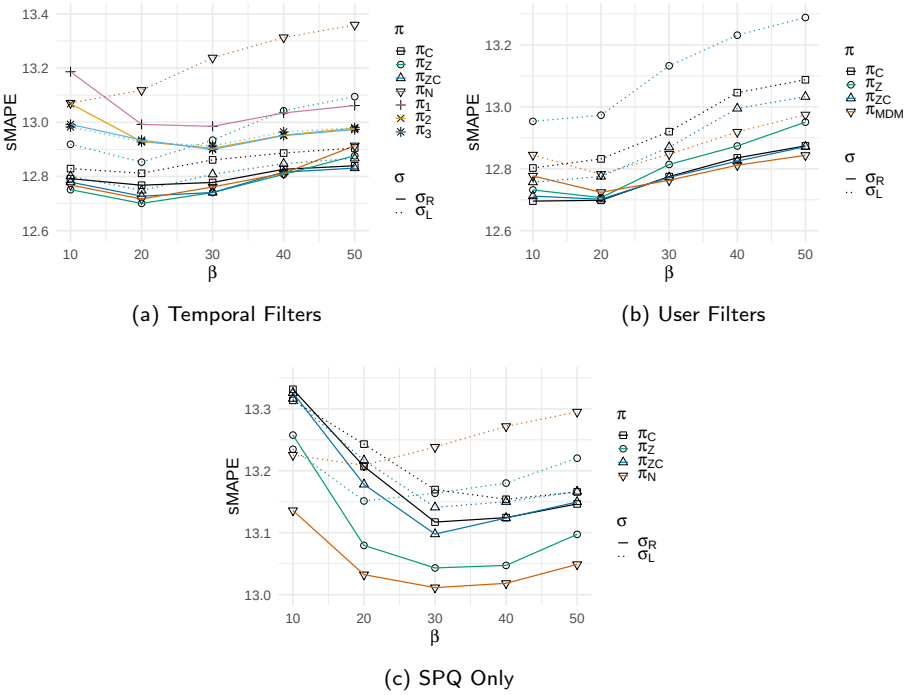
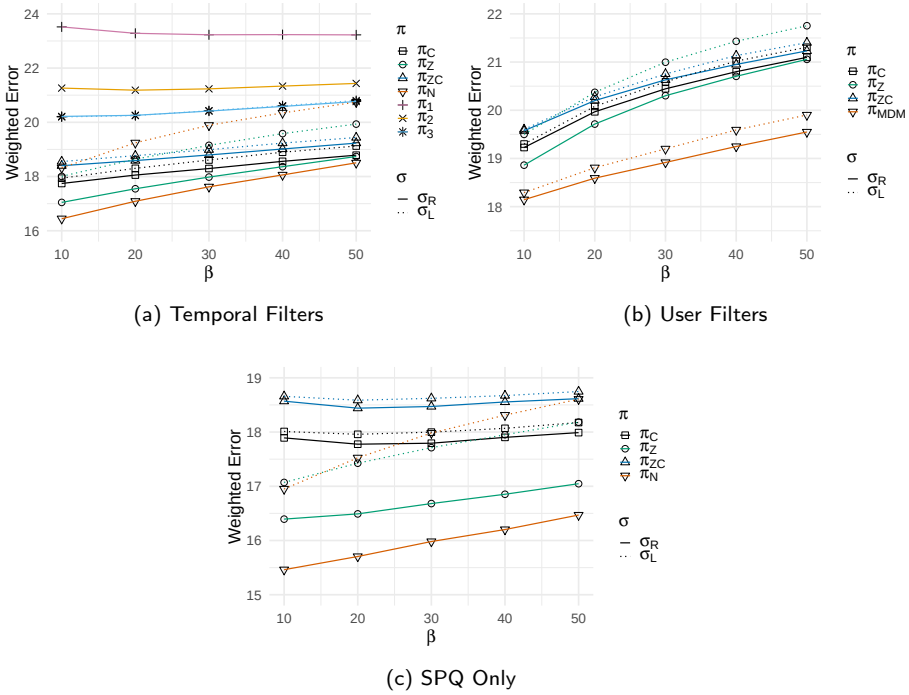


Fig. C.5: sMAPE

known histogram-based methods perform better. For the user filter queries, we also evaluate the π_{MDM} method that partitions queries like π_C but only applies user filters to sub-queries with paths on main roads like motorways or other major roads connecting cities. This partitioning method is derived from the results of a previous study of travel-time estimation methods [2].

Figure C.5a shows the average error for seven different partitioning methods with temporal filters. Here, π_1 performs worst, followed π_2 and π_3, and they achieve their highest accuracy at $\beta = 30$. If only the speed limits are used to estimate the travel time, sMAPE is 34.3% and if all available trajectories for each segment are used, the error is 13.8%. The partitioning methods based on the segment category and/or zone (π_C, π_Z, and π_{ZC}) together with π_N achieve very similar accuracy. Here, the accuracy peaks at $\beta = 20$. Category-based partitioning is the most stable in terms of accuracy, and zone-based partitioning provides the overall best result. The queries using user filters shown in Figure C.5b perform equally well, but with the exception of π_Z do not degrade as much with higher values of β and also obtain their lowest error at $\beta = 20$ and exhibit very similar accuracy to the queries without user filters. The SPQ Only methods in Figure C.5c methods did not manage to



(a) Temporal Filters

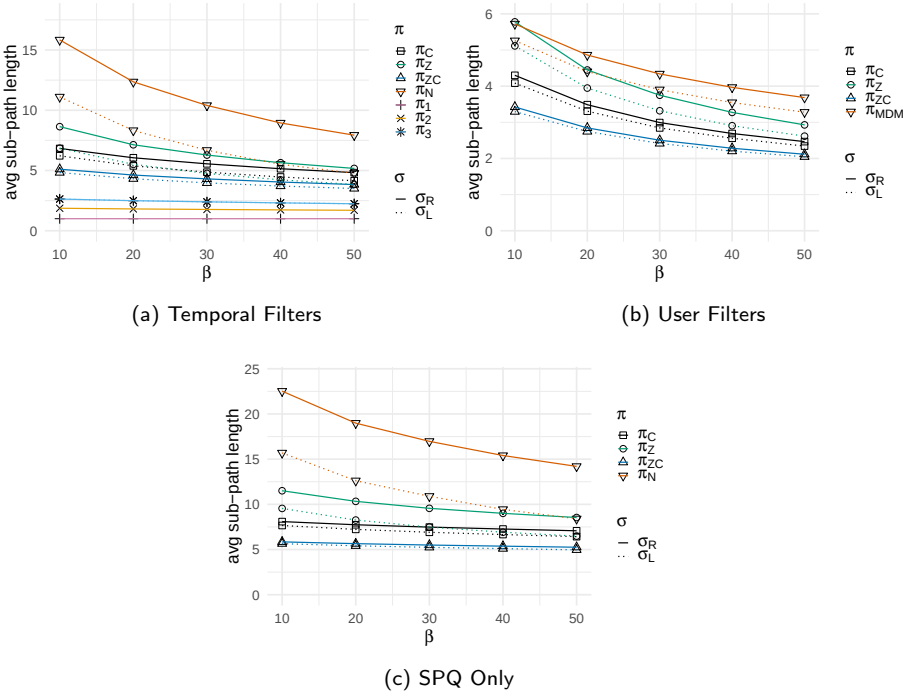
(b) User Filters

(c) SPQ Only

Fig. C.6: Weighted Error

outperform the baseline because it does not use periodic intervals that can observe changing congestion, e.g., longer travel-times during rush hours. In nearly all cases with regular splitting (σ_R) achieves considerably better accuracy than longest prefix splitting (σ_L). In most cases

A similar picture to the sMAPE results can be seen for the temporal filter queries in Figure C.6a, with π_N having the lowest weighted error. If only the speed limits are used to make an estimate, the weighted error is 36.9%, and if all available trajectories for each segment are used, the error is 24.0%. For the user filter queries in Figure C.6b, only π_{MDM} manages to consistently outperform the baseline. The SPQ only queries shown in Figure C.6c show the lowest error with the coarsest partitioning methods. The low error of the SPQ only methods is due to sub-query results being weighted according to the length of the sub-paths and not relative to the share of travel time. Estimates for paths on long segments with high speed limits, e.g., motorways, exhibit already low estimation errors and also tend to improve the most when custom predicates are used [2]. In all cases, σ_L has a higher error than σ_R. Figure C.7 shows the average path lengths of the final sub-queries. We can see that there is an inverse relationship between the weighted error and the



(a) Temporal Filters

(b) User Filters

(c) SPQ Only

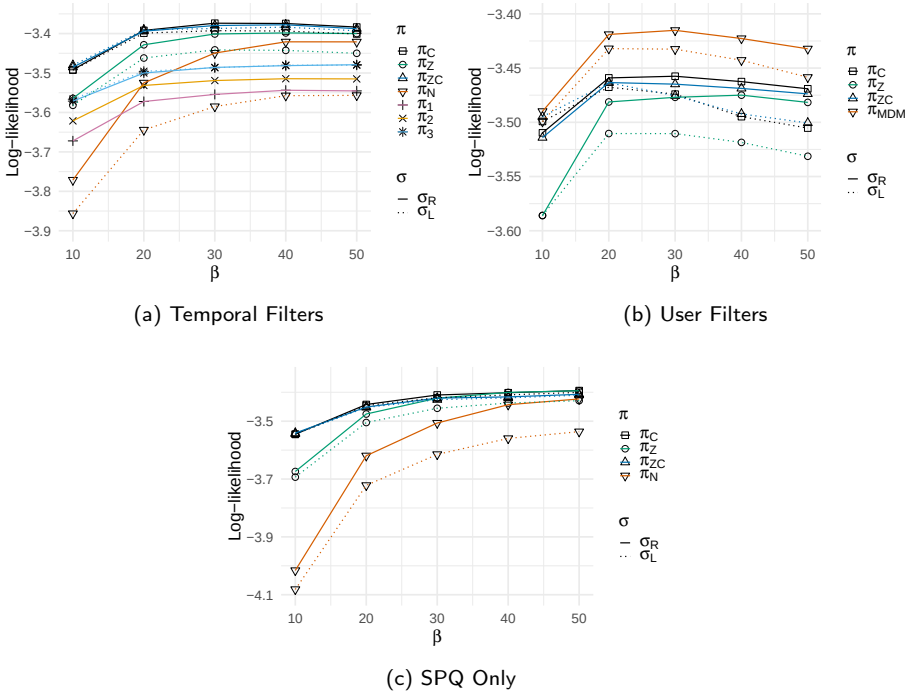
Fig. C.7: Sub-query Path Length

sub-query paths. We can also see that π_Z provides the coarsest partitioning with the exception of π_N, which initially provides none.

Figures C.8a to C.8c show the average log-likelihood with $f(x, H)$ derived form a histogram with a bucket size of $h = 10s$ and different values for β and $\gamma = 0.99$. The queries with only temporal filters and π_Z and π_{ZC} return the most accurate histograms, and the coarser the partitioning method the less accurate the histograms are with low sample sizes. Among the User Filter queries π_{MDM} consistently outperforms the other three partitioning methods. The queries run with π_N do not even outperform the baseline for $\beta < 30$. In all cases, σ_L performs worse than σ_R. We evaluated several values for γ (from 0.90 to 0.99) but the qualitative results did not change.

6.2 Efficiency

The index is implemented in C++17 and compiled with g++ 7.2.0 with the `-O3 -march=native` flags. For the performance test, the SNT-index with a CSS-forest and only a single partition is used. The FM-index is implemented using `sdsl-lite`'s integer-alphabet Huffman-shaped wavelet tree implementation,



(a) Temporal Filters

(b) User Filters

(c) SPQ Only

Fig. C.8: Log-Likelihood

and the suffix array is computed with Yuta Mori's `sais-lite` library [16]. The performance test ran on a server with AMD Opteron 6376 processors and 512 GiB RAM. For the processing time, the average runtime in milliseconds of 6,942 queries is reported in Figure C.9.

The temporal filter queries shown in Figure C.9a perform very similar to the baseline, with π_C and π_{ZC} being slightly faster than regular partitioning. The combination of π_C and σ_L has been omitted in the figure for reasons of scaling since the results are in the range of 50 to 65 ms. In Figure C.9b, it can be seen that the average user filter query takes around 4 to 5 times longer than the temporal filter queries; and with π_{MDM}, queries only take around twice as long since it applies non-temporal predicates only selectively. SPQ only queries have much lower processing times than do the other two query types, and all consistently outperform the baseline. The reason for their low processing times can be seen in Figure C.7c and Procedure C.5. Since their sub-queries tend to cover comparatively long paths, SPQ only queries need to perform considerably fewer temporal index scans than the other query types, which need to be split into more sub-queries to fulfill the cardinality requirements. The average runtime of π_N with σ_L, which is between 30 to

35 ms, has been omitted in Figure C.9c. In all cases σ_L performed poorly in comparison to σ_R.

6.3 Temporal Partitioning

Figure C.10 shows the effect of the temporal partitioning defined in Section 4.3 on memory consumption and setup time. The figures show the results for partition sizes of 7, 30, 90, and 365 days, resulting in a 138, 33, 11, and 3 partitions, respectively. We also examine the case where only one partition is used (FULL). Where applicable, the performance of the index with a B+-forest (BT) is reported as well. For the in-memory B+-trees, we use the `btree_multimap` from Google's `cpp-btree` library [29]. Figure C.10c shows the memory consumption of the different index components, where *Forest* is the memory consumption of the CSS-tree or B+-tree forest, respectively. The size of the forest is not impacted by different partition sizes, but if the partition feature is removed from the index, the memory saved in the tree leafs by omitting the partition identifier w is around 300 MiB for our data set. We can also see that the in-memory B+-tree forest has slightly higher memory requirements than the CSS-forest. The associative container U used to enable user filtering (user) is also not affected by the partitioning and takes up around 65 MB for our data set. The two data structures that comprise the FM-index, the wavelet tree (WT) and the segment counter (C), are affected considerably by partitioning. The segment counter grows linearly with the number of partitions from less than 6 MB to nearly 600 MB since a separate segment count needs to be maintained for every wavelet tree. The compression rate of the wavelet tree degrades considerably with smaller trajectory strings, which for the 7 day partitioning are only a few MBs per partition as opposed to several hundred in a single partition and it grows from around 280 MB to over 4 GB. The memory requirements of the time-of-day histograms required for the cardinality estimator are affected considerably if a histogram is maintained for every non-empty partition for every segment, and the memory required for the histograms soon exceeds the amount required for the index. Figure C.10a shows the memory consumption for three different bucket sizes h (1, 5, and 10 minutes).

The setup times for the index shown in Figure C.10b are not significantly affected by the different partition sizes or tree types and always remain between 425 and 475 seconds. For the setup, the trajectory and map data are loaded from disk.

6.4 Cardinality Estimator

Figure C.10 shows the results for the cardinality estimator. In all cases the results for partitioning method π_Z with regular splitting and $\beta = 20$ are

shown. Figure C.11c shows the q-error of the five different cardinality estimator modes. Here, 5,000 queries are run, after which their cardinalities n are compared with estimate $\hat{\beta}$. The simplest estimate using just the ISA range is on average off by an order of magnitude. The four other modes provide considerably more reliable estimates, with the histogram based methods performing better than the fast ones and the CSS-tree based methods performing slightly better than their B+-tree counterparts.

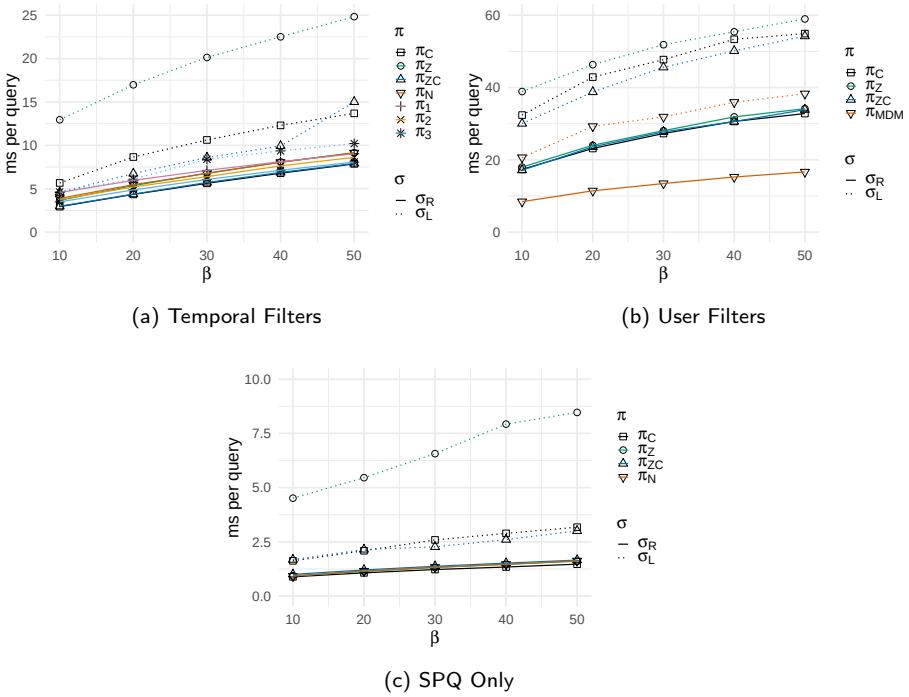
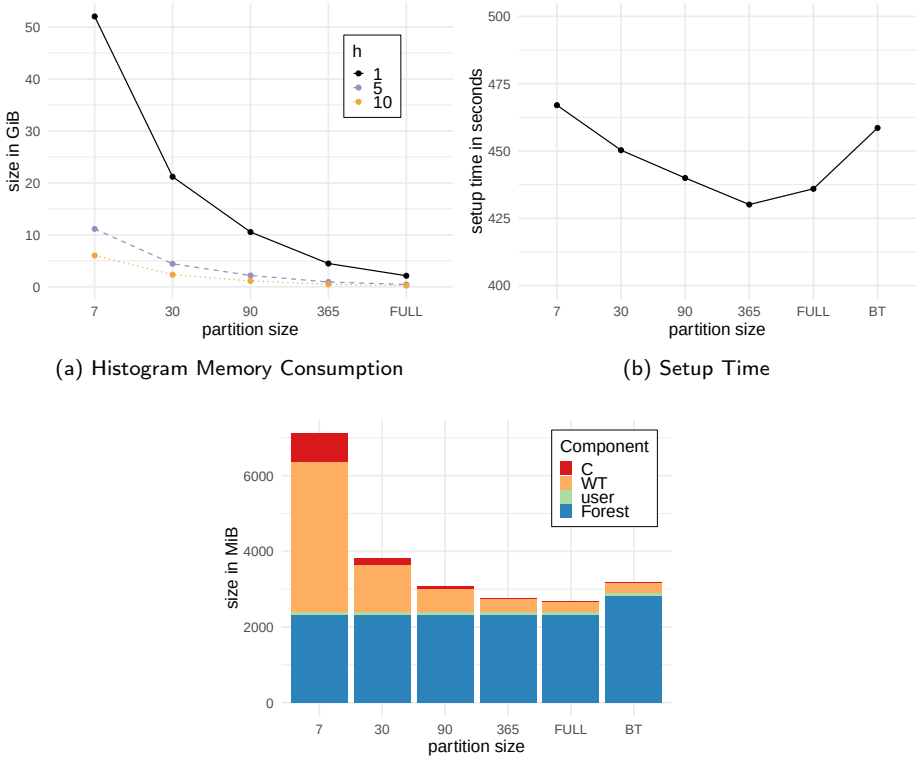


Fig. C.9: Processing Time

Since the selectivity estimates of the estimators might underestimate cardinalities of queries, a query might be split despite covering a sufficient sample size. This may affect the quality of the overall travel-time estimate. Figure C.11b, however, shows that the effects on quality are minuscule compared to the baseline (ISA) and might even yield slight improvements in accuracy.

Figure C.11a shows that partitioning as well as using the cardinality estimators can impact performance significantly. For single, yearly, and quarterly partitions, the query performance changes little, and use of the cardinality estimator reduces query processing times by around 50%. For smaller partitions, however, the effects of using the cardinality estimator diminish; and with weekly partitioning, the B+-tree version of the index performs worse



(a) Histogram Memory Consumption

(b) Setup Time

(c) Index Memory Consumption

Fig. C.10: Temporal Partitioning

with the estimators. The histogram-based CSS-tree version (CSS-Acc) performs worse than the fast version (CSS-Fast), which is most likely due to the amount of time-of-day histograms that have to be scanned to obtain the selectivity sel_{tod}.

6.5 Implications

Overall our data shows that after a certain β is reached no significant gains in accuracy are obtained by increasing it further indicating smaller result sets obtained from fewer SPQs of long paths provide more accurate estimates than larger result sets obtained with short paths. One can also see that evaluating non-temporal predicates comes with a considerable overhead and for the user predicates provides no improvement in quality over the purely temporal methods. If such methods are however applied selectively (e.g. π_{MDM}) the performance overhead is mitigated and the accuracy improves. The naive

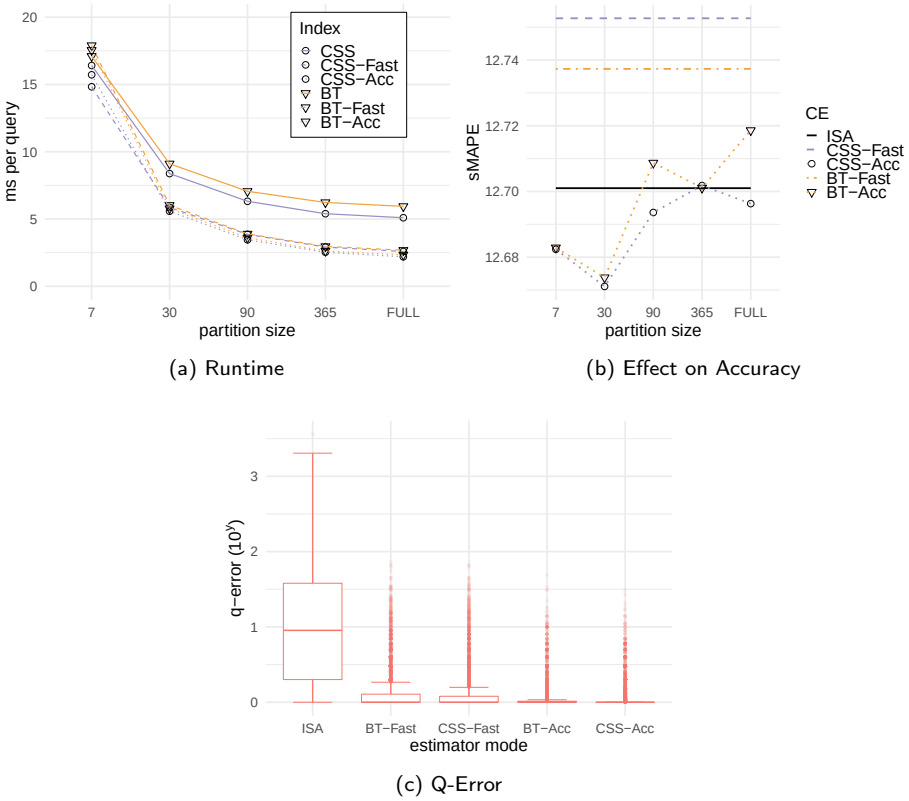


Fig. C.11: Cardinality Estimator

regular splitting method does not only achieve better accuracy but also has a considerably shorter runtime, making it more suitable for a real-time queries. The CSS-tree version of the index is as least as fast or faster than the B+-tree-based version, but the improvements become less noticeable when using the index in conjunction with a cardinality estimator. CSS-trees reduce the memory consumption of the index as well and improve the accuracy of the cardinality estimator with their efficient range lookups. We have also shown that temporal partitioning of the index is viable in some cases, but that using time-of-day histograms to estimate the selectivity of periodic time intervals, despite slight improvements in estimator accuracy and query performance, is hardly worth the memory overhead for the evaluated data set. Additional experiments with larger data sets may offer additional insight into this trade off, but no larger trajectory data sets with user information were available to us. Our results show that modifications aimed at improving query performance often also improve the accuracy of the estimates.

7 Conclusions & Outlook

Travel-time estimations in road networks can be improved considerable by utilizing large NCT data sets not only to provide estimates on a segment level, but also for full paths in the network. To our knowledge no current system supports these path-based estimations which cannot rely on pre-computations. We therefore propose a system that computes travel-time estimations based on trajectories selected at runtime and is able to improve upon the accuracy of existing histogram-based methods by expressing them as a series of strict path queries and adapting their predicates automatically to ensure accurate estimates. The SPQs are processed by our adapted SNT-index which is able to retrieve the traversal times for any path directly from the index. We have shown that the queries can be processed fast enough for real-time applications by utilizing specialized in-memory data structures and cardinality estimators tailored to SPQs. We evaluate our system with a large real-world trajectory data set and find that optimizing queries for performance is not preclusive of accuracy.

Our proposed system leaves several avenues of future work. The current greedy algorithm used for identifying a suitable partitioning and splitting of an SPQ is based on fairly simple heuristics and could be augmented by more sophisticated machine learning methods to improve accuracy of estimations. Approaches that use different values of the parameter β for each sub-query, e.g., smaller sample size requirements in rural zones, could be evaluated. While the processing time of single query might not considerably improve through parallelization the overall query throughput of the system most likely could, making it suitable for online routing applications that support a large number of users. Our approach also does not fully address the issue of data sparseness apart from providing relaxing the predicates if their selectivity is too low. Several approaches to solving the problem of data sparseness have been suggested [7, 30] and could be combined with our system to provide time-dependent travel-time estimates for paths where data is sparse.

Acknowledgments

This research was supported in part by a grant from the Obel Family Foundation and by the DiCyPS project.

References

[1] Eurostat, "Transport, volume and modal split," https://ec.europa.eu/eurostat/web/transport/data/main-tables, 2018.

[2] R. Waury, C. S. Jensen, and K. Torp, "Adaptive Travel-Time Estimation: A Case for Custom Predicate Selection," in *Proceedings of the 19th IEEE International Conference on Mobile Data Management*. IEEE, 2018, pp. 96–105.

[3] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp, "Path-based Queries on Trajectory Data," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2014, pp. 341–350.

[4] Y. Ma, B. Yang, and C. S. Jensen, "Enabling Time-Dependent Uncertain Eco-Weights For Road Networks," in *Proceedings of the Workshop on Managing and Mining Enriched Geo-Spatial Data*. ACM, 2014, pp. 1–6.

[5] S. Winter, "Modeling Costs of Turns in Route Planning," *GeoInformatica*, vol. 6, no. 4, pp. 345–361, 2002.

[6] J. Dai, B. Yang, C. Guo, C. S. Jensen, and J. Hu, "Path Cost Distribution Estimation Using Trajectory Data," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 85–96, 2016.

[7] Y. Wang, Y. Zheng, and Y. Xue, "Travel Time Estimation of a Path using Sparse Trajectories," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2014, pp. 25–34.

[8] C.-H. Wu, J.-M. Ho, and D.-T. Lee, "Travel-Time Prediction With Support Vector Regression," *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 4, pp. 276–281, 2004.

[9] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-Drive: Driving Directions Based on Taxi Trajectories," in *Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2010, pp. 99–108.

[10] D. Wang, J. Zhang, W. Cao, J. Li, and Y. Zheng, "When Will You Arrive? Estimating Travel Time Based on Deep Neural Networks," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. AAAI Press, 2018, pp. 2500–2507.

[11] E. Frentzos, "Indexing Objects Moving on Fixed Networks," in *Proceedings of the 8th International Symposium on Spatial and Temporal Databases*. Springer, 2003, pp. 289–305.

[12] V. T. De Almeida and R. H. Güting, "Indexing the Trajectories of Moving Objects in Networks," *GeoInformatica*, vol. 9, no. 1, pp. 33–60, 2005.

[13] Z. Ding, "UTR-tree: An Index Structure for the Full Uncertain Trajectories of Network-Constrained Moving Objects," in *Proceedings of the 9th IEEE International Conference on Mobile Data Management*. IEEE, 2008, pp. 33–40.

[14] I. S. Popa, K. Zeitouni, V. Oria, D. Barth, and S. Vial, "PARINET: A Tunable Access Method for in-Network Trajectories," in *Proceedings of the 26th IEEE International Conference on Data Engineering*. IEEE, 2010, pp. 177–188.

[15] S. Koide, Y. Tadokoro, and T. Yoshimura, "SNT-index: Spatio-temporal index for vehicular trajectories on a road network based on substring matching," in *Proceedings of the 1st International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. ACM, 2015, pp. 1–8.

[16] Y. Mori, "SAIS: An implementation of the induced sorting algorithm," 2008.

[17] M. Burrows and D. J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm," Digital Equipment Corporation, Tech. Rep., 1994.

[18] S. Gog, T. Beller, A. Moffat, and M. Petri, "From Theory to Practice: Plug and Play with Succinct Data Structures," in *Proceedings of the 13th International Symposium on Experimental Algorithms*. Springer, 2014, pp. 326–337.

[19] J. Rao and K. A. Ross, "Cache Conscious Indexing for Decision-Support in Main Memory," in *Proceedings of the 25th International Conference on Very Large Databases*, 1999, pp. 78–89.

[20] S. Koide, Y. Tadokoro, T. Yoshimura, C. Xiao, and Y. Ishikawa, "Enhanced Indexing and Querying of Trajectories in Road Networks via String Algorithms," *ACM Transactions on Spatial Algorithms and Systems*, vol. 4, no. 1, pp. 1–41, 2018.

[21] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access Path Selection in a Relational Database Management System," in *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*. ACM, 1979, pp. 23–34.

[22] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org," https://www.openstreetmap.org, 2014.

[23] Erhvervsstyrelsen, "zonekort," http://kort.plandata.dk/spatialmap, 2018.

[24] Aalborg University, "ITS Platform," http://www.itsplatform.dk/, 2018.

[25] P. Newson and J. Krumm, "Hidden Markov Map Matching Through Noise and Sparseness," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2009, pp. 336–343.

[26] J. S. Armstrong, *Long-Range Forecasting: From Crystal Ball to Computer*, 2nd ed. John Wiley & Sons, Inc., 1985.

[27] G. Moerkotte, T. Neumann, and G. Steidl, "Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 982–993, 2009.

[28] G. Stefanoni, B. Motik, and E. V. Kostylev, "Estimating the Cardinality of Conjunctive Queries over RDF Data Using Graph Summarisation," in *Proceedings of the 27th International World Wide Web Conference*, 2018, pp. 1043–1052.

[29] Google Inc., "cpp-btree," https://code.google.com/archive/p/cpp-btree/, 2011.

[30] F. Zheng and H. Van Zuylen, "Urban link travel time estimation based on sparse probe vehicle data," *Transportation Research Part C: Emerging Technologies*, vol. 31, pp. 145–157, 2013.

Paper D

A NUMA-aware Trajectory Store for Travel-Time Estimation

Robert Waury, Christian S. Jensen, Kristian Torp

The paper has been accepted for publication in
Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL), 2019.

Abstract

The increasingly massive volumes of vehicle trajectory data that are becoming available hold potential to enable more accurate vehicle travel time estimation than hitherto possible. To enable such uses, we present a multi-threaded in-memory trajectory store that supports efficient and accurate travel-time estimation for road-network paths from network-constrained trajectories. The trajectory store employs advanced indexing to support so-called strict path queries that retrieve all trajectories that traverse a given path to provide accurate travel-time estimations. As a key novel feature, the store is designed and implemented to exploit modern non-uniform memory access (NUMA) systems. We provide a detailed experimental study of the performance of the trajectory store using a synthetic trajectory data set based on real traffic data which shows that query latency can be halved compared to our baseline system.

1 Introduction

Travel time is a fundamental metric in the planning of road transportation. With accurate travel times available, it is possible for transportation companies to optimize the use of their fleets. To compute the travel time of a path in a road network, Dijkstra's algorithm (or similar) is often used. To enable this, a road network is modeled as a weighted graph, where the edge weights are the travel times of the corresponding road segments. A drawback of Dijkstra's algorithm is that it does not take turn directions at intersections into account. This drawback can be eliminated if *network-constrained trajectory* (NCT) data is available. With such data it is possible to have different travel-times, e.g., depending on whether right or left turns occur on an intersection. With NCT data available we cannot only collect aggregate travel-times for edges, but also collect travel-time distributions for edges in the form of histograms. If those histograms are only collected on a per segment basis and a path estimate is obtained through the convolution of all the histograms of the segments along the path, inaccuracies can arise due to the distribution independence assumption that underlies convolution. By adopting a path-based approach, where histograms are also collected for some paths, fewer histograms need to be convolved when computing the distribution of a path. This in turn increases the accuracy of the travel time distribution of a path [1].

The availability of rapidly growing trajectory data sets requires trajectory stores to take advantage of the increasing parallelism of modern multi-core systems to efficiently analyze these data sets. To increase the number of cores in a system, hardware vendors now often ship multi-processor systems with a non-uniform memory access architecture (NUMA) as opposed to a symmetric multi-processor architecture. In a NUMA systems the cores are divided into disjoint groups that each have their own local main memory and are connected to the other groups via a communication network that has considerably lower bandwidth and higher latency than local memory accesses. To take full advantage of the increased parallelism of NUMA systems, it is essential to minimize memory accesses to remote memory regions.

Figure D.1 shows an example memory hierarchy of a NUMA system, where eight processing units (PU) form a single NUMA region with shared memory. If the memory in a different region is accessed, the access occurs via a processor interconnect that is considerably slower than the memory bus. Today, NUMA systems can often be found on single chips since producing multiple smaller and simpler processors and interconnecting them on a die is economically attractive for chip manufacturers since it increases chip yields during the fabrication process [2].

In this paper we propose the, to our knowledge, first NUMA-aware trajectory store which follows a data-oriented architecture. It is designed to ef-

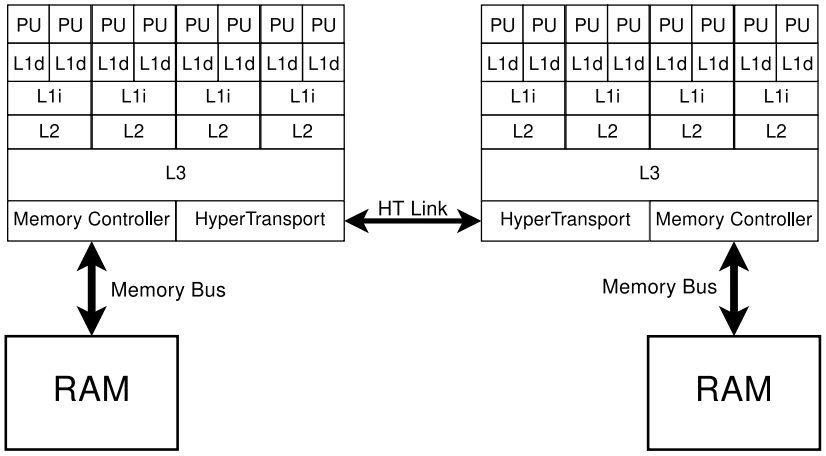


Fig. D.1: Memory Hierarchies in a NUMA system

ficiently query a large in-memory trajectory data set to answer travel-time queries consisting of a road network path and a start time. This is the first NUMA-aware query processing system that coordinates several different worker types which not only handle different data partitions as proposed by earlier systems but also separates the spatial and temporal dimension of a query for purposes of load balancing.

The main contributions of this paper are the following:

- Design of a trajectory store that can simultaneously handle updates and real-time queries.
- An implementation of a scalable NUMA-aware trajectory store.
- A detailed experimental evaluation of the proposed trajectory store.

The rest of the paper is structured as follows. Section 2 provides an overview of related work. Section 3 gives a detailed description of the architecture of the proposed trajectory store, while Section 4 provides implementation details. Section 5 introduces our experimental setup and reports on the experimental results, and Section 6 concludes.

2 Related Work

The proposed system relies on the efficient processing of so-called strict-path queries (SPQ) that retrieve trajectories relevant to a path. We review the most relevant approaches to processing SPQs as well as several NUMA-aware database systems.

2.1 Network-Constrained Trajectory Indexing

Several indexing methods have been proposed to enable efficiently processing SPQs.

NETTRA

The NETTRA index proposed by Krogh et al. [3] enables efficient SPQ processing on trajectory data. An SPQ finds all sub-trajectories that follow the same path between points A and B, i.e., trajectories without detours or extra stops. The NETTRA index uses path encoding based on prime numbers to determine the route of a trajectory. The NETTRA index targets disk-resident data and relies on conventional disk-based indexes such as B+trees for answering queries efficiently. Thus, the NETTRA index is not a main-memory index. Further, it and is designed to be updated with complete trajectories.

SNT-Index

The SNT-index proposed by Koide et al. [4] is an in-memory index for network constrained trajectories (NCT) that is optimized for SPQs. The SNT-index consists of a spatial index and a collection of temporal indexes, one for each road segment. The temporal index of a segment contains one entry for every traversal of that segment, consisting of the timestamp when the segment was entered, a trajectory ID, and a spatial predicate. As a preprocessing step to computing the spatial index, all segment traversals of the trajectory set are represented as strings so that the spatial index can utilize string indexing to identify whether a certain path exists in the trajectory set. The spatial index also computes a spatial filter predicate that consists of a value range that can be evaluated by means of two compare instructions when scanning the temporal indexes. To efficiently obtain this filter predicate, an FM-index [5] based on the Burrows-Wheeler transform [6] of the trajectory string is used. The FM-index can obtain the spatial filter predicate of a path P in $\mathcal{O}(|P| \, log \, \Sigma)$ time, where Σ is the size of the road network. After the predicate is computed, the temporal index of the first segment of the path is scanned, and all matching trajectory IDs are returned. Since the FM-index achieves considerable compression of trajectory strings, most of the memory footprint of the index is caused by the temporal indexes.

Extended SNT-Index

Our trajectory store utilizes the so-called Extended SNT-index [7] that supports not only the efficient retrieval of the IDs of trajectories that follow a path, but also the time needed to traverse the path. This is done by also storing the segment traversal time and its aggregate in the temporal indexes.

Because of the efficient spatial filtering method provided by the SNT-index, the Extended SNT-index can obtain the traversal times of a path with a single query on the FM-index and a scan of the temporal indexes of the first and last segments of the path, regardless of the length of the path. The scan of the first segment, called the build phase, collects the travel times of all trajectories that satisfy the spatial and temporal filters. The build phase is followed by a probe phase in which the path traversal times of the trajectories identified in the previous phase are computed. One of the biggest drawbacks of the SNT-index is that its spatial index, the FM-index, cannot be updated efficiently. To address this, temporal partitioning is used, and one FM-index is maintained for each partition.

2.2 NUMA-Aware Query Processing

Several query processing systems exist that are adapted to the NUMA architecture. They often follow a data-oriented architecture in which data placements aim to achieve improved cache utilization and more efficient memory access patterns. This is achieved through data partitioning and by minimizing coordination between the different partitions. This is similar to optimizations that would be applied to a distributed system.

OLTP

Pandis et al. propose DORA, a DBMS following a data-oriented design that uses a thread-to-data as opposed to a thread-to-transaction assignment for transactional workloads [8]. DORA partitions the data among its worker threads and manages accesses to each partition with thread-local data structures whenever possible thereby reducing the contention on the global lock manager, which is the main bottleneck in multi-threaded DBMSes.

Porobic et al. propose ATraPos which extends this to multi-socket systems [9] which also allows the repartitioning of the data among the workers based on the query load and the system topology.

OLAP

Several NUMA-aware systems optimized for analytical queries exist. Kissinger et al. follow a data-oriented architecture with their ERIS storage engine [10]. Specifically, ERIS partitions the data, and queries to each partition are processed by an "Autonomous Execution Unit" (AEU) that is a thread pinned to a core in the same NUMA region as the partition. ERIS achieves load balancing via repartitioning of the data among the AEUs.

Leis et al. propose a NUMA-aware extension to the HyPer system [11] that achieves load balancing through so-called work stealing.

3 Architecture

Our trajectory store receives two types of input:

Updates segment traversals that are ingested from an input stream and collected into trajectories and indexed.

Travel-Time Queries consisting of a path and a start time that return a travel-time histogram based on the indexed trajectories.

For our trajectory store we also follow a data-oriented architecture since temporal data lends itself well to partitioning. In contrast to the other systems, we implement more than one type of worker since we are not indexing relational data, but data with a spatial and temporal dimension and can therefore utilize specialized indexing techniques for each dimension.

3.1 Overview

As shown in Figure D.2, the system consists of a Dispatcher thread (Disp), an Update Worker (UW), several Spatial Workers (SW) and Temporal Workers (TW), and multiple Convolution Threads (CT). Every worker runs in its own thread, and the threads communicate with each other through either single-producer/single-consumer (SPSC) or multiple-producer/multiple-consumer (MPMC) queues. Queries and updates enter the system through the Dispatcher that enqueues updates into the Update Queue (UQ) and queries into the Query Queue (QQ). The Dispatcher also coordinates batch updates. We proceed to consider each type of worker in turn.

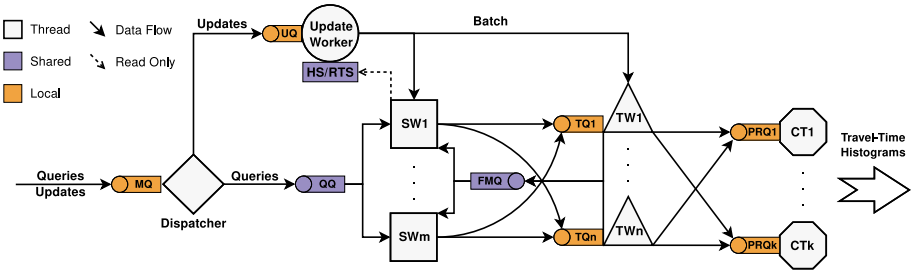
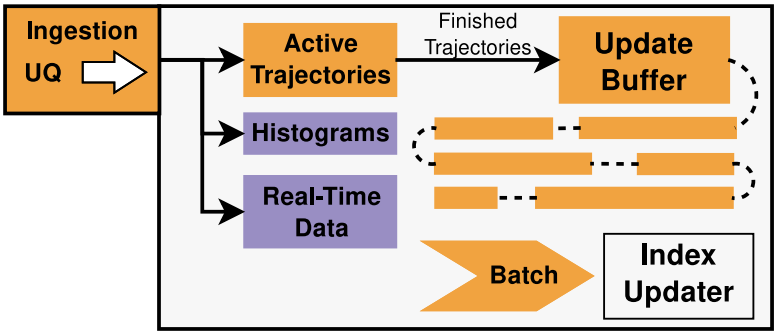


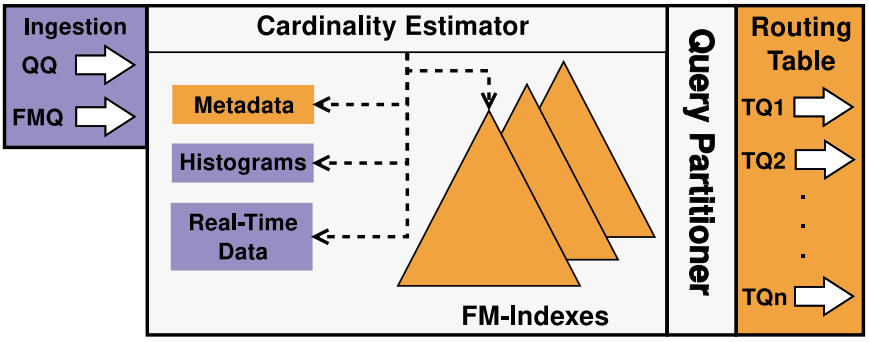
Fig. D.2: Architecture of the Trajectory Store

3.2 Update Worker

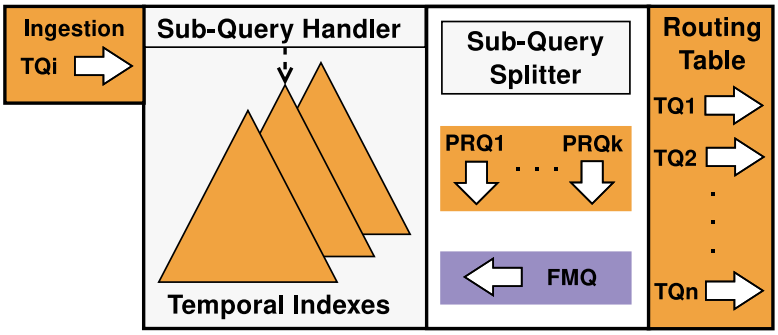
Figure D.3a shows the architecture of the Update Worker. The worker continually polls UQ for new update messages that are tuples containing a trajectory ID, a segment ID, a timestamp, and the traversal time of the segment.



(a) Update Worker



(b) Spatial Worker



(c) Temporal Worker

Fig. D.3: Worker Architecture

When an update arrives, it is inserted into the active trajectory set. If it is the last segment being traversed by an active trajectory, it is inserted into the Update Buffer that maintains a list of all trajectories to be inserted during the next batch update. Each update is also inserted into the Histogram Store (HS) that stores a time-of-day histogram for each segment in order of

the road network to track at which minute of the day a segment has seen traffic. For each segment, the Update Worker also maintains a running average of the most recent traversal times in the Real-Time Store (RTS), which enables the identification of outliers in real-time. When a batch update is triggered by the Dispatcher, all future finished trajectories are inserted into a new Update Buffer instance. The old Update Buffer computes a batch update of all finished trajectories since the last update. First it computes the Burrows-Wheeler transform of the trajectory string in order to compute the FM-index for the batch. Then all segment traversals for the batch are inserted into the temporal indexes of their respective Temporal Workers. Finally, a copy of the new FM-index is inserted into each Spatial Worker, which makes the new trajectory set accessible for SPQs.

3.3 Spatial Worker

As shown in Figure D.2, all Spatial Workers share the MPMC Query Queue (QQ) into which all queries are inserted. Each worker continually polls QQ for queries. Once a query is found, the global real-time data store is checked for any travel-time outliers on the path of the query. If any are found, their travel times are forwarded to the Convolution Thread of that query. After all outliers are removed, the remaining query is partitioned into multiple sub-queries using the network metadata. Queries are partitioned by a simple heuristic [7] to increase query accuracy and parallelism and the resulting sub-queries can be processed independently from each other. Then the time-of-day histograms from HS and the spatial filter predicates from the FM-indexes are used to derive a cardinality estimate for each sub-query. If the estimate is below a desired threshold β the partition is split, and a new estimate is obtained. Once a viable sub-query with a sufficient estimated cardinality is identified, it is enqueued into its respective Temporal Queue (TQ_i).

Since a temporal worker might still not find the required amount of trajectories, a sub-query may have to be split again, which triggers a recomputation of the spatial predicates. If that is necessary, the new sub-queries are inserted into the FM Queue (FMQ) by the Temporal Workers. The FMQ is also continually polled for new sub-queries.

Since any Spatial Worker can process any query, these workers also enable load balancing in the system in the sense that a long running query on one worker does not block the whole system.

3.4 Temporal Worker

Each Temporal Worker maintains temporal indexes of a subset of the segments in the network; these segments are assigned via a partitioning method. Each worker continually polls its Temporal Queue (TQ_i) that contains either a

build job consisting of a path, a time range and spatial predicates, or a probe job consisting of a list of timestamped partial results. If a build job is received, the specified temporal index is scanned for matching entries. If fewer than β entries are found, the sub-query is split and send back to the Spatial Workers via the FMQ to obtain updated spatial predicates. If the query is successful and consists only of a single segment, it is forwarded immediately to its Convolution Thread via the Partial Result Queue (PRQ). If not, a probe query consisting of the partial result is enqueued into the TQ of the Temporal Worker containing the last segment of the path. However, this is only done if the current partition does not own this segment. Otherwise, the probe job is processed immediately locally and the result enqueued into its PRQ.

The Temporal Workers support two types of segment partitioning:

Hash that assigns segments via a hash function $(hash(s_i) \bmod n)$.

Range that assigns an equal-sized range of segment IDs to each worker.

For range partitioning, we use the segment IDs assigned by our map data, where segment IDs are usually assigned by geographical proximity, i.e., adjacent edges are usually continuously numbered.

3.5 Convolution Thread

The Convolution Threads (CT) can be thought of as reducers. Each CT collects the partial result histograms of a subset of the query IDs and convolves them into a travel-time histogram covering the full query path. Once the last result histogram of a sub-query arrives at its CT, the resulting travel-time histogram for the full query path is emitted.

The convolution operation only requires few data points for every sub-query, but the operation is computationally expensive and is therefore performed in its own threads.

4 Implementation

This section describes the implementation details of the proposed main-memory index structure and how it is made NUMA-aware.

4.1 NCT Index

The trajectory store is an updated C++17 implementation of the Extended SNT-Index [7]. As the spatial index, an FM-index based on sdsl-lite's wavelet tree implementation [12] is used. The Burrows-Wheeler transform is computed with sais-lite-lcp [13]. As the temporal indexes, we use in-memory

B+-trees [14]. However, any multi-map that allows for ordered traversal can be used. For inter-thread communication, lock-free implementations of SPSC and MPMC queues are used [15, 16]. The Armadillo math library is used for histogram convolution [17]. All test programs for the experiments are compiled with `clang` version 8.0.1 with the options `-O3 -march=native`.

4.2 Placement Strategy

Usually the operating-system scheduler migrates threads between cores, which can cause suboptimal memory access patterns on NUMA systems because it pollutes the cache. Further, if a thread is moved to another NUMA region, this results in expensive remote memory accesses because Linux by default allocates memory according to a *first touch* policy [18]. Since Linux implements the *pthreads* API, we can control which core a thread is scheduled for by setting its affinity using the function `pthread_setaffinity_np`. This function ensures that a thread is scheduled only on a given set of cores.

In our system, we ensure that all Temporal Workers are scheduled for either the same NUMA region or adjacent ones to ensure that forwarding probe jobs can be done as locally as possible. Threads that share data structures are also scheduled for the same region or adjacent ones.

We do not schedule threads on all available cores since some compute resources are required for background tasks, like preparing batch updates or generating input data, that need to be performed asynchronously.

4.3 Allocation Strategy

Linux provides the `libnuma` C library as a drop-in replacement for the `malloc` and `free` memory-management functions.

`numa_alloc_onnode(size_t size, int node)` Allocates heap memory in a specified NUMA region.

`numa_alloc_local(size_t size)` Allocates heap memory in the region, the process is running in.

`numa_free` Frees memory allocated by the previous two functions.

With this library, we implement C++ allocators that ensure the placement of thread-local as well as shared data structures like the queues in local (and optimal) NUMA region. For example, the Temporal Queues are always allocated in the region where their worker is running.

5 Evaluation

This section describes the experimental setup and methodology and analyzes the experiments.

5.1 Experimental Setup

We proceed to cover the testing environment and methodology.

Data Sets

For the evaluation, we use the real-life road network graph of Northern Denmark provided by OpenStreetMap [19]. It contains more than 750,000 roads and more than 1.4 million directed edges.

To generate a sufficiently large trajectory data set for the experiments we use the ITS Platform data set that contains around 1.4 million map-matched trajectories comprised of over 79 million segment traversals collected in Northern Denmark over a time of over two years [20, 21]. To generate realistic workloads we treat all trajectories as if they were generated on a single day and then choose a random sample from that set for every batch.

Performance Metrics

We evaluate the system by three metrics:

Latency Measures how long it takes for a travel-time query to return a travel-time histogram.

Throughput Measures how many queries (or messages) can be dispatched in a fixed amount of time while still maintaining an acceptable latency.

Scalability Shows how increasing the amount of compute resources or data affects latency and throughput.

Latency is measured in milliseconds from dispatch to result, while throughput is measured in queries (or messages) per second. The testing setup for measuring latency is shown in Figure D.4. A Message Producer (MP) sends update and query messages to the Dispatcher. When a query is sent, a notification consisting of the query ID and the current timestamp is sent to the Result Thread (RT) that, upon receipt of the result travel-time histogram, reports the latency of the query.

The latency is deemed acceptable if it does not exceed 200 milliseconds which can be considered "instantaneous" in interactive applications [22].

We compare our NUMA-aware trajectory store against the "plain" version of our trajectory store that then serves as a baseline. For this, we leave the

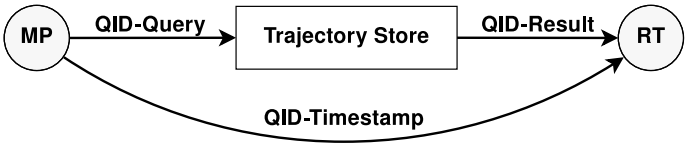


Fig. D.4: Testing Setup

allocation strategy to the standard allocators, and we leave thread placement entirely up to the operating-system scheduler. Apart from those modifications, the baseline runs the exact same code as our NUMA-aware trajectory store. We also evaluate the effect the different partitioning methods have on the performance metrics and how well our solution scales.

Test Hardware

Our tests are performed on the NUMA system described in Table D.1. It consists of four sockets that are connected via a HyperTransport network with the topology shown in Figure D.5a. With this topology, every region has at most a distance of 2 to any other region.

Figure D.5b shows the latency of reads measured in CPU cycles between the different NUMA regions that were obtained with the BenchIT tool [23]. We can see that local reads only incur roughly half the latency compared to reading remote memory. It also shows that small reads in the same region outperform remote reads by nearly an order of magnitude due to the speed of the shared L3 caches for local reads.

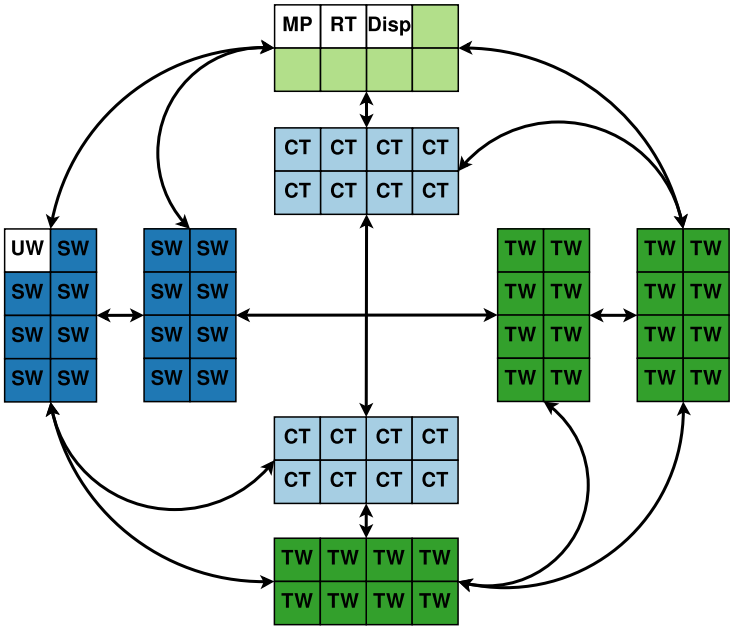
Processors	4x AMD Opteron 6376
Cache (per chip)	L2: 8x 2 MB / L3: 2x 6 MB
RAM	512 GB (64 GB/region)
Cores	64
NUMA Regions	8
OS	Ubuntu 14.04.3 LTS (3.13.0-147)

Table D.1: Test System

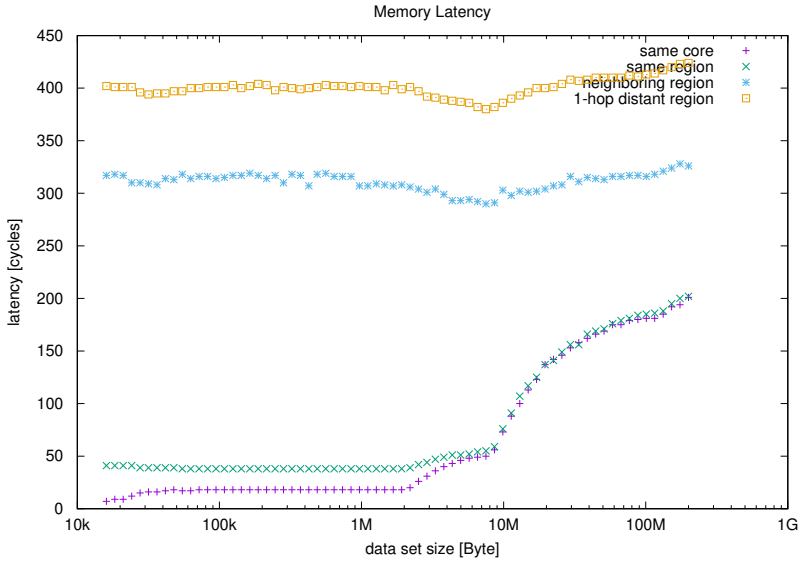
Test Setup

We test how throughput affects latency and how well our system scales in two settings:

Query-Only Loads an index and runs travel-time queries without additional updates.



(a) NUMA topology of our test system with an example thread placement



(b) Latencies of memory accesses on our test system

Fig. D.5: Test System

Real-Time Starts with an empty index that simultaneously receives updates and queries and performs periodical batch updates.

For each setting, three tests are performed with scalability settings described in Table D.2. It lists the batch size, the number of batches inserted, the number of queries run during each batch, and the numbers of workers of each type that are scheduled. A trajectory value of, e.g., 250,000, and a batch value of 10 mean 2.5 million trajectories were indexed in ten partitions (FM-indexes).

Test Type	Test Size	Batches/ Queries/ Trajectories	Throughput	Threads	SW/TW/CT
Query-Only	Region	10/1,000/ 250,000	5-25 Q/s	8	2/2/2
Query-Only	Socket	10/2,000/ 500,000	10-60 Q/s	16	4/4/6
Query-Only	System	10/5,000/ 2,500,000	10-60 Q/s	58	16/24/16
Real-Time	System	3/5,000/ 100,000	5,000 M/s	57	15/24/16
Real-Time	System	3/50,000/ 100,000	50,000 M/s	57	15/24/16
Real-Time	System	3/50,000/ 100,000	100,000 M/s	57	15/24/16

Table D.2: Test Sizes

5.2 Results

This section reports on experimental results for both test settings.

Query-Only Results

Results for the query-only setting are shown in Figure D.6. The black horizontal line denotes the 200 millisecond boundary.

We consider three scalability scenarios. For the optimized version, we schedule all worker threads on either a single region or on a single socket; for the baseline, we just the let the operating-system scheduler decide the placement. In the third scenario, we schedule a worker on nearly every core of the test system while allowing a few cores to remain available for background tasks.

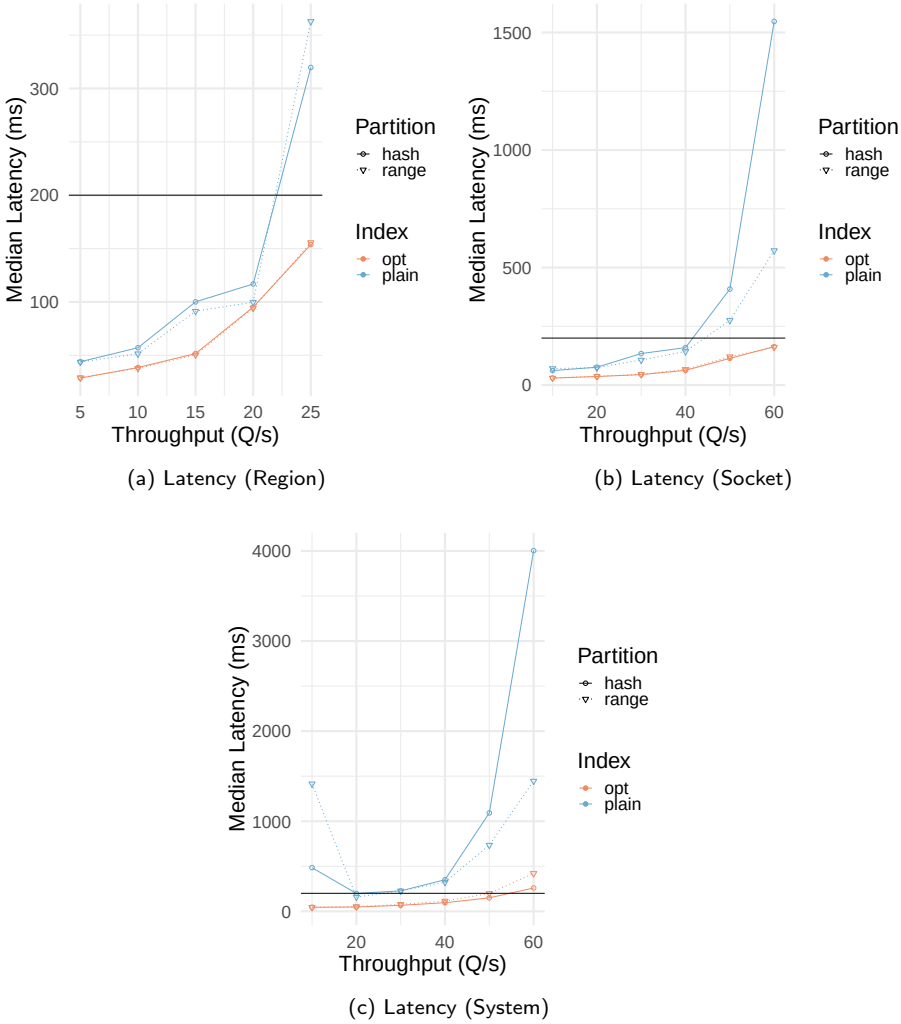


Fig. D.6: Query-Only

For all scalability scenarios, the NUMA-aware version of our trajectory store achieves improved latency by up to a factor of two in low-throughput settings. At higher throughput, the advantage of the NUMA-aware trajectory store is even more pronounced. We can also observe that the difference in performance is larger with higher core counts. For the unoptimized version of the system, we see that range partitioning performs considerably better than hash partitioning due to the better memory locality of this method, which leads to more probe jobs being processed without inter-thread communica-

tion. The better memory locality occurs because segment IDs are clustered spatially.

This effect cannot be observed for the optimized version; and the inverse is actually true when evaluating system level scalability since the better load balancing achieved by hash partitioning becomes noticeable.

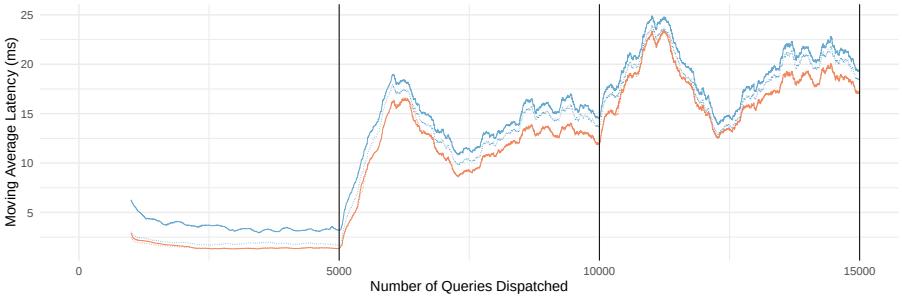
Not only does the optimized version achieve lower latency at all throughput rates, its latency also degrades more gracefully as the load increases. We can also observe that the number of cores can be scaled linearly with the data size while maintaining acceptable latency.

Real-Time Results

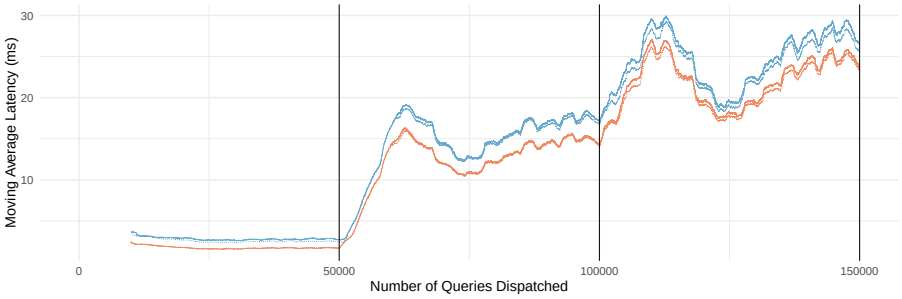
For the real-time tests, several batches containing both updates and queries are sent to the trajectory store. Three batches are inserted at different throughput rates and different query-to-update ratios. For all experiments, we use the thread placement shown in Figure D.5a. The results are shown in Figure D.7, where vertical bars indicate the point where a batch update was triggered. Before the first batch is triggered, only segment-level data from the RTS can be used; and if no data is available the speed limit of the segment is used.

Figure D.7a shows the moving average ($k = 1,000$) of the query latency of three batches, each consisting of 100,000 trajectories and 5,000 queries at a throughput of 5,000 messages per second. Figure D.7b shows the moving average ($k = 10,000$) of the query latency of three batches, each consisting of 100,000 trajectories and 50,000 queries at a throughput of 50,000 messages per second. Figure D.7c shows the moving average ($k = 10,000$) of the query latency of three batches, each consisting of 100,000 trajectories and 50,000 queries at a throughput of 100,000 messages per second.

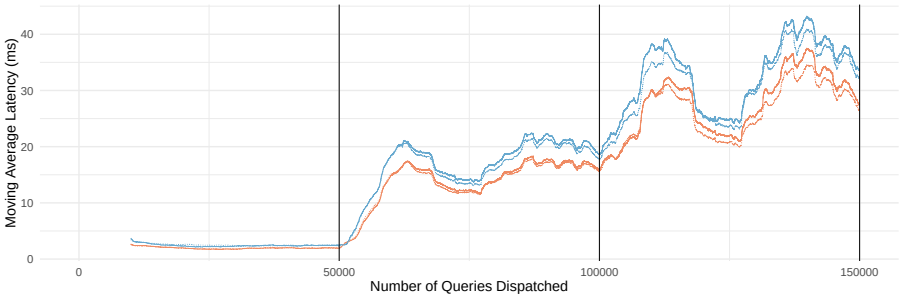
We can observe that even at the highest throughput rate, the average latency never exceeds 50 milliseconds; and even for the baseline, it remains within acceptable levels. The highest average latency can be observed during a batch update, during which the latency nearly doubles until the update is complete. Further, the difference in performance between the two versions is less pronounced than in the Query-Only scenario. However, during spikes in latency, the difference between the NUMA-aware trajectory store and the unoptimized trajectory store is also the biggest. This is consistent with the observation from the Query-Only experiments, where the performance of the optimized version degrades more gracefully under high loads. The difference between the partitioning methods is also less noticeable, with range partitioning only producing small reductions in latency, if any at all.



(a) 5,000 messages/second in three batches of 100,000 trajectories and 5,000 queries



(b) 50,000 messages/second in three batches of 100,000 trajectories and 50,000 queries



(c) 100,000 messages/second in three batches of 100,000 trajectories and 50,000 queries

Fig. D.7: Real-Time Latency

6 Conclusions & Outlook

In order for travel-time estimation in road networks to be able to take advantage of the increasingly massive trajectory data sets, means are needed that enables them to be ingested and indexed efficiently. We propose a novel trajectory store design and implementation that improves the performance of travel-time queries based on so-called strict-path queries and is optimized for non-uniform memory access (NUMA) computing hardware systems. With the rising number of processing units on a single chip, NUMA systems are becoming the norm even on single-socket machines. We show that our architecture can be adapted easily to be NUMA-aware with the modifications required to handle thread placement and NUMA-aware allocation making up less than 3.5% of the trajectory store's code base.

We evaluate our prototype system with a large synthetic data set based on a map-matched real-world trajectory data set. The NUMA-aware trajectory store outperforms a baseline implementation in all scenarios, and it performs especially well under high loads. When queried in a static setting or when being updated concurrently, the trajectory store delivers query latencies suitable for real-time applications.

The proposed trajectory store opens to several avenues of future work. It can be extended further by implementing an additional load balancing mechanism on the Temporal Worker level by repartitioning the segments either based on query load or by trying to maximize the co-location of build and probe jobs to the same worker. Furthermore, the multiple worker type architecture can also be adapted to applications beyond spatio-temporal data processing.

Acknowledgments

This research was supported in part by a grant from the Obel Family Foundation and by the DiCyPS project.

References

[1] R. Waury, C. S. Jensen, and K. Torp, "Adaptive Travel-Time Estimation: A Case for Custom Predicate Selection," in *Proceedings of the 19th IEEE International Conference on Mobile Data Management*. IEEE, 2018, pp. 96–105.

[2] T. Research, "AMD Optimizes EPYC Memory with NUMA," https://www.amd.com/system/files/2018-03/AMD-Optimizes-EPYC-Memory-With-NUMA.pdf, 2018.

[3] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp, "Path-based Queries on Trajectory Data," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2014, pp. 341–350.

[4] S. Koide, Y. Tadokoro, and T. Yoshimura, "SNT-index: Spatio-temporal index for vehicular trajectories on a road network based on substring matching," in *Proceedings of the 1st International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. ACM, 2015, pp. 1–8.

[5] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro, "An Alphabet-Friendly FM-index," in *Proceedings of the International Symposium on String Processing and Information Retrieval*. Springer, 2004, pp. 150–160.

[6] M. Burrows and D. J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm," Digital Equipment Corporation, Tech. Rep., 1994.

[7] R. Waury, C. S. Jensen, S. Koide, Y. Ishikawa, and C. Xiao, "Indexing Trajectories for Travel-Time Histogram Retrieval," in *Proceedings of the 22nd International Conference on Extending Database Technology*, 2019, pp. 157–168.

[8] I. Pandis, R. Johnson, N. Hardavellas, and A. Ailamaki, "Data-Oriented Transaction Execution," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 928–939, 2010.

[9] D. Porobic, E. Liarou, P. Tözün, and A. Ailamaki, "ATraPos: Adaptive Transaction Processing on Hardware Islands," in *Proceedings of the 30th IEEE International Conference on Data Engineering*. IEEE, 2014, pp. 688–699.

[10] T. Kissinger, T. Kiefer, B. Schlegel, D. Habich, D. Molka, and W. Lehner, "ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workloads," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, pp. 1–12, 2014.

[11] V. Leis, P. Boncz, A. Kemper, and T. Neumann, "Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 2014, pp. 743–754.

[12] S. Gog, T. Beller, A. Moffat, and M. Petri, "From Theory to Practice: Plug and Play with Succinct Data Structures," in *Proceedings of the 13th International Symposium on Experimental Algorithms*. Springer, 2014, pp. 326–337.

[13] J. Fischer, "Inducing the LCP-Array," in *Proceedings of the Workshop on Algorithms and Data Structures*. Springer, 2011, pp. 374–385.

[14] Timo Bingmann, "tlx," https://github.com/tlx/tlx, 2019.

[15] C. Desrochers, "A single-producer, single-consumer lock-free queue for c++," https://github.com/cameron314/readerwriterqueue, 2019.

[16] ——, "moodycamel::concurrentqueue," https://github.com/cameron314/concurrentqueue, 2019.

[17] Armadillo contributors, "Armadillo C++ library for linear algebra & scientific computing," http://arma.sourceforge.net/, 2019.

[18] C. Lameter *et al.*, "NUMA (Non-Uniform Memory Access): An Overview." *ACM Queue*, vol. 11, no. 7, p. 40, 2013.

[19] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org," https://www.openstreetmap.org, 2014.

[20] Aalborg University, "ITS Platform," http://www.itsplatform.dk/, 2018.

[21] P. Newson and J. Krumm, "Hidden Markov Map Matching Through Noise and Sparseness," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2009, pp. 336–343.

[22] G. Inc., "Improve Server Response Time," https://developers.google.com/speed/docs/insights/Server?hl=en, 2019.

[23] G. Juckeland, S. Börner, M. Kluge, S. Kölling, W. E. Nagel, S. Pflüger, H. Röding, S. Seidl, T. William, and R. Wloch, "BenchIT - Performance Measurements and Comparison for Scientific Applications," *Advances in Parallel Computing*, vol. 13, pp. 501–508, 2004.

Paper E

Analyzing Trajectories Using a Path-based API

Robert Waury, Christian S. Jensen, Peter Dolog, Kristian Torp

The paper has been published in
Proceedings of the 16th International Symposium on Spatial and Temporal Databases, pp. 198–201, 2019.
DOI: 10.1145/3340964.3340990

Abstract

Large vehicle trajectory data sets can give detailed insight into traffic and congestion that is useful for routing as well as transportation planning. Making information from such data sets available to more users can enable applications that reduce travel time and fuel consumption. However, extracting such information efficiently requires deep knowledge of the underlying schema and indexing methods. To enable more users to extract information from trajectory data, we have developed an API that removes the need to be familiar with the schema. Furthermore, when giving access to trajectory data, privacy concerns often call for the application of anonymization methods before analysis results are made available. In our demonstration, owners of trajectory data are able to experiment with different levels of anonymization to see how this affects the quality of different types of trajectory analysis services implemented on top of a large trajectory data set.

1 Introduction

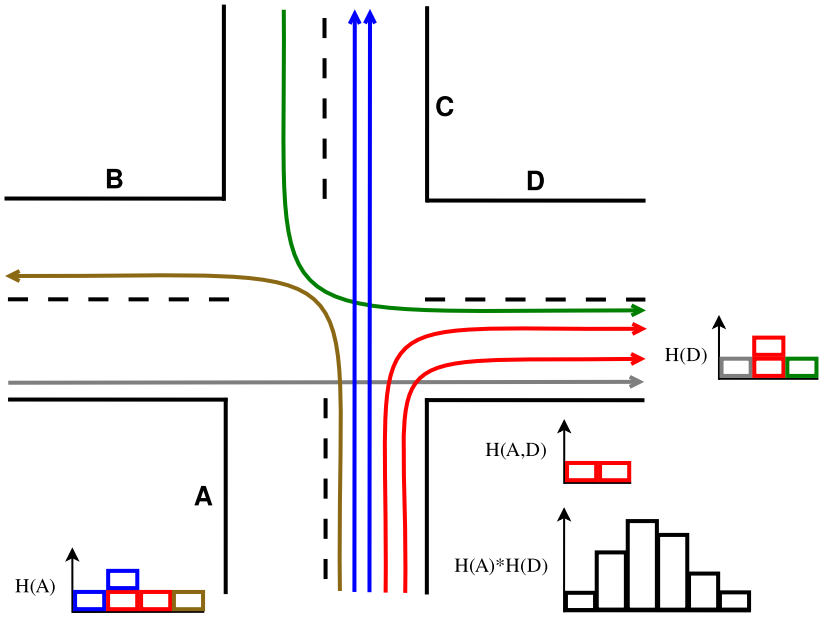


Fig. E.1: Intersection Analysis

A trajectory analysis example is shown in Figure E.1 that illustrates a trajectory set that covers different traversals of a signalized intersection. The figure also illustrates how trajectory data offers benefits over data that only covers single road segments. To estimate the traversal time for the path taking the right turn ($P = \langle A, D \rangle$), one could use all traversal time histograms of segments A ($H(A)$) and D ($H(D)$) and convolve them ($H(A) * H(D)$), or one could consider only trajectories that fully traverse the turn ($H(A, D)$) [1].

Since the former also contains trajectories of cars going straight, which is usually faster than making a right turn, and cars making a left turn, which is usually slower, it is likely to provide an inaccurate estimate with an artificially high variance.

On the other hand, using detailed trajectory data causes privacy concerns since publishing such data would allow to identify drivers' home addresses and places of work by mining the data set for repeating patterns. Therefore, trajectory data sets often cannot be released to the public in full. However, we can allow users to query the data and then answer the queries to an extent that satisfies privacy restrictions either by using anonymization methods or reporting only particular aggregates.

The trajectory analytics services we demonstrate here target multiple use

cases. Municipalities invest substantial resources into analyzing the travel times of road intersections to obtain reliable information for city and traffic planning. Many such analyses can be performed faster and easier by using existing data sets that are, however, currently not available to the municipalities. Another use case is tendering for public transportation contracts where tenders are advertised in all EU countries. Here companies already present on a market have an advantage when bidding, since they can more reliably estimate travel times and costs and adjust their bids accordingly. If trajectory analytics services for a region are made available to all parties, the public sector could expect more competitive offers from companies new to the market.

Such services do not need to be provided via an SQL interface, but can be provided by a RESTful service implemented on top of a path-based API, which allows easy access to the trajectory data without requiring any knowledge about the underlying database schema or indexing methods. Such services are also easier to integrate into a variety of applications. Our goal is to demonstrate how information contained in trajectory data sets can be made easily available to other parties without violating the privacy of the individuals from which the data is collected, i.e., making a trip or a driver identifiable from the published data.

We provide a fast and flexible implementation that allows more meaningful analytics on trajectory data than do other, more prevalent traffic data sources, e.g., loop detectors.

2 The Trip API

The TripAPI provides easy access to a data warehouse while being independent of its schema and indexing methods. It provides access to basic information about routes and segments, e.g., speed and travel-time information

2.1 Data and Parameters

The ITS Platform data set used for the demonstration contains over 1.1 billion GPS points sampled at 1 Hertz from 458 vehicles in and around Aalborg, Denmark during the period from May 2012 to December 2014 [2].

In a preprocessing step, the GPS points are map-matched [3] to the OpenStreetMap model of the road network of Denmark [4]. This results in over 79 million segment traversals that form some 1.4 million trajectories. The map-matched trajectories are stored in the fact table of our PostGIS data warehouse by segment traversals and are organized in a star schema as shown in Figure E.2.

To use any service, the user specifies two GPS coordinates that are then

Function	Arguments	Behavior	Index
NearestSegment	lon, lat	Maps coordinates to the nearest road segment ID (sid)	R-Tree (PostGIS)
SegmentInfo	sid	Retrieves segment information	B+-Tree (PostgreSQL)
TripInfoSegment	sid, dr, tr	Retrieves all trips on the most used path between two segments filtered by date and time range	B+-Tree (PostgreSQL)
TripInfo	sid1, sid2, dr, tr	Retrieves all trip data on the most used path between two segments filtered by date and time range	NETTRA
MostUsedRoute-Trips	sid1, sid2, dr, tr	Provides the number of trips on the most used path between two segments and its path identifier (pid) filtered by date (dr) and time range (tr)	NETTRA
Route	sid1, sid2, pid	Retrieves segment geometries of a path	NETTRA
RouteLength	sid1, sid2, pid	Returns length of a path	NETTRA
GetFreeFlowTT	sid1, sid2, pid	Computes free flow travel-time for a path	NETTRA
NoUnique-VehicleID	sid	Counts number of unique vehicles on a segment	B+-Tree (PostgreSQL)
GetWeatherType	weatherkey	Returns the weather information for a weather measurement obtained with TripInfo	B+-Tree (PostgreSQL)

Table E.1: TripAPI Functions

map-matched to the closest OpenStreetMap segment. The NETTRA index is used to find the most frequently used route between the two segments [5] which has been shown to scale to big trajectory data sets. Users can also apply additional filter predicates, e.g., weather, date range, week day, and

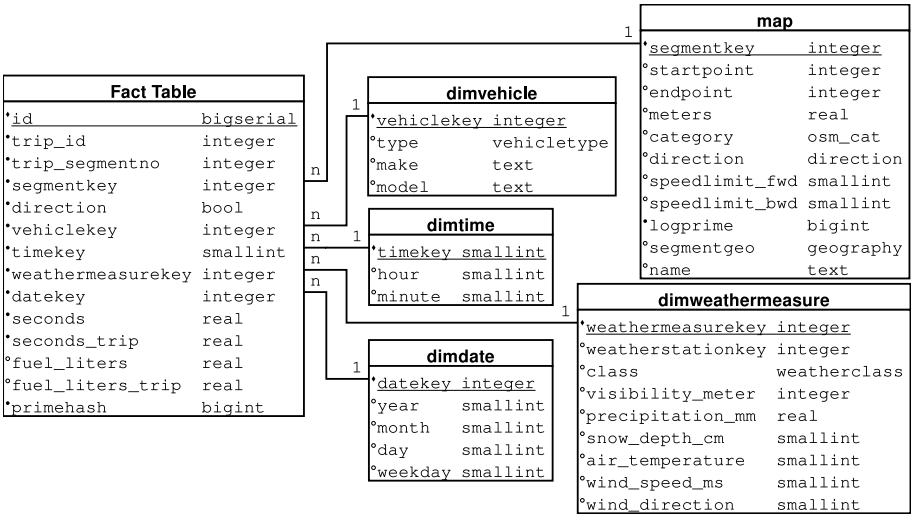


Fig. E.2: Data Warehouse Schema

month, to evaluate their impact on travel times.

2.2 API Functions

The TripAPI provides the functions described in Table E.1, which describes each function's arguments and behavior and states which index is used in the implementation. These functions are implemented using PostGIS spatial queries and the NETTRA index, but could be implemented on top of any database system with spatial capabilities.

The API is used to efficiently retrieve trajectory data and information about road segments. This data is then used to provide a number of anonymized analysis services on an application server. Passing a path identifier (pid) returned by the trip functions, e.g., TripInfo, is necessary since the most frequent path between two segments might change based on the predicates, e.g., time or date range.

3 RESTful Services

We adopt a RESTful architecture for the web services [6] that provides external access to resources for analytics applications based on the TripAPI. Since we focus only on data retrieval and encapsulation, a RESTful architecture fits better to our goals than do other styles such as SOAP [7].

We use a parametric style that adds all parameters of the resource as follows:

`http://.../TravelTime?from={GPS}&to={GPS}`.

The parametric style has the advantage that all query parameters appear right after the resource and was chosen for our demonstration as it easily maps to the dimensions of the data warehouse.

After requesting the URI of our service, the server transfers a representation of a resource as a list of records from the fact table or as an aggregated list of records joined with information from the dimension tables in a JSON document.

4 Anonymized Service Results

Figure E.3 shows how user applications can interact with the service and the data warehouse. An application submits an HTTP GET request that is rewritten by the application server as an SQL query against the data warehouse. The result set is then aggregated and anonymized on the server before being returned to the user.

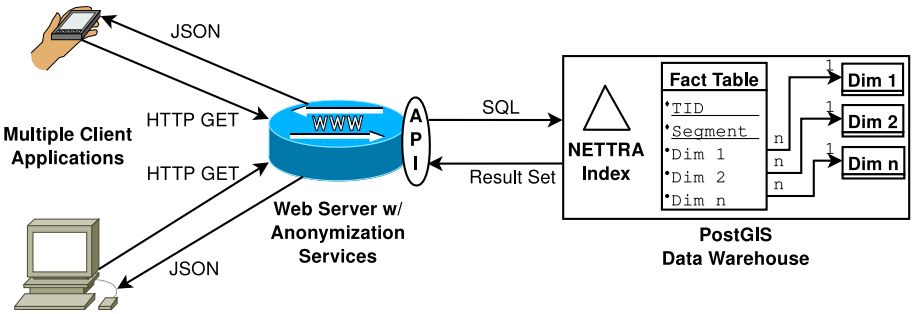


Fig. E.3: System Architecture

4.1 *K*-Anonymity

K-anonymity was introduced by Sweeney and Samarati [8]. To ensure *k*-anonymity of reported data, we need to ensure that the data reported for a user can also belong to at least $k-1$ other users. This can be achieved by not only removing unique identifiers like primary keys but also anonymizing so called "quasi-identifiers" that would otherwise enable re-identification by correlation. We use it for our demonstration because it is a proven yet easy to explain anonymization method with implementations available from third parties.

4.2 TravelTime Service

The TravelTime service provides a k-anonymous average speed histogram of all trajectories for the most traveled path between two segments for every time window within a specified time range. The service can be used to analyze the influence on congestion of the time of day or factors like weather or season.

Figure E.4b shows an average speed histogram for the time of day in 15 minute intervals of a motorway path. If an interval does not contain any trajectories, the average speed of the subsequent interval is reported. Figure E.4c shows the same data, but with $k = 7$, i.e., this time every interval contains the trajectories of at least 7 different users. If too few users are present, it is merged with subsequent intervals until trajectories from at least k users are found. If a path has been traversed by fewer than 7 users, no result is reported.

4.3 Trips Service

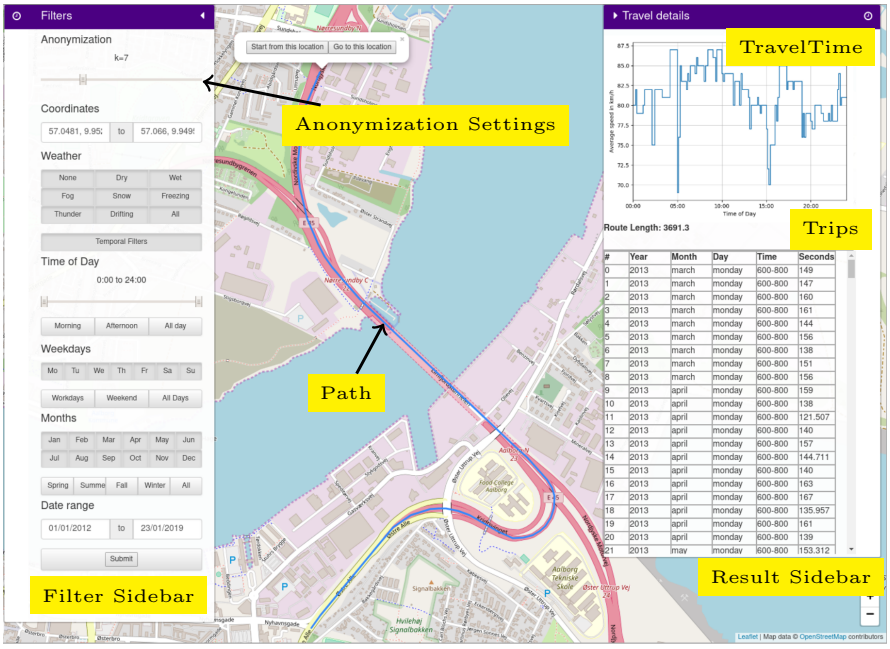
The Trips service shows all travel times of the trajectories matching the desired predicates for a chosen route. It lists the year, month, day of the week, time of day, and the traversal time in seconds. It can be used to estimate the distribution of travel times for a given path. Such data can be anonymized using existing methods by treating every column except the traversal time as a quasi-identifier. A result of the service can be seen in the Result Sidebar in Figure E.4a.

5 Implementation

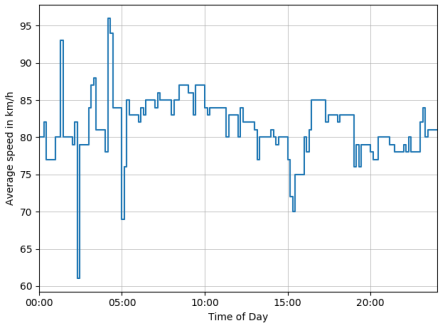
The RESTful service is written in Python and provided via the widely used Flask framework that accesses the database via the TripAPI. The user input is then transformed into a GET request to the server hosting the RESTful service that in turn rewrites the user request into an SQL query. The result set is aggregated/anonymized on the application server before being returned to the user as a JSON document. For anonymizing the trajectory data for the Trips service, a Python implementation [9] of the Mondrian algorithm [10] is used.

6 Demonstration Scenario

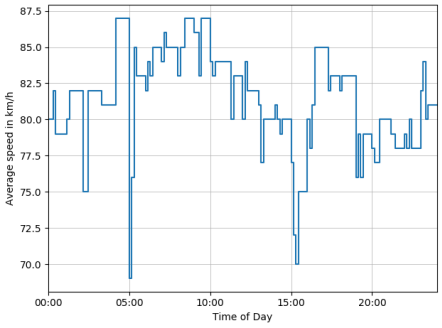
For the demonstration, we utilize a graphical web interface, shown in Figure E.4a, that visualizes the JSON documents returned by the TravelTime and Trips services. After two GPS coordinates are chosen on the map, the



(a) Demo UI



(b) Speed Histogram with $k = 1$



(c) Speed Histogram with $k = 7$

Fig. E.4: Technical Demonstration

most frequently used path between these points is rendered on the map. The TravelTime service also produces an average-speed histogram that is shown next to the map. Below, the table of the anonymized trajectory data from the Trips service is presented. The interface allows users to apply time, date, and weather constraints to the trajectories. In addition, a user can pick a level of anonymization between $k = 1$, i.e., no anonymization, and $k = 25$. The user

can then compare results of the TravelTime service for different k as shown in Figures E.4b and E.4c and observe how the Trips table is affected. For our example path, we can see that during the night and until the early morning, larger intervals are reported for times with only sparse trajectory coverage as well as a smaller range of values. Note that the service experiences only a limited loss of accuracy during rush hours, where many trajectories are available.

Our demonstration allows users to easily and quickly visualize the impact of different data privacy guarantees on the quality of trajectory analyses and how, e.g., the granularity of estimates on different road categories are affected. Based on the results, users can make informed choices on which trade-off between privacy and accuracy to apply when opening their data to analyses, without knowing the details of the underlying database. Furthermore, our demonstration shows that our TripAPI provides a level of abstraction suitable for implementing trajectory analytics on top of large trajectory data sets.

Acknowledgments

This work is supported in part by the optiTruck project (`www.optitruck.eu`) under EU grant agreement No. 713788.

References

[1] R. Waury, C. S. Jensen, and K. Torp, "Adaptive Travel-Time Estimation: A Case for Custom Predicate Selection," in *Proceedings of the 19th IEEE International Conference on Mobile Data Management*. IEEE, 2018, pp. 96–105.

[2] Aalborg University, "ITS Platform," http://www.itsplatform.dk/, 2018.

[3] P. Newson and J. Krumm, "Hidden Markov Map Matching Through Noise and Sparseness," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2009, pp. 336–343.

[4] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org," https://www.openstreetmap.org, 2014.

[5] B. Krogh, N. Pelekis, Y. Theodoridis, and K. Torp, "Path-based Queries on Trajectory Data," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2014, pp. 341–350.

[6] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[7] C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision," in *Proceedings of the 17th ACM International World Wide Web Conference*, 2008, pp. 805–814.

[8] P. Samarati and L. Sweeney, "Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression," SRI International, Tech. Rep., 1998.

[9] Q. Gong, "Basic Mondrian," https://github.com/qiyuangong/Basic_Mondrian, 2018.

[10] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Mondrian Multidimensional K-Anonymity," in *Proceedings of the 22nd IEEE International Conference on Data Engineering*. IEEE, 2006, pp. 25–35.

ISSN (online): 2446-1628
ISBN (online): 978-87-7210-493-5

AALBORG UNIVERSITY PRESS