# History Oblivious Route Recovery on Road Networks

Theodoros Chondrogiannis
University of Konstanz, Germany
theodoros.chondrogiannis@uni-konstanz.de

Johann Bornholdt
University of Konstanz, Germany
johann.bornholdt@uni-konstanz.de

Panagiotis Bouros
Johannes Gutenberg University Mainz, Germany
bouros@uni-mainz.de

Michael Grossniklaus
University of Konstanz, Germany
michael.grossniklaus@uni-konstanz.de

## ABSTRACT

The availability of GPS sensors in vehicles has enabled the collection of trajectory data that can be utilized to improve the quality of location-based services. However, mostly due to privacy concerns, many data sets are published without containing entire trajectories but only the source location, the target location and the duration of recorded trips. In this paper, we study the problem of route recovery from trip data. In contrast to recent works that assume the availability of entire trajectories for past trips, we investigate methods for route recovery in the absence of such historical data, and we present methods for recovering the single most likely route that a vehicle has travelled. Furthermore, we introduce the region recovery problem that aims at determining a small region that is very likely to contain the traveled route. We also introduce region recovery methods for both single trips and trip groups. In a comprehensive experimental evaluation, we study the efficacy of our solutions for both the route and the region recovery problem. For the region recovery problem in particular, we demonstrate the pros and cons of each method along with the trade-off they offer between the size of the recovered region and the likelihood that the region contains the actual route.

## CCS CONCEPTS

• **Mathematics of computing** → **Paths and connectivity problems**; • **Information systems** → *Mobile information processing systems*; • **Applied computing** → *Data recovery*.

## KEYWORDS

route recovery, trajectories, mobility analysis

**ACM Reference Format:**
Theodoros Chondrogiannis, Johann Bornholdt, Panagiotis Bouros, and Michael Grossniklaus. 2022. History Oblivious Route Recovery on Road Networks. In *The 30th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '22), November 1–4, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3557915.3560979



*SIGSPATIAL '22, November 1–4, 2022, Seattle, WA, USA*

ACM ISBN 978-1-4503-9529-8/22/11.
https://doi.org/10.1145/3557915.3560979

## 1 INTRODUCTION

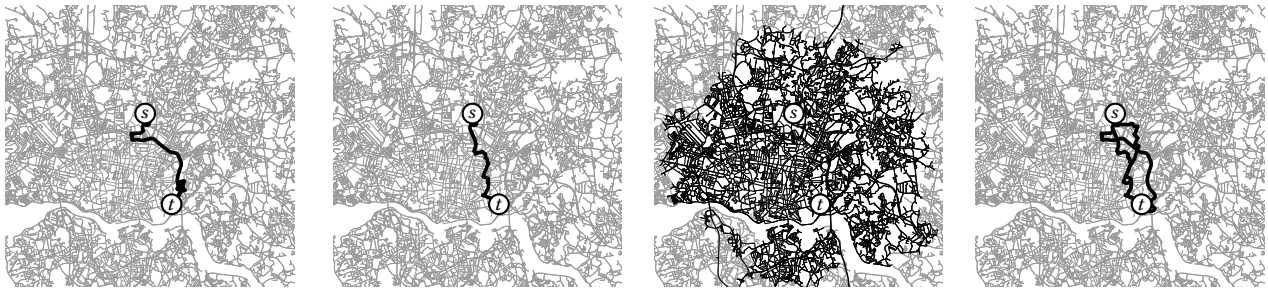
The proliferation of diverse GPS-enabled devices has contributed to the generation of large volumes of trajectory data. Towards providing better and more advanced location-based services, the analysis of trajectory data has received significant attention both from the academic community and industry. Many organizations, especially taxi operators, have published large data sets of trajectories that have been used for the development of solutions in a variety of tasks, e.g., vehicle routing [7], road traffic forecasting [16], and travel time estimation [1].

However, especially due to privacy concerns [32], many vehicle fleet operators do not publish complete, accurate trajectory data. Instead, they publish trip data, i.e., data that report only the starting location, the destination location, the start time and the end time of a trip [31]. Furthermore, even for published trajectory data, there exists the problem of low sampling rate, which results in many consecutively sampled locations being far apart from each other. In such cases, it is unclear which route out of several possible ones the vehicle has followed [19].

Due to the aforementioned scenarios, the problem of *route recovery* has attracted significant attention in the last decade. Given the source location, the target location, and the duration of a trip, the route recovery problem aims at computing the actual route of the trip at hand, as shown in Figure 1a. Existing solutions that tackle the route recovery problem are split into two categories: (1) methods that have been developed in the context of map-matching and aim at recovering routes relying solely on empirical observations [18, 28, 34] and (2) methods that utilize historical information, i.e., past trajectories [17, 33]. On the one hand, map-matching methods have focused on sparse trajectories and require the two locations for which a route must be recovered to be fairly close. As a result, these methods are not applicable for route recovery from trip data. On the other hand, regarding methods in the second category, historical information is not always available.

In this paper, we first revisit the *route recovery problem* and we investigate solutions that operate in the absence of any historical information. More specifically, we utilize recent route planning methods that compute paths more likely to be selected by drivers. For example, one may assume that a driver always chooses the path that yields the lowest travel time under optimal conditions, i.e., no traffic congestion, and therefore use the fastest path as the recovered route, as shown in Figure 1b.

However, due to the limited information available, recovering a single route that exactly matches the route of a given trip is unlikely. As such, we introduce the *region recovery problem* that aims at computing a small sub-network that potentially contains the actual




(a) Original trajectory (b) Fastest path (c) Route containment region (d) PenPE-sky

**Figure 1: Examples of recovered routes and recovered regions.**

route. We then introduce a framework that employs a number of algorithmic solutions, including novel ones, to tackle the region recovery problem. Our solutions aim at balancing accuracy, i.e., the likelihood that the recovered region contains the actual route, with the size of the recovered region. For instance, the recovered region of Figure 1c guarantees that the actual route is contained, but it is too large, while the recovered region of Figure 1d contains part of the actual route but is significantly smaller.

Our contributions in this paper are as follows:

- We study the route recovery problem proposing two solutions; the first based on existing work [25] on near-fastest routing and a novel second that considers the changes between roads of different types (Section 4).
- We introduce the region recovery problem that is to compute a sub-network instead of a single route, and we present a framework for region recovery that first enumerates a set of candidate routes and then selects a meaningful subset of them to form the recovered region (Section 5).
- To improve the accuracy of the region recovery computation, we also devise an approach that receives groups of trips as input. This approach allows information sharing between the route recovery processes for all trips in a group (Section 6).
- Our comprehensive experimental evaluation on real-world data shows that: (1) taking into account changes between roads of different types can benefit route recovery, and (2) our region recovery methods offer different trade-offs between recovery accuracy and recovered region size (Section 7).

We begin in the next section by providing the necessary background and definitions for terms used throughout this paper. Section 3 provides an overview of the related work, while Section 8 gives our concluding remarks along with directions for future work.

## 2 BACKGROUND & PROBLEM DEFINITIONS

A road network $G = (N, E)$ is a *directed weighted* graph with a set of nodes $N$ representing road intersections, and a set of edges $E \subseteq N \times N$ representing road segments. Every edge $(n_i, n_j) \in E$ of a road network $G$ is assigned a *positive* weight that captures the cost of moving from node $n_i$ to node $n_j$. While, this weight can represent any non-negative cost, throughout this paper, we consider the travel time $w_\tau(n_i, n_j)$ as the default edge weight. Given a set of edges $E' \subseteq E$ the sum of the weights of the edges in $E'$ is denoted by $\ell_\tau(E')$. Furthermore, each edge is assigned a label that indicates the type of the road the edge represents, e.g., highway, primary, residential, etc., a speed limit that indicates the maximum speed with which a vehicle is allowed to travel through the edge, and a list of $(x, y)$ coordinates that represent the geometry of the edge. As we consider no traffic information, the travel time of an edge is given by its length divided by its speed limit.

A *route* in a road network is a (simple) *path* $p(s \rightarrow t)$ from a source node $s$ to a target node $t$ that is a *connected* and *cycle-free* sequence of edges $\langle (s, n_i), \ldots, (n_j, t) \rangle$. The travel time $\ell_\tau(p)$ of a path $p$ is the sum of the weights (travel times) of all its contained edges. We refer to the path(s) $p_{fp}(s \rightarrow t)$ that yields the minimum travel time between $s$ and $t$ as the *fastest path(s)*. Furthermore, the term *near-fastest path* has been used in the bibliography [6, 25, 26] to refer to paths that have a bounded travel time of $\alpha \cdot \ell(p_{fp}(s \rightarrow t))$ where $\alpha > 1$.

A (GPS) *trajectory* $\mathcal{T}$ of a moving object is a sequence of sample points $\langle \{\pi_1, \tau_1\} \ldots \{\pi_m, \tau_m\} \rangle$ where $\pi_i$ is a location $(x_i, y_i)$ with longitude-latitude coordinates, and every $\tau$ is a timestamp. Each sample point in the sequence indicates the location $\pi_i$ of the moving object at timestamp $\tau_i$. Given a trajectory $\mathcal{T}$ and a road network $G = (N, E)$, the *map-matching* process aims at finding a route $p_{\mathcal{T}}$ in $G$ that matches with the actual route that the moving object has followed. A *trip* $T$ is a sequence of only two sample points $\langle \{\pi_s, \tau_s\}, \{\pi_t, \tau_t\} \rangle$ indicating only the source location and time and the target location and time of the moving object. The *duration* of a trip $T$ is $dur(T) = \tau_t - \tau_s$.

**Problem Definitions.** In this paper, we first study the route recovery problem, which given a query trip $T$ aims at retrieving a single route that matches the actual route for $T$. The route recovery problem is formally captured by Definition 2.1.

*Definition 2.1 (Route Recovery Problem).* Given a road network $G = (N, E)$ and a past trip $T = \langle \{\pi_s, \tau_s\}, \{\pi_t, \tau_t\} \rangle$ of a moving object $o$, the *route recovery problem* aims at recovering the route $p$ that $o$ has traveled.

While our aim is to find a single route that matches the given trip, with only such limited information available, it is very difficult to perform such a task with high accuracy. As such, we introduce a variant of the route recovery problem namely the *region recovery problem*, formally captured by Definition 2.2.

*Definition 2.2 (Region Recovery Problem).* Given a road network $G = (N, E)$ and a past trip $T = \langle \{\pi_s, \tau_s\}, \{\pi_t, \tau_t\} \rangle$ of a moving object $o$, the *region recovery problem* aims at recovering a region, i.e., a sub-network $G'$ of $G$, that contains the route $p$ that $o$ has traveled.

## 3 RELATED WORK

The route recovery problem has recently attracted significant attention. However, existing works consider a different setup to ours. Furthermore, all these works focus on recovering a single route per trip. To the best of our knowledge, our work is the first one that investigates the recovery of a region instead of a single route. Following from the work of Wu et al. [33], we categorize existing works on the route recovery problem in two groups: (1) methods that are tailored to map-matching and aim at recovering routes without utilizing historical information, and (2) methods that utilize historical information, i.e., entire past trajectories, and use it to infer the most probable taken route using statistical methods or machine-learning.

**Route Recovery for Map-matching.** Most existing methods for map-matching focus on the spatial similarity of a trajectory with the edges of the road network [8]. However, when the input trajectory is sparse, i.e., consecutive coordinates are far apart, most map-matching methods fail to find a matching route. To address this problem, many map-matching algorithms simply compute the fastest path between coordinates that are far apart [21, 30, 35]. However, since one cannot expect drivers to always choose the fastest path [10], other approaches compute paths that minimize costs determined based on empirical observations. Zheng et al. [37] use geometric and topological information of the road network to determine edge weights. Rahmani and Koutsopoulos [24] infer edge weights using a heuristic function that considers delays at traffic lights and left turns.

**Route Recovery using Historical Data.** Methods that utilize historical trajectories usually train a machine-learning model to learn spatial transition probabilities. Jagadeesh and Srikanthan [14] employ a hidden Markov model that learns transition patterns. Zheng et al. [36] model the spatial transition probability between adjacent edges in a road network with one-order Markov model. Banerjee et al. [4] use Gibbs sampling, in which the spatial transition probabilities are modeled using high-order Markov chains. Wu et al. [33] employ inverse reinforcement learning to capture the spatial transition patterns. Due to the fact that we do not assume the availability of historical trajectory data, the aforementioned works are not applicable on our problem setting.

## 4 SINGLE ROUTE RECOVERY

The most straightforward way to assign a route to a given query trip is to compute the fastest path. Assuming that drivers abide by the legal speed limits, the fastest path is the path that yields the lowest travel time between two locations in a road network under optimal conditions, i.e., no traffic. As the fastest path has been used by many map-matching methods for route recovery [21, 30, 35], we include it in our evaluation as a baseline.

**Simplest Near-fastest Path.** Our first approach to select a route other than the fastest path is based on the work of Sacharidis and Bouros [25] on the computation of paths that are easy for drivers to follow. In particular, one of the presented problems deals with the computation of a route that is not much slower than the fastest path, but has low complexity, i.e., involves a small amount of turns. More specifically, given a source $s$ and a target $t$, the *simplest near-fastest route* is the route that has the lowest number of turns among all near-fastest routes from $s$ to $t$. Following from the work of Sacharidis and Bouros, we consider the *near-fastest route with minimum turns* (Min-Turns) as a potential solution for the route recovery problem. During the route recovery process, the travel time of the near-fastest path computed by Min-Turns is bounded by the ground truth duration of the given query trip.

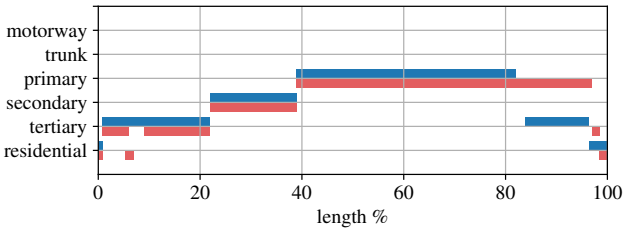



**Figure 2: Road type changes of two sample routes.**

**Minimum Road Hierarchy Peaks.** We now introduce our second approach to choose a path other than the fastest path for the route recovery process. Our approach is based on the observation that road networks are usually characterized by an inherent hierarchical/highway structure [27]. During route planning, roads such as motorways that are multi-lane and with high speed limits are usually given priority over, e.g., residential roads. This is because they allow faster traveling and are less likely to be affected by traffic [23]. However, to ensure optimality, the fastest path may contain switches from roads of higher priority to roads of lower priority. Hence, even though the fastest path yields the lowest travel time under optimal conditions, human drivers may prefer a slightly sub-optimal route. We make the assumption that drivers choose a path that (1) is not much longer than the fastest path, (2) uses roads of high priority as much as possible, and (3) switches to roads with low priority only when it is necessary to reach their destination.

Consider the example in Figure 2, which displays the distribution of the lengths of two routes between the same two locations in the city of Porto over the road types ranked by priority, from highest to lowest. Red indicates the distribution of the length of the fastest path as computed by an algorithm, while blue indicates the distribution of the length of a path with slightly higher duration as chosen by an actual driver. At the beginning of the fastest path, we observe a change from a tertiary to a residential and then back to a tertiary road, while the slower path does not have such a switch. We argue that a driver is unlikely to decide to return to a road with lower priority, as driving in such a road can cause potential delays.

To quantify the aforementioned criteria, we measure the *road hierarchy peaks* in the road type hierarchy of a given route. A peak is defined as a sequence of two switches between road types of lower priority to higher priority and back. Consider again our example in Figure 2. The route represented by the red distribution

has two road hierarchy peaks, i.e., *residential→tertiary→residential* and *secondary→primary→tertiary*, while the route represented by the blue distribution has only one road hierarchy peak, i.e., *secondary→primary→tertiary*. Based on the concept of road hierarchy peaks, we consider the *near-fastest path with minimum road hierarchy peaks* (Min-HP) as a potential solution for the route recovery problem. Similar to Min-Turns, during the route recovery process, the travel time of Min-HP is bounded by the recorded duration of the query trip.

## 5 REGION RECOVERY

Despite the use of empirical criteria, recovering the exact route for a given trip remains a very challenging task, as the choice of a route by drivers depends on many factors including subjective ones. As such, the efficacy of methods that recover a single route for a given trip is limited, as shown by our experiments in Section 7. Even methods that employ machine-learning models and use historical data to train those models, have demonstrated limited efficacy in recovering a single route [17, 33]. To this end, in this section, we abandon the effort to recover a single route and tackle the region recovery problem formally defined in Definition 2.2.

The most straightforward way to define a recovered region for a trip $T$ is to extract the sub-network formed by the union of the nodes and edges from all paths the travel time of which does not exceed the recorded duration of $T$. Following from the concept of single-via paths [2] and assuming that drivers always abide by the speed limit, we can compute a *route containment region*, i.e., a sub-network that certainly contains the actual route.

*Definition 5.1 (Route Containment Region).* Let $G = (N, E)$ be a road network such that each edge $e \in E$ is assigned a weight representing its travel time. Given a trip $T = \langle\{\pi_s, \tau_s\}, \{\pi_t, \tau_t\}\rangle$ where $\pi_s$ and $\pi_t$ are associated with nodes $s, t \in N$ respectively, the *Route Containment Region* of $T$ is the induced sub-network $G' = (N', E')$ of $G$ such that $\forall n \in N'$ for the concatenation of the fastest path from $s$ to $n$ and the fastest path from $n$ to $t$, i.e., $p_{fp}(s \rightarrow n) \circ p_{fp}(n \rightarrow t)$, we have $\ell(p_{fp}(s \rightarrow n) \circ p_{fp}(n \rightarrow t)) \leq dur(T)$.

While the route containment region is bound to contain the actual route, something that we verified by conducting preliminary experiments, in practice it may not be a useful result. Depending on the difference between the travel time of the fastest path between two nodes associated with a trip $T$ and the recorded duration of $T$, the size of the route containment region may be very large. Therefore, a compromise between the size of the recovered region and the chances that the actual route of $T$ is contained is necessary.

To tackle the region recovery problem and achieve a compromise between the size of the recovered region and the chances that the region contains the actual route, we introduce the framework shown in Figure 3. As input we consider a road network $G = (N, E)$ and a query trip $T = \langle\{\pi_s, \tau_s\}, \{\pi_t, \tau_t\}\rangle$. The route recovery process is performed in three steps. In the first step, the edges $(n_{si}, n_{sj})$ and $(n_{ti}, n_{tj})$ that are the closest ones to $\pi_s$ and $\pi_t$, respectively, are computed. Then, either $n_{si}$ or $n_{sj}$ is chosen as the source node and either $n_{ti}$ or $n_{tj}$ is chosen as the target node of the route to be recovered. Note that the accuracy of the route recovery process is not affected as long as we ensure that the two matched edges are part of the recovered region. In the second step, our approach

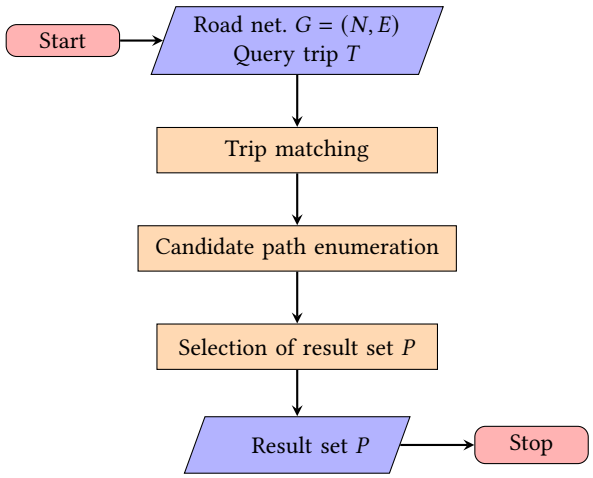



**Figure 3: Route recovery framework.**

applies path enumeration to generate a number of candidate paths. Finally, in the third step, our framework applies filtering methods to select one or more paths out of the candidates that form the result, i.e., the recovered region. In what follows, we elaborate on the second and the third step of our framework.

### 5.1 Candidate Path Generation

Let $G = (N, E)$ be a road network and $s, t \subseteq N$ be two nodes associated with the starting and the ending location of a query trip $T = \langle\{\pi_s, \tau_s\}, \{\pi_t, \tau_t\}\rangle$. A naive approach to compute a set of candidate paths is to enumerate all near-fastest paths from $s$ to $t$ the duration of which does not exceed the trip duration $\tau_t - \tau_s$ [6]. However, the computation of all near-fastest paths from $s$ to $t$ can be prohibitively expensive. Furthermore, even if we accomplish this task, the number of the enumerated paths can be excessively high thus hindering the determination of any meaningful result. Hence, in what follows we present two approaches to construct a small, yet promising set of candidate paths.

**Single-via Paths Enumeration (SvPE).** Our first approach for generating a set of candidate paths is to construct the *single-via paths* [2]. Given a source node $s$ and a target node $t$, the single-via path of a node $n \in N$ is the path formed by concatenating the fastest path from $s$ to $n$ and the fastest path from $n$ to $t$, i.e., $p_{fp}(s \rightarrow n) \circ p_{fp}(n \rightarrow t)$. By definition, the single-via path of a node $n$ is the fastest possible path that connects $s$ and $t$ through $n$. Naturally, there is no need to enumerate all single-via paths, i.e., we consider only single-via paths the travel-time of which does not exceed $dur(T)$. To compute the subset of single-via paths that do not exceed the trip duration, we first execute Dijkstra's algorithm [11] twice to compute two fastest path trees, one from $s$ to every node $n \in N$ and a reverse one from every node $n \in N$ to $t$, and we compute sets $N_s \subseteq N$ and $N_t \subseteq N$ such that $\forall n \in N_s$ : $\ell(p_{fp}(s \rightarrow n)) \leq dur(T)$ and $\forall n \in N_t : p_{fp}(n \rightarrow t) \leq dur(T)$. We then retrieve the single-via path of every node $n \in N_s \cap N_t$ where $\ell(p_{fp}(s \rightarrow n)) + \ell(p_{fp}(n \rightarrow t)) \leq dur(T)$. During the retrieval, all non-simple single-via paths are excluded from the candidate set.

**Penalty-based Path Enumeration (PenPE).** To develop our second approach for enumerating candidate paths, we build upon the Iterative Penalty Method (IPM) originally proposed by Johnson et al. [15] and later improved in the context of alternative routing [3, 9, 13]. The main idea behind IPM is to compute paths by repeatedly running a shortest/fastest path algorithm, e.g., Dijkstra's algorithm. Before each iteration, a penalty is applied on the weights of the edges of each previously computed path. As a result, the fastest path algorithm deviates from the previously computed path and returns a different one (ideally) at each iteration.

The main shortcoming of IPM is that there is no intuition behind the penalty applied in each iteration. Previous works either determine the penalty for each edge weight based on some predefined constant [3, 13] or use randomization. This leads to unpredictable results, i.e., a very large penalty would result in very few and long paths, whereas a very small penalty would require the algorithm to execute more iterations increasing both the number of computed paths and the computational cost. To address this shortcoming and employ edge weight penalization for computing candidate paths, we take into account the recorded duration of the query trip.

Our PenPE technique is based on the following assumption. Given the recorded duration of a trip, assuming that the moving object followed the fastest path in the network at the start time of the trip, the cost of that fastest path has to be equal or greater to the recorded duration. Therefore, for each computed path, the penalty has to be distributed proportionally to the weight of the edges while ensuring that the cost of the path increases in order to match the recorded duration. This process ensures that the cost of each computed path will be approximately equal to the recorded duration after the application of the penalty. Furthermore, in order to ensure that paths computed during previous iterations are not affected by penalties applied much later, the weight of each edge is penalized only once. To this end, given a path $p$ computed during an iteration, the penalty is

$$penalty(n_i, n_j) = (dur(T) - \ell_\tau(p)) \cdot \frac{w_\tau(n_i, n_j)}{\ell_\tau(E_{np})}$$

where $E_{np} \subseteq p$ is the set of edges in $p$ the weights of which have not been penalized during previous iterations of the algorithm. This iterative process is repeated until the cost of the fastest path computed before the penalization exceeds the recorded duration or if all edges on the computed path have already been penalized. Algorithm 1 presents the pseudocode of our penalty-based candidate path enumeration method.

**Algorithm 1:** Penalty-based Path Enumeration

**Input:** Road network $G = (N, E)$, source node $s$, target node $t$, travel time upper bound $\tau_{ub}$, threshold $c$
**Output:** Path set $P$

```
p ← fastest_path (G,s,t)
initialize P ← {p}
while τ_ub − ℓ_τ(p) > c do
    foreach edge (n_i, n_j) ∈ p do
        E_np ← set of non-penalized edges of p
        if (n_i, n_j) has not been penalized then
            τ_pen ← (τ_ub − ℓ_τ(p)) · (w_τ(n_i, n_j)/ℓ_τ(E_np))
            w_τ(n_i, n_j) ← w_τ(n_i, n_j) + τ_pen
    P ← P ∪ {p}
    p ← fastest_path (G,s,t)
restore original edge weights w_τ
return P
```

## 5.2 Candidate Subset Selection

In what follows, we present three methods that enable the selection of a meaningful subset from a given set of candidate routes. As this step follows the process in Figure 3, these methods can be used in combination with either path enumeration method of Section 5.1.

**Local Optimality-based Path Filtering.** Abraham et al. [2] proposed local optimality as an objective criterion to evaluate whether a given path is reasonable, i.e., does not involve unnecessary detours. More specifically, let a path $p$ and a parameter $T < \ell(p)$. Path $p$ is $T$-locally optimal, if every sub-path $p'$ of $p$ such that $\ell(p') > T$ is a fastest path, and for the subpath $p''$ of $p'$ obtained by removing the endpoints of $p'$ we have $\ell(p'') < T$. Since evaluating local optimality is very hard, we employ the $T$-test, a quick 2-approximation scheme, to filter out candidate paths that are not locally optimal. Abraham et al. [2] have shown that if a path $p$ passes the $T$-test, then it is $T$-locally optimal.

**Bicriteria Skyline.** The Min-HP and Min-Turns methods we presented in Section 2 allow us to select out of a set of candidate routes the one that minimizes some criterion other than the travel time. However, any approach that chooses between routes that minimize one out of multiple criteria, disregards any routes that offer a compromise. Hence, our second approach for filtering candidate paths involves returning a set of routes such that no route is dominated by another, i.e., a route dominates another if it is strictly better in all considered criteria. Such a result is achieved by the *skyline* operator [5, 22]. In short, having computed a set of candidate routes, we define the recovered region by the subset of routes that are on the skyline. Based on our experiments in Section 7, we compute the skyline using two criteria, i.e., the travel time and the road hierarchy peaks.

# 6 GROUP-BASED REGION RECOVERY

In this section, we present a novel approach that enables region recovery over groups of trips. More specifically, we introduce *group-based path enumeration* (GrPE), which takes as input a group of trips and returns for each trip a set of candidate paths. The main difference of GrPE to the methods presented in Section 5.1 is that GrPE does not treat the trips in the input group as independent events. GrPE considers trips as part of a group, e.g., grouped by timestamp, where all trips are affected by the conditions of traffic the same way. If GrPE makes an assumption about the travel time of an edge of the road network, e.g., the actual travel time of the edge is double the original one, then this assumption is taken into account for the region recovery of all trips in the input group.

GrPE is based on the main idea behind PenPE for generating candidates. Let a road network $G = (N, E)$ and a group of query trips $\mathcal{S} = \{T_1, \ldots, T_m\}$. The region recovery process starts by computing

the fastest path between the pair of nodes $s_i, t_i \in N$ associated with each trip $T_i \in \mathcal{S}$. However, the approach of PenPE for applying a penalty on the weights of the edges of the computed paths is applicable only when the computed paths share no edges. Recall that during the region recovery for a query trip $T$, PenPE distributes a penalty on the weights of edges of a computed path $p$ to make the penalized travel time of $p$ equal to the recorded duration of $T$. This approach cannot be directly applied by GrPE. If two or more computed paths share some edges, then the weights of those edges have to be penalized by taking into account the recorded duration of all related trips. More specifically, a penalty must be added on the edge weights such that the penalized costs of all paths match the recorded duration of each associated trip.

Towards addressing this issue, we employ a method for applying a multiplicative penalty on the weights of the edges on computed paths, in contrast to the additive penalty of PenPE. More specifically, we model the problem of determining a penalty value as a system of linear equations as follows. Given a set of computed paths $P$ we first compute the set $E_u \subset E$ which contains every edge $e_i \in E_u$ that appears at least once on a path $p_j \in P$. Then, for each path $p_j$ associated with a trip $T_j$, we construct an equation of the form:

$$x_1 \cdot c(e_1) + \ldots + x_{|E_u|} \cdot c(e_{|E_u|}) = dur(T_j).$$

where $x_i$ is the multiplicative penalty that is applied to each edge and $c(e_i)$ is either equal to the weight of edge $e_i$ if the edge lies on the path $p_i$ or 0 otherwise, i.e.,

$$c(e_i) = \begin{cases} w_\tau(e_i), & \text{if } e_i \in p_i \\ 0, & \text{otherwise.} \end{cases}$$

Having constructed such an equation for each path, we now solve the resulting system of linear equations to compute the penalties $x_i$. Obviously, such a system of linear equations is very unlikely to have a unique solution and is more likely to be either underdetermined or overdetermined. So, we employ the *bounded-variable linear least squares* [29] algorithm to obtain an approximate solution with small error. The algorithm also allows us to set both a lower bound $c(e_i) \geq 1$ on the values of the variables to ensure that the penalty can only increase an edge weight and an upper bound to ensure that the penalized weights do not become unrealistically large.

Algorithm 2 presents the pseudocode of our GrPE approach. The algorithm takes as input the road network $G$, a set of query trips $S$ and an additional parameter $\epsilon$ which is used by the termination condition. In Lines 1–7, the algorithm initializes the result map $R$ that stores a set of result paths for each trip in $T_i \in S$, and then initializes each set with the fastest path between the $source-target$ nodes associated with every $T_i$. At the same time, the average error, i.e., the average difference between the recorded duration of a trip and the travel time of the fastest path, is computed. In Lines 8–21, the iterative process to apply penalties and compute more routes is executed. The determination and the application of the penalty on the weights $w_\tau$ of the edges of the road network takes place in Lines 10–15. First, the set of all edges $E_{pen}$ that lie on at least one path computed during the previous round are identified in Lines 10–12. In Line 13, we execute the linear least squares algorithm to determine the penalties for all edges in $E_{pen}$. The weights of the edges are updated in Lines 14–15. Then, in Lines 16–21 we proceed with the computation of the fastest path for each trip considering the updated/penalized edge weights and we update the average error accordingly. This process is repeated until the average error computed during the last round has not changed more than the user-defined constant $\epsilon$. We use this termination condition because the linear least squares algorithm does not guarantee that the average error will ever be zero. Therefore, the iterative penalty and path computation stops when the average error does not change significantly (Line 8). The result map $R$ is returned in Line 23.

**Algorithm 2:** Group-based Path Enumeration

**Input:** Road net. $G = (N, E)$, Set of query trips $S = \{T_1, \ldots, T_{|S|}\}$
**Output:** Result map $R = \langle R[T_1], \ldots, R[T_{|S|}] \rangle$

1 $err_{prev} \leftarrow 0$, $err_{sum} \leftarrow 0$
2 initialize result map $R$
3 **foreach** $T_i \in S$ **do**
4 $\quad p =$ `fastest_path`$(G, T_i.source, T_i.target)$
5 $\quad R[T_i] \leftarrow R[T_i] \cup p$
6 $\quad err_{sum} \leftarrow err_{sum} + dur(T_i) - \ell_\tau(p)$
7 $err_{avg} \leftarrow \frac{err_{sum}}{|S|}$
8 **while** $|err_{avg} - err_{prev}| > \epsilon$ **do**
9 $\quad err_{prev} \leftarrow err_{avg}$
10 $\quad E_{pen} \leftarrow \emptyset$
11 $\quad$ **foreach** $T_i \in S$ **do**
12 $\quad\quad E_{pen} \leftarrow E_{pen} \cup \{\text{last path added to } R[T_i]\}$
13 $\quad Pen =$ `penalty_computation_llsq`$(E_{pen})$
14 $\quad$ **foreach** $e_i \in E_{pen}$ **do**
15 $\quad\quad w_\tau(e_i) \leftarrow w_\tau(e_i) \cdot Pen[e_i]$
16 $\quad err_{sum} \leftarrow 0$
17 $\quad$ **foreach** $T_i \in S$ **do**
18 $\quad\quad p =$ `fastest_path`$(G, T_i.source, T_i.target)$
19 $\quad\quad R[T_i] \leftarrow R[T_i] \cup p$
20 $\quad\quad err_{sum} \leftarrow err_{sum} + dur(T_i) - \ell_\tau(p)$
21 $\quad err_{avg} \leftarrow err_{sum}/|S|$
22 restore original edge weights $w_\tau$
23 return $R$

## 7 EXPERIMENTAL EVALUATION

In order to evaluate the efficacy of all presented methods for the route and the region recovery problems, we conduct a comprehensive experimental evaluation. All our methods were implemented in Python 3.9 using the NetworkX package with the exceptions of Min-HP and Min-Turns, which were partly implemented in C++. This section reports the results of our evaluation.

**Data Sets.** The data sets we use in our evaluation are (1) a subset of the trajectories obtained from taxis in the region of Porto [20], and (2) the road network of the district of Porto obtained from OpenStreetMap (OSM)[1]. The subset of trajectories contains 101,705 trajectories sampled over the period of one month during all hours of each day, excluding all trajectories that consist of only two points.

[1] https://www.openstreetmap.org/

**Table 1: Results summary for route recovery methods.**

| | Shortest | Fastest | Min-HP | Min-Turns |
|---|---|---|---|---|
| precision | 0.559 | **0.613** | 0.611 | 0.575 |
| recall | 0.485 | 0.525 | **0.531** | 0.469 |
| $f_{0.5}$ score | 0.542 | **0.593** | **0.593** | 0.550 |
| $f_1$ score | 0.519 | 0.566 | **0.568** | 0.517 |
| $f_2$ score | 0.498 | 0.541 | **0.545** | 0.487 |
| recall@n | 0.429 | 0.482 | **0.490** | 0.443 |
| accuracy | 0.429 | 0.482 | **0.487** | 0.439 |

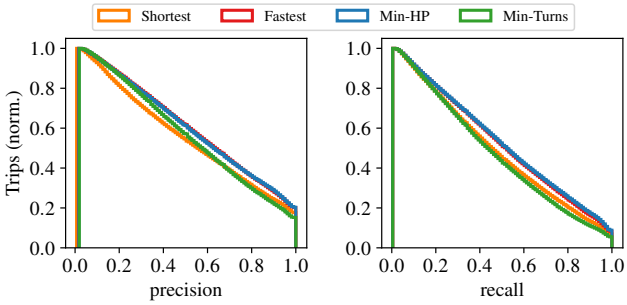



**Figure 4: Reverse-order cumulative histograms of all measures for single route recovery methods.**

On average, each remaining trajectory consists of 49 points. The road network of Porto contains 78,080 nodes and 183,404 edges. A speed limit and a list of labels indicating the type of the road each edge represents are also provided by OSM. We first create a set of trips from the trajectories by extracting the source location, the target location and the duration of each trajectory. Next, we match the source and the target location of each trajectory to nodes of the road network. We then obtain a ground truth for each trip, i.e., the actual route the taxi has traveled, by map-matching the associated trajectory to the road network. For the map-matching process, we use the Fast Map Matching algorithm proposed by Yang and Gidófalvi [34].

**Evaluation Measures.** We adopt a variety of measures to evaluate the efficacy of the presented methods. First, following the paradigm of binary classification, we consider *precision*, *recall*, and f-score [12]. Given an actual route $p_{gt}$ of a trip and the set $E_r$ of recovered edges by either a route or a region recovery method, precision is defined as the ratio of the number $|p_{gt} \cap E_r|$ of correctly recovered edges over the total number $|E_r|$ of recovered edges, and recall is defined as the ratio of the number $|p_{gt} \cap E_r|$ of correctly recovered edges over the number of edges $|p_{gt}|$ in the ground truth. The $f_{beta}$ scores are computed based on precision and recall. In addition, following from previous works on the route recovery problem [17], we report *recall@n* and *accuracy* which are both computed using the length, e.g., in meters, of the edges of the road network. Recall@n is defined as the ratio of the cumulative length of the correctly predicted road segments $p_{gt} \cap E_r$ over the cumulative length of the ground truth $p_{gt}$. Accuracy is defined as the cumulative length of $p_{gt} \cap E_r$ over the maximum value of the length of the ground truth $p_{gt}$ and the cumulative length of the recovered edges $E_r$. Furthermore, for the region recovery methods, we report the size of the recovered region, i.e., the percentage of nodes of the road network that consist the recovered region.

## 7.1 Route Recovery

Table 1 summarizes the results of our experiment to evaluate the efficacy of methods for the route recovery problem. In addition to the methods we presented in Section 4, we also include the shortest path as a route recovery method. First, by comparing the results for the shortest and the fastest path, we confirm that the travel time is more useful as an edge weight for the route recovery process than the distance. Then, we observe that Min-HP is the best approach for four out of five measures, with the exception of precision in which the fastest path comes first. At the same time, the scores for Min-Turns are significantly lower than both Min-HP and the fastest path. In short, the minimum travel time and the minimum road highway peaks seem to be the most effective traits for the route recovery process.

To gain additional insight on the efficacy of the route recovery methods, we conduct further analysis. Due to the limited space we include our results only for precision and recall. Figure 4 shows the cumulative reverse-order histograms of the precision and recall of all route recovery methods. In other words, we report the percentage of trips that have at least a specific precision or recall score. Similar to the results presented in Table 1, the fastest path and Min-HP are the most effective methods for route recovery, outperforming the shortest path and Min-Turns. However, we also observe that both the precision and the recall are high only for a small percentage of trips. In fact, only for around 30% of the trips the precision is over 0.8, while for the recall that percentage is just 10%. These results demonstrate how hard it is to recover a single route that matches the actual route for a trip or even a large part of it accurately.

In Figure 5, we study the effect of the time of the day that a given trip took place to the precision and recall. The results are in line with the previous results reported in Table 1 and in Figure 4, i.e., Min-HP demonstrates the best results coming only slightly ahead of the fastest path, while the shortest path and Min-Turns are clearly below. Another observation is that neither precision nor recall are affected by the different time slots. While one would expect time slots that include rush hour to affect the efficacy of the route recovery process, that does not seem to be the case.

Lastly, in Figure 6 we split trips into 10 buckets based on the error defined as the difference between the recorded duration of the trip and the lowest possible travel time, i.e., $error = \tau_t - \tau_s - \ell_\tau(p_{fp})$. For all methods, we observe that they perform better when the error is low. This decrease in precision and recall is expected. A low error indicates that drivers chose the fastest path or a similar one, while a high error indicates that the choices of drivers were influenced by external factors for which we do not have any information. With regard to the performance of the route recovery methods, the fastest path demonstrates the highest precision and recall for low errors. However, while the error increases, the fastest path is outperformed by Min-HP. The shortest path demonstrates once again the lowest scores in all cases, while Min-Turns comes third in most cases.

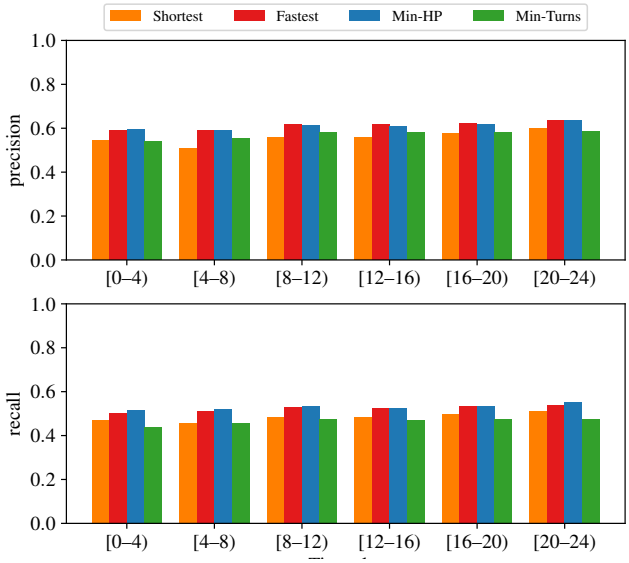



**Figure 5: Precision and recall of single route recovery methods varying time of the day.**

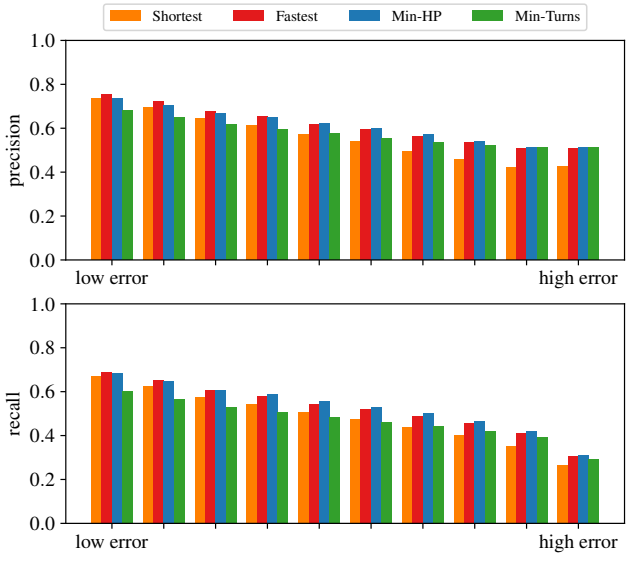



**Figure 6: Precision and recall of single route recovery methods varying real and fastest path duration difference.**

## 7.2 Region Recovery

In this section, we present the results on the efficacy of methods for the region recovery problem. Each method is defined as a combination of a path enumeration method, i.e., SvPE, PenPE, or GrPE, and a candidate subset selection method, i.e., local optimality (LOpt) or bicriteria skyline (Sky) if used. For instance, PenPE-LOpt uses the PenPE path enumeration method in combination with the local optimality-based candidate path filtering. Furthermore, following from the results for the route recovery problem which showed the efficacy of Min-HP, we define the bi-criteria skyline using the travel time and the number of road hierarchy peaks.

Table 2 shows the average scores of all evaluation measures. First, we observe that when path enumeration methods are used without any candidate subset selection, they achieve the highest recall and recall@n. At the same time though, precision and accuracy are too low, thus resulting in low $f_{beta}$ scores as well. Second, the use of candidate subset selection methods leads to a better trade-off between recovered region size and recovery quality. This is demonstrated by the size of the recovered regions and the $f_{beta}$ scores of all the combinations that involve the bi-criteria skyline or the local optimality, with the methods employing bi-criteria skyline candidate subset selection being slightly better. Overall, we observe that solutions based on different candidate path enumeration methods, offer slightly different trade-offs between precision and recall, with SvPE-based methods offering higher recall.

To gain additional insight on the efficacy of the route recovery methods, we further investigate the efficacy of both the candidate path enumeration and candidate subset selection methods. Due to the limited space, we present results only for precision and recall.

*7.2.1 Candidate Path Enumeration.* Figure 7 shows the precision and recall for region recovery using the SvPE and PenPE path enumeration methods without any candidate path selection. In other words, the recovered region is determined using all the candidate paths. On the one hand, SvPE achieves the perfect recall, i.e., the region recovered using SvPE always contains the ground truth. In fact, the result of SvPE is the route containment region (see Definition 5.1). By examining Figure 7 along with Table 2, we observe that the precision of SvPE is very low as the size of the recovered region is high and therefore there are too many false positives. On the other hand, the recall of PenPE is high, but does not reach the recall of SvPE. However, PenPE offers a much better precision. This is due to the fact that the recovered region by PenPE is much smaller than the recovered region of SvPE, resulting in a much smaller amount of false positives.

This difference in precision and recall between SvPE and PenPE as well as the size of each recovered region is also connected to the number of paths each approach enumerates. SvPE enumerates approximately 4,734 candidate paths per trip, while PenPE enumerates approximately only 15 candidate paths per trip.

*7.2.2 Candidate Subset Selection.* In Figure 8, we present our measurements for the candidate subset selection methods presented in Section 5.2. As these methods work in combination with candidate path enumeration methods, we report the precision and recall for all subset selection methods over the results of SvPE and PenPE separately. In general, for both candidate subset selection methods, we observe that by filtering out candidate paths, the recall of each region recovery method drops, i.e., a smaller part of the actual route is contained in the recovered region on average, while the precision increases, i.e., the number of false positives is lower as the recovered region is smaller. More specifically, local optimality-based filtering (LOpt) improves the precision of both SvPE and PenPE only by little. At the same time though, the drop in recall is small. For instance, while PenPE demonstrates a recall of at least 0.8 for more than 80% of the trips, for PenPE-LOpt the percentage of trips for

**Table 2: Results summary for region recovery methods.**

| | SvPE | SvPE-LOpt | SvPE-Sky | PenPE | PenPE-LOpt | PenPE-Sky | GrPE-30 | GrPE-30-LOpt | GrPE-30-Sky |
|---|---|---|---|---|---|---|---|---|---|
| precision | 0.015 | 0.053 | **0.553** | 0.181 | 0.239 | **0.567** | 0.351 | 0.466 | **0.605** |
| recall | **1.0** | 0.953 | 0.606 | **0.901** | 0.816 | 0.596 | **0.693** | 0.631 | 0.555 |
| $f_{0.5}$-score | 0.018 | 0.065 | **0.563** | 0.215 | 0.279 | **0.572** | 0.390 | 0.492 | **0.594** |
| $f_1$-score | 0.029 | 0.100 | **0.579** | 0.301 | 0.370 | **0.581** | 0.466 | 0.536 | **0.579** |
| $f_2$-score | 0.069 | 0.215 | **0.595** | 0.502 | 0.551 | **0.590** | 0.580 | **0.589** | 0.564 |
| recall@n | **1.0** | 0.908 | 0.565 | **0.857** | 0.769 | 0.555 | **0.641** | 0.583 | 0.510 |
| accuracy | 0.012 | 0.041 | **0.466** | 0.174 | 0.230 | **0.469** | 0.331 | 0.413 | **0.483** |
| region size (%) | 9.03 | 1.63 | **0.08** | 0.49 | 0.26 | **0.08** | 0.15 | 0.07 | **0.06** |

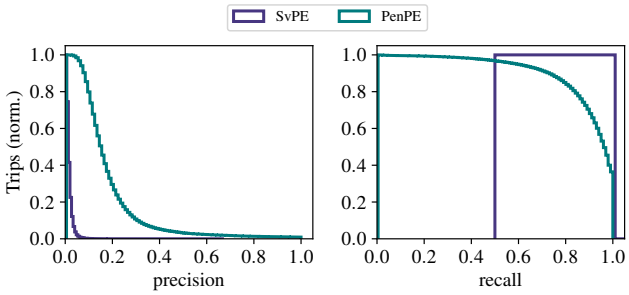



**Figure 7: Precision and recall for path enumeration methods.**

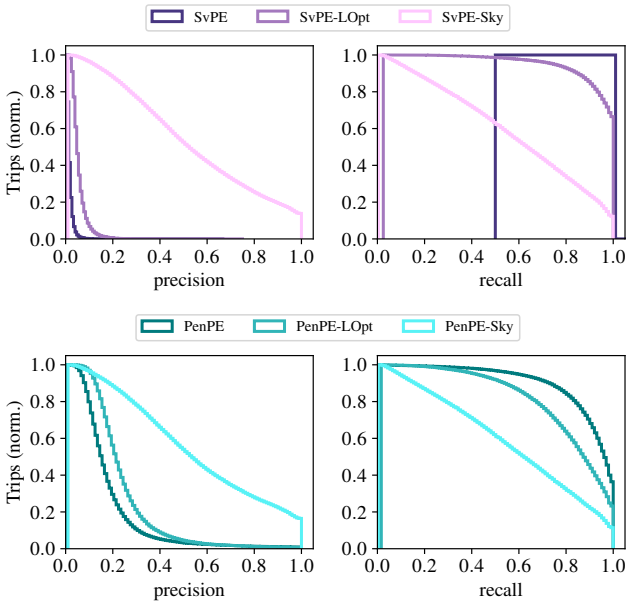



**Figure 8: Precision and recall for candidate path selection methods over SvPE (upper) and PenPE (lower).**

which the recall is at least 0.8 drops to less than 65%. Regarding the bicriteria skyline, we observe that both the improvement in precision and the drop in recall is much higher than when using LOpt. The bicriteria skyline filters out much more candidate paths than LOpt resulting in a much smaller recovered region. In fact, the precision and recall of SvPE-Sky and PenPE-Sky are much closer to the precision and recall of single route recovery methods than other region recovery methods. This leads us to the conclusion that the recovered regions by SvPE-Sky and PenPE-Sky are determined by a very small number of paths.

## 7.3 Group-based Path Enumeration

In this section, we elaborate on the efficacy of the group-based path enumeration. First, we split the trips in our query set into groups based on 15, 30, and 60 minute intervals, denoted by GrPE-15, GrPE-30, and GrPE-60, respectively. In general, using a small interval leads to groups that have fewer trips, but have lower chances that the computed paths overlap. Using a large interval leads to larger groups of trips, but the chances that the conditions in the road networks have changed over time increase. Nevertheless, determining the best clustering method for the input trips is out of the scope of this paper, and is a direction we plan to explore in the future.

Regarding recall, in Figure 9, we observe that GrPE-15 comes first, followed by GrPE-30, with GrPE-60 coming last. However, the result is exactly the opposite for precision, where GrPE-60 comes first and GrPE-15 comes last. This result indicates that using small groups leads to recovering regions that contain a larger part of the actual route than when using large groups, but it also leads to the computation of more routes thus larger recovered regions.

Since, GrPE-30 demonstrates the most balanced behaviour of all three groupings, we investigate its efficacy on region recovery in combination with the candidate subset selection methods presented in Section 5.2. In Figure 9, we observe that LOpt and bicriteria skyline have a similar effect on GrPE as on SvPE and PenPE. More specifically, LOpt increases precision while inflicting only a small drop in recall, while the bicriteria skyline offers even higher precision at the cost of a more significant drop in recall.

Last but not least, in Table 2 we observe that GrPE-30-Sky, demonstrates the highest precision, the highest accuracy and the highest $f_{0.5}$-score among all region recovery methods.

## 8 CONCLUSIONS & FUTURE WORK

In this paper, we studied the route recovery and the region recovery problems from trips in the absence of historical trajectory data. For the route recovery problem, we explored two existing methods and we introduced a novel one that takes that takes road type hierarchy into account. Regarding region recovery, we introduced

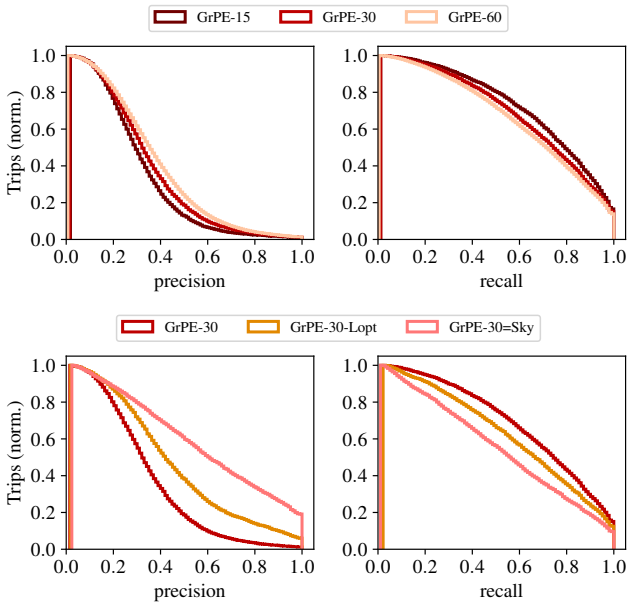



**Figure 9: Precision and recall for GrPE for different groupings of trips (upper) and candidate path selection methods (lower).**

a framework which first enumerates a number of candidate paths, then filters out non-promising candidates, and finally defines the recovered region by the union of the remaining paths. Last but not least, we investigate region recovery from trip groups. Through a comprehensive experimental evaluation, we demonstrated the pros and cons of all presented methods for both the route and the region recovery problem. In the future, we plan to further investigate the group-based route and region recovery by employing additional information that can be shared between processes. We also plan to extend our framework with methods based on machine-learning that learn spatio-temporal features of trips.

## ACKNOWLEDGMENTS

This work is partially supported by Grant No. CH 2464/1-1 of the Deutsche Forschungsgemeinschaft (DFG) and by the Center for the Advanced Study of Collective Behavior at the University of Konstanz, DFG Centre of Excellence 2117 (ID: 422037984). Panagiotis Bouros is a Carl-Zeiss Stiftungsprofessor for "Big Data: In-Memory Databases and Data Analytics". Parts of this work are based on the MSc thesis of David Englert at the University of Konstanz.

## REFERENCES

[1] Sofiane Abbar, Rade Stanojevic, and Mohamed Mokbel. 2020. STAD: Spatio-temporal adjustment of traffic-oblivious travel-time estimation. In *Proc. IEEE MDM*. 79–88.
[2] Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. 2013. Alternative routes in road networks. *Journal of Exp. Algorithmics* 18 (2013), 1–1.
[3] Vedat Akgün, Erhan Erkut, and Rajan Batta. 2000. On finding dissimilar paths. *European Journal of Operational Research* 121, 2 (2000), 232–246.
[4] Prithu Banerjee, Sayan Ranu, and Sriram Raghavan. 2014. Inferring uncertain trajectories from partial observations. In *Proc. IEEE ICDM*. 30–39.
[5] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proc. IEEE ICDE*. 421–430.
[6] W. Matthew Carlyle and R. Kevin Wood. 2005. Near-shortest and K-shortest simple paths. *Networks* 46, 2 (2005), 98–109.
[7] Vaida Ceikute and Christian S Jensen. 2015. Vehicle routing with user-generated trajectory data. In *Proc. IEEE MDM*, Vol. 1. 14–23.
[8] Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. 2020. A survey on map-matching algorithms. In *Proc. Australasian Database Conference*. 121–133.
[9] Dan Cheng, Olga Gkountouna, Andreas Züfle, Dieter Pfoser, and Carola Wenk. 2019. Shortest-Path Diversification through Network Penalization: A Washington DC Area Case Study. In *Proc. IWCTS@SIGSPATIAL Workshop*. 10:1–10:10.
[10] Paolo Cintia and Mirco Nanni. 2016. An effective time-aware map matching process for low sampling GPS data. *arXiv preprint arXiv:1603.07376* (2016).
[11] Edsger W. Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs. *Numer. Math.* 1, 1 (1959), 269–271.
[12] David A Grossman and Ophir Frieder. 2012. *Information Retrieval: Algorithms and Heuristics*. Vol. 15. Springer.
[13] Christian Häcker, Panagiotis Bouros, Theodoros Chondrogiannis, and Ernst Althaus. 2021. Most Diverse Near-Shortest Paths. In *Proc. ACM SIGSPATIAL*. 229–239.
[14] George Rosario Jagadeesh and Thambipillai Srikanthan. 2014. Robust real-time route inference from sparse vehicle position data. In *Proc. IEEE International Conference on Intelligent Transportation Systems*. 296–301.
[15] P E Johnson, D S Joy, D B Clarke, and J M Jacobi. 1993. *HIGHWAY 3.1: An enhanced HIGHWAY routing model: Program description, methodology, and revised user's manual*. Technical Report. U.S. Dept. of Energy, OSTI.
[16] Ibai Lana, Javier Del Ser, Manuel Velez, and Eleni I. Vlahogianni. 2018. Road traffic forecasting: Recent advances and new challenges. *IEEE Intelligent Transportation Systems Magazine* 10, 2 (2018), 93–109.
[17] Xiucheng Li, Gao Cong, and Yun Cheng. 2020. Spatial transition learning on road networks with deep probabilistic models. In *Proc. IEEE ICDE*. 349–360.
[18] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. 2009. Map-matching for low-sampling-rate GPS trajectories. In *Proc. ACM SIGSPATIAL*. 352–361.
[19] Lu Lu, Nan Cao, Siyuan Liu, Lionel Ni, Xiaoru Yuan, and Huamin Qu. 2014. Visual analysis of uncertainty in trajectories. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 509–520.
[20] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. 2013. Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems* 14, 3 (2013), 1393–1402.
[21] Paul Newson and John Krumm. 2009. Hidden Markov map matching through noise and sparseness. In *Proc. ACM SIGSPATIAL*. 336–343.
[22] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2003. An optimal and progressive algorithm for skyline queries. In *Proc. ACM SIGMOD*. 467–478.
[23] Ciaran S. Phibbs and Harold S. Luft. 1995. Correlation of travel time on roads versus straight line distance. *Medical Care Research and Review* 52, 4 (1995), 532–542.
[24] Mahmood Rahmani and Haris N. Koutsopoulos. 2012. Path inference of low-frequency GPS probes for urban networks. In *Proc. IEEE International Conference on Intelligent Transportation Systems*. 1698–1701.
[25] Dimitris Sacharidis and Panagiotis Bouros. 2013. Routing directions: Keeping it fast and simple. In *Proc. ACM SIGSPATIAL*. 164–173.
[26] Dimitris Sacharidis, Panagiotis Bouros, and Theodoros Chondrogiannis. 2017. Finding the most preferred path. In *Proc. ACM SIGSPATIAL*. 1–10.
[27] Peter Sanders and Dominik Schultes. 2005. Highway Hierarchies Hasten Exact Shortest Path Queries. In *Proc. European Symp. on Algorithms*. 568–579.
[28] Mudhakar Srivatsa, Raghu Ganti, Jingjing Wang, and Vinay Kolar. 2013. Map matching: Facts and myths. In *Proc. ACM SIGSPATIAL*. 484–487.
[29] Philip B. Stark and Robert L. Parker. 1995. Bounded-variable least-squares: an algorithm and applications. *Computational Statistics* 10 (1995), 129–129.
[30] Youze Tang, Andy Diwen Zhu, and Xiaokui Xiao. 2012. An efficient algorithm for mapping vehicle trajectories onto road networks. In *Proc. ACM SIGSPATIAL*. 601–604.
[31] New York (N. Y. ). Taxi and Limousine Commission. 2019. New York City Taxi Trip Data, 2009-2018. https://doi.org/10.3886/ICPSR37254.v1
[32] Manolis Terrovitis and Nikos Mamoulis. 2008. Privacy preservation in the publication of trajectories. In *Proc. IEEE MDM*. IEEE, 65–72.
[33] Hao Wu, Jiangyun Mao, Weiwei Sun, Baihua Zheng, Hanyuan Zhang, Ziyang Chen, and Wei Wang. 2016. Probabilistic robust route recovery with spatio-temporal dynamics. In *Proc. ACM SIGKDD*. 1915–1924.
[34] Can Yang and Győző Gidófalvi. 2018. Fast map matching, an algorithm integrating hidden Markov model with precomputation. *International Journal of Geographical Information Science* 32, 3 (2018), 547–570.
[35] Jing Yuan, Yu Zheng, Chengyang Zhang, Xing Xie, and Guang-Zhong Sun. 2010. An interactive-voting based map matching algorithm. In *Proc. IEEE MDM*. 43–52.
[36] Kai Zheng, Yu Zheng, Xing Xie, and Xiaofang Zhou. 2012. Reducing uncertainty of low-sampling-rate trajectories. In *Proc. IEEE ICDE*. IEEE, 1144–1155.
[37] Yuheng Zheng and Mohammed A Quddus. 2011. *Weight-based shortest-path aided map-matching algorithm for low-frequency positioning data*. Technical Report.