A Framework for Predicting Trajectories Using Global and Local Information

William Groves, Ernesto Nunes, and Maria Gini
Department of Computer Science and Engineering
University of Minnesota
{groves, enunes, gini}@cs.umn.edu

ABSTRACT

We propose a novel framework for predicting the paths of vehicles that move on a road network. The framework leverages global and local patterns in spatio-temporal data. From a large corpus of GPS trajectories, we predict the subsequent path of an in-progress vehicle trajectory using only spatio-temporal features from the data. Our framework consists of three components: (1) a component that abstracts GPS location data into a graph at the neighborhood or street level, (2) a component that generates policies obtained from the graph data, and (3) a component that predicts the subsequent path of an in-progress trajectory. Hierarchical clustering is used to construct the city graph, where the clusters facilitate a compact representation of the trajectory data to make processing large data sets tractable and efficient. We propose four alternative policy generation algorithms: a frequency-based algorithm (FreqCount), a correlation-based algorithm (EigenStrat), a spectral clustering-based algorithm (LapStrat), and a Markov Chain-based algorithm (MCStrat). The algorithms explore either global patterns (FreqCount and EigenStrat) or local patterns (MCStrat) in the data, with the exception of LapStrat which explores both. We present an analysis of the performance of the alternative prediction algorithms using a large real-world taxi data set.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; I.2.11 [**Distributed Artificial Intelligence**]: Intelligent Agents

Keywords

urban mobility, route prediction, spatio-temporal analysis, GPS, large-scale data, smart cities, big data

1. INTRODUCTION

The ubiquitous use of mobile devices that store driving-related spatio-temporal information provides researchers and practitioners with data that can be used to understand driving patterns in cities, predict congestion points, and design location-based services for drivers. In order to discover characteristic patterns in large spatio-temporal data sets, we exploit methods that encode spatial relations, such as topology and direction of travel, as well as temporal relations. To the urban planner, the methods we propose can be used to discover common paths and uncover alternative routes that could help alleviate traffic congestion. Such knowledge can also be used to sequence and prioritize the maintenance of roads. The future of smart cities will drive the need for greater use of such technologies.

This paper proposes a framework for trajectory prediction that involves three components: (1) abstraction from global positioning system (GPS) location data into a city graph, (2) graph-based policy generation which constructs policies from a large data set of observations, and (3) a prediction process which forecasts future vehicle movements based on in-progress paths and on learned policies. The proposed novel combination of techniques yields predictions that are an improvement over purely statistical techniques.

Challenges: One challenge is that GPS waypoint data can be noisy, may have low sampling rates, and can be unevenly distributed across city regions. Noise is caused by factors such as high rise buildings, cold starting of data collection, or poor GPS signal reception. Noise may cause the collected trajectories to include locations that are not in the real trajectories. Hence, if not cleaned, noisy waypoints used in learning can lead to incorrect predictions. To address this, we first preprocess the GPS data to clean the data and extract trajectories of individual vehicles (see Section 6.4).

The uneven density of trajectories across a city makes it harder to generate data abstractions that accurately represent the geometry of the road network, even when the amount of data collected is large. Data abstraction is essential to make efficient predictions, but if the abstraction underrepresents roads in sparsely covered areas, the road clusters may be too coarse in those areas and reduce the quality of the predictions. To alleviate this problem, we use hierarchical clustering on the trajectory data, clustering together GPS waypoints that are spatially close.

Approach: After initial preprocessing and location cluster generation, trajectories are mapped into sequences of location clusters, connecting with directed edges pairs of clusters that have sufficient support, and producing a city graph. This representation is used to learn policies (i.e., a data structure providing probabilistic information about the most likely movements between clusters).

For policy learning, we propose several competing algorithms of varying complexity. Each of the algorithms leverages different aspects of the data set to produce policies. Predictions about potential future paths from the last known location are made using policies relevant to an in-progress trajectory. The resulting predicted paths are then scored by comparing with the ground truth path.

Our work combines data mining techniques that discover global structure in the data by identifying characteristic patterns and data

CF'14, May 20 - 22 2014, Cagliari, Italy

http://dx.doi.org/10.1145/2597917.2597934 .

clusters, and local probabilistic methods that predict short-term routes, based on past driving trajectories. Global methods learn about large-scale relationships (i.e., relationships between locations that are far apart in the road network). Local methods instead learn about highly localized relationships between locations (e.g. given the traversal of a specific road, the probability distributions of nearby and adjacent roads is estimated).

In summary, our main contributions are as follows:

1. We propose a systematic framework for trajectory prediction from a large data set of observations. The framework provides abstractions to make learning tractable and efficient given the large number (500,000) of taxi trips in a large city. The proposed framework could be deployed as a core component of an intelligent driver agent (run on-board a vehicle) or used as a centralized aggregation method of real-time GPS data streams to provide predictions for drivers of the expected traffic in near-future paths they might take.
2. We make predictions using only the spatio-temporal data stream from vehicles without using any external features. By not using additional features, our approach can be easily applied to other spatio-temporal domains.
3. A key to the success of our data representation is the encoding of direction with location in the clustering process. The resulting clusters produce shapes that match well the geometry of the physical routes data. This data representation is very compact and facilitates efficient learning.
4. We propose, analyze, and compare four prediction algorithms that leverage the global as well as local patterns. FreqCount and EigenStrat use only global patterns. One algorithm, LapStrat, uses a mix of global and local structures. MCStrat uses highly localized patterns. We show experimentally that LapStrat offers higher prediction performance compared to the other methods.

Section 2 discusses literature relevant to trajectory prediction. Section 3 covers the data used and the preprocessing steps, the clustering technique for building a city graph from the trajectories, the construction of policies from the city graph, and the baseline algorithm FreqCount for policy prediction. Section 4 describes the experimental setup. Section 5 describes our prediction algorithms. Results and lessons learned are in Section 6. Concluding remarks and future work are covered in Section 7.

2. RELATED WORK

Due to data sparseness and the low-sampling rate of trajectory data, aggregated models of the physical world are needed to facilitate generalization between trajectories. Even when a high quality digital map is available, using a logical description of the world is useful for prediction. Li et al. [7] utilize a simple abstraction by partitioning the map of the city using a fixed size rectilinear grid and use this representation in their passenger-finding strategies. Yuan et al. [12] use a landmark graph to build relationships between the road segments the taxis traverse frequently. Two nodes in the graph are connected if trajectories between them are used frequently enough. Their online algorithm finds the fastest route to a destination at a given departure time.

Observing that grids do not capture the differences in data density in regions of a traffic network, Liu et al. [8] create regions of a map using Connected Components Labeling, an image-based edge detection algorithm that takes a road map (similar to Figure 1) as input. This representation is conceptually similar to the clustering methods we propose because it takes into account the geometry of the road network. It differs from our approach in that we directly use the trajectory data to build the graph while they use an external road network map.

Eigendecomposition has been used extensively to analyze and summarize the characteristic structure of data sets. For instance, Lakhina et al. [6] use principal component analysis (PCA) to summarize network flows that pass through an internet service provider. Zhang et al. [13] identify two weaknesses that make PCA less effective on real-world data (i.e. sensitivity to outliers in the data, and concerns about its interpretation), and present Laplacian eigenanalysis as an alternative. The difference is that the Laplacian matrix considers similarity measurements only between close neighbors, while PCA considers relationships between all pairs of points. The intuition is that small differences in similarity are informative only for locations that are close to each other. These studies focus on the clustering power of the eigen-based methods to find structures in the data. Our work goes beyond summarizing the structure of the taxi routes, and uses the eigenanalysis clusters to predict the subsequent path of an in-progress taxi trajectory.

Research in trip prediction based on observations of driver behavior has enjoyed some recent popularity. Krumm et al. [5] predict the next turn a driver will take by choosing with higher likelihood a turn that links more destinations or is more time efficient. Ziebart et al. [15] propose algorithms to predict turn, route, and destination. The study combines a Markov model and inverse reinforcement learning to provide accurate predictions for each of their prediction tasks. Veloso et al. [10] propose a Naive Bayes model to predict a taxi's destination, using time of the day, day of the week, weather, and land use as features.

All these studies use features beyond location data to improve prediction accuracy. Our work instead uses GPS data alone to analyze travel patterns and trajectory sequences to predict routes.

3. DATA ABSTRACTIONS

This section describes the abstraction process starting from the raw GPS data stream and ending with the construction of policies.

3.1 Trajectory preprocessing

The GPS trajectories we use for our experiments are taken from the publicly available Beijing Taxi data set which includes 1 to 5-minute resolution location data for over ten-thousand taxis for one week in 2009 [12]. There are approximately half a million taxi rides in the data set. Beijing, China is reported to have seventy-thousand registered taxis, so this data set represents a significant cross-section of all taxi traffic for a one-week period [14].

Because the data set contains only location (latitude and longitude of a waypoint) and time information for each taxi, it is useful to process the data into segments based on individual taxi fares.

The data has sufficient detail to enable inference on when a taxi ride is completed: for example, a taxi waiting for a fare will be stopped at a taxi stand for many minutes [14]. Using these inferences, we separate the data into taxi rides. We are interested in predicting short term movements of in-progress taxi trajectories, so the exact start and end points of a trajectory are not critical.

Let $V = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_q\}$ be the set of *q trajectories*. We divide the set into V_{TR}, V_{TE}, V_{VA} which are the training, test, and validation sets, respectively. A trajectory $\boldsymbol{v_i}$ is a sequence of n time-ordered GPS coordinates (the length n may differ for each trajectory v_i): $\boldsymbol{v_i} = [c_1^{\boldsymbol{v_i}}, \ldots c_j^{\boldsymbol{v_i}}, \ldots, c_n^{\boldsymbol{v_i}}]$. Each GPS coordinate is the latitude and longitude of a waypoint. Since we are interested in the direction of the motion, we extend the coordinates by computing for each waypoint the current direction as the normalized x and y displacement from the previous waypoint. Hence, coordinate c_j in trajectory $\boldsymbol{v_i}$ is $c_j^{\boldsymbol{v_i}} = (x_j, y_j, dx_j, dy_j)$.

If there was no previous displacement (the taxi stayed in place), then the most recent direction value is used (forward smearing). If there is no information in the sequence, the first direction observed in the sequence is used (backward smearing). We scale dx and dy by using a *direction contribution coefficient*. The useful range of this coefficient is between 0 (direction contributes nothing to the clustering process) and the average inter-cluster distance (any two co-located waypoints with opposite direction values are in different clusters). For this data set, the maximal range of the direction contribution coefficient used is 5 km. The coefficient allows us to control the influence of directionality in the clustering process.

3.2 From GPS data to a city graph map

To facilitate analysis, we encode the taxi trajectories as a sequence of edges in a city graph. The city graph consists of nodes that correspond each to a physical area and edges that correspond to vehicle movements between areas. Earlier work [5, 10, 4] used a rectangular grid for nodes and edges to only immediately adjacent nodes (in a North, East, South, and West pattern). This work departs from that approach using instead a clustering-based city graph. Other papers also use a network graph (e.g., [12, 3]), but this work departs from others by generating the location clusters directly from the GPS waypoints without using a road network map. By using the waypoints directly, the prediction framework presented here (1) does not rely on the accuracy of the road map and (2) can be directly applied to other domains (i.e. animal migration patterns) for which no network map exists.

We divide the city into regions by a clustering process over a random sample of GPS waypoints from the training trajectory data set, after we augmented the GPS coordinates with the current direction, as described earlier.

We cluster the data using the Ward hierarchical clustering algorithm [11] to generate z clusters. Ward's algorithm is a bottom up method that joins together clusters that minimize the sum of intra-cluster distances, hence, it tends to join smaller clusters first. We have chosen this clustering algorithm over simple geometric approaches, such as k-means, because the cluster regions from this algorithm conform better to the geometry of the underlying data. The best value for z, the direction contribution coefficient, and other parameters are found during sensitivity testing (see Section 6).

The clusters simultaneously encode direction and location. Direction is important to distinguish between co-located points that have different directions. For example, given a direction contribution coefficient of 5 km, two points that are co-located but 180° opposite in direction will have an Euclidean distance of 10 km.[1]

Even if co-located, points with dissimilar direction components should be placed in different clusters. For example, two vehicle waypoints located on the same highway segment but with (180°) opposite directional components should be placed in different clusters. When clustering is based on location alone, these points would be in the same cluster regardless of the direction of travel. Adding direction better informs the prediction algorithms than using location clusters alone. Direction encoding is a novel contribution, and an experiment in Section 6.1 shows benefits from this approach.

The city is encoded as a graph with z nodes, each representing a cluster, connected by directed edges. We indicate with s_i the mean center of the ith cluster in the Euclidean (x, y, dx, dy)-space. With a slight abuse of notation, s_i is used to indicate the cluster and its centroid. The set of clusters is denoted as S where $S = \{s_1, \ldots, s_i, \ldots, s_z\}$. An edge, $e_{s_i s_j}$ is created between two clusters when it is one of the top w observed transitions exiting from s_i. The value of w was found experimentally in the parameter optimization process. We found that different values perform best for each of the prediction algorithms.

We define $\mathbb{I}(c_j, s_i)$ to be an indicator function that returns 1 if waypoint c_j is closer to the cluster centroid s_i than to any other centroid and 0 otherwise.

Each waypoint in a trajectory (sequence of GPS waypoints) is mapped to a node (cluster) in the city graph. Given two adjacent waypoints c_i and c_{i+1} in a trajectory, the indicator function Φ (Eq. 1) determines if the movement between the waypoint c_i and c_{i+1} corresponds to a movement between two clusters that are connected by an edge in the city graph or not. Rare transitions between location clusters (due to unusual movements or due to noise in the GPS data) are not encoded in the city graph. The value of Φ is 1 if there is an edge between the pair of clusters and 0 otherwise:

$$\Phi(c_j^{\boldsymbol{v}_i}, c_k^{\boldsymbol{v}_i}, e_{s_l s_m}) = \begin{cases} 1, & \text{if } \mathbb{I}(c_j^{\boldsymbol{v}_i}, s_l) * \mathbb{I}(c_k^{\boldsymbol{v}_i}, s_m) = 1 \\ 0 & \text{Otherwise} \end{cases} \quad (1)$$

A conceptual example of a city graph is in Figure 4.

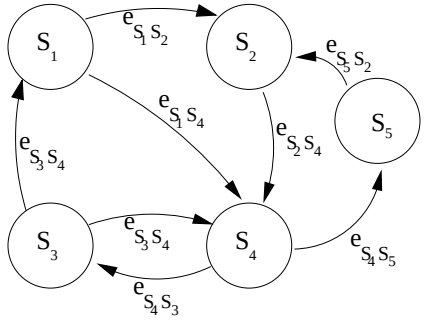


Figure 4: A conceptual example of the city graph abstraction. Location clusters are represented as nodes, edges between nodes appear only if there is sufficient support.

This hierarchical partitioning of the city improves over a purely grid-based encoding. Rectilinear grid partitioning fails to encode the geometry of clusters with irregular shapes. With our clustering process, the learned shapes of the city graph, instead, follow the underlying geometry of the road network (i.e. clusters will usually be centered over a road). In addition, our approach considers local waypoint density, such that more waypoints lead to better differentiation of dissimilar (e.g distant) waypoints and to more clusters.

Figures 1-3 provide examples illustrating the benefits of clustering with direction. Given a grid of all South direction points (180°), the cluster boundaries are drawn in Figure 2. The black dots are GPS taxi waypoints in the South direction ($135^\circ <$ heading $< 225^\circ$) and show taxi density for the cluster. Given the same cluster node set, Figure 3 shows the cluster boundaries for a grid of East direction points (90° heading) and all East direction ($45^\circ <$ heading $< 135^\circ$) waypoints. Adjacent roads with a similar direction are often clustered together. Correspondence between major roads and clusters can be observed by comparing Figures 1 and 2. Less dense regions (e.g. the Western region) will have fewer clusters than more dense regions (e.g. the Eastern region).

3.3 Data structures to support abstraction

To facilitate the learning process, we build a set of *edge data vectors* $\Pi = \{\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_q\}$ from the set of trajectories V that have been mapped to the clusters in the city graph. Each trajectory

[1]For example, given two GPS data points with direction (all values are in km): $a = (9, 8, 5, 0)$ $b = (9, 8, -5, 0)$. The Euclidean distance is $\sqrt{(9-9)^2 + (8-8)^2 + (5-(-5))^2 + (0-0)^2} = 10$ km.

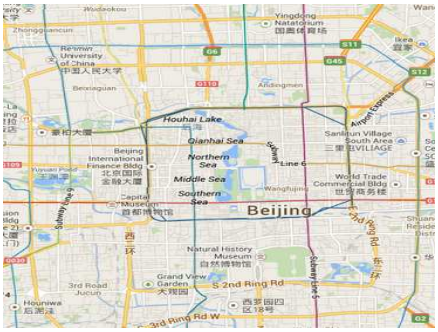


Figure 1: Beijing city road map showing major roads.

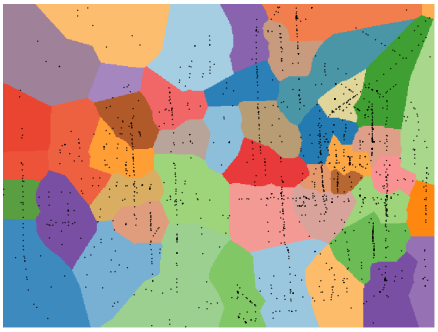


Figure 2: Cluster boundaries for a grid of South direction waypoints; South direction dominant waypoints (dots) from V_{TR}.

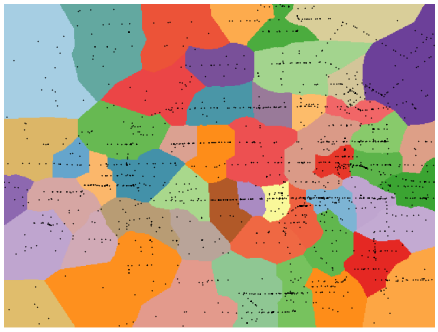


Figure 3: Cluster boundaries for a grid of East direction waypoints; East direction dominant waypoints (dots) from V_{TR}.

has a corresponding edge data vector, so Π has q elements, which is the number of trajectories in V as well.

An edge data vector $\boldsymbol{\pi}^{v_i}$ has as many elements as the number of edges in the city graph. The value of each element of $\boldsymbol{\pi}^{v_i}$ is 1 if there is an edge between the two corresponding clusters in trajectory v_i and 0 otherwise.

$$\boldsymbol{\pi}^{\boldsymbol{v}_i} = \{\delta^{\boldsymbol{v}_i}_{s_1,s_2}, \ldots, \delta^{\boldsymbol{v}_i}_{s_l,s_m}, \ldots\}, \text{where} \tag{2}$$

$$\delta^{\boldsymbol{v}_i}_{s_l,s_m} = \begin{cases} 1, & \text{if } \sum_{j=1}^{n-1} \Phi(c_j^{\boldsymbol{v}_i}, c_{j+1}^{\boldsymbol{v}_i}, e_{s_l,s_m}) \geq 1 \\ 0 & \text{Otherwise} \end{cases} \tag{3}$$

A graphical example showing a trajectory converted into a edge data vector is shown in Figure 5.

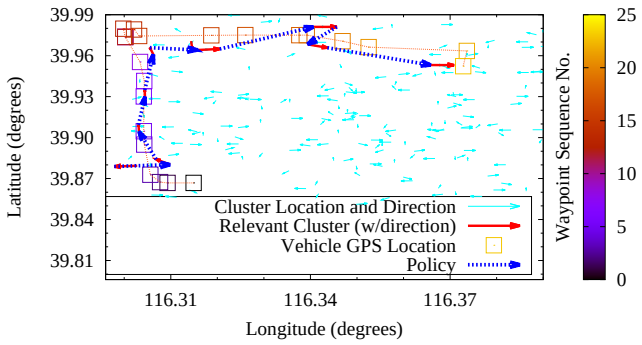


Figure 5: A trajectory converted to an edge data vector in the city graph. The waypoints (represented as squares) are mapped into cluster locations. The set of transitions is used to set values the edge data vector (represented as dotted lines).

Given a complete trajectory $\boldsymbol{v_i}$, for learning and prediction purposes we consider a *partial trajectory*, which, in this work, is the first half of a full trajectory. Hence, $\boldsymbol{v}_i^{partial} = [c_1^{\boldsymbol{v}_i}, c_2^{\boldsymbol{v}_i}, \ldots, c_{n/2}^{\boldsymbol{v}_i}]$. The last location of a partial trajectory $\boldsymbol{v}_i^{last} = [c_{n/2}^{\boldsymbol{v}_i}]$ is used to begin the prediction task. The partial trajectory is used for training, while the second half of the trajectory is used as ground truth to validate our predictions.

We also create a set of *location data vectors*, one for each cluster in the city graph. Each location data vector is created from the waypoints in each partial trajectory $\boldsymbol{v}_i^{partial}$. Each element of a location data vector has value 1 if the corresponding waypoint from $\boldsymbol{v}_i^{partial}$ is closest to the cluster centroid s_i than to any other centroid and 0 otherwise.

$$\boldsymbol{\theta}^{\boldsymbol{v}_i^{partial}} = [\omega_{s_1}, \omega_{s_2}, \ldots, \omega_{s_z}], \text{where} \tag{4}$$

$$\omega_{s_i} = \begin{cases} 1, & \text{if } \sum_{j=1}^{n} \mathbb{I}(c_j^{\boldsymbol{v}_i^{partial}}, s_i) \geq 1 \\ 0 & \text{Otherwise} \end{cases} \tag{5}$$

For convenience, a graphical representation of the data structures used for prediction is shown in Table 1.

Table 1: Data structures and associated dimensions used in this framework. Note: $z << q$ for this domain.

Location Data		Edge Data	
ω	(1×1)	δ	(1×1)
$\boldsymbol{\theta}$	$(1 \times z)$	$\boldsymbol{\pi}$	$(1 \times q)$
$\Theta_{V_{TR}}$	$(\lvert V_{TR}\rvert \times z)$	$\Pi_{V_{TR}}$	$(\lvert V_{TR}\rvert \times q)$

3.4 Policy learning using FreqCount

FreqCount is the simplest of the methods we present for learning a policy and predicting the next location a taxi will visit from the taxi's current location. We use it as a baseline.

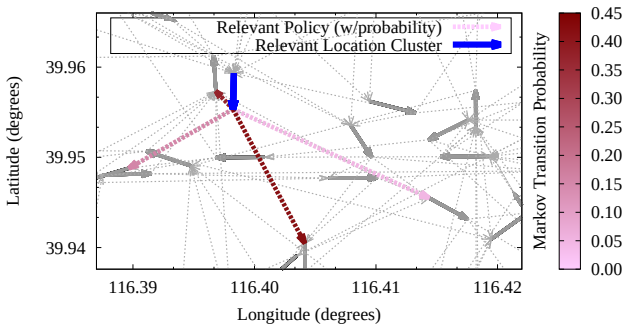
FreqCount first learns a policy, $\boldsymbol{\pi}^{\text{FreqCount}}$, from past trajectory data. Each element of the policy vector for FreqCount is the frequency of the corresponding transitions (movement between clusters in the city graph) in the training data.

With a slight abuse of notation, we use δ_{s_l,s_m}, which we used for binary values in edge data vectors, to indicate a transition probability between nodes in the city graph:

$$\delta^{\boldsymbol{\pi}^{\text{FreqCount}}}_{s_i,s_j} = \frac{\sum_{\boldsymbol{v} \in V_{TR}} \delta^{\boldsymbol{v}}_{s_i,s_j}}{\sum_{\boldsymbol{v} \in V_{TR}} \sum_{k=1}^{z} \delta^{\boldsymbol{v}}_{s_i,s_k}} \tag{6}$$

Figure 6 shows the relative frequencies of transitions exiting a specific graph cluster location using all data from the training set V_{TR}. The relative frequency of each transition exiting from the highlighted cluster is the ratio of the count of that transition divided by the total count of the cluster (Equation 6). The resulting policy vector provides the conditional probability of each edge exiting from that location.

Since FreqCount does not consider locations visited before the current location, it attributes equal probabilities to all outgoing edges



(a)

(b)

Figure 6: (a) Visualization of FreqCount policy for a specific cluster showing the probabilities of several alternative future transitions from a single highlighted location cluster. The transition probabilities exiting from each cluster sum to 1. (b) The physical road map corresponding to the clusters shown in (a).

from a cluster. For this reason, it predicts poorly where many vehicle trips intersect each other (such as at a highway interchange). The more complex methods described later make better predictions by also considering previous locations in the path.

4. EXPERIMENTAL SETUP

Given an in-progress taxi trajectory, we want to make predictions of the short-term future movement of the vehicle. To simulate this task for experimentation, we use a collection of partial trajectories (e.g. Figure 7) generated from complete trajectories from the test set V_{TE}. After FreqCount or one of the algorithms we will present next has generated one or more policy vectors relevant to the partial, policy iteration is performed to predict future locations. The inferred future locations (e.g. Figure 8) are compared against the actual complete taxi trajectory (e.g. Figure 9). Prediction results are scored by comparing the predicted set of paths with the actual vehicle path using Hausdorff distance.

4.1 Performance Measure

The Hausdorff distance is the distance measure we use to compare the ground truth path with a predicted path. Intuitively, it measures the maximum distance between any point on the true path and the nearest point on the predicted path [3]. The properties of this distance measure are highly desirable for short-term route predictions. For example, if the predicted path is longer than the true path (i.e. path C in Figure 10), the distance value is not increased relative to the exact length. Conversely, if the predicted path is too

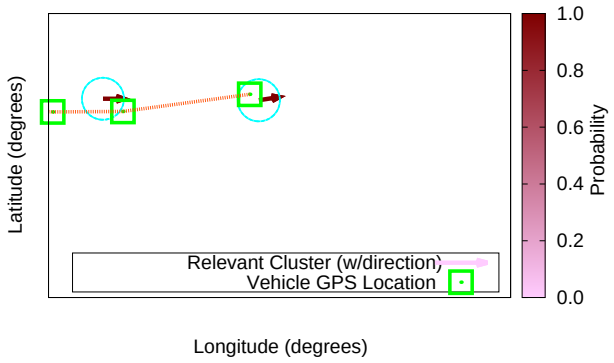


Figure 7: A sample edge data vector $\pi^{v_i^{partial}}$.

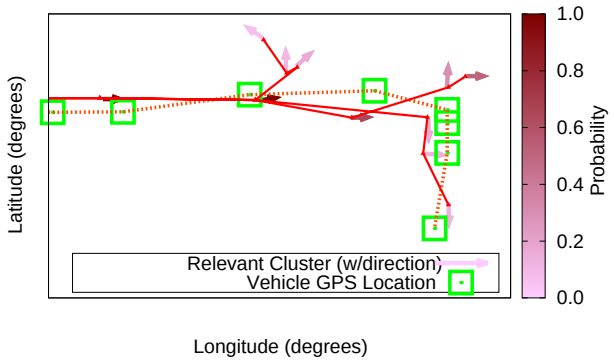


Figure 8: Predictions from FreqCount with horizon of 3.

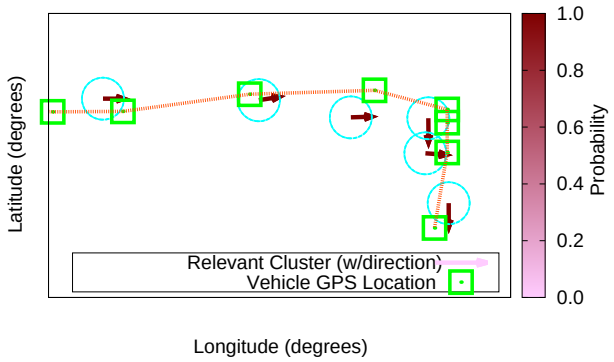


Figure 9: Ground truth trajectory π^{v_i} corresponding to Fig. 7.

short (the model does not predict a path of sufficient length), it will have a larger distance value because a large portion of the (longer) true path will be far from the (shorter) predicted path (i.e. path A in Figure 10). This objective function encourages algorithms to make sufficiently long predictions to match the true path.

While the Hausdorff distance is not an obvious choice for a performance measure, it is strongly preferred to measurements on the graph representation alone, such as accuracy or correlation, because it implicitly captures both prediction error (due to the prediction algorithm) and representation error (due to the cluster abstraction) simultaneously.

As the parameter search process simultaneously varies the learning and representation algorithms, a performance measure that captures the total error of our predictions is valuable for preventing the search process from finding a representation that over simplifies the

representation to achieve high prediction accuracy.

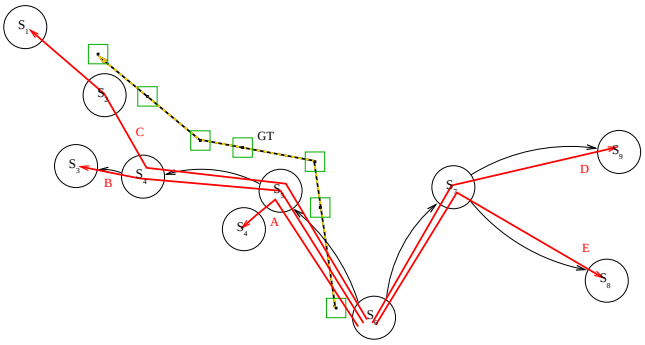


Figure 10: Comparison between ground truth GPS waypoint sequence (GT, dotted line) and a set of 5 predicted paths (solid arrows). Location clusters are denoted as circles; GPS waypoints are denoted as squares.

4.2 Policy Iteration

Policy iteration (Algorithm 1) is applied to $\boldsymbol{\theta}^{last}$, the last location of a partial trajectory, to predict future actions. In the simplest prediction algorithm (FreqCount), there is one policy $\boldsymbol{\pi}$ used to make predictions about subsequent locations, starting from $\boldsymbol{\theta}^{last}$. In policy iteration, the policy vector coefficients are used as probabilities of transitions to subsequent locations. If the vehicle is at location cluster s_i at time step t (the last location in the partial trajectory), then the probability of entering cluster s_j in the next step $t+1$ is computed as: $P^{t+1}(s_j) = P(s_j|s_i) * P^t(s_i) = \delta^{\boldsymbol{\pi}}_{s_i,s_j} * \omega^t_{s_i}$. This algorithm is used by the four prediction methods described in this paper.

With a slight abuse of notation, in the policy iteration algorithm we use ω_{s_i} as the probability that a trajectory visits cluster s_i. Earlier we used ω_{s_i} in location data vectors as a binary value.

Algorithm 1: Policy Iteration

Input: Location vector with last location of taxi ($\boldsymbol{\theta}^{last}$), policy list ($\Pi$), prediction horizon (*numiter*)
Output: Location vector containing visit probabilities for future locations $\hat{\boldsymbol{\theta}}$

$\boldsymbol{\theta}^{accum} \leftarrow [0, \ldots, 0]$ and $|\boldsymbol{\theta}^{accum}| = z$;
for $\boldsymbol{\pi} \in \Pi$ **do**
  $\boldsymbol{\theta}^0 \leftarrow \boldsymbol{\theta}^{last}$;
  for $t = 1$ *to numiter* **do**
    $\boldsymbol{\theta}^t = [\omega^t_{s_1}, \omega^t_{s_2}, \ldots, \omega^t_{s_i}, \ldots, \omega^t_{s_z}]$, where $\omega^t_{s_i} = \sum_{s_j \in S} (\omega^{t-1}_{s_j} * \delta^{\boldsymbol{\pi}}_{s_j,s_i})$;
  for $t = 0$ *to numiter* **do**
    $\boldsymbol{\theta}^{\boldsymbol{\pi}} = [\omega^{\boldsymbol{\pi}}_{s_1}, \omega^{\boldsymbol{\pi}}_{s_2}, \ldots, \omega^{\boldsymbol{\pi}}_{s_i}, \ldots, \omega^{\boldsymbol{\pi}}_{s_z}]$, where $\omega^{\boldsymbol{\pi}}_{s_i} = \max(\omega^{\boldsymbol{\pi}}_{s_i}, \omega^t_{s_i})$;
  $\boldsymbol{\theta}^{accum} = [\omega^{accum}_{s_1}, \omega^{accum}_{s_2}, \ldots, \omega^{accum}_{s_i}, \ldots, \omega^{accum}_{s_z}]$, where $\omega^{accum}_{s_i} = \omega^{accum}_{s_i} + \frac{\omega^{\boldsymbol{\pi}}_{s_i}}{|\Pi|}$;
$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{accum}$

For each relevant policy $\boldsymbol{\pi}$, the algorithm returns $\boldsymbol{\theta}^{\boldsymbol{\pi}}$ which contains the probability values of the vehicle visiting each location cluster in the graph. If multiple policies are relevant (possible in Algorithms 2, 3, and 4), then policy iteration is repeated once for each relevant policy, and the final state probabilities ($\omega^{\boldsymbol{\theta}^{accum}}_{s_i} \forall s_i \in S$) are the mean values for location cluster probabilities for all policies in the relevant policy list.

5. MORE SOPHISTICATED ALGORITHMS FOR BUILDING THE POLICY

We present three additional methods that use local structures, global structures, or a mix of both to build policies and to predict short-term trajectories.

5.1 EigenStrat: Eigen Analysis of Covariance

We expect highly positively and highly negatively correlated transition pairs to be useful for prediction. For example, if a taxi is traveling west at the beginning of an expressway, it is likely to also be traveling west at the end of the expressway (large positive correlation). Conversely, if a taxi is traveling west at the beginning of an expressway, it is highly unlikely to travel east at the same location (large negative correlation). Such relationships are compactly encoded using eigenvectors.

EigenStrat exploits these linear covariance relationships between transitions in the graph. These results, in the form of eigenvectors of the graph transition covariance matrix, (1) can be matched to partial trajectories for purposes of prediction and (2) can be used to study behaviors in the system. Steps 1-4 in Algorithm 2 focus on model generation (computation of eigenvectors). For each pair of edges δ_{s_i,s_j} and δ_{s_k,s_l}, the covariance σ is computed using the trajectories in the training set V_{TR} as:

$$\sigma(\delta_{s_i,s_j}, \delta_{s_k,s_l}) = E(\delta_{s_i,s_j} * \delta_{s_k,s_l}) - E(\delta_{s_i,s_j}) * E(\delta_{s_k,s_l}) \quad (7)$$

where $E(.)$ is the expected value. The $dims$ largest eigenvectors are computed from the covariance matrix. These form a collection of characteristic *eigen-strategies* from the training data.

Algorithm 2: EigenStrat

Input: Π_{TR}, number of principal components (*dims*), minimum angle between policies (α), prediction horizon (*horizon*), edge data vector ($\boldsymbol{\pi}^{\boldsymbol{v}_i{}^{partial}}$)
Output: Location vector of predicted visit probabilities $\hat{\boldsymbol{\theta}}$

1 Generate covariance matrix $C_{|\boldsymbol{\pi}|\times|\boldsymbol{\pi}|}$ (where $\boldsymbol{\pi} \in \Pi_{\text{TR}}$) between transitions on the graph;
2 Get *dims* eigenvectors of C with largest eigenvalues;
3 Compute cosine similarity between $\boldsymbol{\pi}^{\boldsymbol{v}_i{}^{partial}}$ and the principal components ($\boldsymbol{\pi}_j, j = 1 \ldots dims$): $\boldsymbol{\Pi}_{rel} = \{\boldsymbol{\pi}_j || cos(\boldsymbol{\pi}_j, \boldsymbol{\pi}^{\boldsymbol{v}_i{}^{partial}})| > \alpha\}$;
4 If the $cos(\boldsymbol{\pi}_j, \boldsymbol{\pi}^{\boldsymbol{v}_i{}^{partial}}) < 0$, then flip the sign of the coefficients for this eigenpolicy.;
5 Use Algorithm 1 with Π_{rel} on $\boldsymbol{v}_i^{partial}$ for *horizon* iterations to compute $\hat{\boldsymbol{\theta}}$

When predicting from an in-progress trajectory, the algorithm takes the edge data vector generated from the partial trajectory $\boldsymbol{\pi}^{\boldsymbol{v}_i^{partial}}$, a maximum angle to use as the relevancy threshold α, and the eigen-strategies as Π. Eigen-strategies having an angular distance less than α to $\boldsymbol{\pi}^{\boldsymbol{v}_i^{partial}}$ are added to Π_{rel}, which will contain policies that are almost parallel to (similar to) the partial trajectory in terms of angular distance. This collection is then used for policy iteration. Optimal values for α and *dims* are learned experimentally.

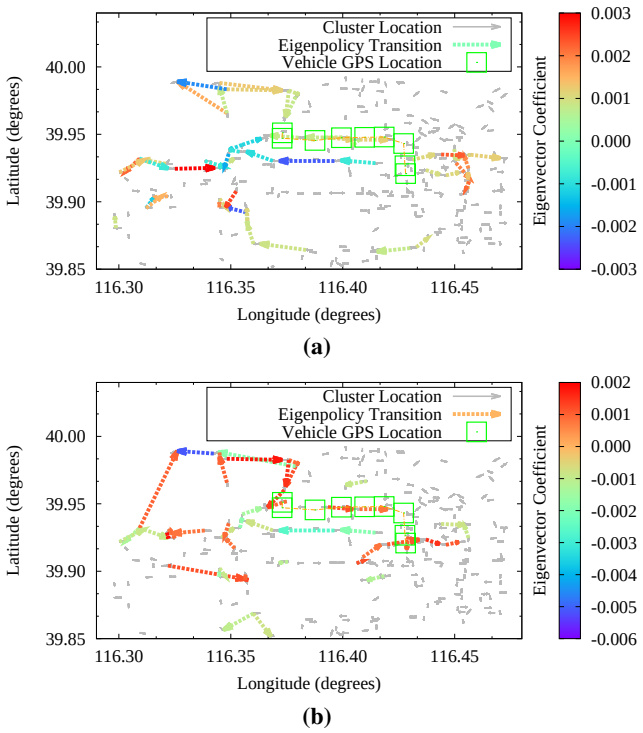


Figure 11: Two eigenpolicies relevant to a partial trajectory showing different strategic patterns found in the training set.

Eigenpolicies facilitate exploration of strategic decisions. Figure 11 show two relevant eigenpolicies that match the partial trajectory shaded by boxes. The eigenpolicies are different both to the east and west of the partial trajectory. Both policies show that taxis moving along the partial trajectory are less likely to turn west at the end of the partial trajectory (as evidenced by the opposite coefficient signs). Instead, the predicted trajectory is comprised of points that have the same coefficient signs as the points in the partial trajectory. Positive eigenvector coefficients show positively correlated transitions (usually traversed together by trajectories in data), transitions with opposite sign pairs are negatively correlated.

In addition, the predicted policies in Figure 11 show two alternative roads a taxi driver can take going east from the partial trajectory. We believe this analysis can be beneficial to urban planners who can take advantage of it to better manage city traffic.

5.2 LapStrat: Spectral Clustering-Inspired Algorithm

Using the edge representation of taxi trajectories, a large matrix of edge data vectors is constructed from the training set. LapStrat (Algorithm 3) seeks to group similar policies together and produce from each group an edge data vector centroid (Step 4), used as a policy. The entire training set is distilled into a small set of policies which are then matched to an incoming edge data vector for prediction. The policy iteration (Algorithm 1) over the relevant matched policies is then used to infer future driver movements.

This prediction method uses spectral clustering to build the policy set (approach of Shi and Malik [9] using an unnormalized graph Laplacian). Spectral clustering operates upon a similarity graph. We use Jaccard index to compute the similarity matrix between edge data vector pairs. We chose the Jaccard index because it finds similarities between vectors that are almost parallel. This is important in cases where two highways only have one meeting point; in this case, if the highways are alternative routes to the same intersection, they should be similar with respect to the intersection point. The inputs to the Jaccard index are two vectors representing either edge data vectors generated in Section 3.3 or policies generated by a policy learning algorithm (in Sections 3.4 or 5). $J(\boldsymbol{\pi}_i, \boldsymbol{\pi}_j)$ is the Jaccard similarity for pair $\boldsymbol{\pi_i}$ and $\boldsymbol{\pi_j}$. The Laplacian is computed by subtracting the *degree matrix* from the similarity matrix. We choose the *dims* eigenvectors with smallest eigenvalues, and perform k-means to find clusters in the reduced dimension. The optimal value for *dims* is learned experimentally.

Algorithm 3: LapStrat

Input: Π_{TR}, dimension (*dims*), number of clusters (k), similarity threshold (Lap Epsilon, ϵ), policy relatedness threshold (Lap Relatedness, ρ), prediction horizon (*horizon*), edge data vector ($\boldsymbol{\pi}^{\boldsymbol{v}_i{}^{partial}}$)

Output: Location vector of predicted visit probabilities $\hat{\boldsymbol{\theta}}$

1 Generate similarity matrix $W_{|\Pi_{\text{TR}}|\times|\Pi_{\text{TR}}|}$ where

$$w_{ij} = \begin{cases} J(\boldsymbol{\pi}_i, \boldsymbol{\pi}_j), & \text{if } J(\boldsymbol{\pi}_i, \boldsymbol{\pi}_j) \geq \epsilon \\ 0 & \text{Otherwise} \end{cases};$$

2 Generate Laplacian L: $L = D - W$ and $\forall d_{ij} \in D$

$$d_{ij} = \begin{cases} \sum_{z=1}^{|\Pi_{TR}|} w_{iz}, & \text{if } i = z \\ 0 & \text{Otherwise} \end{cases}$$

3 Get the *dims* eigenvectors with smallest eigenvalues;

4 Use k-means to find the mean centroids ($\boldsymbol{\pi}_j, j = 1 \ldots k$) of k policy clusters;

5 Find all centroids similar to $\boldsymbol{\pi}^{\boldsymbol{v}_i{}^{partial}}$:
$\Pi_{rel} = \{\boldsymbol{\pi}_j | J(\boldsymbol{\pi}_j, \boldsymbol{\pi}^{\boldsymbol{v}_i{}^{partial}}) > \rho\}$;

6 Use Algorithm 1 with Π_{rel} on $\boldsymbol{v}_i^{partial}$ for *horizon* iterations to compute $\hat{\boldsymbol{\theta}}$

5.3 MCStrat: Markov Chain-Based Algorithm

The recent path history of a taxi can be informative about where it will go beyond its current location. While EigenStrat and LapStrat encode some of this information, MCStrat explicitly tries to match the partial trajectory with full trajectories in the training set that contain the last k locations of the partial trajectory. Matches in the city graph can be thought of as matches on a Markov chain of length k. The Markov chain approach uses local, recent information from $\boldsymbol{v}_i^{partial}$, the partial trajectory to predict from. This method is inspired by [5] but differs in graph construction.

Algorithm 4: MCStrat

Input: V_{TR}, matching length (l), edge data vector ($\boldsymbol{\pi}^{\boldsymbol{v}_i^{partial}}$), prediction horizon (*horizon*), similarity thresh. (ϵ_{sim})

Output: Predicted location vector of visit probabilities $\hat{\boldsymbol{\theta}}$

1 Generate relevant policies V_{rel}
$V_{rel} = \boldsymbol{v}_i | \text{match}(k, \boldsymbol{\pi}^{\boldsymbol{v}_i^{partial}}, \boldsymbol{\pi}^{\boldsymbol{v}_i}) \geq \epsilon_{sim}, \boldsymbol{v}_i \in V_{\text{TR}}$;

2 Generate a composite single relevant policy $\boldsymbol{\pi}_{rel}$
$\boldsymbol{\pi}_{rel} = \delta_{s_1,s_2}^{\boldsymbol{\pi}^{rel}}, \ldots, \delta_{s_i,s_j}^{\boldsymbol{\pi}^{rel}}, \ldots$, where
$\delta_{s_i,s_j}^{\boldsymbol{\pi}^{rel}} = \frac{\sum_{\boldsymbol{v} \in V_{rel}} \delta_{s_i,s_j}^{\boldsymbol{v}}}{\sum_{\boldsymbol{v} \in V_{rel}} \sum_{l=1}^{M} \delta_{s_i,s_k}^{\boldsymbol{v}}}$;

3 Use Algorithm 1 with $\boldsymbol{\pi}_{rel}$ on $\boldsymbol{v}_i^{partial}$ for *horizon* iterations to compute $\hat{\boldsymbol{\theta}}$

Given the last l edges traversed by a taxi, the algorithm finds all complete trajectories in the training set that contain the same l edges and builds a set of relevant policies V_{rel} using the match function. match$(l, \boldsymbol{a}, \boldsymbol{b})$, returns 1 only if at least the last k transitions in the policy generated by trajectory $\boldsymbol{a}$ are also found in $\boldsymbol{b}$. The matching process is not exact, we add a similarity threshold parameter ϵ_{sim}, such that even if two policies do not match on some transitions but their similarity is above the threshold, they are still considered similar. V_{rel} is used to build a composite single relevant policy $\boldsymbol{\pi}_{rel}$, that obeys the Markov assumption, so the resulting policy preserves the probability mass.

Using the composite $\boldsymbol{\pi}_{rel}$, policy iteration is then performed on the last location vector computed from $\boldsymbol{v}_i^{partial}$. This method uses very localized information to make predictions. When sufficient local information exists (i.e. when there are enough observations matching the Markov chain), these predictions are of high quality.

6. RESULTS AND ANALYSIS

The data set contains approximately 500,000 in-progress trajectories (of 1 hour or less in length) from 10,000 taxis. We split the data set randomly into three disjoint sets to facilitate experimentation: 90% in the training set, and 5% in both the test and validation sets. For each model type, the training set V_{TR} is used to generate the model. Model parameters are optimized using the test set V_{TE}. Scores are computed using predictions made on partial trajectories from the validation set V_{VA}. On average, an in-progress partial trajectory has 2.4 transitions in the graph with mean total length of 5.56 km; the complete partial trajectories have 3.3 future transitions (4.63 km mean length) from the last waypoint of the partial trajectory.

Running the algorithms with the parameter values obtained from the parameter optimization we find that LapStrat produces the lowest mean Haudsdorff distance in experiments both with directional contribution information and without (see Table 2).

Table 2: Statistical results by prediction algorithm, reported as mean values over all the samples in the validation set for the best parameter configurations. The best method is in bold. The Wilcoxon signed-rank test is performed to compare the direction and the without direction experiment for each algorithm. Statistical significance denoted by *.

Method	with Direction		without Direction	
	Mean Hausdorff Distance (km)	Mean Prediction Length (km)	Mean Hausdorff Distance (km)	Mean Prediction Length (km)
FreqCount	2.676	16.6	2.693	17.0
EigenStrat	2.641*	13.9	2.675	17.8
LapStrat	**2.425****	17.0	2.509	17.0
MCStrat	2.587**	6.84	2.589	15.6

* $p < 0.05$; ** $p < 0.01$

Figure 12 shows that errors are greater for longer predictions. This figure highlights differences between the methods based on the required distance that must be predicted. For very short length predictions (< 4 km), the performance of the methods is similar. This is reasonable because predictions made for near future travel are most dependent on the current location information and additional information about farther distant movements is unlikely to be informative. For intermediate length trajectories ($4 - 10$ km), the MCStrat algorithm performs better than the three global-based algorithms. The MCStrat configuration (Markov Chain Length $=$ 3) and these experiments indicate that near term information is highly informative for this distance. For long length predictions (> 10 km), MCStrat has larger errors than the three global methods, with LapStrat performing best. MCStrat produces predictions of shorter length than other algorithms as shown by the Mean Prediction Length values reported in Table 2. Long length predictions benefit from knowledge of global patterns which LapStrat can leverage.

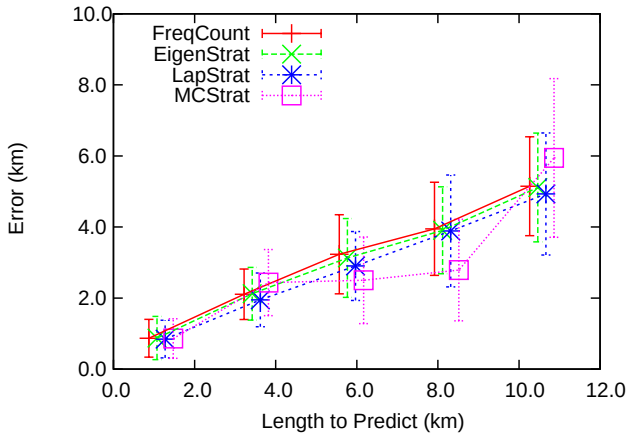


Figure 12: Hausdorff error and std. dev. for each algorithm (at best configuration on with direction data) binned by length of prediction (distance of ground truth) for all validation set trajectories.

Of the global methods, LapStrat performs better than others at all lengths. This may be due to the way that the algorithm clusters common behaviors into groups: it builds clusters based on local similarity (by using pair-wise comparisons of individual policies developed from the training set trajectories). A policy not sufficiently similar to any other in the training set will not be assigned to any cluster. Also, relationships must be sufficiently close (the similarity measure for a pair of edge data vectors must exceed the Lap Epsilon threshold) to be considered at all. These properties differentiate LapStrat from the other global methods and cluster quality is improved.

6.1 Effect of Directional Contribution

While it is possible to see the effect of adding the directional contribution waypoint representation through the sensitivity testing, having an independent experiment testing this aspect of learning avoids any dependencies with the other parameters. Separate parameter searches for each algorithm on data both with and without any directional contribution are shown in Table 2. The statistical significance testing performed using the Wilcoxon signed-rank test shows the directional contribution improves results for EigenStrat, LapStrat, and MCStrat. The directional version improves predictions by facilitating discrimination between co-located points with different directions.

6.2 Parameter Optimization

The parameter settings for the algorithms presented in the results are obtained systematically using the hyperparameter optimization algorithm Tree of Parzen Estimators [1]. This approach is preferred over simpler methods like random sampling or grid search due to greater sampling efficiency. The optimization process is run us-

ing scores computed on the test set, and the reported values are computed using the validation set (to avoid overfitting from the optimization process). The scoring results from the best configuration for each algorithm are listed in Table 2.

Additionally, sensitivity testing is performed for each algorithm parameter using the best configuration as a starting point. The sensitivity is observed for each parameter by fixing the values of all other parameters with values from the best observed configuration and only varying a single parameter throughout its range of valid assignments. These analyses are shown for each algorithm in Figures 13-16. Please note that the four clustering parameters (no. of policy iterations, direction contribution component, transitions per cluster, and cluster count) are repeated for each algorithm as the cluster configuration most compatible may be different for each algorithm.

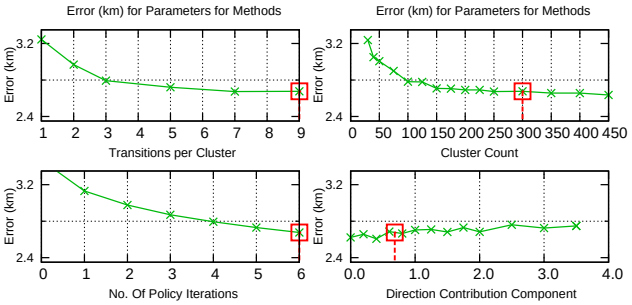


Figure 13: Sensitivity testing for parameters for FreqCount.

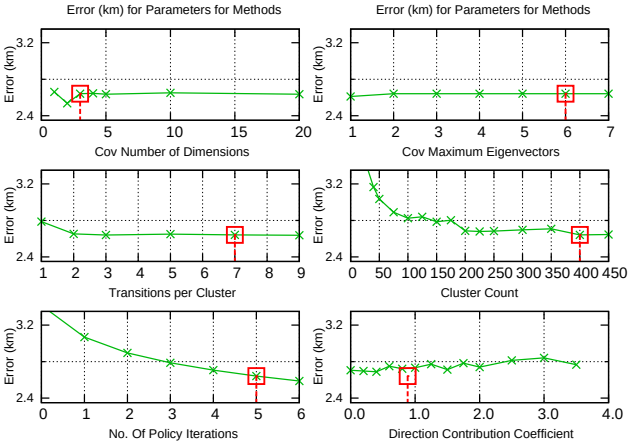


Figure 14: Sensitivity testing for parameters for EigenStrat.

6.3 Complexity Analysis

We consider both the computational and the storage complexity of the proposed algorithms. The time complexity of each principally depends on the representation used for the policy data. The time complexity of building the edge data vector training set (using the city graph abstraction) depends on the complexity of the city graph clustering algorithm and the cost of applying cluster labels to the training data. We use a bottom-up hierarchical clustering approach to cluster the waypoints. The time complexity of clustering is $O(|V_{TR}|^3)$ in the worst-case. Nearest neighbor classification is used to apply cluster labels to the policy training data. The total cost of this labeling is $O(|\Pi_{TR}| \times log(|\boldsymbol{\theta}|))$. The overall cost of

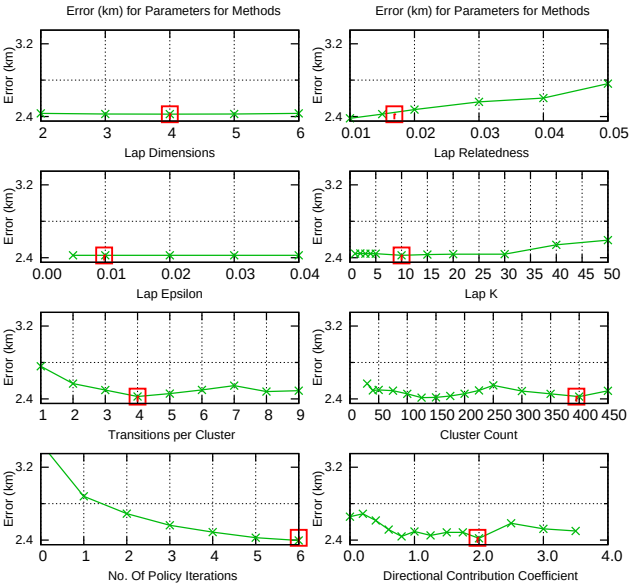


Figure 15: Sensitivity testing for parameters for LapStrat.

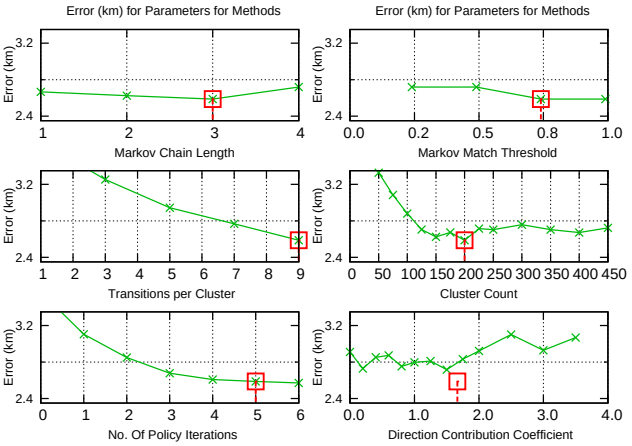


Figure 16: Sensitivity testing for parameters for MCStrat.

building the policy training data is the sum of both time complexities.

The time complexity for LapStrat and EigenStrat can be expressed as the combined cost of building the similarity graph and building the covariance matrix, and the eigen-decomposition. Building the similarity graph and covariance matrix is a $O(|\Pi_{TR}|^2)$ operation on the number of training samples. Eigen-decomposition has a worst-case time complexity of $O(|\Pi_{TR}|^3)$ [2].

MCStrat has a worst-case $O(|\Pi_{TR}|^2)$ time complexity. It scans the entire training set for each evaluated edge data vector. The FreqCount method is the cheapest, it has an $O(|\Pi_{TR}|)$ time complexity, because it does a linear scan on the length of the policy training set when computing the transition probabilities.

A comparison of the storage complexity of the methods appears in Table 3. During model construction LapStrat stores pairwise comparisons of edge data vectors across the entire training set,

Table 3: Space complexity of methods.

Model	Model Construction	Model Storage
FreqCount	$O(\|\boldsymbol{\pi}\|)$	$O(\|\boldsymbol{\pi}\|)$
EigenStrat	$O((\|\boldsymbol{\pi}\|)^2)$	$O(\text{dims} \times \|\boldsymbol{\pi}\|)$
LapStrat	$O((\|\Pi_{TR}\|)^2)$	$O(k \times \|\boldsymbol{\pi}\|)$
MCStrat	$O(1)$	$O(\|\Pi_{TR}\| \times \|\boldsymbol{\pi}\|)$

hence the high storage complexity. On the other hand, the EigenStrat method requires building a covariance matrix of dimensions $|\boldsymbol{\pi}|\times|\boldsymbol{\pi}|$. The model storage for these methods have similar storage complexity, because both methods store characteristic policies.

FreqCount reads one edge data vector at a time from the training set and stores the sum of the coefficients, hence, it requires storage proportional to the length of the stored policy. MCStrat does not require any storage during model construction; however, it must read the entire training set to perform prediction.

To mitigate the complexity of the methods, we sampled from the data and built data abstractions that facilitate computation and storage. For LapStrat, the similarity graph is very sparse (roughly 97%), We used sparse representation of the data to allow for efficient computations. This sparsity can be leveraged to improve computation time by using a sparse algorithm for the eigendecomposition.

6.4 Practical Lessons Learned from Trajectory Data Analysis

Experiments involving GPS data collected in the real world often require data preprocessing to facilitate good predictions. For example, this taxi data set required preprocessing to remove waypoint outliers due to device cold starting and poor GPS signals. These points were mostly identifiable through four criteria used to make trajectories: (1) GPS points with latitude/longitude coordinates far from the study area (outside the bounding box of the major roads surrounding Beijing) were removed; (2) GPS points far (> 30 km) from the preceding/succeeding waypoints are removed; (3) Trajectories must be one hour in length or less; (4) The minimum average speed of a trajectory must be at least 3 km/h. These criteria will remove trajectories that are due to the data collection device being left on while the taxi is parked and will also identify specific GPS waypoints that contain excessive noise. While some erroneous data are still likely to have been admitted into the data set, this prepreprocessing appears, on manual inspection, to have removed the majority of the poor quality data.

7. CONCLUSIONS

Overall, the global methods assume the actions from each cluster location are fixed and paths are implicitly clustered into distinct but repeated goals: in this domain, each observation is a set of actions a driver takes in fulfillment of a specific goal. For example, to take a passenger from the airport to his/her home. In contrast, the local methods only consider actions taken in the immediate neighborhood and are most effective only for short-length predictions.

Although the performance of the framework is already promising, some aspects deserve more research. (1) A non-negligible part of the errors in our predictions is due to representation (map matching) errors. Building inference algorithms to reduce uncertainty caused by the low sampling rate trajectories and regions with low trajectory counts is an interesting direction for future work. (2) Applying algorithms for reducing sample repetition of the input to spectral analysis is also a future direction. This makes spectral clustering more useful when working with very large amounts of data.

The framework presented here can be applied to other spatio-temporal domains where only basic location and time information is collected, such as sensor networks or mobile phone networks. Another direction involves applying the proposed framework to predicting trajectories in non-road based domains (e.g. animal foraging). We believe that spectral analysis can uncover the most frequent paths taken.

8. REFERENCES

[1] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, et al. Algorithms for hyper-parameter optimization. In *Conf. on Neural Information Processing Systems (NIPS)*, 2011.

[2] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[3] J. Froehlich and J. Krumm. Route prediction from trip observations. In *Society of Automotive Engineers (SAE) 2008 World Congress*, 2008.

[4] W. Groves, E. Nunes, and M. Gini. Predicting globally and locally:. In *Proc. of the 2013 Int'l Workshop on Ubiquitous Data Mining at IJCAI*, pages 5–9, 2013.

[5] J. Krumm. Where will they turn: predicting turn proportions at intersections. *Personal and Ubiquitous Computing*, 14(7):591–599, 2010.

[6] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. *Perform. Eval. Rev.*, 32(1):61–72, 2004.

[7] B. Li, D. Zhang, L. Sun, C. Chen, S. Li, G. Qi, and Q. Yang. Hunting or waiting? discovering passenger-finding strategies from a large-scale real-world taxi dataset. In *IEEE Int'l Conf. on Pervasive Computing and Communications Workshops*, pages 63–68, 2011.

[8] W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xing. Discovering spatio-temporal causal interactions in traffic data streams. In *Proc. ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 1010–1018, 2011.

[9] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[10] M. Veloso, S. Phithakkitnukoon, and C. Bento. Urban mobility study using taxi traces. In *Proc. 2011 Int'l Workshop on Trajectory Data Mining and Analysis*, pages 23–30, 2011.

[11] J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[12] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *Proc. 18th SIGSPATIAL Int'l Conf. on Advances in GIS*, pages 99–108, 2010.

[13] J. Zhang, P. Niyogi, and M. S. McPeek. Laplacian eigenfunctions learn population structure. *PLoS ONE*, 4(12):e7928, 12 2009.

[14] Y. Zhu, Y. Zheng, L. Zhang, D. Santani, X. Xie, and Q. Yang. Inferring Taxi Status Using GPS Trajectories. *ArXiv e-prints*, May 2012.

[15] B. Ziebart, A. Maas, A. Dey, and J. Bagnell. Navigate like a cabbie: probabilistic reasoning from observed context-aware behavior. In *Proc. Conf. on Ubiquitous Computing (UbiComp)*, pages 322–331, 2008.