Doctoral Thesis

Mining Algorithms for Network-Constrained Trajectories

Gook-Pil Roh (노국필)

Division of Electrical and Computer Engineering

(Computer Science and Engineering)

Pohang University of Science and Technology

2012

움직임에 제약이 있는 개체의 궤적을 위한 마이닝 기법

Mining Algorithms for Network-Constrained Trajectories

Mining Algorithms for Network-Constrained Trajectories

by

Gook-Pil Roh
Division of Electrical and Computer Engineering
(Computer Science and Engineering)
Pohang University of Science and Technology

A dissertation submitted to the faculty of the Pohang University of Science and Technology in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Division of Electrical and Computer Engineering (Computer Science and Engineering).

Pohang, Korea
06. 27. 2012
Approved by
Seung-won Hwang

Academic Advisor

Mining Algorithms for Network-Constrained Trajectories

Gook-Pil Roh

The undersigned have examined this dissertation and hereby certify that it is worthy of acceptance for a doctoral degree from POSTECH.

06/27/2012

Committee Chair	Seung-won Hwang
Member	Joon Hee Han
Member	Seungyong Lee
Member	Hee-Kap Ahn
Member	Byoungkee Yi

DECE 노국필, Gook-Pil Roh,
20053006 Mining Algorithms for Network-Constrained Trajectories,
움직임에 제약이 있는 개체의 궤적을 위한 마이닝 기법,
Division of Electrical and Computer Engineering, 2012, 114P,
Advisor: Seung-won Hwang. Text in English.

ABSTRACT

With the advent of ubiquitous computing, a massive amount of trajectory data has been published and shared in many websites. Such computing also motivates the needs of the online mining of such data, to fit user-specific preferences or context (*e.g.*, time of the day). While many trajectory analysis algorithms have been proposed, they typically do not consider the restrictions of the underlying road network and have focused on a spatio-temporal query. This dissertation discusses desirable properties for mining the road network trajectories. As the existing work does not fully satisfy these properties, we develop (1) trajectory representation and (2) distance measure that satisfy all the desirable properties we identified. Based on the representation and distance measure, we discuss how to efficiently evaluate similarity search queries which include three types of similarity semantics– whole, subpattern, and reverse subpattern. With the distance measure that reflects the spatial proximity of the road network trajectories, we develop efficient clustering algorithms that reduce the number of distance computations during the clustering process. Moreover, we devise a clustering algorithm considering selection conditions representing user contexts to fit user-specific preferences or context (*e.g.*, time of the day). Our experimental results demonstrate the efficiency and effectiveness of our proposed method using real-life trajectory data.

To my wife Jihye Ku
for her patience and understanding

Contents

List of Figures

List of Algorithms

List of Tables

1 Introduction

With the advent of ubiquitous computing devices equipped with GPS, RFID, or sensors, we can easily record the movement of moving objects, or *trajectories*. Recently, much research work has been studied to evaluate the spatio-temporal query, such as range and nearest neighbor queries, on trajectory database from the given query point [1, 2, 3, 4].

While most existing work focuses on supporting unconstrained trajectory, we focus on moving objects with constraints on their movements. In particular, we deal with the trajectories of vehicles or mobile users on the road. Such moving objects can move only along the roads, thus their trajectories are constrained by the *road network*. Such a trajectory is known as the *network trajectory*.

Toward the goal, we first identify five desirable properties that have not been fully satisfied by any of the prior work:

R1: Road network. As we assume that the moving object moves only along the road, the moving object should not be located off the road, (see Fig. 1.1a), and such off-road locations should not affect measuring the distance between trajectories.

R2: Spatial proximity. The distance measure between trajectories should reflect the spatial proximity from the viewpoint of the road network. For instance, in

Fig 1.1b, the proximity between B and C shares more road segments than A and C, which has to be reflected to the distance measure.

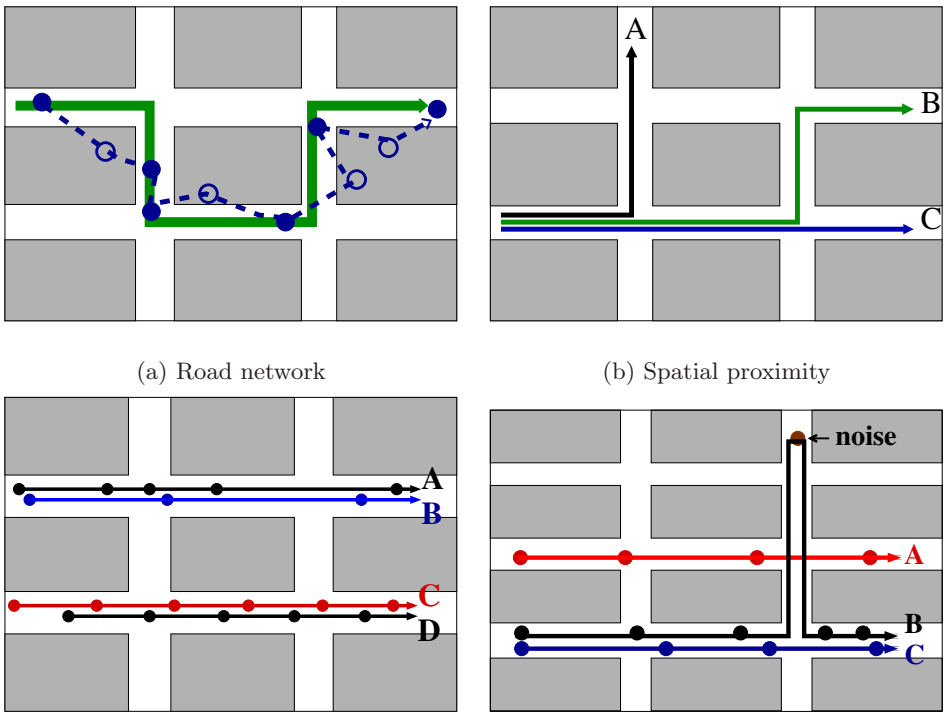
R3: Sampling rate / speed invariant. Due to the difference in sampling rates or speeds, two objects moving along the same route could be represented by two different trajectories (see A and B in Fig. 1.1c). Even when the two objects have the same sampling rates and move along the same route, if the sampling is not synchronized, their trajectories can still differ (see C and D in Fig. 1.1c). The distance measure should not be affected by when or how often the locations are sampled.

R4: Robust to noise. Due to measurement errors or communication failures, trajectories may contain noises, as illustrated in Fig. 1.1d. If a distance measure is sensitive to noise, it cannot reflect the similarity between B and C, and may report A is closer to C. To avoid the problem, the distance measure should be robust to noise, in order to identify an unusual movement as noise and eliminate it in similarity computation.

R5: Metric. In general, it is desirable for distance measures to be *metric*, because irrelevant objects can be pruned out with no false dismissal leveraging existing index structures. However, this property is not required but rather recommended, as non-metric measures can achieve the same advantages, if appropriate indexing schemes can be developed [5].

In addition to supporting network-constrained trajectory representing a sequence of "locations" over time, we also consider "semantically rich" trajectories that can be collected from social network services. For example, Flickr[1] allows users to geo-tag their photos and annotate with comments and tags, which enables users to describe their trips as a trajectory of photos and information, as Figure 1.2 shows a geo-tagged photo of the opera house on the map with annotated information such as title,

[1] flickr.com



(a) Road network

(b) Spatial proximity

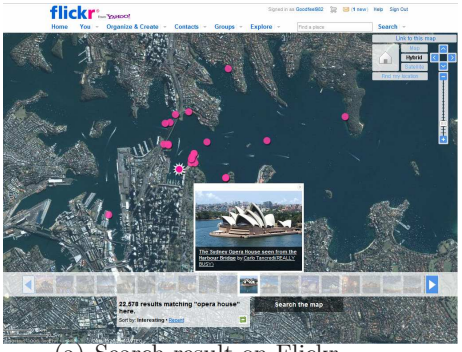
(c) Sampling rate invariant

(d) Robust to noise

Figure 1.1: Illustration of desirable properties identified

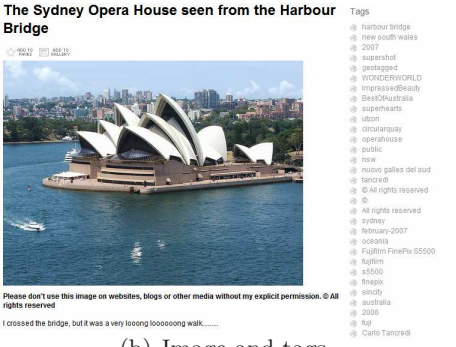
username, or timestamp. Such additional information enables to bridge the semantic gap between the location and the actual activity on the location, by distinguishing different activities happened on the same location, or the same activities on different locations, as Figure 1.3 illustrates. Recent work [6], distinguishing activities from locations using user comments, can be viewed as sharing a similar goal of supporting semantic-rich location-based service.

The scale of such semantic-rich trajectories, *e.g.*, millions of geo-tagged images added monthly to Flickr, naturally motivates us to support advanced data analysis (*e.g.*, similarity search and clustering), *e.g.*, recommending interesting travel itineraries [7].



(a) Search result on Flickr

(tag="opera house"; location="Sydney")



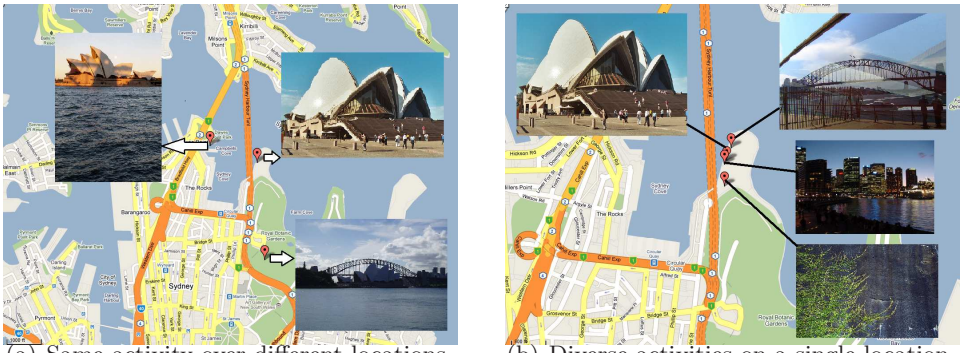
(b) Image and tags

Figure 1.2: Geo-tagged image example on Flickr

In this dissertation, we propose mining algorithms for network-constrained trajectories. First, we aim at supporting effective similarity queries over trajectories. Toward this goal, we propose new types of queries that search for matching trajectory patterns in a database. While such pattern matching queries have been successfully adopted for querying time series data, our work, to the best of our knowledge, is the first to support pattern matching queries over trajectory data. Second, we propose "online" clustering algorithms for semantic-rich trajectories, as we motivate in the following example scenario.

Example 1. *While existing travel recommendations,* e.g., *travel guide or portal, focus on the "offline" computation rendering the same itinerary for all users, travel recommendation on ubiquitous devices naturally motivates "online" recommendation adapting to changes in user-specific context. For instance, depending on temporal context, such as time of the day or season of the year, the recommendation should be different.*

We study online trajectory clustering algorithms, combining the following semantic context criteria:



(a) Same activity over different locations (b) Diverse activities on a single location

Figure 1.3: Semantic gaps between location and activity

- Hard criteria: Users may consider only the trajectories satisfying some Boolean semantic criteria, such as $time = evening$ or $user = tom$.
- Soft criteria: Users may embed various semantic information in the similarity measure, such as image similarity.

We focus on combining both of "hard semantic criteria" and "soft semantic criteria" with clustering, that can be easily provided by the user at the query time. However, existing clustering algorithms cannot address this problem due to the following challenges. *First,* most existing work focuses on the offline "unconstrained" location trajectories, such as hurricane or animal trajectories. In contrast, in many location-based services, trajectory recommendation should reflect the proximity with respect to the underlying constraint, *e.g.*, road networks or subway routes, because two nearby locations can be very hard to reach if no roads connect the two locations. In contrast, we study the online computation of network-constrained trajectories. *Second,* to support hard semantic criteria, we need to support "Boolean+clustering" queries efficiently. While existing clustering algorithms can be adopted after Boolean conditions are evaluated, such naive adoption incurs significantly high computational cost, which motivates an integrated algorithm evaluating both.

Toward this goal, we design a data summary structure where pre-computed initial clusters of trajectories are stored. With the data summary structure we can expedite online computation to merging these initial clusters. We store statistical information on selection attributes to reflect Boolean selections in the merging process.

Summing up, we identify our main contributions in this thesis as follows:

- We identify desirable properties for mining road network trajectories.
- We devise a trajectory representation and similarity measure that satisfy the properties.
- We propose an efficient algorithm for similarity search queires that include three types of simialrity semantics–whole, subpattern, and reverse subpattern.
- We motivate and study the problem of supporting Boolean+clustering queries for location trajectories and semantic-rich trajectories.
- We design a summary structure for pre-materializing initial trajectory clusters.
- We propose an efficient clustering algorithm building on the proposed summary structure.
- We extensively validate the effectiveness and the efficiency of our proposed framework using real-life trajectory data.

2 Related work

2.1 Similarity measures

In this section, we survey prior research efforts on (a) querying trajectories and (b) modeling trajectories and their similarity. We also present some background information necessary for the proposed methods.

Queries: There have been prior research efforts for querying trajectories constrained by networks [8, 9, 10, 11]. A data model of network-constrained moving objects has been proposed in [12], followed by many index structures proposed to efficiently perform the spatio-temporal window query (3D interval) in [8, 9, 10]. Lin *et al.* [13] and Frentzos *et al.* [14] proposed a similarity search of moving object trajectories, which is similar to our whole pattern matching query. In a clear contrast, we focus on whole, subpattern, and reverse subpattern matching queries, of which the last two types of queries have never been studied before for trajectory data.

Trajectory representation and similarity: Next, we survey prior work on the trajectory and similarity modeling, which we categorize into two categories– the sequence based [15, 16, 17, 18, 19, 20, 21] and the shape-based similarity measures [22, 23, 13, 24]. The sequence-based similarity measures build upon trajectories

represented as a *sequence* of points in two or three dimensional space, comparing how closely the two sequences align with each other. In contrast, shape-based similarity measures build upon trajectories as a *set* of points or line segments and they are thus not sensitive to the order of points or segments. We stress that, while these measures do not consider the road network, our work, for the first time to the best of our knowledge, supports pattern matching queries for the network-constrained trajectories.

2.1.1 Sequence-based similarity measure

In this area of research, a trajectory is represented as a multi-dimensional time series. From this point of view, there have been prior research efforts to extend similarity metrics used in time series analysis to trajectories. Many similarity measures were proposed, such as Euclidean distance [15, 25], Longest Common Subsequence (LCSS) [16], Edit Distance on Real sequence (EDR) [17], and Edit distance with Real penalty (ERP) [18], most of which quantify how closely the two sequences align with each other.

LCSS, by quantizing the distance between a pair of elements as either 0 or 1, is rather robust to noise. However, it is known that LCSS could be inaccurate by neglecting to consider *gap g* between similar subsequences. To address this problem, Lei Chen *et al.* [17] propose EDR which considers the gap penalty between the two matching subsequences. ERP is a variant of EDR, which does not quantize the distance between elements.

The sequence-based similarity measures do not consider network constraints and thus do not satisfy **R1**. LCSS and EDR moderately satisfy **R2** by quantizing the distance between a pair of elements. All the sequence-based similarity measures are affected by the different sampling rates (**R3**) of trajectories. LCSS and EDR are rather robust to noise by quantizing element distance (**R4**), while ERP is not.

Among them only ERP is a metric (**R5**).

Several distance measures for trajectories have been proposed for clustering [19, 20] or searching [21] for similar patterns. Gudmundsson *et al.* [19] studies how to efficiently detect the spatio-temporal patterns (flock, leadership, convergence, and Encounter) from trajectories, while Jeung *et al.* [20] propose a relaxed version of flock pattern, called convoy. Among these notions, flock, convoy, and leadership notions share similarity with our notion of similarity, while we will describe differences below. Similarly, Bakalov *et al.* [21] define the distance measure between symbolic representations of trajectories and study similarity search.

However, similar to LCSS or EDR, all of the above measures [19, 20, 21] do not satisfy **R1** and **R3**, because they consider trajectories of free movements. In addition, [19, 20] do not fulfill the requirement **R4**. and [21] does not satifsy **R2**.

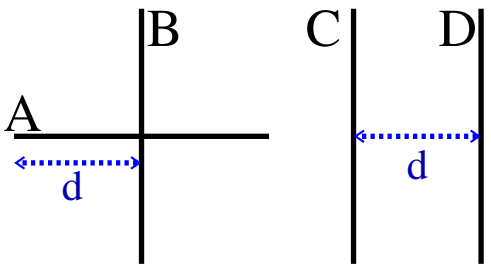
2.1.2 Shape-based similarity measure

Another area of research discusses the shape-based similarity measures for trajectory data. The shape-based similarity measures differ from sequence-based similarity measures in that they do not consider the order of elements of trajectory. In other words, the shape-based similarity measures ignore the time information of trajectories and appraise the similarity with respect to the *shapes* of trajectories. In this line of work, trajectories are usually represented as a point set or polyline.

Point set

For trajectories represented as point sets, Hausdorff distance has been widely used to measure "resemblance" between the two shapes. In particular, Lou *et al.* [26] and Junejo *et al.* [27] adopt Hausdorff distance as a distance measure for the trajectory.

Given two finite point sets A and B, the Hausdorff distance $D_H(A, B)$ is defined



(a) (b)

Figure 2.1: The limitation of Hausdorff distance

as

$$D_H(A, B) = \max\{d_H(A, B), d_H(B, A)\},$$

where

$$d_H(A, B) = \max_{a \in A} \min_{b \in B} \|a, b\|$$

and $\|a, b\|$ is the Euclidean distance between a and b. The directed Hausdorff distance $d_H(A, B)$ takes the maximum among all the distances from each point of A to its closest point on B. The (bidirectional) Hausdorff distance is the maximum of the two directed Hausdorff distances.

Hausdorff distance builds upon the trajectories represented as point sets on Euclidean space, and thus does not satisfy the requirement **R1**. Hausdorff distance supports **R2** with the following limitation– Fig. 2.1 (a) and (b) show two trajectories with the same Hausdorff distance d, while the pair in (b), C and D, is more similar. Regarding **R3**, Hausdorff distance does not handle different sampling rates of moving objects, as the same trajectory can be represented as two different point sets depending on the sampling rate. It is generally known that Hausdorff distance is a metric (**R5**) and sensitive to noise (**R4**).

Polyline

For trajectories represented as a set of line segments, polylines, Gao *et al.* [28] proposed Line segment Hausdorff distance (LHD) and Chen *et al.* [29] improve LHD to be robust to noise. Though these measures were invented for pattern recognition, such as face matching and logo recognition, they can be adopted to the problem of matching trajectories if we represent trajectories as polylines, as Lee *et al.* employed the distance measure, proposed in [28], for trajectory clustering problem in [22].

Basically, LHD follows the same intuition of Hausdorff distance, except that LHD defines a new distance metric $d(\cdot, \cdot)$ as a weighted L_2-norm of feature vector, which consists of the angle distance, parallel distance, and perpendicular distance of two lines, as illustrated in Fig. 2.2– The angle distance is the smallest intersecting angle between two line segments, *i.e.*, θ in Fig. 2.2. The parallel and perpendicular distance are $\min\{a, b\}$ and c respectively in Fig. 2.2. Due to the similarity shared between LHD and the Hausdorff distance, the two metrics fulfill the same set of desirable properties.

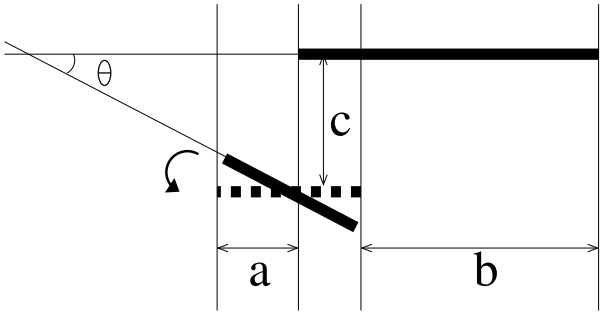


Figure 2.2: Distance computation of LHD

Fréchet distance [30] was proposed to measure the resemblance between two polygonal curves, which is generally known to better capture the spatial proximity than the Hausdorff distance. To illustrate, in Fig. 2.3, Fréchet distance can detect the difference of two very different trajectories, while the Hausdorff distance of these two

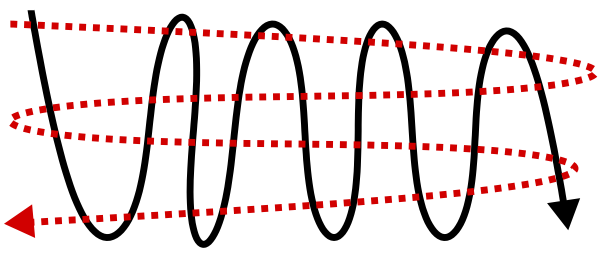


Figure 2.3: Two trajectories with small Hausdorff distance

trajectories is very small. However, computing the Fréchet distance is more complex, *i.e.*, $O(mn \log(nm))$ [31] for two polygonal curves with m and n segments.

A discrete version of the Fréchet distance, which is an approximation of Fréchet distance, is introduced in [32]. It is based on the idea of looking at all possible couplings between the endpoints of the line segments of the polygonal curves. The discrete Fréchet distance is an upper bound for the origial Fréchet distance. The difference between those two measures is bounded by the length of the longest edge of the polygonal curves. The discrete Fréchet distance can be computed in O(mn) time using an dynamic programming algorithm. Fréchet distance and its discrete version consider the order of the endpoints of trajectories.

Alternatively, the Euclidean distance between polylines can be used as a distance measure [24]. As preprocessing, spatial or temporal normalization is performed to deal with trajectories of different lengths– First, spatial normalization, using linear interpolation, transforms the polyline sets into new sets with equal spatial length. Temporal normalization then follows to re-sample the trajectory per fixed time intervals. This normalization enables to support trajectories with different sampling rates (**R3**). However, this measure does not consider the road network (**R1**) and cannot fully capture spatial proximity (**R2**). Also, it suffers from noise and is not a metric, which suggests that **R4** and **R5** are not satisfied.

Table 2.1: Summary of previous research

Requirements		**R1**	**R2**	**R3**	**R4**	**R5**
Sequence	LCSS [16],EDR [17]	X	△	X	O	X
	ERP [18]	X	O	X	X	O
	FLOCK [19],CONVOY [20]	X	X	X	X	X
	TS2 [21]	X	△	△	X	O
Shape	Hausdorff (Lee *et al.* [22])	X	△	X	X	O
	Frèchet [23]	X	O	X	X	O
	normalization [24]	X	△	O	X	X
	Lin *et al.* [13]	X	X	X	O	X

Recently, Bin Lin *et al.* [13] proposed another shape-based similarity measure and its approximation, when trajectories are represented as polylines connecting each adjacent pair of observed points. Due to the high computational cost of their proposed distance measure, they present an approximation scheme as well, building on the grid representation. This grid-based approach reduces both the computational cost and sensitivity to noise. However, this proposed measure [13] satisfies the requirement **R4**.

We summarize the fulfillment of requirements for the existing distance measures in Table 2.1. The symbols 'O', 'X', and △ mean 'satisfy', 'not satisfy', and 'partially satisfy' respectively.

Time-order dependency

In contrast, to generally support both order-dependent and order-independent measures, we focus on devising an order-independent measure as a lower bound for both distance measures. To illustrate the differences, Figure 2.4 illustrates two trajecto-

ries that move along the same route but in opposite directions. Assume that the two trajectories have the same sampling points $(t_0, \cdots, t_5)$. T_A moves from t_5 to t_0 and T_B moves from t_0 to t_5. When the distance measure becomes sensitive to the order of sampling points, *i.e.*, an order-dependent measure, T_A is thought to be totally different from T_B. In contrast, when using order-independent measures, the two trajectories are considered as being identical.

This suggests that ranking pairwise distances by order-independent measure "underestimates" the order-dependent distances. We can thus obtain a ranking of order-dependent distances by using order-independent metrics (if required by an application), by simply adding a post-filtering step to discard the "false positives". We can thus focus on order-independent metrics.

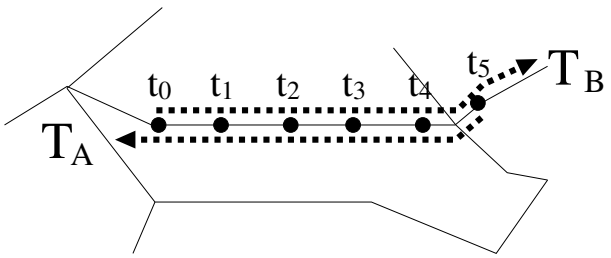


Figure 2.4: Effects on the order-dependence of distance measure

2.2 Clustering algorithms for trajectories

Lately, clustering trajectories has been studied actively, but most work assumes trajectories without road network constraints, as in hurricane tracking or animal movement data. For trajectories without constraints, trajectories are typically represented by either a *set* or *sequence* of locations, for which similarity metrics have been proposed in [26, 27, 33, 34, 35, 36, 13] and [15, 25, 16, 24, 17, 18, 37] respectively. In a clear contrast, we focuses on clustering trajectories constrained by road networks,

Once similarity metrics are defined, existing algorithms can be applied to group trajectories into clusters. Specifically, recent work proposes to adopt EM (Expectation Maximization) algorithm [38] and density-based clustering algorithm [36] for the task. However, these algorithms are not suitable for our proposed problem of online clustering with Boolean selection conditions– [38] requires slow convergence, which prohibits online response, while [36] requires to pre-materialize an index structure for online computation, which becomes ineffective with the presence of Boolean selection conditions.

Our proposed clustering algorithm distinguishes itself with the above existing work, by designing a distance metric for network-constrained "location" trajectories and developing an online algorithm. This dissertation, based on this preliminary work, extends the metrics and algorithms to supporting semantic-rich trajectories, archiving not just location changes, but also the semantic information such as images, text, userID, and tags. In particular, we study to support Boolean selections on semantic features. For efficient support, we propose to minimize online computation requirements, as similarly adopted for numerical data in [39]. However, adopting [39] for our proposed problem is not desirable, as this work assumes two objects with similar selection attribute values are similar in terms of the similarity metric. This

assumption does not hold in our problem, where similarity and selection attributes are allowed to be independent, *e.g.*, similarity on image features and selection on time stamps.

The most relevant work [6] distinguishes users' activities from location using user comments on the trajectory, which can be viewed as sharing a similar goal. Our work can similarly support this, either by embedding tf-idf textual similarity in the similarity function (offline), or supporting Boolean conditions or texts or tags, such as $tag = dining$ (online). A key distinction of our work from existing work is to provide a systematic framework for online clustering with arbitrary Boolean semantic conditions.

In the field of computer vision, clustering trajectories is often used to feed prototype trajectories into a model for detecting abnormal trajectories in the surveillance system. In order to define the model of the surveillance system, it is of high interest to associate objects with common observed trajectories and identify "outliers" that cannot be associated. However, as the number of trajectories grows, it becomes impractical to group and identify common patterns manually. To automate this process, neural network based approaches [40, 41], a node-based model [42, 43], and hierarchical models [44, 45, 46] have been proposed. However, these models cannot represent network-constrained movements as in our paper, which is not critical in a narrow-range surveillance application where search space is often bounded by the range of a single camera.

Meanwhile, for wider-range surveillance by multiple cameras, our trajectory clustering method, which is based on the network-constrained trajectory, can be adopted, to accurately detect the behaviors of moving vehicles or mobile users constrained by network.

3

Distance measure for road network trajectories

3.1 Trajectory representation

In this section, we propose to represent a trajectory as a sequence of road segments in a road network. We define a road segment as its starting and ending positions and require it not to intersect with any other segment. (Note that this requirement is not restrictive, since a crossing segment pair can be alternatively represented as four non-intersecting segments.) We also require a trajectory to be *connected* in that the ending point of a segment should be the starting point of the next segment in the sequence. With this representation, we can represent the restriction of road networks and the spatial proximity of road segments, without being affected by different sampling rates of moving objects.

Trajectories of *raw data* from positioning devices such as GPS consist of quadruplets (x, y, p, v), where x and y are 2D location points, while p and v are the time stamp and the velocity respectively. Most of the prior researches take the raw data "as is" as input, which cannot represent the restrictions of the underlying road networks.

In contrast, our proposed trajectory representation uses a sequence of road segments, which requires a pre-processing phase to transform the raw data into a con-

nected sequence of road segments. One alternative is to adopt map-matching algorithms for this process, especially an *incremental* map-matching algorithm [47] devised for fast map-matching. However, this algorithm requires the connectivity information, such as an adjacency matrix, of the entire road networks, let alone the need to tune five parameters used for scoring. In contrast, we devise a memory-efficient scheme, requiring to maintain only the vicinity area (which is less than 0.01% of the entire network in our evaluations).

To devise an efficient mapping, there are two challenges: *First,* for each 2D point, we need to find the road segment it belongs to. *Second,* in case two consecutive road segments are not connected, which can happen due to the error of the GPS device or infrequent sampling, we need to connect the two, to satisfy our assumption.

For the first challenge, we adopt R-tree [48] to index all the road segments of the given road network. This approach is similar to the method that employs the R-tree index to find the road segment that covers a query point in [49]. Each road segment is transformed to the Minimum Bounding Rectangle (MBR) whose diagonal is the straight line, connecting $r_{i,s}$ and $r_{i,e}$ of road segment, and sides are parallel to the coordinate axes. Once these MBRs are inserted into the R-tree, the nearest road segment for the given 2D point can be identified by simply performing a nearest neighbor search on the R-tree.

For the second challenge of connecting two *disconnected* consecutive road segments, when the ending point of one segment does not match the starting point of the next, we use the shortest path approach that recovers the missing path by finding the shortest path between the ending position and the starting position of the two segments. This approach is based on an assumption that a moving object insists on taking the shortest path to reach its destination. For this solution, we can employ one of the heuristic algorithms listed in [50], adopting existing shortest path algorithms

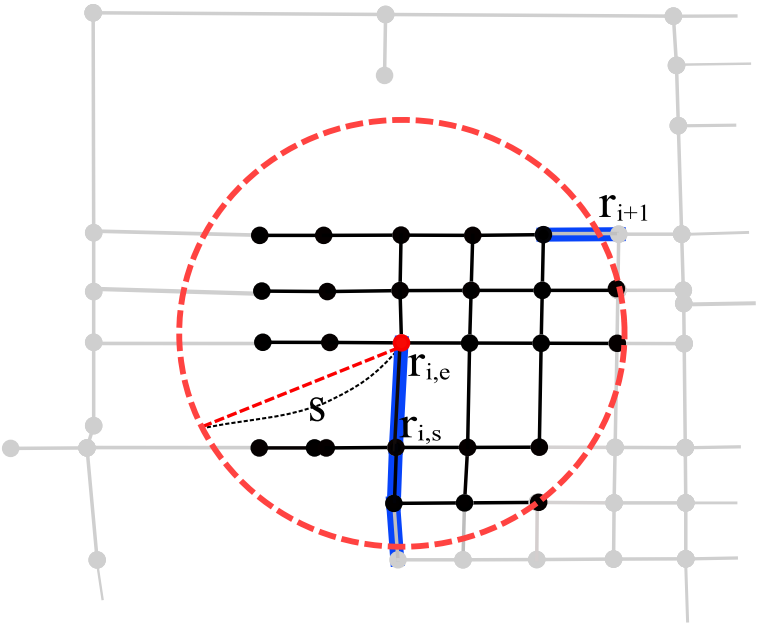


Figure 3.1: Example of vicinity circle/graph

on large-scale real road networks.

However, running a shortest path algorithm, such as Dijkstra's, for the entire road network incurs prohibitive cost, *i.e.*, $O(n^2)$, where n is the number of nodes in the graph representing the road network. Considering there are usually millions of nodes in the road network, we need to restrict the search space. More specifically, we restrict our search to road segments in the "vicinity circle" of the disconnected road segments. To ensure correctness, we generate a vicinity graph G that includes all road segments on the shortest path.

To generate a vicinity graph G, for the given disconnected road segments r_i and r_{i+1}, we find a vicinity circle, centered at the ending position of the preceding road segment, *i.e.* $r_{i,e}$. Fig. 3.1 illustrates the vicinity circle of the disconnected road segment r_i and r_{i+1} with its radius $s = v_{max} \times (ts(r_{i+1}) - ts(r_i))$ where v_{max} is the maximum speed of the moving object and $ts(r_i)$ is the time stamp of sample point

which is transformed to r_i. If r_i includes multiple sample points, we take the last one. Note that, this circular region covers the maximum *driving* distance a moving object can travel along the roads from r_i to r_{i+1}. All the road segments contained by the vicinity circle are retrieved (*e.g.*, the road segments in dark in Fig. 3.1) and then inserted into the vicinity graph G.

Once the vicinity graph is identified, we run a shortest path algorithm to find the shortest path segments from $r_{i,e}$ to r_{i+1}. In the case r_{i+1} is disconnected from r_{i+2}, it is not clear which endpoint is the starting point. In such a case, we should identify the shortest path from the single source point to both endpoints of r_{i+1}, *i.e.*, "one-to-some" shortest path. For such a case, we choose the Dijkstra's approximate bucket implementation [51], which was empirically validated in [52] to perform well for one-to-some shortest path problem.

To eliminate noise caused by measurement errors, we reasonably assume that noise does not occur in k consecutive road segments, possibly with the help of error correction technology of GPS devices. (k is an application dependent parameter.) With this assumption, we calculate the vicinity area until we find the road segment which is intersected by the vicinity area in k consecutive road segments from r_i. If r_{i+k} is the road segment that is first intersected by the vicinity area, we drop the road segments from r_{i+1} to r_{i+k-1}.

3.2 Distance measure

In this section, we present a novel distance measure for network-constrained trajectories. As we discussed in chapter 2, we focus on order-independent distance measure to generally support both order-dependent and order-independent measures. In particular, we extend the intuition of Hausdorff distance to our trajectory representation– Informally, the Hausdorff distance between two sequences of segments, is the longest distance an adversary can force you to travel from one segment to another.

More specifically, we first determine how to measure distance between two road segments. Using this measure, we then define the distance measure for trajectories. First, we define the distance $d(r_i, r_j)$ between two road segments r_i and r_j, based on which we will define the distance between the trajectories later on. The distance between road segments is defined as follows:

Definition 1 (Road segment distance)**.**

$$d(r_i, r_j) = \max\left\{ \overrightarrow{d}(r_i, r_j), \overrightarrow{d}(r_j, r_i) \right\}, \tag{3.1}$$

where $\overrightarrow{d}(r_i, r_j)$ *is a one-way road segment distance from* r_i *to* r_j *which is defined below.*

Definition 2 (*one-way* road segment distance)**.**

$$\overrightarrow{d}(r_i, r_j) = \max\left\{ \begin{array}{l} \min\{\|r_{i,s}, r_{j,s}\|, \|r_{i,s}, r_{j,e}\|\}, \\ \min\{\|r_{i,e}, r_{j,s}\|, \|r_{i,e}, r_{j,e}\|\} \end{array} \right\}, \tag{3.2}$$

where $\|a, b\|$ *is the Euclidean distance between* a *and* b*.*

The road segment distance is the longest of two possible one-way distances to travel from one segment to the other.

One may argue that $\|a, b\|$ should be the shortest path distance between a and b, to satisfy the **R1** property. Such an alternative, however, incurs prohibitive computational cost, because we have to search a large portion of the road network if a and b are road segments located far from each other. Therefore, the shortest path distance is practically infeasible for comparing trajectories.

Based on the road segment distance above, we now move on to define the distance measure between two trajectories.

Definition 3 (Trajectory distance)**.** *Given two trajectories* $T_a = [a_1, \ldots, a_n]$ *and* $T_b = [b_1, \ldots, b_m]$*, the distance between them is defined as follows:*

$$D(T_a, T_b) = \max\left\{\overrightarrow{D}(T_a, T_b), \overrightarrow{D}(T_b, T_a)\right\}, \tag{3.3}$$

where $\overrightarrow{D}(T_a, T_b)$ *is one-way trajectory distance from* T_a *to* T_b*.*

Definition 4 (*One-way* trajectory distance)**.** *Given two trajectories* $T_a = [a_1, \ldots, a_n]$ *and* $T_b = [b_1, \ldots, b_m]$*, the one-way trajectory distance is defined by*

$$\overrightarrow{D}(T_a, T_b) = \max_{a_i \in T_a} \min_{b_j \in T_b} d(a_i, b_j) \tag{3.4}$$

Similar to Hausdorff distance, our distance measure can be explained as the longest path from each road segment to its closest road segment in another trajectory. To illustrate, consider two trajectories of moving objects A and B in Fig. 3.2. The two trajectories share the same movement up to the point where A keeps moving straight while B makes a right turn. That is, the distance between A and B is 0, until the two objects take different paths, when the distance to travel for A to reach B, or vice versa, is either the distance of A making a U-turn to B, as represented as a dashed line in the figure, or B making a U-turn to A. As the former distance is longer, we use the length of the dashed line as the distance between two trajectories.

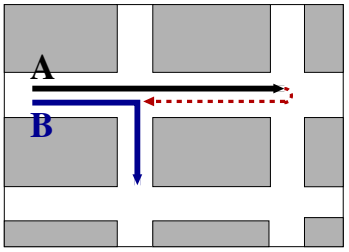


Figure 3.2: Meaning of proposed distance measure

Next, we argue how our proposed representation and distance metric fulfill the properties identified in Section 3.1. First, our trajectory representation as a sequence of connected road segments, satisfies the three requirements **R1**, **R2**, and **R3**– Our representation considers the restriction of road networks and the spatial proximity of the segments, and it is not affected by the different sampling rates of the objects. Second, we argue how our proposed distance measure also satisfies **R4** and **R5**.

While our distance itself does directly satisfy R4 property, we achieve it by means of the robust trajectory representation as described in the previous section. Lastly, we show that our distance measure is a metric, which ensures efficient query processing using an index structure, as we will later discuss.

Theorem 1. *The trajectory distance is a metric.*

Proof. The reflexivity and symmetry directly follow from Eq. 3.3 and 3.4. To prove the triangle inequality, we first define ϵ-neighborhood road segments of trajectory T_a, denoted by $N_\epsilon(T_a)$, as follows: $N_\epsilon(T_a) = \{r \in R | d(r, a_i) \leq \epsilon, 1 \leq i \leq n\}$. Then,

$$\begin{aligned}
\overrightarrow{D}(T_a, T_c) &= \inf\{\epsilon > 0 | T_a \subset N_\epsilon(T_c)\} \\
&= \inf\left\{\delta + \delta' | T_a \subset N_{\delta+\delta'}(T_c)\right\} \\
&\leq \inf\left\{\delta + \delta' | T_a \subset N_\delta(T_b) \text{ and } T_b \subset N_{\delta'}(T_c)\right\} \\
&\leq \inf\{\delta | T_a \subset N_\delta(T_b)\} + \inf\left\{\delta' | T_b \subset N_\delta(T_c)\right\} \\
&= \overrightarrow{D}(T_a, T_b) + \overrightarrow{D}(T_b, T_c).
\end{aligned}$$

Similarly, $\overrightarrow{D}(T_c, T_a) \leq \overrightarrow{D}(T_b, T_a) + \overrightarrow{D}(T_c, T_b)$. □

Note that we can adopt discrete Fréchet distance framework instead of Hausdroff distance framwork. However, discrete Fréchet distance can be used for only whole pattern matching. Discrete Fréchet distance itself does not cover semantics of the subpattern matching and reverse subpattern matching. We can consider an additional preprocessing step, such as partitioning a trajectory into multiple sub-trajectories of smaller length, for discrete Fréchet distance to support subpattern and reverse subpattern matchings. However, such preprocessing step increases the time complexity of query processing and the space complexity of the index structure. In contrast to Fréchet distance, Hausdorff distance framework perfectly captures three types of similarity notions. Furthermore, it requires just one index structure. We will discuss further in chapter 4.3. Furthermore, another advantage of our framework is supporting all three types of queries using a single index structure. We will explain how the index structure built for whole pattern patching query can be used for subpattern and reverse subpattern in Chapter 4.3.2 and 4.3.3, respectively.

4

Similarity search on network-constrained trajectories

Most of the previous research efforts focus on efficiently evaluating the *spatio-temporal query*, such as supporting range and K nearest neighbor (KNN) queries, from the given query point, for location-based services (*e.g.*, [1]). In [53], many index structures are surveyed for efficient query processing on the spatio-temporal database. In clear contrast, we support new types of queries that search for matching trajectory patterns in a database. While such pattern matching queries have been successfully adopted for querying time series data, our work, to the best of our knowledge, is the first to support pattern matching queries over trajectory data.

4.1 Problem definition

Whole pattern matching: Whole pattern matching captures the global similarity between two trajectories and compares them in their entireties without ignoring any part of them. Consider two example trajectories in Fig. 4.1. Suppose trajectory T_1 represents a commute from work to home of some user U_1. He can then query with T_1 to retrieve other trajectories sharing similar road segments, *e.g.*, T_2 to carpool with.

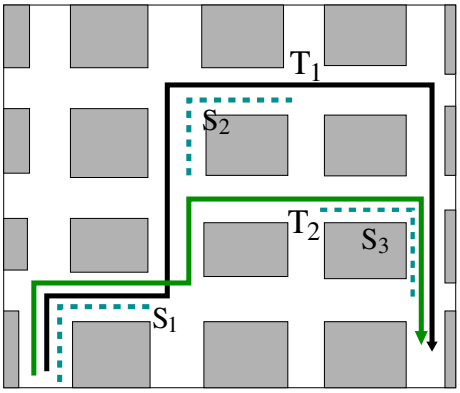


Figure 4.1: Illustration of three patten matching types

Subpattern matching: Subpattern matching on the other hand considers the query trajectory as whole and some interesting parts of database trajectories that best match the query trajectory and ignores the rest of them. To illustrate, suppose there is a road construction plan, say S_3 in Fig. 4.1. Then, the drivers who may be affected by the construction are those whose commute routes contain S_3 as a subpattern, which is T_2 in this particular case.

Reverse subpattern matching: Alternatively, the subpattern matching problem can be reversed. Suppose we consider a plan for a new bus route. Revese subpattern matching can be useful to find the drivers who may use the bus instead of driving to commute. For example, if T_1 in Fig. 4.1 is the new bus route, then S_1 and S_2 are the candidates.

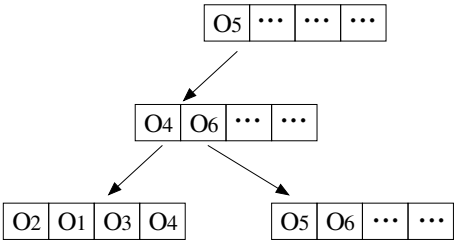
4.2 Index structure

In this section, we discuss how to index trajectories for the efficient evaluations of proposed queries. Without such an index structure, we have to repeat computing the distance between a query trajectory T_q and all trajectories T in the database. In contrast, an effective index structure enables the elimination of distance computations for the trajectories that do not affect query results.

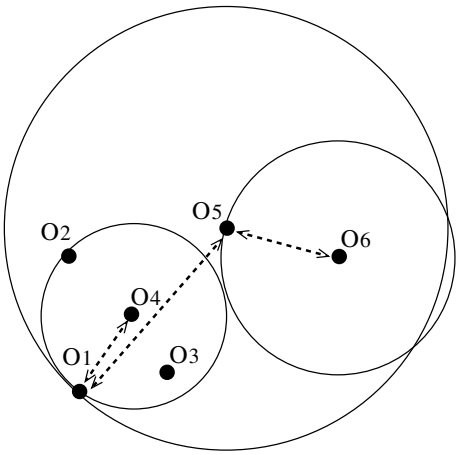
As our proposed distance measure fulfills the metric property, we can use any index structures designed for the metric space, *i.e. metric tree*, such as the vp-tree [54], the mvp-tree [55], the gh-tree [56], the GNAT [57] and the M-tree [58]. These metric trees all exploit the triangular inequality of distance metric to prune out irrelevant objects.

Among these trees, we pick M-tree as our choice of the index structure for the following reasons: First, the M-tree is the only one that is optimized for large secondary memory-based data sets [59]. Other metric trees are main memory index structures, which cannot handle the trajectory database whose size is bigger than the capacity of the main memory. Second, the M-tree supports dynamic updates, such as insertion and deletion from the database, which often occur in a trajectory database. Last, M-tree is reported to be more scalable than other approaches in [58].

As preliminaries, we briefly overview how M-tree works: The leaf node of M-tree contains the actual objects, whereas the non-leaf node stores the representative objects, which are selected among objects in the sub-tree using a selection algorithm described in [58]. Both leaf and non-leaf node also include the distance to the parent object. In the case of non-leaf node, two additional features are stored, in addition to the representative object: a pointer to the sub-tree and a covering radius that is the distance between the representative object and the farthest object from the



(a) M-tree structure



(b) Covering radii

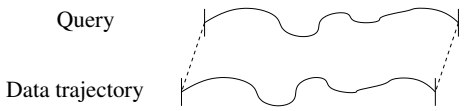
Figure 4.2: Example of M-tree index

representative object in the sub-tree. We illustrate an example of the M-tree structure in Fig. 4.2(a) with the covering radii denoted by dashed arrows for three representative objects, *i.e.* O_4, O_5, and O_6, in Fig. 4.2(b). For the given trajectory database, we can build a single M-tree, by inserting each trajectory using trajectory distance measure (Equation 3.3) as metric function.

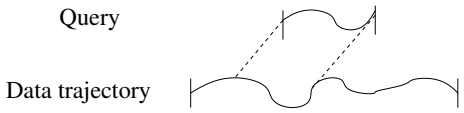
4.3 Query processing

Given a query trajectory, we support three types of pattern matching: whole, subpattern, and reverse subpattern matching, for finding "similar" trajectories to a query trajectory in the database. However, as illustrated in Section 4.1, for each type of matching, the definition of "similar" trajectory varies. First, in whole pattern matching, the query trajectory is typically of similar length to the trajectories in the database and the query retrieves the result trajectories that are globally similar to the query trajectory. Second, in subpattern matching, the query trajectory is typically much shorter than the trajectories in the database and the query retrieves the results containing some subpatterns that are similar to the query pattern. Lastly, in reverse subpattern matching, the query is typically much longer than the trajectories in the database and the query retrieves the trajectories in the database that match a certain part of the query pattern.

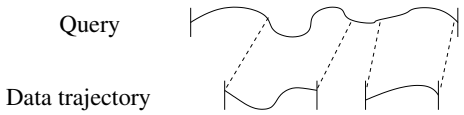
Fig. 4.3 illustrates these three types of pattern matching.



(a) Whole pattern matching



(b) Subpattern matching



(c) Reverse subpattern matching

Figure 4.3: Three types of pattern matching queries

In the following sections, we present algorithms for (a) range queries finding the trajectories with distances smaller than user-specified threshold ϵ and (b) KNN (K Nearest Neighbor) queries finding the k trajectories with the smallest distance, and prove their correctness.

4.3.1 Whole pattern matching

Whole pattern matching searches for the trajectories whose movement patterns are globally similar to the given query trajectory. To support the range and KNN search, we can simply employ the range and KNN search algorithm of M-tree [58] without any modification, as our distance measure is metric (Theorem 1). During the traversal of M-tree, for each non-leaf node, we need to decide whether its sub-tree can be pruned out– Fig. 4.4 shows the pruning condition of the whole pattern matching. If the distance d between a query trajectory T_q and a representative trajectory $rp(O_i)$ of a non-leaf node O_i is bigger than the sum of search range ϵ and scope $s(O_i)$, the distance between every object in the sub-tree of O_i and T_q is also bigger than ϵ, which suggests that O_i can be safely pruned out, as we formally state below:

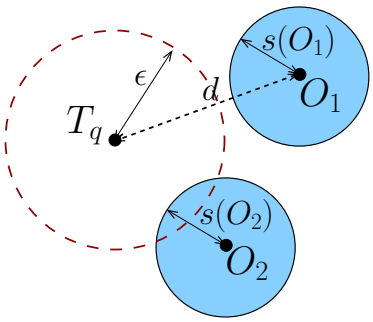


Figure 4.4: Pruning for whole pattern matching

Theorem 2 (Pruning condition for whole patterns)**.** *Given a query trajectory T_q and a user-specified threshold ϵ, no trajectory T in O satisfies $D(T_q, T) \leq e$, if*

$D(T_q, rp(O)) > s(O) + \epsilon$.

Proof. Immediate from [58], as our distance measure is metric (Theorem 1). □

With this pruning condition, we can conclude that, in the case of Fig. 4.4, we do not need to investigate the child nodes of O_1, because the pruning condition $d > \epsilon + s(O_1)$ is satisfied. This condition is used in both range and KNN search.

4.3.2 Subpattern matching

Subpattern matching searches for all the trajectories which include a similar pattern to a query trajectory. While we used *two-way* distance for the whole pattern matching, as we need to consider all the road segments of the trajectories to globally match the query, in the subpattern matching, we only need to consider the closest segment to the query, to quantify how well the specific closest segment to the query matches the query pattern. The *one-way* distance, $\overrightarrow{D}(T_q, T)$, perfectly captures such a notion, which is only affected by the nearest road segment of a trajectory T from each road segment of T_q. For subpattern matching, we can thus simply modify the distance notion $D(T_q, T)$ in Algorithm 1 and 2, to use *one-way* distance $\overrightarrow{D}(T_q, T)$ instead, to identify the closest subpattern to the query pattern, which suggests that we can reuse the index structure built for whole pattern.

To illustrate why *one-way* distance is more proper for the subpattern matching, consider two trajectories (A and B) and a query trajectory (dashed line) in Fig. 4.5. Both of their *one-way* distances from a query are 0, as they contain the exactly same road segment, or subpattern. Meanwhile, their *two-way* distance, considering all the road segments, differ, as they begin to take different paths at one point. For our purpose of matching a subpattern, *one-way* distance, $\overrightarrow{D}(T_q, T)$, is thus more proper.

We need to update the pruning condition to use *one-way* distance instead. Fig. 4.6

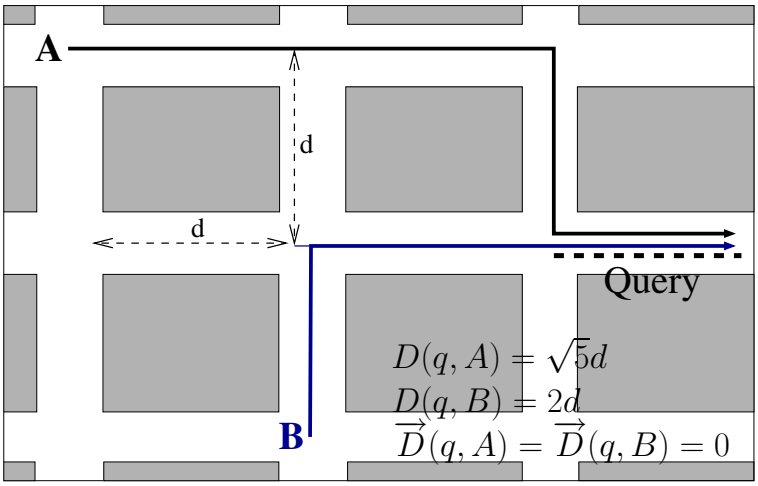


Figure 4.5: *One-way* distance

depicts the pruning condition for the subpattern matching. If the *one-way* distance $\overrightarrow{D}(T_q, rp(O))$ between a query trajectory and the representative trajectory $rp(O)$ of a non-leaf node O, is bigger than $\epsilon + s(O)$, all the trajectories in the sub tree of O can be safely pruned out, as we will formally prove below. Note that $s(O)$ is a *two-way* distance between $rp(O)$ and the farthest trajectory ($T_{farthest}$) belongs to the sub tree of O, and the inequality $D(T_{farthest}, rp(O)) \geq \overrightarrow{D}(T_{farthest}, rp(O))$ holds by Equation 3.3.

Theorem 3 (Pruning condition for subpatterns)**.** *Given a query trajectory* T_q *and a user-specified threshold* ϵ*, no trajectory* T *in* O *satisfies* $\overrightarrow{D}(T_q, T) \leq e$*, if* $\overrightarrow{D}(T_q, rp(O)) > s(O) + \epsilon$.

Proof. If $\overrightarrow{D}(T_q, rp(O)) > s(O) + \epsilon$, we want to show the following inequality holds.

$$\overrightarrow{D}(T_q, T) > \epsilon$$

Due to the fact that the *one-way* distance satisfies the triangular inequality,

$$\overrightarrow{D}(T_q, rp(O)) \leq \overrightarrow{D}(T_q, T) + \overrightarrow{D}(T, rp(O))$$

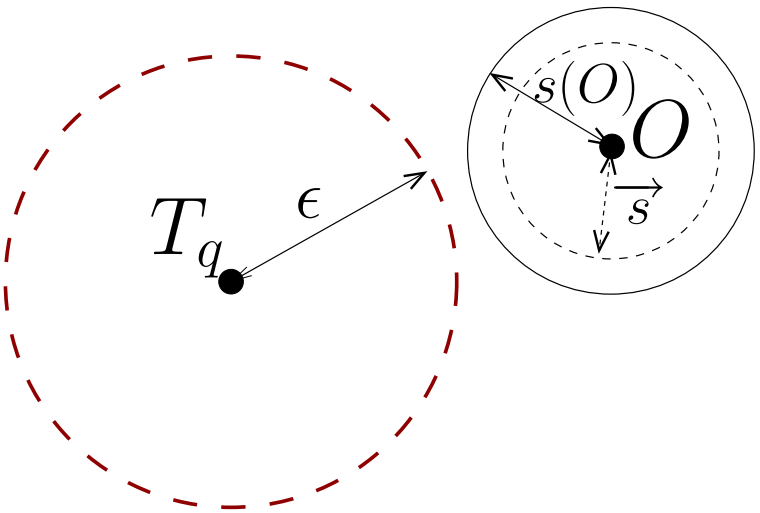


Figure 4.6: Pruning for subpattern matching

$$\begin{aligned}
\overrightarrow{D}(T_q, T) &\geq \overrightarrow{D}(T_q, rp(O)) - \overrightarrow{D}(T, rp(O)) \\
&\geq \overrightarrow{D}(T_q, rp(O)) - s(O) \ (\because \overrightarrow{D}(T, rp(O)) \leq s(O)) \\
&> \epsilon \\
\therefore \overrightarrow{D}(T_q, rp(O)) &> s(O) + \epsilon \longrightarrow \overrightarrow{D}(T_q, T) > \epsilon
\end{aligned}$$

□

Based on the above proof, KNN and range search algorithms for subpattern matching can be straightforwardly implemented by replacing (a) distance metrics and (b) pruning conditions to use one-way distance. More specifically:

- All instances of distance function $D(T_q, rp(o_i))$ and $D(T_q, o_i)$ in Algorithm 1 and 2 now refer to $\overrightarrow{D}(T_q, rp(o_i))$ and $\overrightarrow{D}(T_q, o_i)$, respectively.
- Pruning condition is changed from Theorem 2 to Theorem 3 (occurring twice in line 5 of Algorithm 1 and line 9 of Algorithm 2 respectively)

4.3.3 Reverse subpattern matching

Reverse subpattern matching is used when the database contains short trajectories and the users want to know which trajectories are included in the query trajectory.

For this matching, another one-way distance $\overrightarrow{D}(T, T_q)$ can capture the notion well, by identifying the closest query segment to trajectory data, instead of $\overrightarrow{D}(T_q, T)$ used for subpattern matching.

In other words, we can reuse the index structure built for whole pattern by a simple modification of range and KNN search algorithms. For reverse subpattern matching, we replace the distance metrics and pruning conditions from Algorithm 1 and 2 as follows:

- All instances of distance function $D(T_q, rp(o_i))$ and $D(T_q, o_i)$ in Algorithm 1 and 2 now refer to $\overrightarrow{D}(rp(o_i), T_q)$ and $\overrightarrow{D}(o_i, T_q)$, respectively.
- Pruning condition is changed from Theorem 2 to Theorem 4 (occurring twice in line 5 of Algorithm 1 and line 9 of Algorithm 2 respectively)

Theorem 4 (Pruning condition for the reverse subpatterns)**.** *Given a query trajectory T_q and a user-specified threshold ϵ, if $\overrightarrow{D}(rp(O), T_q) > s(O) + \epsilon$, there is no trajectory T in O satisfies $\overrightarrow{D}(T, T_q) \leq \epsilon$.*

Proof. The proof of Theorem 4 is similar to the proof for Theorem 3. If $\overrightarrow{D}(rp(O), T_q) > s(O) + \epsilon$, we want to show the following inequality holds.

$$\overrightarrow{D}(T, T_q) > \epsilon$$

From the fact that the *one-way* distance satisfies the triangular inequality, the above inequality can be proved.

$$\overrightarrow{D}(rp(O), T_q) \leq \overrightarrow{D}(rp(O), T) + \overrightarrow{D}(T, T_q)$$

$$\begin{aligned}\overrightarrow{D}(T, T_q) &\geq \overrightarrow{D}(rp(O), T_q) - \overrightarrow{D}(rp(O), T) \\ &\geq \overrightarrow{D}(rp(O), T_q) - s(O) \ (\because \overrightarrow{D}(rp(O), T) \leq s(O)) \\ &> \epsilon \\ \therefore \overrightarrow{D}(rp(O), T_q) > s(O) + \epsilon &\longrightarrow \overrightarrow{D}(T, T_q) > \epsilon\end{aligned}$$

□

Algorithm 1: Range search algorithm (for whole pattern matching)

Input: O: node, T_q: query trajectory, ϵ: search_radius

Output: L: set of trajectories

```
1  L ← ∅;
2  O ← Root_node;
3  if O is not a leaf node then
4      for ∀ o_i in O do
5          if D(T_q, rp(o_i)) ≤ ε + s(o_i) then
6              whole_range_search( node(o_i), T_q, ε);
7  else
8      for ∀ o_i in O do
9          if D(T_q, o_i) ≤ e then
10             add o_i to L;
```

Algorithm 2: KNN search algorithm (for whole pattern matching)

Input: O: node, T_q: query trajectory , k:knn

Output: NN: k nearest trajectories

```
1  PriorityQueue ← Root_node;
2  for i ← 0 to k do
3      NN[i] ← [_, ∞];
4  while PriorityQueue ≠ ∅ do
5      Next_node ← pop(PriorityQueue);
6      if Next_node is not a leaf node then
7          for o_i ∈ O do
8              if D(T_q, rp(o_i)) ≤ d_k + s(o_i) then
9                  add [o_i, D(T_q, o_i) − s(o_i)] to PriorityQueue;
10                 if D(T_q, o_i) + s(o_i) < d_k then
11                     d_k ←update_NN(o_i,D(T_q, o_i) + s(o_i));
12                     remove all entries whose distance < d_k from PriorityQueue;
13     else
14         for o_i ∈ O do
15             if D(T_q, o_i) ≤ d_k then
16                 d_k ← update_NN(o_i,D(T_q, o_i));
17                 remove all entries whose distance < d_k from PriorityQueue;
```

4.3.4 Spatial query

In this section, we discuss how to support common spatial queries, which are listed in [60], over trajectory data. More specifically, we present an efficient method to evaluate spatial queries, such as point, window, intersection, and enclosure query, using the three matching queries proposed in previous sections. Note, however, that what we propose here may not be the best way to process spatial queries for which there are other techniques with customized indexes that may outperform our method, such as [61, 8, 9, 10]. Rather we intend to demonstrate that our framework can also handle spatial queries as well without necessitating special indexes.

Point query

The point query finds all trajectories overlapping a given point p [60]. In our framework, the point query can be performed as follows. We find the nearest road segment r_p to p, which we use as a query trajectory and perform the range search (setting $\epsilon = 0$) of the subpattern matching for this query trajectory.

With slightly more restriction, this computation can be even faster– If the query point p is restricted to either the starting or ending point of each road segment, we can treat p itself as a road segment with the same starting and ending point and perform the range query of the subpattern matching (skipping to find the nearest road segment).

Window query

The point query is a special case of the window query, finding all trajectories intersecting with a d-dimensional interval which is denoted by Q_w. In our framework, the window query retrieves the trajectories that are either (1) enclosed or (2) intersected

by the query window.

The solution for the window query is not as straightforward as the one for point query– Unlike the query point, which can be converted into a query trajectory, it is not clear how to convert a query window into trajectories. We thus develop two approaches– *naïve* and *interfree*.

- **Naïve:** One naïve approach is to first identify all road segments enclosed and intersected by the region of Q_w, by querying Q_w on the R-tree of the road network. We then perform the range search (setting $\epsilon = 0$) of the subpattern matchings for all road segments intersected and enclosed by Q_w. Then, the union of the results from each query can be returned to the user.

  This naïve approach is easy to implement but has the following shortcomings. As the window size Q_w grows, the performance quickly deteriorates, as the number of enclosed and intersected road segments increases, as we will empirically show in Section 4.3.4. Another shortcoming is that the same trajectory is retrieved redundantly from multiple subpattern matching queries.

- **Interfree:** To address the above shortcomings, we propose an alternative approach, which combines subpattern and reverse subpattern matching. We call this approach *interfree*, as it significantly improves the naïve approach, by restricting pattern matching only to intersected road segments.

  More specifically, we distinguish the trajectories that are (1) enclosed and (2) intersected by Q_w– To illustrate, Fig. 4.7 contrasts the two types of trajectories, using two intersected trajectories Y and W and one enclosed trajectory Z. Note that these two types of trajectories are disjoint.

  We then process two types of trajectories separately and later merge the results. For the intersected trajectories, we can perform subpattern matching query for

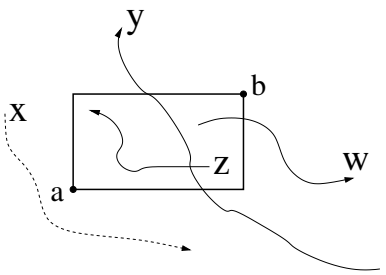
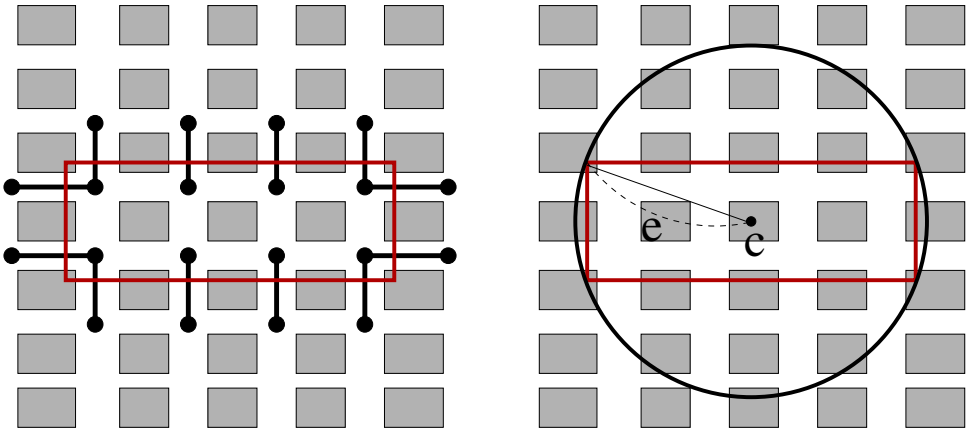


Figure 4.7: Example of the intersection and containment trajectory



(a) Intersected road segments (b) Circumcircle of the query window

Figure 4.8: Interfree approach: two types of queries for (a) the intersected trajectory and (b) the enclosed trajectory

each road segment, as we did in the naïve approach. However, such matchings are restricted only to the intersected road segments, as Fig. 4.8(a) illustrates.

Meanwhile, for the enclosed trajectory in Fig. 4.8(b), we perform the range search of the reverse subpattern matching at the center of the circle which circumscribes Q_w. Note that all road segments of the result trajectories are within e (the radius of the circumcircle) distance from c (the center of the circumcircle), as the reverse subpattern matching retrieves the trajectory T

such that the one-way distance, $\overrightarrow{D}(T, c)$ is less than e. During the computation of the one-way distance, $\overrightarrow{D}(T, c)$, the road segment distance (Equation 3.1) between t_i and c is the longest of $\|t_{i,s}, c\|$ and $\|t_{i,e}, c\|$. By the definition of the one-way distance, $\overrightarrow{D}(T, c)$ takes the maximum distance among the road segment distances (the point c is regarded as one road segment whose starting and ending positions are equal). Therefore all the road segments of the result trajectory are within e distance from c.

However, these results may include the trajectories that contain the road segments outside of Q_w. We thus need a post-filtering phase, to filter out such trajectories, by comparing Q_w with the MBR covering T and pruning out T from the result, if Q_w does not fully cover the MBR of T.

There is still room for further optimization of the *interfree* approach. In case of intersected trajectories, we can process the entire set of intersected road segments all together in a batch mode to avoid repeated accesses to the same page. In case of enclosed trajectories, it would be problematic if the query window is very elongated either horizontally or vertically, because the corresponding circumcircle then could be far larger than the window itself, which may lead to an excessive number of false alarms. To remedy this, we propose to divide the query window into smaller, yet more square-like fragments and process multiple reverse subpattern matching queries caused by the partitioning in a batch mode just as in the case of intersected trajectories. The exact derivation of the optimal partitioning strategy is in itself an independent research topic and beyond the scope of this dissertation, hence we leave it as future research.

For the optimal partitioning of the query window we define a cost function, C_{enc}.

Definition 5 (Cost function for the optimal partitioning)**.** *For a given query window*

whose width is x and height is y, assuming that y is larger than x, C_{enc} is defined as follows.

$$C_{enc} = \sum_{i=1}^{n} \pi \left(\left(\frac{x}{2}\right)^2 + \left(\frac{p_i}{2}\right)^2 \right) - xy \tag{4.1}$$

where n (≥ 1) is the number of partitions and p_i is a width of each partition.

Note that C_{enc} is the difference between the sum of circumcircle's area for each partition and the area of a given query window.

To find the optimal value of n and p_i which minimize the cost function, we first fix n and calculate the optimal value of p_i. We let $p_i = a_i y$, where $0 < a_i \leq 1$ and $\sum_{j=1}^{n} a_i = 1$.

$$\begin{aligned} C_{enc} &= \sum_{i=1}^{n} \pi \left(\left(\frac{x}{2}\right)^2 + \left(\frac{a_i y}{2}\right)^2 \right) - xy \\ &= \frac{\pi}{4} \sum_{i=1}^{n} x^2 + \frac{\pi y^2}{4} \sum_{i=1}^{n} a_i^2 - xy \end{aligned}$$

Because $\frac{\pi}{4} \sum_{i=1}^{n} x^2$ and xy are constant, we need to mizimize $\frac{\pi y^2}{4} \sum_{i=1}^{n} a_i^2$. For any n, this formula has the minimum value when a_i is $\frac{1}{n}$.

Now we substitute $\frac{y}{n}$ for p_i of Equation 4.1.

$$\begin{aligned} C_{enc} &= \sum_{i=1}^{n} \pi \left(\left(\frac{x}{2}\right)^2 + \left(\frac{y}{2n}\right)^2 \right) - xy \\ &= \frac{n\pi}{4} x^2 + \frac{\pi}{4n} y^2 - xy \end{aligned} \tag{4.2}$$

We obtain n which minimize the cost function by differentiating Equation 4.2.

$$\begin{aligned} \frac{d}{dn} C_{enc} = \frac{\pi x^2}{4} + \frac{\pi y^2}{4n^2} &= 0 \\ \frac{y^2}{n^2} &= x^2 \\ n &= \frac{y}{x} \end{aligned}$$

Therefore, p_i (the width of divided window) is x. That is, the cost function is minimized when each partition is a square.

Intersection and enclosure query

Finally, we further generalize the window query, to support intersection and enclosure query for query windows of arbitrary shapes. That is, given a query window O_q of arbitrary shape, the intersection query finds all objects having at least one point in common with O_q and the enclosure query finds objects enclosing O_q [60], while the window query restricts the query object to the iso-oriented rectangle.

The simplest case is when the query object O_q is a trajectory– In such a case, the enclosure query is equivalent to range search ($\epsilon = 0$) of the reverse subpattern matching and the intersection query is reduced to the query of finding all trajectories having at least one *road segment* in common with a query trajectory. For such a query, we can perform the range search ($\epsilon = 0$) of subpattern matching for each road segment of a query trajectory, then union the results of subpattern matchings.

In the general case when query object O_q may not be a trajectory, we convert a query object with arbitrary shapes, *e.g.* polygon, circle and so on, to the MBR covering the query object. After we perform the window query with the MBR as a query window, each of the result trajectories is then checked with O_q to prune out the results that are not intersected (for the intersection query) or enclosed (for the enclosure query) with the query object, as post-filtering.

4.4 Experimental evaluation

In this section, we report the results of experiments conducted to verify the effectiveness and efficiency of our proposed framework. We begin by describing experimental settings. We used real trajectory data [62] to show the quality of the matching trajectories. There are 213 trajectories, each of which is a sequence of positions received by GPS in the Cook and Dupage county of Illinois. Additionally, small random noises were added to the original data to demonstrate the robustness of the proposed measure. We observe that there are three "ground truth" groups in the real trajectory data and the distance between inter-group members are larger than the distance between intra-group members. These trajectories were then transformed to sequences of road segments based on the method described in Section 3.2, with length ranging from 8 to 742. We obtained the road network data of the two counties in a TIGER/LINE format from the U.S. Census Bureau [63] consisting of 201, 540 road segments in the road network. All experiments were conducted on a machine with a Pentium-4 CPU (3.2GHz) and 1 GBytes of main memory, running a version of the Linux operating system. We implemented the range and KNN search algorithms for each pattern matching query on top of the M-tree.

We now move on to discuss the result of three pattern matching queries. In the dataset, we observed that 93 trajectories share similar patterns, of which we randomly picked one as a query trajectory for both whole and reverse subpattern matching (see Figure 4.9(a)). It consists of 444 road segments, of which we also randomly extracted 49 consecutive road segments as a query trajectory for the subpattern matching (see Figure 4.9(b)).

Using the query trajectores from above, we performed all three types of pattern

Rank	1 ~ 79	81	87	96	97
Whole					
Rank	1 ~ 8	9	29	36	44
Sub					
Rank	1	2	13	47	51
Reverse Sub					

Table 4.1: Visualization of trajectories for each pattern matching

(a) Whole and reverse sub (b) Sub

Figure 4.9: Query trajectories

matchings and visualize the results in the order of similarity to the query for each type of matching. In Fig. 4.1, a thick pattern indicates a query trajectory and a thin pattern indicates a matching trajectory. From the whole pattern matching results, we can observe that the top 93 results are the trajectories that either exactly match the entire query or the majority of road segments. The results ranked $81th$ and $87th$ share the majority of road segments of the query, with minor deviations in the middle. Recall that, there exist three ground-truth clusters in the real-life set. The top 93 results belong to one such cluster (travelling across counties), and the $96th$ and $97th$ results represent the other two clusters (travelling within a county), respectively. Because of this clustered nature, top 93 results are highly similar to the query, while the similarity of lower ranked results deviate rather dramatically.

Top results from subpattern matching include longer trajectories that subsume the query pattern. Note that as this query pattern is a part of the query used for the whole pattern matching, most of them overlap with those from the whole matching. We thus highlight only the trajectories that do not overlap in Fig. 4.1, such as the trajectories with rank 9, 29, and 44 in the figure.

Lastly, in reverse subpattern matching, top results include the exact match and short trajectories subsumed by the query trajectory. The top-most trajectory is the query trajectory itself, as the query trajectory was selected from the trajectory

Table 4.2: Quality comparison with other measures

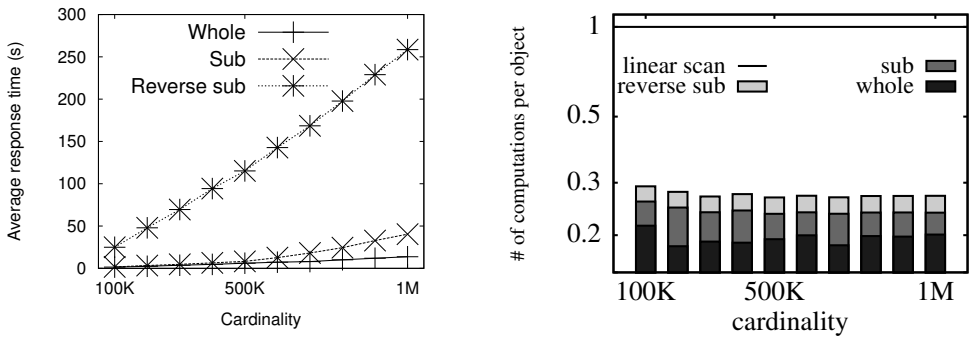
Groups (# of traj.)	Num. of Correctly Classified Trajectories (Recall %)			
	EDR [17]	ERP [18]	OHD [13]	Proposed
G1 (96)	70 (73%)	70 (73%)	87 (90%)	**96 (100%)**
G2 (24)	4 (17%)	8 (33%)	3 (13%)	**24 (100%)**
G3 (93)	49 (53%)	60 (65%)	67 (72%)	**93 (100%)**

database. In Fig. 4.1, we highlight the results that do not overlap with other query results.

Summing up, we can observe from the result trajectories of the three queries in Fig. 4.1 that the proposed distance metric well reflects the semantics of each pattern matching query.

We now move on to discuss the effectiveness of our distance measure comparing with other measures using real-life road network trajectories [62]. As trajectories in this set clearly fall into three groups, we use such groups as "ground truth" results and compare the recall of all metrics in Table 4.2. Specifically, for each group of size m, we select one trajectory in the group as query and compare the top-m results from each metric. When using our metric, we identify all trajectories in the group and nothing else (*i.e.*, perfect precision/recall), while the recall of other metric varies from 17% to 90%.

We also empirically study whether our proposed metric is in any way redundant to existing metrics, by comparing the rank correlations between measures using widely-adopted Kendall's tau coefficient [64]. Specifically, using one trajectory from each group as a query, we rank the whole trajectories by similarity and measure the rank similarities. If our proposed metric correlates perfectly with the existing metric, Kendall's tau coefficient is 1, suggesting our metric would be redundant. Meanwhile,



(a) Average response time (b) Distance computations

Figure 4.10: Performance of pattern matching queries

such coefficient ranges were rather low, from −0.004 (in EDR) to 0.44 (in OHD).

We next turn to the performance aspects of the proposed method. One key performance parameter in the use of M-tree in our framework is the node size– as the page size increases, the number of I/O decreases and the performance thus improves up to a certain point, but as the CPU cost increases over a larger page size, the performance eventually starts to worsen after a trade-off point. To find it, we conducted additional experiments in which we observed that the performance significantly improves when the page size increases from 4 to 32 Kbytes, but plateaus after 32 (see Figure 4.11 (a) and (b)). Similarly in Figure 4.11 (c), we can observe that the response time decreases up to 32 Kbytes and starts to increase from the point. We thus set the node size as 32 Kbytes in subsequent experiments.)

Next, we also validated the scalability of the proposed method over a synthetic trajectory datasets, which are generated by the network-based data generator in [65], in various experimental settings, varying the retrieval size, the length of the trajectory and the cardinality of the database. Naturally, the performance of our proposed method depends on the cost for searching M-tree, as we adopt search methods of M-tree for efficient query processing.

Our reported response time is the average of 100 synthetically generated queries, which are generated using the trajectory data generator. In particular, when generating query trajectories, we set the length of query trajectory equal to that of the trajectories in the database for the whole pattern matching, 0.1 times for the subpattern matching, and 2 times for the reverse subpattern matching queries in Fig. 4.12, 4.13, and 4.14.

Fig. 4.12 reports the response time over varying k. Observe that, in all three types of pattern matching, our proposed framework significantly outperforms the linear scan. For instance, in the whole pattern matching, our proposed method is 60 times faster than the linear scan.

Fig. 4.13 shows the average response over varying query trajectory length. Observe that, the speedup, a ratio of average response time of linear scan to that of the proposed method, increases as the length grows. In the case of whole pattern matching, the speedup is 55 when the length is 50, while the speedups for sub and reverse subpattern matching are about 5.2 and 15.5, respectively.

Fig. 4.14 shows the scalability of pattern matching over varying cardinality of the trajectory database. Similarly, our proposed method consistently outperforms the linear scan in all three matchings. The speedup of our proposed method is up to 69, 5.2, and 14.6 over the linear scan in the whole, subpattern and reverse subpattern matching respectively.

Finally, to demonstrate the pruning effect of Theorem 2, 3, and 4, we report the distance computation ratio per trajectory in Fig. 4.10. The ratio of the number of distance computation required by our proposed framework to that of linear scan is 0.2, 0.23, and 0.27 for the whole, subpattern and reverse subpattern matching respectively. Note that the ratio is the lowest for the whole pattern matching, as its pruning condition using the two-way distance is relatively stricter compared to the

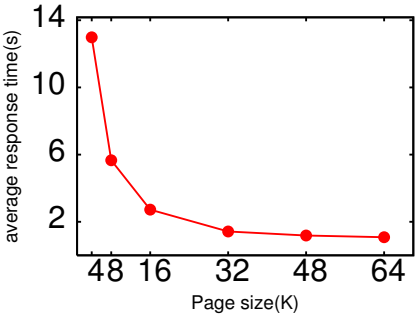
other matchings using only the one-way distance.

We now discuss the performance of spatial query in our proposed framework. We only investigate the performance of the window query in terms of the average response time. This is because the point query is a special case of the window query and the performance of other spatial queries, such as the intersection and the enclosure query, is affected by that of the window query.

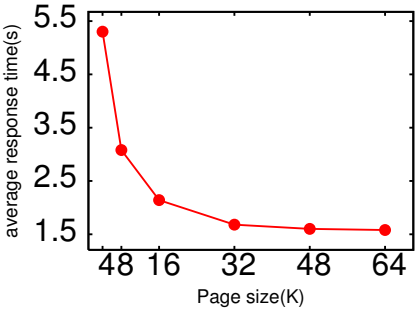
We randomly generate 50 window queries, by generating 50 points to use as a upper-left corner, based on which we compute a lower-right corner by adding $\sqrt{f}W$ to each axis, when f is the parameter controlling the area of the query window and W controlling the side of the query window. The default value for W is 150 (meter).

We compare our two proposed approaches in Section 4.3.4: the naïve approach and the *interfree* approach. Recall that, the *interfree* enhances the naïve approach performing the subpattern matchings for every road segment intersected with a query window. As a result, unlike the naïve approach incurring proportional cost to the number of road segments that are either (1) intersected and (2) enclosed, our proposed *interfree* approach reduces the number of such matchings, by performing one reverse subpattern matching for all the enclosed segments. Its cost is proportional only to the intersected segments.

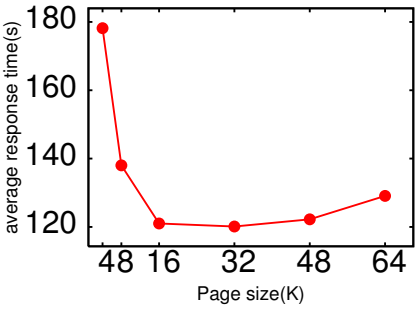
To compare the two approaches empirically, we plot average response time of the two over varying window size f in Fig. 4.15. Observe that, our proposed approach *interfree* outperforms the naïve approach and the performance gap only increases as the query window size grows. Note, this observation is consistent with our analysis– The cost of *interfree* and naïve approach is proportional to the number of (1) intersected and (2) intersected plus enclosed segments, which changes over query window size as Fig. 4.16 illustrates with the average of 50 window queries, which explains similar cost patterns observed in Fig. 4.15.



(a) whole

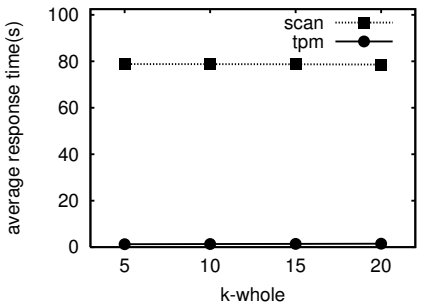


(b) sub

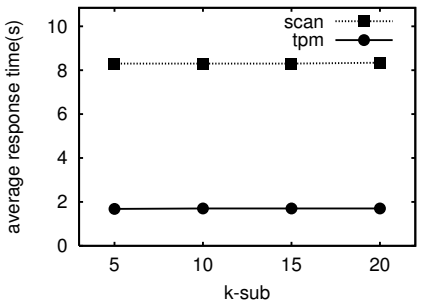


(c) reverse sub

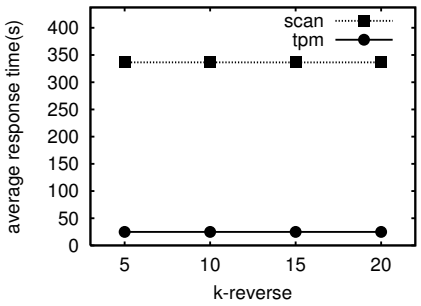
Figure 4.11: Average response time for different node sizes (default setting)



(a) whole

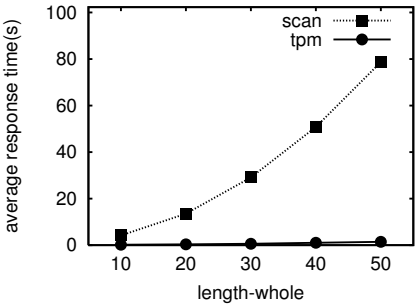


(b) sub

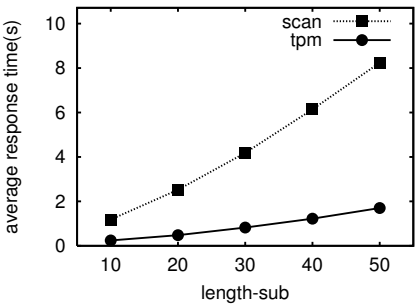


(c) Reverse sub

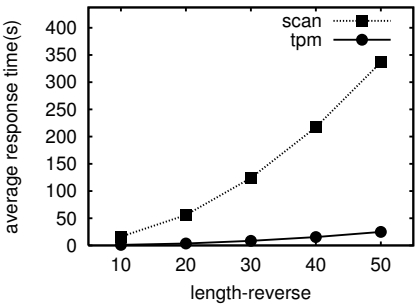
Figure 4.12: Time vs. k (length=50 size=100K)



(a) whole

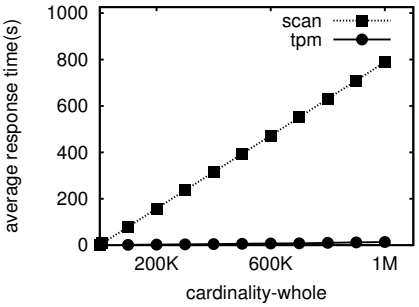


(b) sub

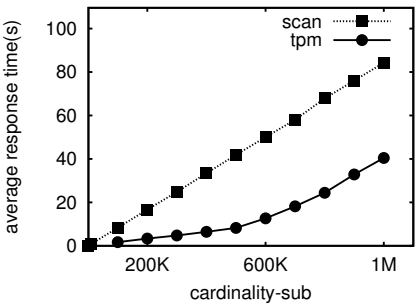


(c) Reverse sub

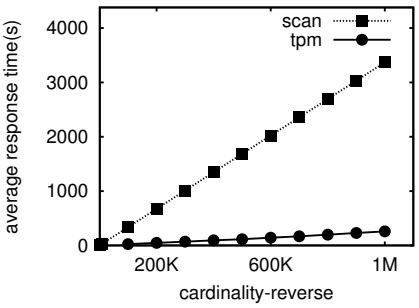
Figure 4.13: Time vs. *length* (k=15 size=100K)



(a) whole



(b) sub



(c) Reverse sub

Figure 4.14: Time and # of distance computations vs. *cardinality* (k=15 length=50)

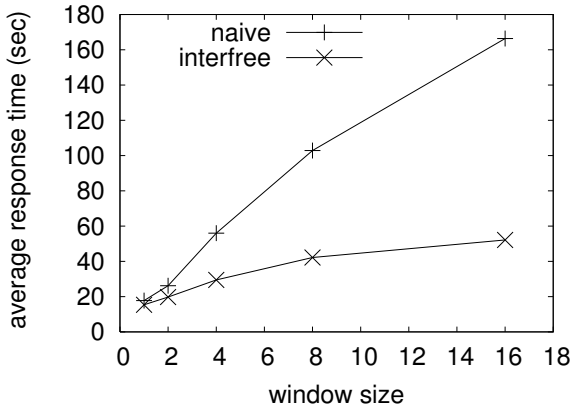


Figure 4.15: Average response time for different window sizes

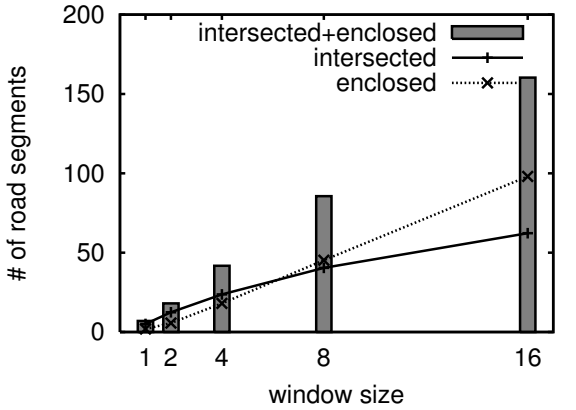


Figure 4.16: The number of intersected/enclosed road segments for different window size

4.5 Summary

In this chapter, we studied how to support pattern matching queries for trajectories constrained by road networks. Toward this goal, we first investigate the requirements for the trajectory representation and distance measure for our target problem. As none of the existing work fully satisfies the requirements identified, we then devise the trajectory representation and the distance measure, which fulfill all the requirements.

More specifically, we study the three pattern matching queries (whole, subpattern, and reverse subpattern matching) to search for similar trajectories to the given query trajectory. Though the notion of *similarity* varies across different types of queries, we proposed a unified framework efficiently supporting range and KNN queries for all three types of matching based on M-tree and pruning rules. We validated the quality of results by visualizing the results for different types of queries over real-life road network trajectories. Comparison with other distance measures showed that the proposed distance measure is more appropriate for road network trajectories.

There are also opportunities for future research. One opportunity is to study a specialized index structure for the reverse subpattern matching, which is inherently more complex compared to other matchings, as our M-tree based framework, though achieving the speed up of 14.6 over the baseline approach, still incurs high computational cost.

5

Clustering network-constrained trajectories

In this chapter, we study how to develop an efficient clustering algorithm for network-constrained trajectories. Clustering trajectories can be used to identify distinct groups in which trajectories have more similar moving patterns than those in other groups. To support semantic context criteria as we discussed in chapter 1, this work focuses on clustering the vehicle trajectories of users on road networks. While there has been prior research work done on clustering trajectories, none of them considered the movement constraints imposed by the underlying road networks– Two locations that are close based on Euclidean distance may not be reachable over road networks, if no road exists between the two locations. In addition, we propose an efficient method to support "Boolean+clustering" queries. For efficient support, the proposed method minimizes online computation requirements using a distance approximation and a pre-computed data summary structure.

5.1 Problem definition

In this section, we formally define our problem. We model a road network as a graph, where an edge is a road segment and a vertex is an intersection of adjacent segments, as we formally state below:

Definition 6 (Road network)**.** *A road network RN is defined as a graph* $G_{RN} = (V, E)$*, where* V *is a set of intersections of the road network, and* E *is a set of road segments* $r_i \in E$ *such that*

$$r_i = (r_{i,s}, r_{i,e}),$$

where $r_{i,s}, r_{i,e} \in V$ *and there exists an road between* $r_{i,s}$ *and* $r_{i,e}$*.*

Trajectories on road network can thus be represented by a set of connected road segments, or edges in the above-mentioned graph. Matching GPS logs to road segments have been studied in existing map matching literature [66].

Definition 7 (Trajectory)**.** *Given a road network* $G_{RN} = (V, E)$*, a trajectory of length* l *is defined as*

$$TR_i = \left[t_1^i, \ldots, t_l^i\right],$$

where $t_j^i \in E$*,* $1 \leq j \leq l$*, and* t_j^i *and* t_{j+1}^i *are connected.*

We now formally define our problem, of finding a set of disjoint clusters $\mathbf{C} = \{c_1, \ldots, c_n\}$, which satisfies the given Boolean condition set B. Boolean condition set B can be a complex Boolean condition such as conjunctions and disjunctions of sub-conditions.

Definition 8 (Problem statement)**.** *Given input trajectories* T *and Boolean selection condition set* B*, a cluster* $c_i \in \mathbf{C}$ *is a set of trajectories in* T *such that*

1. $c_i \neq \varnothing$*,* $i = 1, \ldots, n$;

2. $\bigcup_{i=1}^{n} c_i = \mathbf{C}$;

3. $c_i \cap c_i = \varnothing$, $i, j = 1, \ldots, n$ *and* $i \neq j$.

Our goal is to find $\mathbf{C_{opt}}$ *maximizing inter-cluster distance* D_{inter}*, i.e.,*

$$\mathbf{C_{opt}} = \arg\max_{\mathbf{C^l}} \frac{1}{|C^l|} \sum_i \sum_j D_{inter}(c_i, c_j),$$

where any trajectory $TR \in \forall c_i$ *satisfies Boolean conditions in* B.

5.2 Clustering algorithm

In this section, we present two clustering algorithms– NNCluster and SemCluster for location and semantic-rich network trajectories respectively.

In particular, we first present the common skeleton for the two algorithms in Section 5.2.1, with two key components– *InitialCluster* and *MergeIC*. Section 5.2.2 and 5.2.3 discuss how these components are implemented differently for NNCluster and SemCluster respectively.

5.2.1 Overview of NNCluster and SemCluster

This section presents the overview of NNCluster and SemCluster for location and semantic-rich trajectories respectively.

Figure 5.1 illustrates how our algorithms build upon two key components– *Initial-Cluster* and *MergeIC*. *InitialCluster* materializes the initial groupings of trajectories into mini-clusters. (Hereafter, we call this initial group as *initial cluster*.) Based on this structure, clustering algorithms perform *MergeIC*, which iteratively merges initial clusters into the meaningful result clusters.

A key difference of SemCluster, though sharing the same *InitialCluster* module, is to call *InitialCluster* module offline and pre-materialize them into structures in

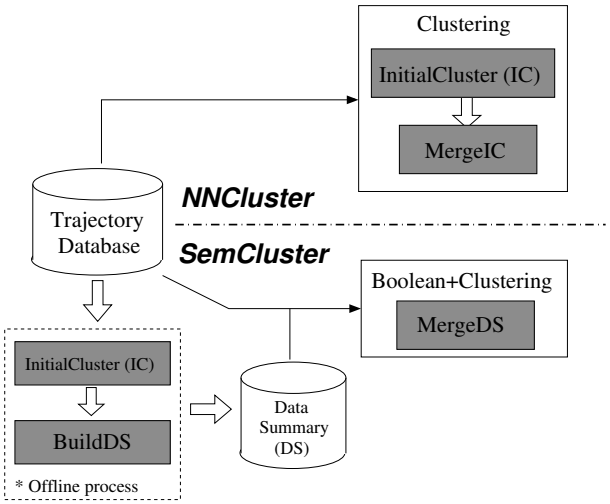


Figure 5.1: Overview of NNCluster and SemCluster

BuildDS. *MergeDS* module then integrates the given Boolean condition for the merge process, as we will discuss in the following sections.

5.2.2 NNCluster

We first discuss how we implement *InitialCluster* and *MergeIC* for location trajectories, in the following two sections respectively.

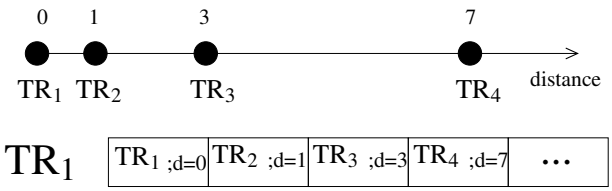
InitialCluster

As the first step, we group the trajectories into initial clusters that are likely to belong to the same cluster. To decide which trajectories to put into the same initial cluster, we identify a set of trajectories sharing the common nearest neighbor, widely adopted in spatial structures such as Voronoi diagram [67]. This problem can be abstracted as a graph problem, by denoting each object as a node, connected to another node that is the nearest neighbor. Given this graph, building the initial clusters can be reduced to listing all maximal cliques, which is known as NP-complete problems [68].

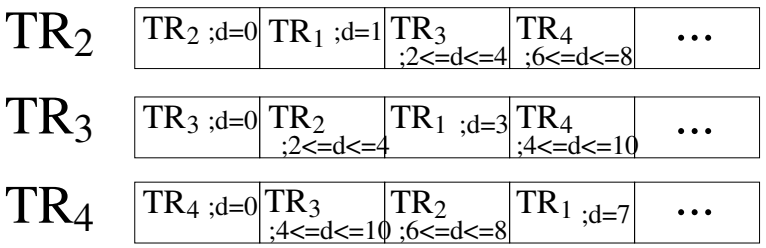
Our goal is to approximate this process, by relaxing the notion of initial clusters, as those sharing many k-nearest neighbors. To efficiently construct initial clusters, we compare k-nearest neighbors of each trajectory with those of another trajectory. The two trajectories are then merged into a cluster if the ratio between the common nearest neighbors is greater than a user-defined threshold value.

More specifically, we repeat the following processes until no further merging is possible. To reduce the distance computation, at each iteration, we randomly select a *seed* trajectory TR_{seed} among trajectories that have yet to be assigned to any initial cluster and compute a sorted list for TR_{seed}, which contains the distances between TR_{seed} and all of the other trajectories in the ascending order. Figure 5.2(a) shows an example of such a sorted list for TR_{seed}, where TR_1 is selected as a *seed* trajectory. From the top trajectory in the list, *e.g.*, TR_2 in Figure 5.2(a), we proceed to test whether the ratio of common nearest neighbors of TR_2 and TR_{seed} is greater than the user-defined threshold value. To find the nearest neighbors of TR_2, we need to compute a sorted list for TR_2 as TR_{seed}. By building exact sorted lists only for seed trajectories, we can avoid the prohibitive cost of building exact lists for every trajectory.

For the remaining "non-seed" trajectories, instead of building exact sorted lists, we generate approximate lists that are based on the exact lists of seed trajectories. This suggests that once we compute the *exact* sorted list for a *seed* trajectory, we can then estimate the sorted lists of the remaining non-seed ones by approximating the distance based on the *exact* sorted lists on the seed trajectories. Hereafter we call an approximated sorted list a *derived* sorted list. For *derived* sorted lists, each element of the list has a lower bound distance and an upper bound distance. Trajectories are sorted first by the lower bound and the ties in lower bound (if they exist) are broken by upper bounds.



(a) Exact sorted list for TR_1



(b) *Derived* sorted lists

Figure 5.2: Example of approximating sorted list (TR_1 is a seed trajectory)

For example, we can illustrate how to approximate the distance between TR_2 and TR_3, when we already have the sorted lists for TR_1 as illustrated in Figure 5.2(a). Figure 5.2(b) shows *derived* sorted lists for TR_2, TR_3, and TR_4 after we compute the exact sorted list for the trajectory TR_1.

As our distance measure satisfies the property of triangle inequality, the following two inequalities are also satisfied.

$$D(TR_2, TR_3) \leq D(TR_1, TR_2) + D(TR_1, TR_3) \tag{5.1}$$

$$D(TR_1, TR_3) \leq D(TR_1, TR_2) + D(TR_2, TR_3) \tag{5.2}$$

By using the above two inequalities, we can compute the lower and upper bound of $D(TR_2, TR_3)$ as follows:

$$\begin{aligned} & D(TR_1, TR_3) - D(TR_1, TR_2) \\ & \leq D(TR_2, TR_3) \\ & \leq D(TR_1, TR_2) + D(TR_1, TR_3). \end{aligned} \tag{5.3}$$

We stress that the above property holds for any distance metric satisfying triangular inequality, which makes our optimization applicable to any such distance metric. The same bound was leveraged in different application scenarios: [69] exploits the triangle inequality to accelerate k-means clustering and [70] uses the triangle inequality to build a specialized index structure for the fast retrieval of k neighbors. However, these works cannot be used for our problem: First work requires the number of clusters k which is often unknown. Second work requires to build an index structure, which is infeasible for our online scenario.

After a *derived* sorted list is established for a *non-seed* trajectory TR_i, we can compare k-nearest trajectories of TR_i with those of TR_{seed}. If the ratio of the common trajectories to the union of the two sets of k-nearest neighbors is greater than the user-defined threshold, we can merge TR_i with TR_{seed}.

Algorithm 3 outlines the algorithm in constructing initial clusters. It selects a *seed* trajectory TR_{seed} from non-visited trajectories (line 5) and then calculates the sorted list for TR_{seed} (lines 8–10). For non-visited trajectories, *derived* sorted lists are generated using Equation 5.3 (line 14) and then its neighbors (NN_{TR_i}) are compared with neighbor trajectories of TR_{seed} (line 16). If there is no trajectory left to visit, the algorithm return the result of clustering and terminates.

Merge InitialCluster

Once the initial clusters are materialized, clustering algorithms iteratively merges them into the right number of clusters. This section discusses the two challenges of (1) how exactly we merge and (2) how we find the right number of clusters deciding when to terminate.

For the first challenge of *merging*, as merging these initial clusters into disjoint clusters maximizing inter-cluster distance D_{inter} is known to be NP-hard [71], we

Algorithm 3: $InitialCluster(M, k, D)$

input : T: a set of trajectories, k: # of nearest neighbors,
D: distance function

output: C: a set of clusters

```
1  begin
2      C ⟵ ∅;
3      Visit ⟵ ∅;
4      while |Visit| < |T| do
5          seed ← randomly selected from T such that seed ∉ Visit;
6          Visit ← Visit ∪ seed;
7          L ← ∅;
8          foreach t ∈ T do
9              L.insert(t, D(seed, t));
10         SL ← sort(L);
11         NN_seed ← nnSearch(k, SL);
12         C_cur ← seed;
13         foreach t ∈ NN_seed do
14             SL_t ← sort(approximateList(SL, t));
15             NN_t ← nnSearch(k, SL_t);
16             if |NN_seed ∩ NN_t|/|NN_seed ∪ NN_t| ≥ ε then
17                 C_cur ← C_cur ∪ t;
18         C ← C ∪ C_cur;
19     return C
```

develop a greedy approach, which continues merging a pair of clusters in the ascending order of the inter-cluster distance until a stop condition is satisfied.

There are two challenges in this step: *First* we need to define the distance measure between clusters. *Second*, a stop condition is required to determine when the expansion of the initial cluster terminates.

To address the fist challenge, we define the distance between clusters as the distance between representative trajectories of clusters. As a representative trajectory, we choose the trajectory that minimizes the average distance between the trajectories in a cluster. We give a formal definition of the representative trajectory as follows:

Definition 9 (Representative trajectory)**.**
Given cluster $c_i = \{TR_1, \ldots, TR_m\}$*, a representative trajectory,* $rt(c_i)$*, is defined by*

$$rt(c_i) = \arg\min_{TR_i \in c_i} \frac{1}{m} \sum_{j=1}^{m} D(TR_i, TR_j). \tag{5.4}$$

We define the distance between two clusters as follows:

Definition 10 (Inter-cluster distance)**.** *Given two clusters,* c_i *and* c_j*, the distance between them is defined by*

$$D_{inter}(c_i, c_j) = D(rt(c_i), rt(c_j)), \tag{5.5}$$

where $rt(\cdot)$ *is the representative trajectory of a cluster and* $D(\cdot, \cdot)$ *is the trajectory distance (Equation 3.3).*

For the second challenge of deciding when to terminate, we first introduce quality measures which have been used to validate the clustering results. Then we present a method to exploit the quality measure as a stop condition for our clustering method.

Many quality measures (*e.g.*, Dunn Index, Silhouette Index, and Davies-Bouldin (DB) Index) have been introduced previously to address issues of this type[1].

[1]The interested readers may find an exhaustive study of clustering validation in [72].

These quality measures were originally developed to validate clustering results *after* a clustering process terminates. Therefore, the cost for computing a quality measure has not been considered in the context of their application. However, to apply these measures in order to invoke a termination condition, we need to consider the computational overhead. We thus choose the DB Index [73] from the range of quality measures available, as this measure can be computed based on the distances computed during clustering and does not require additional computation on the values. Given a set of cluster $\mathbf{C} = \{c_1, \ldots, c_n\}$, the DB index is defined as:

$$DB(\mathbf{C}) = \frac{1}{n} \sum_{i=1}^{n} R_i, \tag{5.6}$$

where $R_i = \max_{j=1,\ldots,nj \neq i} \frac{s_i + s_j}{d_{ij}}$, d_{ij} is the distance between cluster c_i and cluster c_j, and s_i is the variance of cluster c_i. The DB index approaches zero value if the clusters are more compact and well-separated.

We store the value of the DB index at each step when merging clusters. Then, we report the "l_{opt}-th" clustering result which minimizes the DB index, *i.e.*,

$$l_{opt} = \arg\min_{i \in [0,L]} DB(\mathbf{C}_i),$$

where there are L steps of merging. Note that we ignore the last step, because the DB index becomes 0 if all of the trajectories are assigned to a single cluster.

Algorithm 4 formally describes $MergeIC$, which continues to merge clusters until the current number of clusters becomes 1 (line 6). In each step of the while loop, it performs merging initial clusters. After while loop, the set of clusters having the minimum value of DB index is output as a result of clustering.

5.2.3 SemCluster

This section presents how to efficiently evaluate "Boolean+clustering" queries for given arbitrary Boolean conditions. As NNCluster is an online algorithm, it can be

Algorithm 4: $MergeIC$: merging initial clusters

input : IC: a set of initial clusters, k: # of nearest neighbors,
D: distance function

output: $C^{l_{opt}}$: a set of clusters

```
1  begin
2      l ← 0;
3      C^l ← IC;
4      foreach c_i ∈ |C^l do
5          c_i.r ← representative(c_i);                    /* Equation 5.4 */
6      C^l.db ← DB(C^l);                                   /* Equation 5.6 */
7      while |C^l| > 1 do
8          find the closest pair of clusters c_i, c_j ∈ C^l;
9          c_ij ← c_i ∪ c_j;
10         c_ij.r ← representative(c^ij);                  /* Equation 5.4 */
11         C^{l+1} ← {C^l − c_i − c_j} ∪ c_ij;
12         C^{l+1}.db ← DB(C^{l+1});                       /* Equation 5.6 */
13         l ← l + 1;
14     l_opt ← arg min_i C^i.db;
15     return C^{l_opt}
```

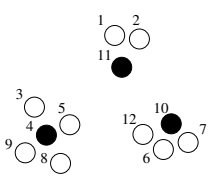
naively adopted for Boolean+clustering queries, by evaluating Boolean selection first and performing NNCluster only on the qualifying results. However, such naive adoption incurs high cost, especially when the selectivity of Boolean conditions is high.

To address this challenge, the underlying idea of SemCluster is to minimize the distance computations in runtime by performing *InitialCluster* offline and pre-materializing initial clusters into "data summary" using *BuildDS* module. In particular, we present (1) how we pre-materialize a data summary structure and (2) how we perform "Boolean+clustering" queries using the data summary structure, in the following two sections respectively.

BuildDS

This section describes how we build a data summary structure offline, which materializes the initial clusters that are returned by *InitialCluster*. In addition to the information of initial clusters, to efficiently execute *MergeDS*, we also store the inter-cluster distance between initial clusters in the data summary structure. We will discuss how we use the data summary structures for *MergeDS* in Section 5.2.3.

The data summary structure consists of two components: initial cluster information and distance relationship between initial clusters. Once initial clusters are identified using *InitialCluster* described in Section 5.2.2, we store the information of initial clusters, such as the representative trajectory of an initial cluster and the number of members, by computing the representative in each group using Equation 5.4. We choose a hash table to store the information of initial clusters for fast retrieval. More specifically, for each trajectory of an initial cluster, the key-value pair of $(tid, (refid, s))$ is inserted into a hash table, where tid is an identifier of trajectory, $refid$ is an identifier of representative trajectory, and s is the number of trajectories



Hash table

(tid, (refid, s))
1, (11,3)
2, (11,3)
3, (4,5)
4, (4,3)
5, (4,5)
6, (10,4)
7, (10,4)
8, (4,5)
9, (4,5)
10, (10,4)
11, (11,3)
12, (10,4)

Distance matrix

	4	7	11
4			
7			
11			

Figure 5.3: Example of data summary structure

that belong to the initial cluster represented by $refid$.

In addition, we also compute and store the distance relationship between all pairs of representative trajectories using Equation 5.5, which will reduce the number of the distance computations in runtime. If there are n initial clusters, we build a $n \times n$ distance matrix DM, where DM_{ij} contains inter-cluster distance between c_i and c_j.

Figure 5.3 illustrates a data summary structure for example trajectories. In Figure 5.3, there are 12 trajectories, each of which is represented as ◯ and their ID number, in three initial groups. We marked representative trajectories as solid circles. For example, the trajectory 1 belongs to the cluster that has three member trajectories and its representative is trajectory 11. Therefore, the key-value pair of $(1, (11, 3))$ is stored in the hash table.

Algorithm 5 describes the process of building a data summary. It first runs *InitialCluster* to obtain initial clusters for trajectories in database (line **??**) and then calculates the representative trajectories for each initial cluster (line 4). For each member trajectory of a initial cluster, we insert a key-value pair into the hash table (line 6). Lastly, we compute the distance between representative trajectories (line 7-9).

Algorithm 5: *BuildDS*: build the data summary

input : IC: a set of initial clusters returned by *InitialCluster*

D: distance function

output: $Hashtable$: a hash table of (tid, (refid,s)),

DM: a distance matrix between representatives

```
1  begin
2      Hashtable ← ∅;
3      foreach c_i ∈ IC do
4          c_i.r ← representative(c_i);
5          foreach TR_j ∈ c_i do
6              Hashtable, insert(TR_j.id, ((c_i.r).id, |c_i|));
7      for i ← 0 to |IC| do
8          for j ← 0 to |IC| do
9              DM_ij ← D(c_i.r, c_j.r);
10     return (Hashtable, DM)
```

MergeDS

This section presents how we use the above pre-materialized data summary to merge initial clusters into meaning groups.

For the given Boolean conditions, we first identify initial clusters for trajectories satisfying Boolean conditions by looking up the hash table of data summary. As a result of Boolean selection, a set of trajectory identifiers is obtained by searching the database. We then look up the hash table with each trajectory identifiers as a key. Using a hash table, we can efficiently identify initial clusters of trajectories that satisfy Boolean conditions.

During looking up the hash table, we maintain the information of member trajectories in each initial cluster for selection attributes, to efficiently quantify the number of trajectories satisfying the given condition. We will exploit this information to efficiently determine the representative trajectory.

We discuss how we merge initial clusters identified in the previous step. As mentioned in Section 5.2.2 our clustering problem, finding disjoint clusters maximizing inter-cluster distance D_{inter} is known to be NP-hard [71]. We also exploit the same approach for SemCluster, which merges a pair of clusters in the ascending order of the inter-distance between clusters until a stop condition is satisfied.

Once initial clusters are identified, we merge identified clusters in the ascending order of inter-cluster distance. Note that the inter-cluster distance is already computed and stored in the data summary structure, *i.e.*distance matrix. Therefore, SemCluster does not require any distance computation for determining the order of merging clusters. In each step of merging c_i and c_j, a new representative trajectory is elected between the representatives of c_i and c_j, *i.e.*, $c_i.r$ and $c_j.r$, by comparing the number of trajectories in c_i and c_j satisfying the given Boolean conditions. The

representative from the larger cluster will be promoted to be a new representative of $c_i \cup c_j$. Similarly to NNCluster, we store the value of the DB index at each merging step. Then, we report the clustering result that minimizes the DB index.

Function LoadDS(Hashtable,Tlist): find initial clusters from the hash table

input : $Hashtable$: a hash table,$Tlist$: a set of trajectories

output: C: a set of clusters

```
1  begin
2      C ← ∅;
3      foreach t_i ∈ Tlist do
4          (refid, s) = lookup(Hashtable, t_i.tid);
5          if c_i exists such that c_i.r == refid then
6              c_i ← c_i ∪ t_i;
7              c_i.count ← c_i.count + 1;
8          else
9              c_new ← t_i;
10             c_new.r ← refid;
11             c_new.origsize ← s;
12     foreach c_i ∈ C do
13         if c_i.count < c_i.origsize then
14             c_i.r ← representative(c_i);          /* Equation 5.4 */
```

Algorithm 6 and function *LoadDS* present the process of merging cluster using the data summary structure. We identify trajectories satisfying given Boolean conditions B, in a hash table (line 3). We then identify initial clusters from the data summary with *LoadDS* function. In function *LoadDS*, we identify initial clusters by looking

Algorithm 6: *MergeDS*: Merging clusters with the data summary

input : *Hashtable*: a hash table of (tid, (refid,s)),

DM: a distance matrix between representatives,

B: Boolean conditions, *DB*: a trajectory database

output: $C^{l_{opt}}$: a set of clusters

```
1  begin
2      l ← 0;
3      Tlist ← Search(DB, B);
4      C^l ← LoadDS(Hashtable, Tlist);
5      C^l.db ← DB(C^l);                                  /* Equation 5.6 */
6      while |C^l| > 1 do
7          find the closest pair of clusters c_i, c_j ∈ C^l;
8          c_ij ← c_i ∪ c_j;
9          if c_i.count > c_j.count then
10             c_ij.r ← c_i.r;
11         else
12             c_ij.r ← c_j.r;
13         c_ij.count ← c_i.count + c_j.count;
14         C^{l+1} ← {C^l − c_i − c_j} ∪ c_ij;
15         C^{l+1}.db ← DB(C^{l+1});                      /* Equation 5.6 */
16         l ← l + 1;
17     l_opt ← arg min_i C^i.db;
18     return C^{l_opt}
```

up the hash table and compute the representative of each identified cluster. In each step of the while loop, it continues to merge the closest pair of clusters (line 8) and then compute the DB index (line 9). After the while loop, we report the clustering result that minimizes the DB index.

5.3 Experimental evaluation

In this section, we validate the efficiency and the effectiveness of proposed clustering methods over real-life location and semantic-rich trajectory data in Section 5.3.1 and 5.3.2 respectively.

All experiments were conducted on a machine with a Pentium-4 (3.2GHz) CPU and 1 GBytes for its main memory, running a version of the Linux operating system. We implemented our proposed algorithms using the C programming language. All trajectories are stored in Oracle Berkeley DB (version 4.6). B-tree index was built on the database for efficient retrieval of Boolean selections.

5.3.1 Location trajectories

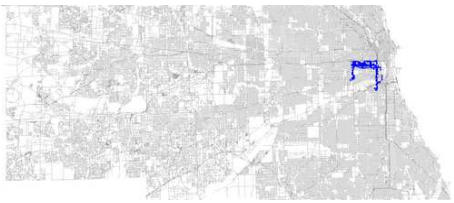
In this section, we first validate the efficiency and the effectiveness of NNCluster with real-life location trajectory data.

Experimental settings

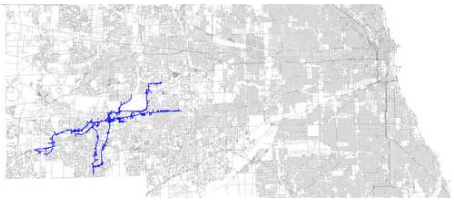
As our experimental dataset, we used a real-life trajectory dataset[2] that contained 214 trajectories collected as a sequence of positions from GPS receivers in two counties of Illinois, *i.e.*, Cook and DuPage county. The length of the trajectory ranged from 18 to 1486.

To get "ground truth" clustering results, we manually classified the real trajectory data into three groups according to their movement patterns. Figure 5.4 visualizes the trajectories of each group on the road network and also shows statistics such as average, minimum, and maximum values of the trajectory length for each group. In group i, there were 97 trajectories restricted within a small region of Cook county

[2] `http://cs.uic.edu/~boxu/mp2p/gps_data.html`



(a) Group iAvg:184, Min:30, Max:1486



(b) Group iiAvg:174, Min:18, Max:488

(c) Group iiiAvg:901, Min:580, Max:1200

Figure 5.4: Three groups of the real-life trajectories

(see Figure 5.4(a)) Figure 5.4(b) illustrates the 23 trajectories of group ii, which were located within DuPage county. Group iiiincluded the trajectories of those traveling from one county to another, covering 93 trajectories in the dataset.

We can observe from this real-life dataset that trajectories in the same cluster can vary in "density" within the group. To illustrate, the trajectories in group iiiare identical to each other except for two trajectories that are missing road segments. This suggests that the density of the identical trajectories is high, while that of the remaining two is much lower.

As density-based clustering algorithms are known to fail when clusters have "het-

erogeneous" densities within the group, applying the algorithms to this set is not desirable. In contrast, this section shows how our proposed algorithm identifies highly accurate clusters for real-life data.

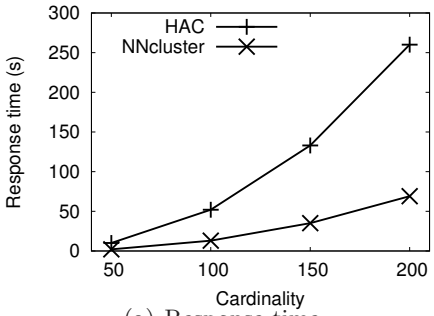
Efficiency

In this section, we will validate the efficiency of NNCluster compared to agglomerative hierarchical clustering as a baseline algorithm. More specifically, to adopt agglomerative hierarchical clustering method to our problem, we first compute a distance matrix (DM) that contains the distances between all pairs of clusters. DM_{ij} contains the distance between the two clusters. At each particular step, we merge two clusters that have the minimum distance value in DM. Assume that TR_i and TR_j are merged into a new cluster. Whenever a new cluster is generated, we then update DM by re-calculating the distances of the new cluster against all of the existing clusters (except TR_i and TR_j). This algorithm terminates when all of the trajectories are merged into a single cluster.

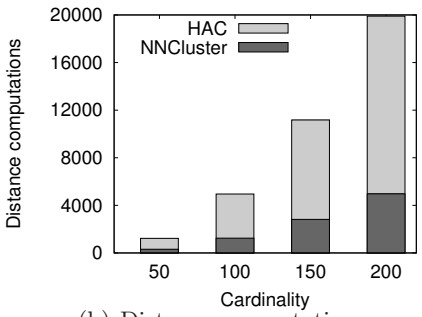
To observe the performance of our algorithm with changing cardinality, we evaluated sample trajectories from the real-life dataset discussed above, of size 50, 100, 150, and 200. To ensure the representativeness of these samples, we made sure that each sample set maintained the same distribution over three categories. In other words, we sampled to make sure that the ratio of categories in each of three categories remained unchanged both in the original dataset and sample sets.

As shown in Figure 5.5(a), our proposed clustering algorithm outperformed the baseline algorithm in all of the experimental settings in this section. Overall, our proposed algorithm was 4 or 5 times faster than the baseline algorithm.

Figure 5.5(b) shows the number of distance computations, performed by our proposed algorithm and the baseline algorithm. It is clear that our proposed algorithm



(a) Response time



(b) Distance computations

Figure 5.5: Efficiency test for different cardinalities

reduces the number of distance computations by four times as much, as similarly observed trend for the response time in Figure 5.5(a). These results suggest that our proposed algorithm can efficiently cluster trajectories by reducing the number of distance computations, which directly influences the overall performance of clustering algorithms.

Quality of the clustering result

In this section, we first validate the quality of the clustering results, produced by NNCluster, by comparing the clusters of NNCluster with the original class of trajectories.

In particular, we focus on the smallest sample set of 50 trajectories used in Sec-

Group	Size	Trajectory IDs
i	22	121 ∼ 142
ii	6	1 11 100 112 ∼ 114
iii	22	0 10 13 14 15 17 18 101 ∼ 111 117 ∼ 120

Table 5.1: Sample trajectory data

tion 5.3.1, to visualize the clustering results with the dendrograms in this section. Table 5.1 presents the sample trajectory data with a list of trajectory IDs extracted from each "ground truth" group.

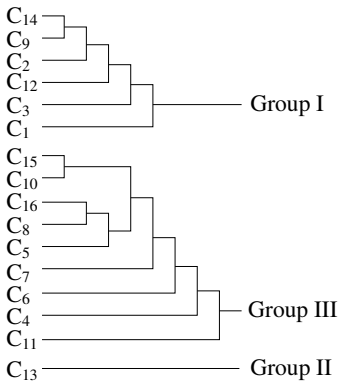
We now validate the quality, by comparing clusters generated by our algorithm, with the those of the ground-truth clustering (of the three groups of trajectories discussed in Figure 5.4). Recall that, NNCluster starts with a set of initial clusters and then merges them in a bottom-up fashion, which can be represented as a cluster hierarchy. We summarized the initial clusters in Figure 5.6(a), and then represented their clustering hierarchy in Figure 5.6(b) with a dendrogram.

It is clear that, initially, there were 16 clusters identified by NNCluster. Figure 5.6(a) lists the initial clusters with their representative trajectories marked with an asterisk. The quality of initial clusters can be validated by the fact that every initial cluster belongs to the same cluster group.

After merging the initial clusters, *i.e.*, at the second step of NNCluster, NNCluster finally produces three clusters as shown in Figure5.6(b). It is apparent that the three clusters produced by NNCluster are equivalent to the three ground-truth cluster groups. This observation suggests that NNCluster correctly partitions the sample trajectory data.

cluster id	member(s)
C_1	128* 139 129 141 140
C_2	127*
C_3	142* 131 132
C_4	102*
C_5	108* 107
C_6	17*
C_7	14*
C_8	119*
C_9	137* 124 130 122 121 136
C_{10}	109* 118 18 120 103 101
C_{11}	0*
C_{12}	125* 126
C_{13}	100* 112 11 1 114 113
C_{14}	133* 123 134 138 135
C_{15}	105* 111 10
C_{16}	110* 104 106 13 15 117

(a) Initial clusters of NNCluster



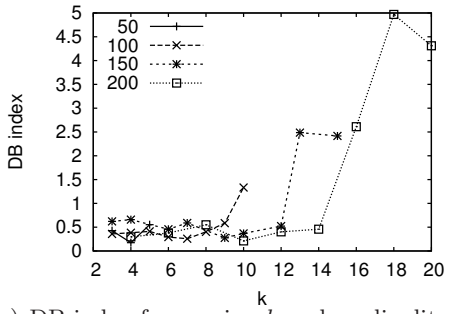
(b) Dendrogram

Figure 5.6: Clustering results of NNCluster

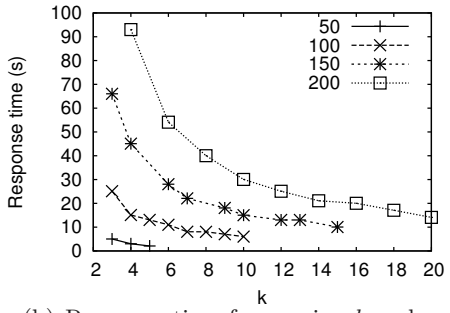
Effects on parameter k

In this section, we discuss the effect of the user parameter k, when deciding the number of neighbors to consider, on the performance of NNCluster.

Intuitively, as observed above, k determines the trade-off between the performance and the quality of NNCluster. For a smaller k, NNCluster identifies more initial clusters of smaller sizes, and eventually, when $k = 1$, our algorithm degenerates to the baseline algorithm. For a larger k, NNCluster identifies a smaller number of bigger initial clusters and terminates earlier, but starting with large initial clusters may negatively affect the output cluster quality. In an extreme case when $k = n$, NNCluster will identify a single cluster containing all of the objects.



(a) DB index for varying k and cardinality



(b) Response time for varying k and cardinality

Figure 5.7: Effects on the parameter k

It is thus important for NNCluster to tune k to the right value. To find the right value of k, we conducted extensive experiments by varying the value of k over the four sample dataset as described in Section 5.3.1. Figure 5.7 shows the decreasing quality (in Figure 5.7(a)) and response time (in Figure 5.7(b)) for larger k.

It is clear that, as k increases, the quality is not affected up to a certain point and that the performance of NNCluster improves. Based on this observation, we suggest that the value of k is set to the largest possible value that does not degenerate the quality. In all of the sample dataset, NNCluster shows best performance and a reasonable quality when k is set to an amount of 7 percent of sample dataset.

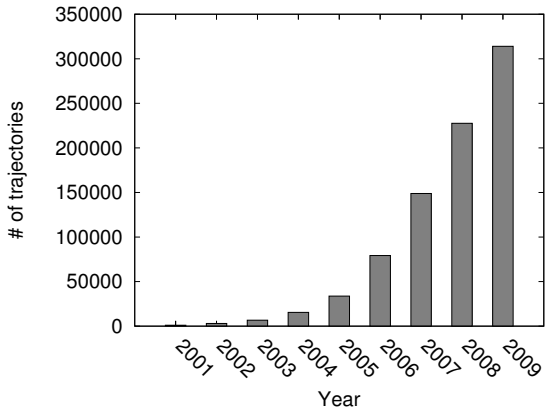


Figure 5.8: The number of trajectories from 2001 to 2009

5.3.2 Semantic-rich trajectories

In this section, we report the result of experiments conducted to verify the effectiveness and efficiency of SemCluster for semantic-rich trajectories.

Experimental settings

For semantic-rich trajectories, we collect geotagged images from Flickr website using Flickr API. More specifically, we collected $1,505,768$ geo-tagged images, which are taken from 1 January 2001 to 31 December 2009 in Australia. From the collected images, we first group them by their userID and sort images of each group in the order of the taken time. We then split a image sequence of each user into disjoint subsequences by discretizing 24 hours into 12 range groups. The number of trajectories exponentially has increased over years as shown in Figure 5.8.

As a result, $314,000$ semantic-rich trajectories are stored in the database.

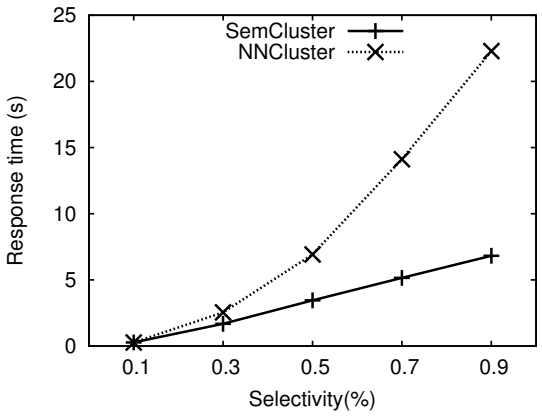


Figure 5.9: Response time vs. selectivity

Efficiency

In this section, we validate the efficiency of SemCluster, compared with NNCluster using the response time as metric, by varying the selectivity of Boolean conditions. Note that, in contrast to NNCluster, SemCluster does not require any distance computations in runtime. Therefore, the performance gap between NNCluster and SemCluster will increases if semantic-aware distance measure is used, because of computing the image similarity.

As shown in Figure 5.9, SemCluster outperforms NNCluster in all experimental settings. The speedup of SemCluster increases as the selectivity grows. We achieve this speedup by pre-computing trajectory distance computation when building the data summary structure, and performing only references of inter-cluster distance in distance matrix at runtime.

Quality of the clustering result

In this section, we validate the quality of clustering results, produced by SemCluster. More specifically, we visualize the clustering results for Boolean selections on

(a) Results for S1 and S3 (b) Results for S2

Figure 5.10: Daily trend of trajectories

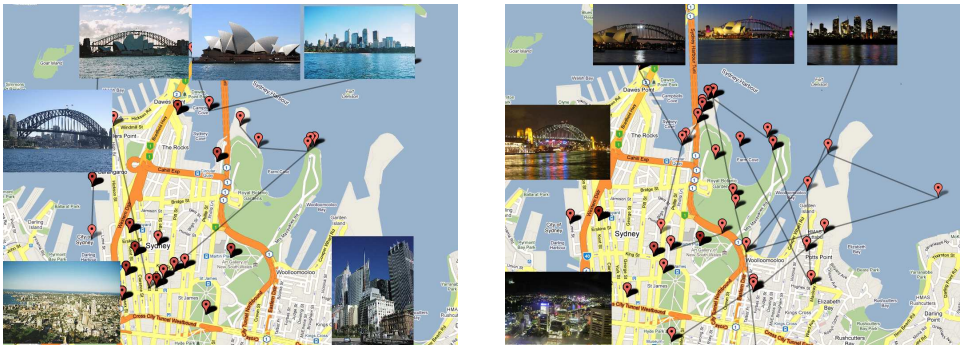
timestamps or userID. As our location trajectories also have these semantic information, we report our results for both semantic-rich and location data sets.

Selection on temporal attributes Scenarios for combining clustering with selection conditions on timestamp include finding the clusters for different time of the day or different season of the year. In this section, we demonstrate the clustering results for trajectories in the morning (S1), during lunchtime (S2) and in the evening (S3), which correspond to 44, 57, and 100 trajectories in the dataset.

Figure 5.10(a) and (b) contrast different clustering results. In Figure 5.10(a), for selection conditions S1 and S3, our framework identifies two clusters, representing a long and short commutes. In contrast, during lunchtime, the representative cluster only shows the location changes near the workplace.

Similarly, Figure 5.11(a) and (b) contrast results for S1 and S3– Note the cluster identified for S3 (evening) shows night view spots near coastline, while the results for S1 (morning) includes more locations in downtown.

Selection on userID Similarly, selection conditions can be added to userID to identify user-specific trends. As location trajectory dataset consists of GPS locations of two users, we contrast the clustering results of these two users in Figure 5.12, which accurately contrasts person A and person B making a long and short commute



(a) Morning (6am-12pm) (b) Evening (6pm-12am)

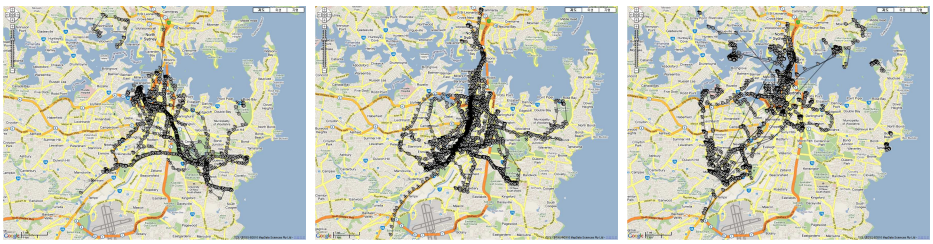
Figure 5.11: Daily trend of semantic-rich trajectories

respectively.

(a) Person A (b) Person B

Figure 5.12: Representative trajectories of individuals

We also observe user-specific trends in the clustering result of semantic-rich trajectories. We use Flickr user ID as a selection attribute. Figure 5.13 illustrates trajectories satisfying the Boolean condition "userID=29052940@N02" (S4), "userID=29064767@N04" (S5), and "userIC=30265340@N00" (S6). We present the clustering result in Figure 5.14, 5.15 and 5.16 for the Boolean condition S4, S5, and S6, respectively. For presentation purpose, we plot only the representative trajectory and we do not visualize small clusters with size less then 15% of the largest one. Figure 5.14, 5.15, and 5.16 contrast the clustering results for S4, S5, and S6, which show the user-specific trends in semantic-rich trajectories.

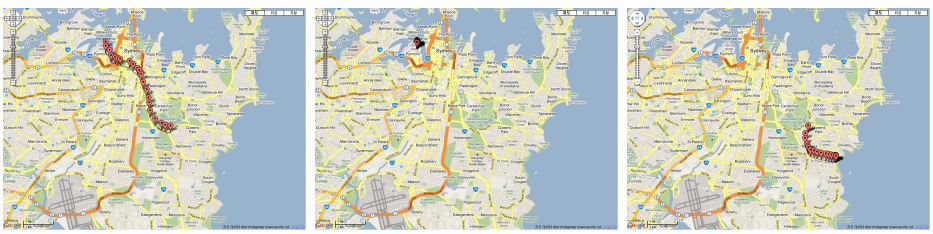


(a) Trajectories satisfying **S4** (68 trajectories)

(b) Trajectories satisfying **S5** (288 trajectories)

(c) Trajectories satisfying **S6** (108 trajectories)

Figure 5.13: Trajectories of individuals

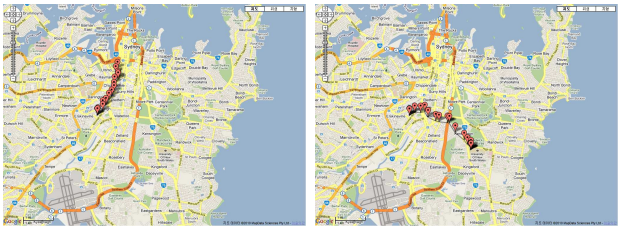


(a) C_1 (30 trajectories)

(b) C_2 (22 trajectories)

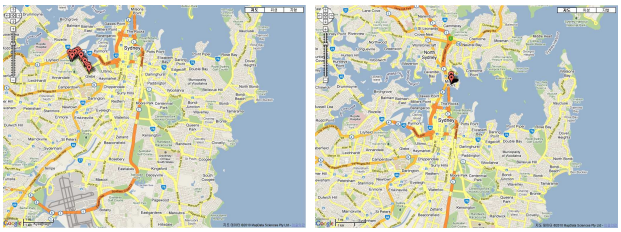
(c) C_3 (10 trajectories)

Figure 5.14: Representative trajectories for **S4**



(a) C_1 (229 trajectories) (b) C_3 (31 trajectories)

Figure 5.15: Representative trajectories for **S5**



(a) C_1 (76 trajectories) (b) C_2 (20 trajectories)

Figure 5.16: Representative trajectories for **S6**

5.4 Summary

In this chapter, we first proposed methods how to cluster network trajectories in an on-line environment. To work towards this goal, we first investigated the proximity for network trajectories, *i.e.*, road-network proximity, and then devised a distance measure, which fulfilled the required properties. With the proposed distance measure, we presented an efficient clustering algorithm NNCluster, which is based on an intuitive notion of common nearest neighbors.

We validated the efficiency and the quality of the clustering results using real-life road network trajectories. To validate the efficiency of NNCluster, we conducted experiments on sample trajectory data varying the cardinality of the trajectories. NNCluster outperformed the baseline approach in all samples. We then validated the quality of the clustering results of NNCluster. NNCluster correctly partitioned the trajectory data, which made it equivalent to the result from ground-truth clustering. Our results demonstrate that NNCluster identifies identical clusters to ground-truth clustering but incurs only the 20% of the baseline computation cost.

Next, we studied how to integrate a Boolean selection with clustering for mining semantic-rich trajectories. Specifically, we design a new framework pre-materializing a data summary structure whole dataset and propose a greedy clustering algorithm building on such summary structure. We validated the effectiveness and the efficiency of our results using real-life location trajectories and semantic-rich trajectories collected from the Flickr website. Our proposed framework enables to capture temporal characteristics of clusters and outperforms the baseline algorithm which was proposed in our preliminary work. For future work, we are investigating the efficient maintenance of the summary structure, in case of frequent updates in the trajectory database.

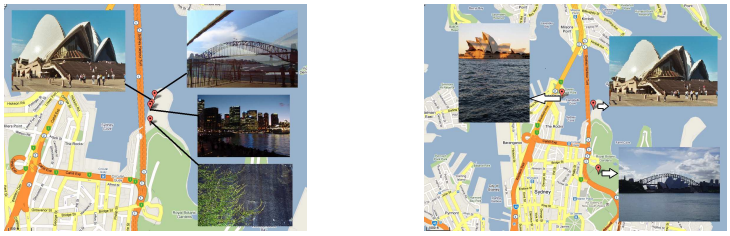
6

Semantic-rich Trajectory Mining

With the advent of ubiquitous computing, a massive amount of geo-tagged photographs has been collected and shared in online media-sharing services such as Flickr [74] and Google picasa [75]. For example, Flickr hosts over 190 millions public geo-tagged images up to July 2012. The enriched online multimedia resources open up a new world of opportunities to discover geographic knowledge and information related to location. Therefore, analyzing geo-tagged images has been focused recently in many research communities such as database, computer vision, and information retrieval.

In this chapter, we propose a method for discovering landmarks as the first step of semantic-rich trajectory mining. A landmark is typically defined a prominent geographic feature containing easily recognizable things, such as buildings and towers. As growing online media-sharing services, the popular appeal of landmarks results in a large amount of related images. We focus on geo-tagged images which connect geographical, time, and visual information together and provide possibilities to discover visual patterns and knowledge of a particular location.

To analyze geo-tagged images, there are two challenges: (1) spatial entity disambiguation and (2) spatial entity resolution. When using only geographical informa-



(a) Spatial entity disambiguation (b) Spatial entity resolution

Figure 6.1: Limitations of location-based landmarks

tion, such as latitude and longitude, all images within a GPS error bound are treated as the same landmark. For example, Fig. 6.1(a) shows four geo-tagged images ,which include different landmark entities, within a GPS error bound. Images located far away from each other can contain the same object as shown in Fig. 6.1(b).

Our goals are (1) to discover landmarks and their representative images over geo-tagged image dataset and (2) to devise an efficient search method to find a landmark for a given specific geo-location.

Toward the goal, we first use image features of the geo-tagged images, using Microsoft Photosynth [76], to extract the adjacency graph between images where two images are connected if they share the same object. Clustering this graph maps images to real-life landmark entities, based on which we find landmarks and their representative images. This representation enables our method to address both (1) spatial entity disambiguation and (2) spatial entity resolution.

As an application scenario, our proposed method can be adopted to semantic trajectory representation [77, 78, 79, 80]. The semantic trajectory representation is to summarize the raw GPS trajectories as a sequence of key points that describes the tourist's interests. To find such key points in a trajectory, most of existing work either link the trajectory to external databases such as POI (Point Of Interests) database or use sub-sampling elements of trajectories. In contrast, once we build the database

that consists of discovered landmarks, we can use those landmarks, which is related to geo-locations, as key points. It is more general than the existing work, because we cannot guarantee the existence of POI database for all application domains.

6.1 Discovering landmarks

This section presents how to discover landmarks from geo-tagged image database. Our observation is that a popular object appears frequently in the photo database. In addition, there is a spatial locality of photos where the same object was taken. With above two observations, we invent new object-level landmark detection framework as illustrated in Fig. 6.2.

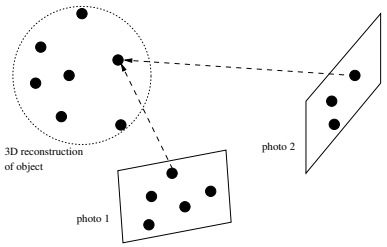
6.1.1 Landmark extraction

From the photos, we identify objects of the given photo by comparing image content of the given photos.

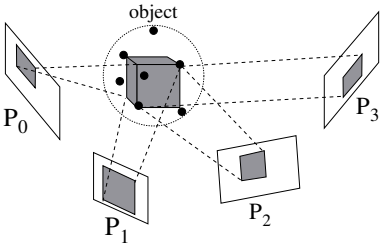
We adopt a popular descriptor for photo called SIFT [81]. SIFT applies Laplacian of Gaussian filters at several different scales to identify candidate points, that are iteratively refined by removing points around edges or those in noisy regions. For each image, SIFT generates a set of 128-dimensional vector of size several hundreds.

We then apply Bundler [82] which is a research purpose prototype of Photo-Synth [76] to identify shared points between photos. Bundler reconstructs 3D structures for the photos using key points we retrieved using SIFT feature. The structures are composed of points in 3D space and we can find the points two photos share from these structures. By computing the coordinates of the shared 3D points back into photo coordinate, we obtain the overlap of two photos.

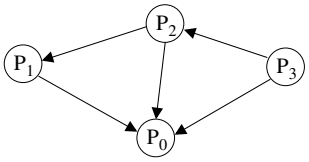
For example, we reconstruct an object using Bundler with photos (see Fig. 6.2(a)). For the given two photos that used in the reconstruction, we find their shared points and then obtain , for each photo, the region that bounds these points (see Fig. 6.2(b)).



(a) Shared points between photos



(b) Shared area from the shared points



(c) Graph representation

Figure 6.2: Object-level landmark detection

6.1.2 Visual summarization

This section describes how we find a representative image of each landmark, which will describe geographical knowledge well. In contrast to the existing work that have been proposed to find landmarks, our landmark detection algorithm seeks representative photos in an object level.

In previous section, we already identified objects from image database. Based on this, we now find the relationship between images that share the same object or not. We encode the object-sharing relationship to a graph as shown in Fig. 6.2(c). In the graph, a node indicates a photo and an edge between two nodes is connected if the two nodes share the same object. The weight on each edge is computed by measuring the overlap region between the two photos.

We then perform HITS algorithm on the graph to find the representative photo. If some nodes have many incomming links from other nodes in the graph, those nodes contain the object that appears frequently in other photos. HITS algorithm is exploited to identify such nodes. After we perform HITS algorithm [83] on the graph, we assign a node with the highest *authority* value as a representative of landmark.

6.1.3 Searching on landmark database

Given a geo-location, i.e., latitude and longitude, we propose a method to search for a list of landmarks. For efficiency, we can use R-tree index structure which is built on landmark database with respect to the geo-location and its corresponding landmark (see Fig. 6.3).

We fist search for landmark candidates using a range query on the R-tree within the GPS error bound, typically 100m, from the given geo-location. From the candidates, We can perform the post-processing step comparing visual similarity between a

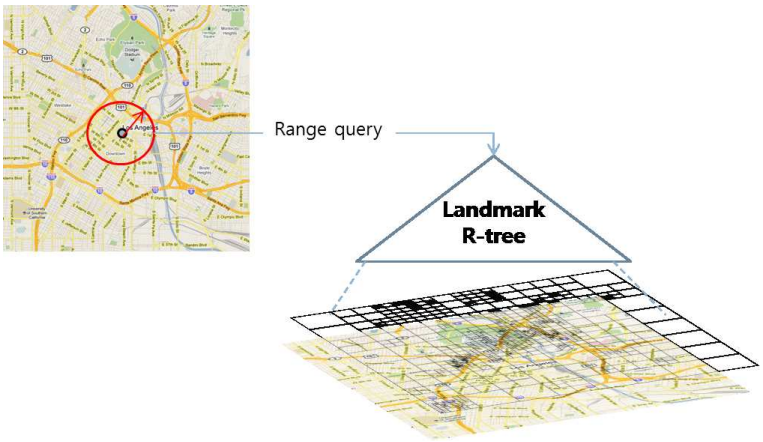
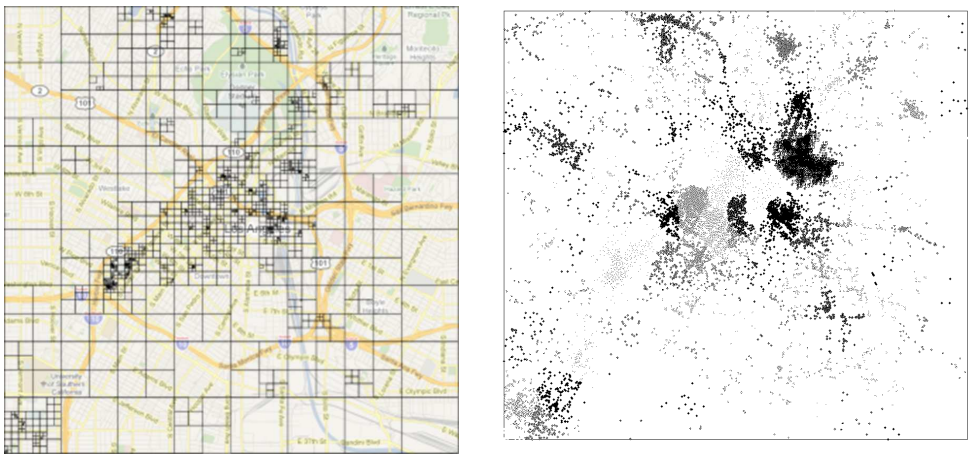


Figure 6.3: R-tree index structure for the landmark database

representative image of each candidate and a give image, if exists, using SIFT feature matching.

6.2 Experimental evaluation

In this section, we verify the effectiveness of our proposed landmark detection method over the dataset that we collected geotagged photos from Flickr website using Flickr open API. We collected 114,105 photos which had been taken in downtown Los Angeles from 2005 to 2010.



(a) partition (b) plotting geotagged photos

Figure 6.4: Dataset (1564 cells)

6.2.1 Landmark extraction

Fig. 6.4 shows the area where we collected the dataset and the result of space partitioning. We found 1564 cells in our search space.

Fig. 6.5 shows representative images of landmarks that are discovered by our method. Note that they are well known sites and buildings located in downtown Los Angeles.

Figure 6.5: Examples of landmarks detected by our algorithm

6.2.2 Comparing the landmark with real-life entities of Foursqure

To verify the quality of landmarks discovered by our method, we exploit the experience of Foursquare users, in which we can find which places are visited frequently by users and photos taken by visitors. This approach can be considered as indirect user study.

Table 6.1 shows the top 25 places that Foursqure provides, which includes the name, category, the number of users that visited the place, and the number of photos taken. We rank the place by a popularity measure, i.e., the ratio of the number of photos to the number of users that visited the place. The popularity measure that we use assumes that users take more photos when they are interested in the place. With this measure, we can observe that places with the lower value of popularity measure belong to categories where indoor activity often occurs (e.g., restaurant, gallery, and bar).

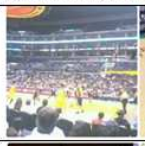
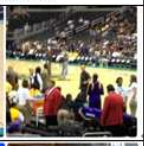
We further investigate to show the quality of our landmarks by comparing the images that users can see in Foursqure website with our representative images of landmark. Foursqure provide the functionality uploading images when the user check

in some place. We compare the quality of such images with our representative images.

Table 6.2, 6.3, and 6.4 show images that are linked with each place in Foursqure website. We selected images that appears in the first page when we search the place in that website. We first observed that, as we expected, places where indoor activity often takes place have images that does not include recognizable landmarks. For example, restaurant, brewery, and coffee shop consists of lots of images of food.

NAME	Category	#of users	# of photos	#of photos/# of users	Our method
STAPLES Center	Stadium	36679	3615	0.098557758	O
Dodger Stadium	Baseball Stadium	23089	2171	0.094027459	O
Walt Disney Concert Hall	Concert Hall	5356	420	0.078416729	O
Bottega Louie	Italian Restaurant	10960	670	0.061131387	X
Doll Factory (L.A. Derby Dolls)	Event Space	1291	70	0.054221534	X
Nokia Theatre	Concert Hall	8107	392	0.048353275	X
The Westin Bonaventure Hotel & Suites	Hotel	5351	257	0.048028406	O
Philippe The Original	Sandwich Place	7781	368	0.047294692	O
Union Station	Train Station	14372	666	0.046340106	O
Los Angeles Chinatown	Other Great Outdoors	3947	173	0.043830758	O
Rooftop Bar at The Standard, Downtown LA	Lounge	5566	243	0.043657923	X
L.A. LIVE	General Entertainment	10884	466	0.042815141	X
The Standard, Downtown LA	Hotel	5693	231	0.040576146	O
Urth Caffe	Coffee Shop	4666	189	0.040505787	X
Club Nokia	Rock Club	6008	235	0.039114514	X
Wurstkuche	German Restaurant	9203	352	0.038248397	X
JW Marriott LA Live	Hotel	6394	226	0.035345637	X
Los Angeles Convention Center	Convention Center	11095	374	0.033708878	O
Little Tokyo	Other Great Outdoors	5120	170	0.033203125	O
ESPN Zone	Sports Bar	6261	181	0.02890912	X
University of Southern California	University	7395	211	0.028532792	X
Regal LA LIVE Stadium 14	Multiplex	6444	162	0.025139665	X
Yard House	Brewery	11252	261	0.023195876	X
Downtown Los Angeles Artwalk	Art Gallery	4333	99	0.022847911	X
Los Angeles Fashion District	Clothing Store	4491	89	0.019817413	X

Table 6.1: Landmarks provided by Foursqure.com in downtown Los Angeles

Name	Category	Visual information provided by Foursquare	Our representative image
STAPLES Center	Stadium	  	
Union Station	Train Station	  	
Dodger Stadium	Baseball Stadium	  	
University of Southern California	University	  	N/A
L.A. LIVE	General Entertainment	  	N/A

Name	Category	Visual information provided by Foursquare	Our representative image
Bottega Louie	Italian Restaurant	  	N/A
Los Angeles Convention Center	Convention Center	  	
Yard House	Brewery	  	N/A
Wurstküche	German Restaurant	  	N/A
Regal LA LIVE Stadium 14	Multiplex	  	N/A

Table 6.2: Representative images of our method and Foursqure's images (1/3)

Name	Category	Visual information provided by Foursquare	Our representative image
JW Marriott LA Live	Hotel		N/A
The Westin Bonaventure Hotel & Suites	Hotel		N/A
Urth Caffé	Coffee Shop		N/A
Nokia Theatre	Concert Hall		N/A
Philippe The Original	Sandwich Place	 	

Name	Category	Visual information provided by Foursquare	Our representative image
Little Tokyo	Other Great Outdoors		
The Standard, Downtown LA	Hotel		
ESPN Zone	Sports Bar		N/A
Doll Factory (L.A. Derby Dolls)	Event Space		N/A
Los Angeles Chinatown	Other Great Outdoors		

Table 6.3: Representative images of our method and Foursqure's images (2/3)

Name	Category	Visual information provided by Foursquare	Our representative image
Club Nokia	Rock Club		N/A
Rooftop Bar at The Standard,	Lounge		N/A
Walt Disney Concert Hall	Concert Hall		
Downtown Los Angeles Artwalk	Art Gallery		N/A
Los Angeles Fashion District	Clothing Store		N/A

Table 6.4: Representative images of our method and Foursqure's images (3/3)

7 Conclusion

This dissertation discusses (1) how to support pattern matching queries for trajectories constrained by road networks and (2) how to integrate a Boolean selection with clustering for mining semantic-rich trajectories. Specifically, we first investigate the requirements for the trajectory representation and distance measure for our target problem. As none of the existing work fully satisfies the requirements identified, we then devise the trajectory representation and the distance measure, which fulfill all the requirements.

This dissertation has introduced the three pattern matching queries (whole, subpattern, and reverse subpattern matching) to search for similar trajectories to the given query trajectory. Though the notion of *similarity* varies across different types of queries, we proposed a unified framework efficiently supporting range and KNN queries for all three types of matching based on M-tree and pruning rules. We validated the quality of results by visualizing the results for different types of queries over real-life road network trajectories. Comparison with other distance measures showed that the proposed distance measure is more appropriate for road network trajectories.

We then study how to integrate a Boolean selection with clustering for mining semantic-rich trajectories. Specifically, we design a new framework pre-materializing

a data summary structure whole dataset and propose a greedy clustering algorithm building on such summary structure. We validated the effectiveness and the efficiency of our results using real-life location trajectories and semantic-rich trajectories collected from the Flickr website. Our proposed framework enables to capture temporal characteristics of clusters and outperforms the baseline algorithm which was proposed in our preliminary work.

한글 요약문

유비쿼터스 컴퓨팅의 발달로 대용량의 궤적 데이터가 생성되고 공유되고 있다. 이러한 궤적 데이터는 그 양이 많을 뿐 아니라, 굉장히 빠른 속도로 생성되며 다양한 타입의 미디어 데이터 (텍스트, 이미지, 비디오)를 포함하고 있다. 이러한 데이터의 특징과 다양한 사용자의 기호, 심지어 동일한 사용자라 할지라도 시시각각 변하는 사용자의 컨텍스트를 만족하게 하기 위해서는 효율적인 데이터 마이닝 기법이 필요하다. 또한, 다양한 타입의 미디어를 분석하는 것은 기존의 데이터에서 알 수 없었던 새로운 지식을 추출하는 것이 가능하다. 본 학위 논문은 이러한 궤적 데이터 중에서 움직임에 제약이 있는 이동 개체의 마이닝 기법에 대한 연구 결과를 담고 있다. 기존의 궤적 데이터에 대한 연구는 주로 움직임에 제약이 없는 이동 개체에 대한 마이닝 기법에 초점을 맞추고 있었다. 태풍이나 동물의 궤적 데이터가 이러한 타입의 궤적 데이터에 속한다. 하지만 실제 애플리케이션에서는 이동 개체의 움직임에 제약이 있는 경우가 대부분이다. 이처럼 움직임에 제약이 있는 이동 개체가 만들어내는 궤적 데이터는 기존의 방법을 적용하였을 때에 그 결과의 정확도가 떨어진다. 따라서 본 학위 논문에서는 움직임에 제약이 있는 이동 개체의 마이닝 기법이 만족해야 할 속성들을 제안하고 이를 만족하기 위한 궤적의 표현기법과 거리 함수를 제안하였다. 이를 기반으로 유사검색과 클러스터링 방법론을 제안하였다. 유사검색은 궤적 데이터에 필요한 세 가지의 패턴을 정의하며, 각 패턴의 의미를 완벽히 반영하고 있는 하우스도프 디스턴스를 사용하여 단일의 인덱스를 사용하여 모든 패턴의 쿼리를 처리할 수 있도록 고안하였다. 클러스터링은 온라인 환경을 고려하여 효율적일 수 있도록 거리 계산을 줄이는 방법을 개발하였으며, 이는 전통적인 데이터베이스 질의인 불린(Boolean) 질의와 통합하여 좀 더 효율성을 높였다. 마지막으로 궤적 데이터에 포함되어있는 미디어 데이터를 분석하여 사용자의 의도와 행동을 더 잘 이해할 수 있는 기반을 마련하였다. 우리는 각 마이닝 기법에 대해서 실제 궤적 데이터를 사용하여 효율성과 유효성을 보였다.

Bibliography

[1] Christian S. Jensen, Dan Lin, and Beng Chin Ooi. Query and update efficient b+-tree based indexing of moving objects. In *Proc. VLDB'04*, pages 768–779, 2004.

[2] Haibo Hu, Jianliang Xu, and Dik Lun Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *Proc. SIGMOD*, pages 479–490, 2005.

[3] X. Xiong, M.F. Mokbel, and W.G. Aref. Sea-cnn: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. *Proc. ICDE'05*, pages 643–654, April 2005.

[4] Su Chen, Beng Chin Ooi, Kian-Lee Tan, and Mario A. Nascimento. ST2B-tree: a self-tunable spatio-temporal b+-tree index for moving objects. In *Proc. SIGMOD*, pages 29–42, 2008.

[5] Eamonn J. Keogh. Exact indexing of dynamic time warping. In *Proc. VLDB*, pages 406–417, 2002.

[6] Vincent W. Zheng, Yu Zheng, Xing Xie, and Qiang Yang. Collaborative location and activity recommendations with gps history data. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 1029–1038, New York, NY, USA, 2010. ACM.

[7] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proc. WWW*, pages 791–800, 2009.

[8] Dieter Pfoser and Christian S. Jensen. Trajectory indexing using movement constraints*. *GeoInformatica*, 9(2):93–115, 2005.

[9] Victor Teixeira De Almeida and Ralf Hartnut Guting. Indexing the trajectories of moving objects in networks. *GeoInformatica*, 9(1):33–60, 2005.

[10] Xiang Li and Hui Lin. Indexing network-constrained trajectories for connectivity-based queries. *International Journal of Geographical Information Science*, 20(3):303–328, 2006.

[11] Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, and Nikos Mamoulis. Continuous nearest neighbor monitoring in road networks. In *Proc. VLDB*, pages 43–54, 2006.

[12] Michalis Vazirgiannis and Ouri Wolfson. A spatiotemporal model and language for moving objects on road networks. In *Proc. SSTD*, pages 20–35, 2001.

[13] Bin Lin and Jianwen Su. One way distance: For shape based similarity search of moving object trajectories. *Geoinformatica*, 12(2):117–142, 2008.

[14] Elias Frentzos, Kostas Gratsias, and Yannis Theodoridis. Index-based most similar trajectory search. In *Proc. ICDE*, pages 816–825, 2007.

[15] Seok-Lyong Lee, Seok-Ju Chun, Deok-Hwan Kim, Ju-Hong Lee, and Chin-Wan Chung. Similarity search for multidimensional data sequences. In *Proc. ICDE*, pages 599–608, 2000.

[16] Michail Vlachos, Dimitrios Gunopulos, and George Kollios. Discovering similar multidimensional trajectories. In *Proc. ICDE*, pages 673–684, 2002.

[17] Lei Chen, M. Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proc. SIGMOD*, pages 491–502, 2005.

[18] Lei Chen and Raymond T. Ng. On the marriage of lp-norms and edit distance. In *Proc. VLDB*, pages 792–803, 2004.

[19] Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckmann. Efficient detection of motion patterns in spatio-temporal data sets. In *Proc. GIS*, pages 250–257, 2004.

[20] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S. Jensen, and Heng Tao Shen. Discovery of convoys in trajectory databases. In *Proc. VLDB*, pages 1068–1080, 2008.

[21] Petko Bakalov, Eamonn Keogh, and Vassilis J. Tsotras. TS2-tree - an efficient similarity based organization for trajectory data. In *Proc. GIS*, pages 1–4, 2007.

[22] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proc. SIGMOD*, pages 593–604, 2007.

[23] Helmut Alt, Christian Knauer, and Carola Wenk. Matching polygonal curves with respect to the fréchet distance. In *Proc. STACS*, pages 63–74, 2001.

[24] Yutaka Yanagisawa, Jun-ichi Akahani, and Tetsuji Satoh. Shape-based similarity query for trajectory of mobile objects. In *Proc. MDM*, pages 63–77, 2003.

[25] Yuhan Cai and Raymond Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *Proc. SIGMOD'04*, pages 599–610, 2004.

[26] Jiangung Lou, Qifeng Liu, Tieniu Tan, and Weiming Hu. Semantic interpretation of object activities in a surveillance system. In *Proc. ICPR'02*, page 30777, 2002.

[27] Imran N. Junejo, Omar Javed, and Mubarak Shah. Multi feature path modeling for video surveillance. In *Proc. ICPR'04*, pages 716–719, 2004.

[28] Yongsheng Gao and Maylor K. H. Leung. Line segment hausdorff distance on face matching. *Pattern Recognition*, 35(2):361–371, 2002.

[29] Jingying Chen, Maylor K. Leung, and Yongsheng Gao. Noisy logo recognition using line segment hausdorff distance. *Pattern Recognition*, 36(4):943–955, 2003.

[30] M. Fréchet. Sur quelques point do calcul fonctionnel. *Rendiconti delcircolo Mathematico di palermo*, 22:1–74, 1906.

[31] Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995.

[32] Thomas Eiter and Heikki Mannila. Thomas Eiter and Heikki Mannila, Computing discrete Frechet distance, Tech. Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

[33] Joachim Gudmundsson, Marc J. van Kreveld, and Bettina Speckmann. Efficient detection of motion patterns in spatio-temporal data sets. In *GIS*, pages 250–257, 2004.

[34] David Biliotti, Gianluca Antonini, and Jean-Philippe Thiran. Multi-layer hierarchical clustering of pedestrian trajectories for automatic counting of people in video sequences. In *Proc. WACV/MOTION*, pages 50–57, 2005.

[35] Hu Cao and Ouri Wolfson. Nonmaterialized motion information in transport networks. In *Proc. ICDT*, pages 173–188, 2005.

[36] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proc. SIGMOD*, pages 593–604, 2007.

[37] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S. Jensen, and Heng Tao Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1(1):1068–1080, 2008.

[38] Scott Gaffney and Padhraic Smyth. Trajectory clustering with mixtures of regression models. In *Proc. KDD*, pages 63–72, 1999.

[39] Chengkai Li, Min Wang, Lipyeow Lim, Haixun Wang, and Kevin Chen-Chuan Chang. Supporting ranking and clustering as generalized order-by and group-by. In *Proc. SIGMOD*, pages 127–138, 2007.

[40] Neil Johnson and David Hogg. Learning the distribution of object trajectories for event recognition. *Image and Vision Computing*, 14(8):609 – 615, 1996. ¡ce:title¿6th British Machine Vision Conference¡/ce:title¿.

[41] J. Owens and A. Hunter. Application of the self-organising map to trajectory classification. In *Visual Surveillance, 2000. Proceedings. Third IEEE International Workshop on*, pages 77 –83, 2000.

[42] Esther B. Koller-meier and Luc Van Gool. Modeling and recognition of human actions using a stochastic approach. In *2 nd European Workshop on Advanced Video-Based Surveillance Systems*, pages 17–28, 2001.

[43] Jiangung Lou, Qifeng Liu, Tieniu Tan, and Weiming Hu. Semantic interpretation of object activities in a surveillance system. In *Pattern Recognition, 2002.*

Proceedings. 16th International Conference on, volume 3, pages 777 – 780 vol.3, 2002.

[44] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee. Using adaptive tracking to classify and monitor activities in a site. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR '98, pages 22–, Washington, DC, USA, 1998. IEEE Computer Society.

[45] Chris Stauffer and W. Eric L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):747–757, August 2000.

[46] Weiming Hu, Xuejuan Xiao, Zhouyu Fu, D. Xie, Tieniu Tan, and S. Maybank. A system for learning statistical motion patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(9):1450 –1464, sept. 2006.

[47] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proc. VLDB*, pages 853–864, 2005.

[48] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proc. SIGMOD*, pages 322–331, 1990.

[49] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *Proc. VLDB*, pages 802–813, 2003.

[50] G.A. Klunder and H.N. Post. The shortest path problem on large-scale real-road networks. *Networks*, 48(4):182–194, 2006.

[51] Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: theory and experimental evaluation. *Math. Program.*, 73(2):129–174, 1996.

[52] F. Benjamin Zhan and Charles E. Noon. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32(1):65–73, 1998.

[53] Mohamed F. Mokbel, Thanaa M. Ghanem, and Walid G. Aref. Spatio-temporal access methods. *IEEE Data Eng. Bull.*, 26(2):40–49, 2003.

[54] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. SODA'93*, pages 311–321, 1993.

[55] Tolga Bozkaya and Meral Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. SIGMOD'97*, pages 357–368, New York, NY, USA, 1997. ACM.

[56] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information processing letters*, 40(4):175–179, 1991.

[57] Sergey Brin. Near neighbor search in large metric spaces. In *Proc. VLDB'95*, pages 574–584, 1995.

[58] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. VLDB*, pages 426–435, 1997.

[59] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001.

[60] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[61] Elias Frentzos. Indexing objects moving on fixed networks. In *Proc. SSTD'03*, pages 289–305, 2003.

[62] `http://cs.uic.edu/~boxu/mp2p/gps_data.html`.

[63] `http://www.census.gov/geo/www/tiger`.

[64] Maurice Kendall and Jean D. Gibbons. *Rank correlation methods, 5th ed.* A Charles Griffin Book, 1990.

[65] Thomas Brinkhoff. Generating traffic data. *IEEE Data Eng. Bull.*, 26(2):19–25, 2003.

[66] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proc. VLDB*, pages 853–864, 2005.

[67] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.

[68] Richard M. Karp. *Reducibility Among Combinatorial Problems. In: Miller, R.E., Thatcher, J.W. (Eds.), Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[69] Charles Elkan. Using the triangle inequality to accelerate k-means. In *ICML'03*, pages 147–153, 2003.

[70] Marzena Kryszkiewicz and Piotr Lasek. A neighborhood-based clustering by means of the triangle inequality. In *Proceedings of the 11th international conference on Intelligent data engineering and automated learning*, IDEAL'10, pages 284–291, 2010.

[71] Yee Leung, Jiang-She Zhang, and Zong-Ben Xu. Clustering by scale-space filtering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1396–1410, 2000.

[72] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Int. J. Intell. Inf. Syst.*, 17(2-3):107–145, 2001.

[73] David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227, 1979.

[74] `http://www.flickr.com`.

[75] `http://picasa.google.com`.

[76] `http://photosynth.net`.

[77] Xiang Li, Christophe Claramunt, Cyril Ray, and Hui Lin. A semantic-based approach to the representation of network-constrained trajectory data. In *Progress in Spatial Data Handling*, pages 451–464, 2006.

[78] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Jose Antonio Fernandes de Macedo, Bart Moelans, and Alejandro Vaisman. A model for enriching trajectories with semantic geographical information. In *ACM GIS*, pages 22:1–22:8, 2007.

[79] Zhixian Yan. Towards semantic trajectory data analysis: A conceptual and computational approach. In *VLDB PhD Workshop*, 2009.

[80] Falko Schmid, Kai-Florian Richter, and Patrick Laube. Semantic trajectory compression. In *SSTD*, pages 411–416, 2009.

[81] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[82] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2008.

[83] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. In *SODA*, pages 668–677, 1998.

Acknowledgements (감사의 글)

학위 과정을 무사히 마칠 수 있게 도와주신 모든 지도교수님들께 (박상현 교수님, 이병기 교수님, 이승용 교수님, 황승원 교수님) 감사의 말씀을 드립니다. 박사 학위 심사를 위해서 바쁘신 와중에도 시간을 내어 주신 한준희 교수님, 안희갑 교수님 진심으로 감사의 마음을 전합니다. 제게 주신 조언들 가슴 깊이 새기고 살아가겠습니다.

사랑하는 그리고 자랑스러운 우리 IDS 연구실 식구들, 우리같이 했던 소중한 추억 들을 한 번씩 꺼내볼 수 있게 매년 홈커밍데이엔 모두들 볼 수 있었으면 좋겠습니다. 당신들을 알게 된 것은 제게 큰 행운입니다. 감사합니다.

아버지, 어머니, 그리고 누나, 멀리 있어서 자주 보지는 못했지만, 목소리만 들어도 힘이 났었고, 그 힘으로 여기까지 올 수 있었던 게 아닐까 생각합니다. 늘 곁에 있어줘서 고맙습니다. 사랑합니다.

사랑하는 아내 지혜, 당신의 인내심과 이해심 없이는 이 모든 게 가능하지 않았을 거라는 생각이 들어요. 400km의 장거리 연애 때에도 결혼 후 한동안 가장으로서 우리 가정을 책임져야 했을 때에도 당신에게 늘 미안한 마음 갖고 있었어요. 그 미안함은 내 삶이 끝나는 날까지 할부로 갚아 나갈게요. 그동안의 시간을 이해해줘서 고마워요. 사랑합니다.

Curriculum Vitae

Name. Gook-Pil Roh

Research interests

1. Spatio-temporal Data Mining
2. Graph Data Mining
3. Information Retrieval and Search Engines

Education

Ph.D. (2012), POSTECH, in Computer Science and Engineering

M.S. (2005), POSTECH, in Computer Science and Engineering

B.S. (2003), Ajou University, in Information and Computer Engineering

Publications

1. Gook-Pil Roh and Seung-won Hwang "TPM: supporting pattern matching queries for road-network trajectory data." *EDBT*: 554-557, 2011.
2. Gook-Pil Roh and Seung-won Hwang "Online Clustering Algorithms for Semantic-Rich Network Trajectories." *Journal of Computing Science and Engineering.* Vol. 5, No. 4, pp. 346-353, 2011.
3. Gook-Pil Roh, Jong-Won Roh, Seung-won Hwang, and Byoung-Kee Yi " Supporting Pattern Matching Queries Over Trajectories on Road Networks." *IEEE Transactions on Knowledge and Data Engineering* Vol. 23, No. 11, 2011.

4. Gook-Pil Roh and Seung-won Hwang “NNCluster: An Efficient Clustering Algorithm for Road Network Trajectories.” *DASFAA*: 47-61, 2010

5. Woojun Yi, Gook-Pil Roh, Seonggoo Kang, and Sanghyun Park “An Effective Searching Method for Timestamped Event Sequences.” *KISS Spring Conference*: 782-784, 2003