TPM: Supporting pattern matching queries for road-network trajectory data

Gook-Pil Roh
Department of Computer Science & Engineering, POSTECH, Pohang, Korea.
noh9pil@postech.ac.kr

Seung-won Hwang
Department of Computer Science & Engineering, POSTECH, Pohang, Korea.
swhwang@postech.ac.kr

ABSTRACT

With the advent of ubiquitous computing, we can easily collect large scale trajectory data from moving vehicles. This paper presents TPM (Trajectory Pattern Miner), a software aimed at pattern matching queries for road-network trajectory data, which complements existing efforts focusing on (a) a spatio-temporal window query for location-based service or (b) Euclidean space with no restriction. To overcome limitations of prior research, TPM supports three types of pattern matching queries– whole, subpattern, and reverse subpattern matching for road-network trajectories. We demonstrate application scenarios for each type of pattern matching queries using large-scale real-life trajectory data.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining, Spatial database and GIS*

General Terms

Algorithms, Measurement, Experimentation

1. INTRODUCTION

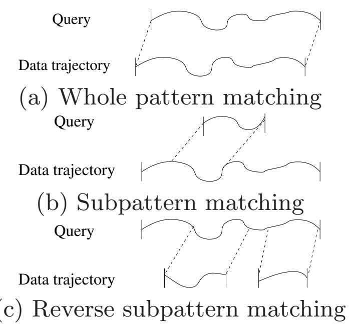
With the advent of ubiquitous computing, we can easily acquire the locations of moving objects, *e.g.*, vehicle locations acquired from global positioning system (GPS). Querying over the paths of such objects, or *trajectories*, has thus gained attention lately.

However, 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 [8, 12, 9, 6].

In addition, prior work typically assumes objects can move anywhere and considers Euclidean space as search space. However, in many real-life applications, the movements of objects are often constrained by obstacles such as buildings or trees.

In this demonstration, we use scenarios of finding training partners, among the bike trajectories people posted on Bikely [1]. The goal in this scenario is (a) finding partners preferring similar routes, while (b) such routes are constrained by the availability of bike trails (*i.e.*, underlying road network). For instance, two routes, that are spatially adjacent, may not be a good match, if separated by a highway and no bike trail to connect the two trajectories. For this scenario, we present TPM, a mining software supporting pattern matching queries over trajectories constrained by the given road network, overcoming limitations of prior research identified above.

In particular, we support three types of pattern matching queries: whole, subpattern, and reverse subpattern matching. Each pattern matching query has distinct similarity notion. First, whole pattern matching captures the global similarity between two trajectories and compares them in their entireties without ignoring any part of them. Second, subpattern matching 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. Lastly, the subpattern matching problem can be reversed. 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. 1 illustrates these three types of pattern matching.



(a) Whole pattern matching
(b) Subpattern matching
(c) Reverse subpattern matching

Figure 1: Three types of pattern matching queries

The rest of the paper is organized as follows. Section 2 presents the trajectory modeling such as trajectory representation and distance measure. Section 3 presents index structure for efficient query processing. In Section 4, we describes matching algorithms for each type of pattern matching. Section 5 presents application scenarios.

2. TRAJECTORY MODELING

We model a trajectory as a *connected* sequence of road segments, such that each road segment of a trajectory should be connected to the next road segment in the sequence. A road segment is defined by its starting and ending positions. 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.

The *raw* trajectories from positioning devices such as GPS, typically consists of quadruplets (x, y, p, v), where x and y are location points (2D points), while p and v are the time stamp and the speed 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 sequence of road segments. We leave the details on how to convert raw trajectories into the sequence of road segments, in our technical paper [10].

Once converted, we use the following distance measures reflecting the constraints of the underlying road-network, which we discussed in [10].

Definition 1 (Road segment distance [10]). *Given two road segments r_i and r_j, the road segment distance is defined as follows:*

$$d(r_i, r_j) = \max\left\{\overrightarrow{d}(r_i, r_j), \overrightarrow{d}(r_j, r_i)\right\}, \tag{1}$$

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

Definition 2 (Trajectory distance [10]). *Given two trajectories $T_a = [a_1, \dots, a_n]$ and $T_b = [b_1, \dots, b_m]$, the distance between them is defined as follows:*

$$D(T_a, T_b) = \max\left\{\overrightarrow{D}(T_a, T_b), \overrightarrow{D}(T_b, T_a)\right\}, \tag{2}$$

where $\overrightarrow{D}(T_a, T_b) = \max_{a_i \in T_a} \min_{b_j \in T_b} d(a_i, b_j)$ is one-way trajectory distance from T_a to T_b.

We extended the intuition of Hausdorff distance to our trajectory representation. Similar to Hausdorff distance, the distance measure can be explained as the longest path from each road segment to its closest road segment in another trajectory and the distance measure is metric [10].

3. INDEX STRUCTURE

For an efficient query processing, we index trajectories with M-tree [7] index structure based on the distance measure [10] between trajectories.

As our distance measure fulfills the metric property [10], we can use any index structures designed for the metric space, *i.e. metric tree*, such as the gh-tree [11], the GNAT [5] and the M-tree [7]. 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: M-tree is the only one that is optimized for large secondary memory-based data sets [4]. 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. In addition, M-tree is reported to be more scalable than other approaches in [7].

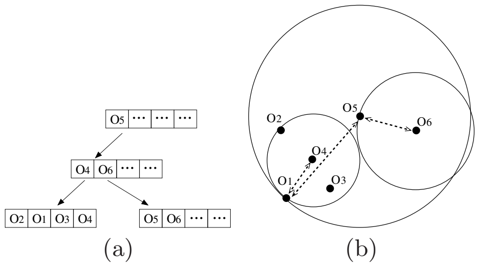


Figure 2: M-tree example: (a) M-tree structure and (b) covering radii

As preliminaries, we briefly overview how M-tree works: The leaf node of M-tree contains the 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 [7]. 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 representative object in the sub-tree. We illustrate an example of the M-tree structure in Fig. 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. 2(b).

4. QUERY PROCESSING

Given a query trajectory, TPM supports the three types of pattern matchings: whole, subpattern, and reverse subpattern pattern matching. For each type of pattern matching, TPM supports two types of similarity searches: range search and k-NN (nearest neighbor) search. The range search is to find the trajectories with distance smaller than user-specified threshold value. The k-NN search is to find the k trajectories with the smallest distance.

4.1 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 [7]. During the traversal of M-tree, for each non-leaf node, we need to decide whether its sub-tree can be pruned out– Fig. 3 shows the pruning condition of the whole pattern matching. If the distance d between a query trajectory T_q and a representative trajectory $tr(O_i)$ 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. With this pruning condition, we can conclude that, in the case of Fig. 3, we do not need to investigate the child nodes of O_1, because the pruning condition $d > \epsilon + s(O_1)$ is satisfied.

4.2 Subpattern matching

Subpattern matching searches for all the trajectories which

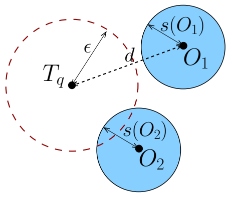


Figure 3: Pruning for whole pattern matching

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. If the *one-way* distance $\overrightarrow{D}(T_q, tr(O))$ between a query trajectory and the representative trajectory $tr(O)$ 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.

For subpattern matching, we can thus simply modify the pruning condition by using *one-way* distance $\overrightarrow{D}(T_q, T)$ to identify the closest subpattern to the query pattern. KNN and range search algorithms for subpattern matching can be straightforwardly implemented by replacing the pruning conditions to use one-way distance, $\overrightarrow{D}(T_q, T)$.

4.3 Reverse subpattern matching

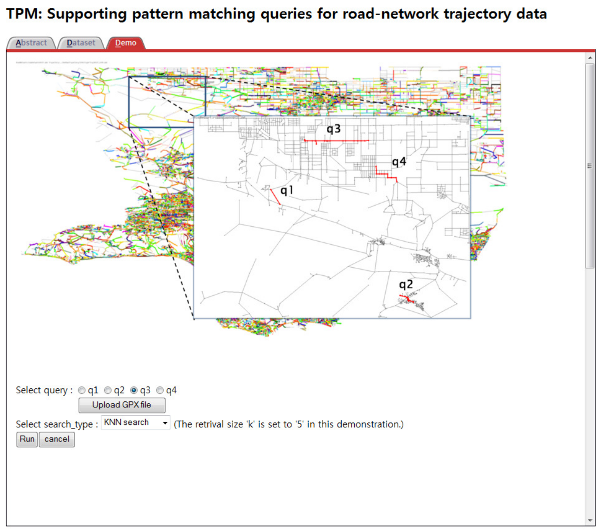
Reverse subpattern matching is used when the database contains short trajectories and the users want to know which trajectories are included in the query trajectory. For this matching, another one-way distance $\overrightarrow{D}(T, T_q)$ can capture the notion well, by identifying the closest query segment to trajectory data, instead of $\overrightarrow{D}(T_q, T)$ used for subpattern matching. Similarly to subattern matching, KNN and range search algorithms for reverse subpattern matching are straightforwardly implemented by replacing the pruning conditions to use one-way distance, $\overrightarrow{D}(T, T_q)$.

5. DEMONSTRATION SCENARIOS

We designed our demo to demonstrate three types of pattern matching queries (whole, subpattern, and reverse subpattern matching) for the real dataset that we collected from Bikely website [1].

Specifically, we collected 12971 real-life bike trajectories in California from Bikely. The average number of sampling points in trajectories is 470 and the average length of trajectories is 38.54 (km). We obtained the road network data of California in a TIGER/LINE format from U.S. Census Bureau [2].

Figure 4 illustrates our user interface of TPM where users can issue a query by uploading a query trajectory file in the GPX format that most GPS softwares support or choosing a query trajectory among example trajectories. Given the query trajectory, TPM shows the top ranked trajectories, superimposed on underlying road network, for all three types of pattern matching queries.


TPM: Supporting pattern matching queries for road-network trajectory data

Select query : ( ) q1 ( ) q2 [✓ q3] ( ) q4

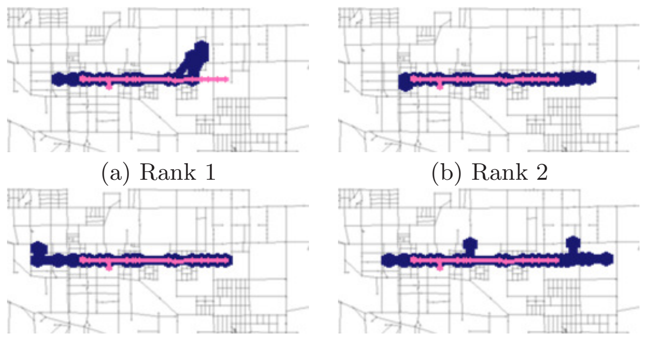
Select search_type : KNN search (The retrival size 'k' is set to '5' in this demonstration.)

Figure 4: User interface of TPM

We implemented TPM in C/C++ programming language on top of R-tree [3] (for indexing road networks) and M-tree [7] (for indexing trajectories), and web interface in Perl programming language. Our Web-based demo is available at http://ids.postech.ac.kr/TPM.

We present application scenarios for each type of pattern matching and illustrate top ranked trajectories that TPM returns in each scenario. Note that a thick polyline indicates a data trajectory while a thin polyline indicates a query trajectory in Fig. 5, 6, and 7.

EXAMPLE 1. ***Whole pattern matching***
Suppose trajectory q_3 of Fig. 4 represents a commute from work to home of some user. He can then query a whole pattern matching with q_3 to retrieve other trajectories sharing globally similar routes to train with. Fig. 5 illustrates top-4 trajectories that are retrieved by whole pattern matching query when the query trajectory is q_3.

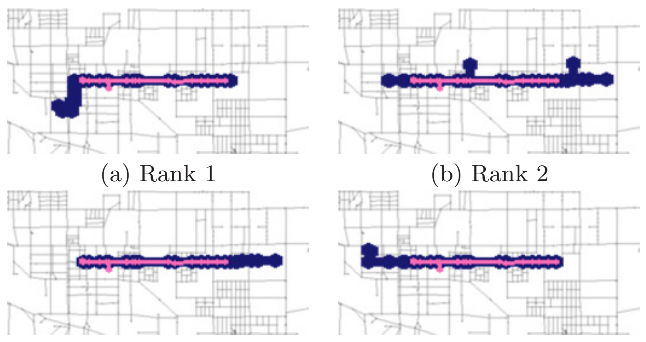


(a) Rank 1 (b) Rank 2
(c) Rank 3 (d) Rank 4

Figure 5: Top-4 trajectories of whole pattern

EXAMPLE 2. ***Subpattern matching***
Suppose that there is a road construction plan, say q_3 in Fig. 4. An administrator plans to alert the users who may

be affected by the construction. Then, he can query a subpattern matching with q_3, which can retrieve users whose trajectories contain q_3 as a subpattern (see Fig. 6).

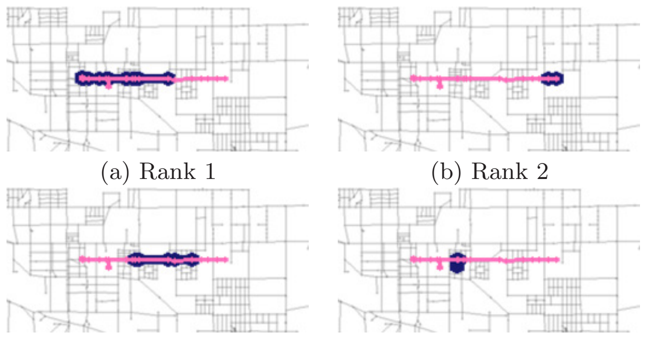


(a) Rank 1 (b) Rank 2

(c) Rank 3 (d) Rank 4

Figure 6: Top-4 trajectories of subpattern

EXAMPLE 3. ***Reverse subpattern matching***
Suppose a user considers a plan for a new bicycle route and he wishes to know information for the route before he begins to ride. Reverse subpattern matching can be useful to find users who already have an experience in some parts of a new route. For example, if q_3 in Fig. 4 is the new bicycle route, trajectories of Fig. 7 are the candidates.



(a) Rank 1 (b) Rank 2

(c) Rank 3 (d) Rank 4

Figure 7: Top-4 trajectories of reverse subpattern

6. ACKNOWLEDGMENTS

This work was supported by Microsoft Research Asia and the National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development.

7. REFERENCES

[1] www.bikely.com.

[2] http://www.census.gov/geo/www/tiger.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proc. SIGMOD*, pages 322–331, 1990.

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

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

[6] S. Chen, B. C. Ooi, K.-L. Tan, and M. A. Nascimento. ST2B-tree: a self-tunable spatio-temporal b+-tree index for moving objects. In *Proc. SIGMOD*, pages 29–42, 2008.

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

[8] H. Hu, J. Xu, and D. L. Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *Proc. SIGMOD*, pages 479–490, 2005.

[9] S. Rasetic, J. Sander, J. Elding, and M. A. Nascimento. A trajectory splitting model for efficient spatio-temporal indexing. In *Proc. VLDB'05*, pages 934–945, 2005.

[10] G. Roh, J. Roh, S. Hwang, and B. Yi. Supporting pattern matching queries over trajectories on road networks. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints), 2010.

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

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