FlexTrack: A System for Querying Flexible Patterns in Trajectory Databases

Marcos R. Vieira[1], Petko Bakalov[2], and Vassilis J. Tsotras[1]

[1]UC Riverside [2]ESRI

`{mvieira,tsotras}@cs.ucr.edu,pbakalov@esri.com`

Abstract. We describe the *FlexTrack* system for querying trajectories using *flexible pattern queries*. Such queries are composed of a sequence of simple spatio-temporal predicates, e.g., range and nearest-neighbors, as well as complex motion pattern predicates, e.g., predicates that contain *variables* and constraints. Users can interactively select spatio-temporal predicates to construct such pattern queries using a hierarchy of regions that partition the spatial domain. Several different query processing algorithms are currently implemented and available in the *FlexTrack* system.

1 Introduction

In this paper we describe *FlexTrack*, a system that allows users to query, in a very intuitive way, trajectory databases using *flexible patterns* [1, 2]. A flexible pattern query (or pattern query for short) is specified over a fixed set of areas that partition the spatial domain and is defined as a combination of predicates that allow the end user to focus on specific parts of the trajectories that are of interest. For example, the pattern query "Find all trajectories that first were in downtown LA, later passed by Santa Monica, and then were closest to LAX" provides a mixture of range and Nearest-Neighbor (NN) predicates that have to be satisfied in the specific order. Essentially, flexible patterns cover that part of the query spectrum between the single predicate spatio-temporal queries, such as the range predicate that covers certain time instances of the trajectory life (e.g. "Find all trajectories that passed by area A at 11pm"), and similarity/clustering based ones, such as extracting similar movement patterns and periodicities from a trajectory archive that cover the whole lifespan of the trajectory (e.g. "Find all trajectories that are similar to a given query trajectory according to some similarity measure").

In order to provide more expressive power, flexible pattern queries can also include *variables* as predicates. An example of a query with a variable is "Find all taxi cabs that visited the same city district twice in the last 1 hour". Here the area of interest is not known in advance but it is specified by its properties (visited twice in the last 1 hour). We term these variable-enabled pattern queries as "flexible" as they provide a powerful way to query trajectories. Both the fixed and variable spatial predicates can express explicit temporal constraints (e.g., "between 10am and 11am") and/or implicit temporal ordering between

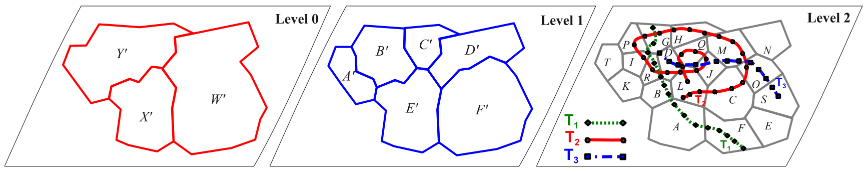


Fig. 1. Example of a set of regions defined using a hierarchy of 3 levels.

them ("anytime later"). Flexible predicate queries can also include "numerical" constraints (NN and their variants) to provide "best fit" capabilities to the query language. Using this general and powerful querying framework, the user can "focus" the search only on the portions/events in a trajectory's lifetime that are of interest.

2 The Flexible Pattern Query Language

In this section we provide the definition of key elements in the *FlexTrack* system, as well as the description of the query language syntax.

A trajectory T_{id} is defined as a list of locations collected for a specific moving object over an ordered sequence of timestamps, and is stored as a sequence of w pairs $\{(ls_1, ts_1), \ldots (ls_w, ts_w)\}$, where $ls_i \in \mathbb{R}^d$ is the object location recorded at timestamp ts_i ($ts_{i-1} < ts_i$). In the *FlexTrack* system, the spatial domain is partitioned by a leveled hierarchy, where at each level l the spatial domain is divided by a fixed set Σ_l of non-overlapping regions, as shown in Figure 1. A region in level l is formed by the union of regions in the previous level $l-1$. Regions correspond to areas of interest (e.g. *school districts*, *airports*) and form the alphabet $\Sigma = \bigcup_l \Sigma_l = \{A, B, C, \ldots\}$. Note the non-overlapping property between regions at a given level (e.g., W", X", Y" in level 0), while regions from different levels can overlap (e.g., regions W" in level 0 and F' in level 1).

In the *FlexTrack* query language, a spatio-temporal predicate $\mathcal{P}$ is defined by a triplet $\langle op, \mathcal{R}[, t]\rangle$, where $\mathcal{R}$ corresponds to a predefined spatial region in Σ or a *variable* in Γ ($\mathcal{R} \in \{\Sigma \cup \Gamma\}$), *op* describes the topological relationship (e.g. *meet*, *overlap*, *inside*) that the trajectory and the spatial region must satisfy over the (optional) time interval t ($t := (t_{from} : t_{to}) \mid t_s \mid t_r$). A predefined spatial region is explicitly specified by the user in the query predicate (e.g. "the convention center"). In contrast, a *variable*, e.g. "@x", denotes an arbitrary region using the symbols in $\Gamma = \{@a, @b, @c, \ldots\}$. Unless otherwise specified, a *variable* takes a single value (instance) from a given level Σ_l (e.g. $@a{=}C$), where the level l is specified in the query. Conceptually, *variables* work as placeholders for explicit spatial regions and can become instantiated (bound to a specific region) during the query evaluation.

Such spatio-temporal predicates $\mathcal{P}$ however cannot be used to specify distance based constraints (e.g., "best-fit" type of queries, like NN, that find trajectories which best match a specified pattern). This is because topological predicates involved are binary in nature and thus cannot capture distance based properties of the trajectories. To solve this problem we introduce the optional $\mathcal{D}$ part of a pattern query $\mathcal{Q}$ which allows us to describe distance-based or other constraints among the *variables* in $\mathcal{S}$ and the predefined regions (for more details, see [1]).

Having defined spatio-temporal predicates and the distance based constraints, we can now define a pattern query $\mathcal{Q} = (\mathcal{S}\ [\cup\ \mathcal{D}])$ as a combination of a sequential pattern $\mathcal{S}$ and (possibly) a set of constraints $\mathcal{D}$, where a trajectory matches $\mathcal{Q}$ if it satisfies both $\mathcal{S}$ and $\mathcal{D}$ parts. Here $\mathcal{S} := \mathcal{S}.\mathcal{S} \mid \mathcal{P} \mid !\mathcal{P} \mid \mathcal{P}^{\#} \mid ?^{+} \mid ?^{*}$ corresponds to a sequence of spatio-temporal predicates, while $\mathcal{D}$ represents a collection of distance functions (e.g. *NN*) and constraints (e.g. @x!=@y, @z={A,D,F}) that may contain regions defined in $\mathcal{S}$. The wild-card "?" is also considered a variable, however it refers to any region without occurring multiple times within a $\mathcal{S}$.

The use of the same set of *variables* in describing both the topological predicates and the numerical conditions provides a very powerful language to query trajectories. To describe a query in *FlexTrack*, the user can use fixed regions for the parts of the trajectory where the behavior should satisfy known (strict) requirements, and *variables* for those sections where the exact behavior is not known but can be described by *variables* and the constraints between them.

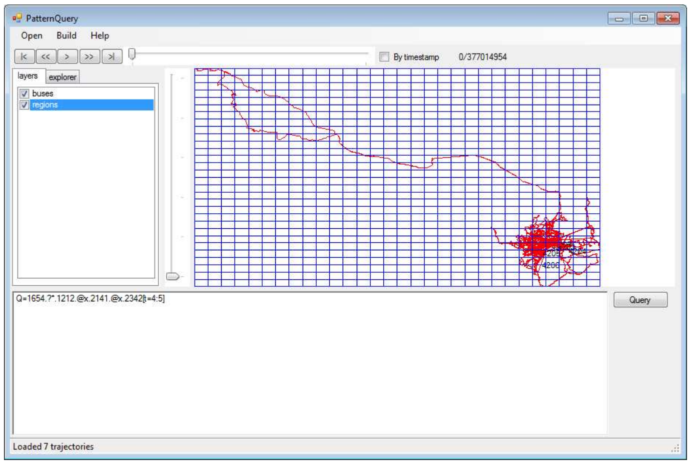
3 Pattern Query Evaluation

We continue with a description of the system architecture, its major components and evaluation algorithms.

In order to efficiently evaluate *flexible pattern queries*, the *FlexTrack* system employs two lightweight index structures in the form of ordered lists that are stored in addition to the raw trajectory data. There is one *region-list* (*R-list*) per region and one *trajectory-list* (*T-list*) per trajectory. The *R-list* $\mathcal{L}_{\mathcal{I}}$ of a given region $\mathcal{I} \in \Sigma$ acts as an inverted index that contains all trajectories that passed by region $\mathcal{I}$. Each entry in $\mathcal{L}_{\mathcal{I}}$ contains a trajectory identifier T_{id}, the time interval (*ts-entry*:*ts-exit*] during which the trajectory was inside $\mathcal{I}$, and a pointer to the *T-list* of T_{id}. Entries in a *R-list* are ordered first by T_{id} and then by *ts-entry.*

The only requirement for the region partitioning is that regions should be non-overlapping. In practice, there may be a difference between the regions presented to the end user as Σ and what is used internally for space partitioning. In the *FlexTrack* system we use a uniform grid to partition the space and we overestimate the regions in Σ by approximating each one of them with the smallest collection of grid cells that completely encloses the region. Because of the overestimation, false positives may be generated from regions that do not completely fit the set of covering grid cells. They, however, can be removed with a verification step using the original trajectory data.

In order to fast prune trajectories that do not satisfy $\mathcal{S}$, the *FlexTrack* system uses the *T-list*, where each trajectory is approximated by the sequence of regions it visited in each level of the partitioning space. A record in the *T-list* of T_{id} contains the region and the time interval (*ts-entry*:*ts-exit*] during which this region was visited by T_{id}, ordered by *ts-entry.* In addition, entries in *T-list* maintain pointers to the *ts-entry* part in the original trajectory data. Given those index



```
Q=1654.?*.1212.@x.2141.@x.2342[t=4:5]
```

```
Loaded 7 trajectories
```

Fig. 2. The main interface of the *FlexTrack* System.

structures available, we propose four different strategies for evaluating flexible pattern queries (for the details on how $\mathcal{D}$ is evaluated, see [1]):

1. *Index Join Pattern* (*IJP*): this method is based on a merge join operation performed over the *R-lists* for every fixed predicate in $\mathcal{S}$. The *IJP* uses the *R-lists* for pruning and the *T-lists* for the *variable* binding;

2. *Dynamic Programming Pattern* (*DPP*): this method performs a subsequence matching between every predicate in $\mathcal{S}$ (including *variables*) and the trajectory approximations stored as the *T-lists*. The *DPP* uses mainly the *T-lists* for the subsequence matching and performs an intersection-based filtering with the *R-lists* to find candidate trajectories based on the fixed predicates in $\mathcal{S}$;

3. *Extended-KMP* (*E-KMP*): this method is similar to *DPP*, but uses the Knuth-Morris-Pratt algorithm [3] to find subsequence matches between the trajectory representations and the query pattern;

4. *Extended-NFA* (*E-NFA*): this is an NFA-based approach to deal with all predicates of our proposed language. This method also performs an intersection-based pruning on the *R-lists* to fast prune trajectories that do not satisfy the fixed spatial predicates in S.

4 Demonstration

For our demonstration we will use the *Trucks* and *Buses* datasets that contain moving object trajectories collected from the greater metropolitan area of Athens, Greece (`www.rtreeportal.org`). The *Trucks* dataset contains 112,203 locations generated from 276 moving objects. The *Buses* dataset has 66,096 locations from 145 moving objects. For the purposes of the demonstration we partition the spatial domain into regions using uniform grid with three levels. The granularity at levels 0, 1 and 2 is, respectively, 100×100, 50×50 and 25×25.

The first step in the query evaluation is to load the trajectory dataset from secondary storage. The next step is to create the index structures (*R-list* and *T-*

list) used by our evaluation algorithms. During this process the users can tune several parameters (e.g. grid size, number of levels) for optimal performance. Using the system main interface, shown in Figure 2, users can visualize the trajectories in the spatial domain for a particular time interval. This property allows users to inspect, navigating in space and time, which regions have high concentration of trajectories. The system also has the property to "replay" the movement of the trajectories timestamp-by-timestamp.

After the data is loaded and the index structures are created, the user can create pattern queries using the Σ alphabet. The user can zoom in/out to select a lower/higher level of interest in the hierarchy. This allows the user to form a query with mixed size predicates where more detailed, lower level regions correspond to areas of particular interest, and less detailed, higher level regions are used otherwise. The user can also select variables or distance-based constraints at any level of the hierarchy. In addition to that, the user can create predicates that contain a set of regions or is defined by a maximum bounding rectangle (i.e. range predicate).

After the user's query $\mathcal{Q}$ is composed using the GUI it is then translated into the system's internal representation, as described in Section 2, and passed to the query engine. The pattern query is then evaluated using one of the four query evaluation algorithms available in the *FlexTrack* system (*IJP*, *DPP*, *E-KMP* or *E-NFA*). The trajectories in the result set are then plotted on the visualization canvas. Users can then zoom in/out and select parts of the trajectories by specifying the time interval of interest. The system also allows users to "replay" the movement of all the trajectories in the result set. Upon request, the system can provide textual description of trajectories using the regions in Σ.

5 Conclusion

This paper describes the *FlexTrack* system, which allows users to intuitively query trajectory databases by specifying complex motion pattern queries. Using the system GUI, users can easily construct those pattern queries that are further translated into a regular expression-like representation, which is then evaluated by the query evaluation module. Because of its expressive power, fast performance and intuitive user interface, the system can be of great help for users that work with large spatio-temporal archives.

Acknowledgements: This research was partially supported by NSF IIS grants 0705916, 0803410 and 0910859. Vieira's work was funded by a CAPES/Fulbright Ph.D fellowship.

References

1. Vieira, M.R., Bakalov, P., Tsotras, V.J.: Querying trajectories using flexible patterns. In: EDBT. (2010) 406–417
2. Vieira, M.R., Martínez, E.F., Bakalov, P., Martínez, V.F., Tsotras, V.J.: Querying spatio-temporal patterns in mobile phone-call databases. In: MDM. (2010) 239–248
3. Knuth, D., Morris, J., Pratt, V.: Fast pattern matching in strings. SIAM J. on Computing (1977)