Processing Location Based Queries through a Streaming Fashion

Rui Zhang

Department of Computing and Information Systems
University of Melbourne, Australia

Email: rui@csse.unimelb.edu.au

Abstract

In the past few years, the wide application of online map applications and location based services has significantly changed the ways of our life. Today, it is typical for us to search for directions or businesses on the Internet and find our ways through cell phones. Lots of the location based services require providing answers to users continuously so that the users can interact with the applications in real time. For example, a tourist may ask for the nearest three restaurants to be reported continuously while traveling in a city, so that he or she may choose to go to one at any time. As the answers may change over time, the continuous nature of the applications poses new challenges on methods for processing the queries efficiently.

In this talk, we look at a few key strategies to process location based queries in a streaming fashion such as incremental and shared computation, safe regions, and time-constraint processing. We illustrate these strategies through the algorithms to several important types of continuous location based queries.

Keywords: Location based queries, stream data

1 Introduction

In the past few years, the wide application of online map applications and location based services has significantly changed the ways of our life. Today, it is typical for us to search for directions or businesses on the Internet and find our ways through cell phones. Lots of the location based services require providing answers to users continuously so that the users can interact with the applications in real time. For example, a tourist may ask for the nearest three restaurants to be reported continuously while traveling in a city, so that he or she may choose to go to one at any time. As the answers may change over time, the continuous nature of the applications poses new challenges on methods for processing the queries efficiently.

In this paper we first review the studies on two components of location based queries: spatial queries and temporal queries. Then we introduce the background of streaming location based queries. We discuss three key techniques in streaming location based queries, including incremental computation and shared computation, safe region and time constraining. We concludes the paper with a discussion on current research trends.

8th Web Information Systems and Applications Conference (WISA), November 4-6, Chongqing, 2011

2 Studies on Traditional Spatial and Temporal Queries

Early studies (before 2000) process spatial and temporal data in separate. Before 1995 some basic types of spatial queries on static data were studies, i.e., the point queries and range queries. During 1995 to 2000, nearest neighbor (NN) queries were studied. In addition, cost models were proposed to estimate the cost of processing spatial queries. Indexing techniques for managing high-dimensional data were also studied. Meanwhile, early studies on temporal data focused on version indexes and time interval indexes. After 2000 more sophisticated types of spatial queries and temporal queries were studies like reverse nearest neighbor queries, spatial join queries, skyline queries and similarity queries on time series. Studies start to process spatial and temporal data together. Point queries, range queries and NN queries on moving objects were studies. Queries on data streams were also studied. Between 2005 and 2008, trajectory data, location selection problems and continuous queries on moving objects were the hot topics. We will discuss research trends on recent years at the end of the paper.

3 Streaming Location based Queries

Streaming queries are characterized by its continuous nature. Query answer may change anytime due to continuous change of the query itself or the data. As a result, prompt answer is important and highly efficient algorithms are in demand.

Different types of changes in streaming spatio-temporal queries may require different techniques to handle. In this paper we discuss there types of changes: (i) static query, data objects moving, (ii) static data objects, query moving, and (iii) both query and data objects moving. Key techniques for handling these different changes are: (i) incremental computation and shared computation, (ii) safe region, and (iii) time constraining.

3.1 Key Technique 1: Incremental Computation and Shared Computation

We use a study on Continuous Retrieval of 3D Objects [9, 21] as an example to illustrate the incremental computation/shared computation technique.

Recent advances in mobile computing have delivered competitive rendering capabilities, which have enabled a new realism of visualizing 3D representation of objects on small computing devices such as cell phones or head-mounted displays. For example, in LifeClipper, a head-mounted display is used to offer virtually enhanced travel experiences for tourists visiting foreign locations. Virtual 3D objects are added to the view according to the current position and viewing direction of the user. While the current status of this particular project requires the tourist

to carry a portable storage device for providing the data, we envision that users should only need to wear a head-mounted display. In our scenarios, clients can obtain the data on the fly through a wireless connection. For example, a tourist can use a mobile device, such as a smartphone, to see the 3D interior details of restaurants along a street without physically entering them.

The data access issue in the application fits into a common client-server model. This model consists of a *mobile client*, a *server*, a large set of *3D objects* located on the server, and a *wireless link* between the client and the server. The server stores information about the 3D objects. The client has a *view* attached to it. At any time, according to the client's location and view direction, the client retrieves all the objects within the range of its view from the server through the wireless link, and then renders the objects in the display. As the client moves, new objects will continuously be retrieved. The process can be viewed as a *continuous window query on 3D object databases*. To guarantee a realistic visual experience, the results in the query window have to be retrieved at a high rate.

The data retrieval yields the highest delay when the client changes its view rapidly. This is because (i) in the same amount of time, there are more objects that is swept by the client when the view moves at a high speed, leading to more objects to be retrieved per time unit from the server, (ii) for a moving client, the usable bandwidth of a connection in a network such as mobile phone networks drop to a fraction of the bandwidth that is available for clients at rest.

The papers [9, 21] provide a systematic solution to this data retrieval problem. The solution is based on the key insight that when the client's view is moving at a high speed, the client is only interested in and capable of absorbing high level information from the environment viewed. Even if we visualize the objects in full details when the client is in fast motion, the user will not be able to see most of the details presented. Therefore, the papers choose to represent objects in multiple resolutions and retrieve only the data necessary to visualize the objects at the required resolutions based on the speed of the movement of the client's view. Specifically, the papers use wavelet representations of 3D objects. Wavelets are ideal for the problem because of the following two advantages: (i) we can easily represent an object in different resolutions using different wavelet coefficients; (ii) for an increased object resolution, we only need to retrieve the difference between the two resolutions, which incurs only incremental costs.

Based on the wavelet representations of 3D objects, the papers solve the problem with the following key techniques:

- Motion-aware data retrieval: (i) representing 3D objects in multiple resolutions through wavelet decomposition, (ii) retrieval of data necessary for a certain resolution, determined by the speed of the change in the view frame, (iii) incremental retrieval of the difference when increasing the resolution.
- Motion-aware buffer management: A pre-fetching and caching strategy based on the view's movement.

3.1.1 Object Movement based Incremental Data Retrieval

3D objects can be approximated by their surfaces using triangular meshes. Let M^j be a triangular mesh representation of the surface of a 3D object at resolution j. An object can be represented in different levels of resolution by a sequence of meshes M^0,M^1,...,M^J, where, M^0 is the *base mesh* and M^J is the *final mesh*.

The wavelet decomposition of a mesh M^J produces a base mesh M^0 and the sets, $\{W_0, W_1, ..., W_{J-1}\}$, of wavelet coefficients. Each W_i contains a set of wavelet coefficients at level i which represent the missing details between mesh M^i and M^{i+1}. Wavelet coefficients describing a 3D object can be used as a selection criteria whether they are necessary or not at a given point of time for a client. A set of coefficients is selected according to the client's region of interest (query window), and the geometrical influences (how much effect a certain coefficient has on the visualization) of coefficients. The geometric influence of a coefficient may be determined by the speed of navigation, the resolution level of the screen, or the terminal's processing power of the client.

In the motion-aware data retrieval scheme, a client uses a function to map the speed of the view to a resolution of 3D objects that are necessary for the visualization at a given point of time. The client can tune this function depending upon its environment, e.g., display size, available bandwidth. Since 3D objects are decomposed and represented in multiple resolutions using wavelets, the client maps the speed to wavelet coefficients for the objects with required resolutions. The clients incrementally retrieve spatial data from a data server. As the client moves, it sends queries to the server with different timestamps. The client only retrieves the data that is not already retrieved by the previous query.

3.1.2 Object Movement based Buffer Management

The buffer management scheme uses a state estimation based motion-prediction scheme to determine the probability distribution of the data to be accessed. It has three stages: (i) estimate the client's path and probabilities of surrounding cell blocks to be visited, (ii) select the list of blocks to be put into the buffer from each of the directions, (iii) retrieve objects from the server for the predicted blocks which are currently not in the client's buffer.

The study uses a cost model for a multi-dimensional non-uniform motion of a client. The cost model for the buffer management scheme has two major parts: (i) latency and (ii) data transfer costs. Assume that the data space is divided into grid-like blocks. When the client visits a new region (i.e., not found in the local buffer), it retrieves a number of blocks from the server and puts it in the local buffer. Thus, the client does not need to contact with the server as long as it remains in the buffered region. The latency in the system can be reduced by lowering the cache misses (i.e., when the data is not found in the local buffer) to a small value. Also, the client has a limited buffer and some of the pre-fetched data may not be actually accessed by the client. Thus, the buffer management scheme for a client should also avoid the retrieval of redundant data to minimize the wasted bandwidth. The study uses the *Kalman filter* to predict future positions of a client from its recent locations and compute the future error covariances to determine the confidence level of those predictions. Based on the covariances and predictions, the study calculates the probabilities of the client to visit neighboring regions.

The buffer management scheme also utilizes the motion of clients to adapt a multi-resolution strategy. The idea is that a client moving at higher speeds buffers more objects with lower resolutions than that of a slowly moving client. The process is similar to the motion-aware continuous data retrieval strategy.

3.2 Key Technique 2: Safe Region

The safe region technique is commonly used in processing continuous kNN queries, which continuously return k NNs for a moving query point. We use two such studies [11, 20] to illustrate the technique.

Consider the following two scenarios. A driver in a GPS-equipped car issues a continuous query to find the nearest gas station while driving in a city. A tourist uses a location-aware mobile device to issue a continuous query for the nearest restaurant while walking to a museum. The queries are sent to a server that processes the queries and returns the answers. In these scenarios, the server has to continuously maintain the answer set which may change depending on the location of the query point. These scenarios are typical examples of *moving k nearest neighbor queries* (M*k*NN). A straightforward way to process a M*k*NN query is using a *sampling-based* method, which processes the M*k*NN query as a kNN query at sampled locations. This method does not provide answers between sampled locations. In order to provide an (almost) continuous answer to the query, a high sampling rate is required, which makes the method inefficient due to the frequent processing of kNN queries. The concept of the *safe region* provides a more effective way to achieve continuous answers to location-based spatial queries. In a safe-region-based method, an answer is returned with a safe region. As long as the query point stays in the safe region, the answer remains the same. When the query point moves out of the safe region, another answer with its associated region is returned. Therefore, a safe-region-based method always (that is, continuously) provides accurate answers without the need for sampling. This approach also requires much less frequent communication between the mobile client and the server.

3.2.1 A Classic Technique - Voronoi Diagram

A classic example of safe-region-based techniques is the *Voronoi Diagram*. The Voronoi Diagram is a well known space decomposition determined by distances to a given discrete set of objects, typically, a set of points. Specifically, the Voronoi Diagram of a set of points $P = \{\vec{p}_1, \vec{p}_2, ..., \vec{p}_n\}$ is defined as a set of cells where each cell $V(\vec{p}_i)$ is a region of space that consists of all points of the data space that are closer to $\vec{p}_i$ than any other point in P.

Processing a 1NN query using the Voronoi Diagram involves: (i) locating which Voronoi cell the query point falls into; and (ii) identifying the associated object. The answer remains valid as long as the query point stays in a Voronoi cell.

The Voronoi Diagram can be generalized to the k^{th}-*order Voronoi Diagram (kVD)*. In a *k*VD, each region is associated with the set of *k* nearest neighbors, termed *kNN set* or *k NNs*, rather than only the nearest one. The *k*VD can handle M*k*NN queries in the same manner as the basic first order VD handles 1NN queries.

The *k*VD has the following shortcomings: (i) Expensive precomputation. The *k*VD requires precomputing all the *k*VD cells and access to all the data points. Both computation and storage costs are high.(ii) No support for dynamically changing *k* values. The *k*VD can only accommodate *k*NN queries with a specific *k* value; the ordered *k*VD can only accommodate *k*NN queries with *k* values no larger than the order of the diagram. As a result, the technique is not suitable for situations where the value of *k* is unknown in advance or can change dynamically.(iii) Inefficient update operations. Many cells have to be recomputed for each insertion or deletion on the dataset.

The expensive precomputation is especially not justified if the query point is confined to a small region of the whole data space. For example, if a car is moving in a small part of a city, then it is unnecessary to compute the *k*VD for the entire city. Furthermore, one may require different *k*VDs for different needs. For example, a driver may need to find a gas station with a restroom facility while another driver needs one with a special type of fuel. Precomputing *k*VDs for all possible scenarios may be prohibitive.

An improved algorithm locally computes a *k*VD cell, which mitigates the precomputation and update problems (shortcomings 1 and 3). However, the algorithm is still relatively expensive and does not address the problem of dynamically changing *k* values (shortcoming 2).

3.2.2 A Novel Technique - V*-Diagram

In the papers [11, 20], a technique called the *V*-Diagram* for M*k*NN queries is proposed. The V*-Diagram has the following key advantages: (i) It requires no precomputation. (ii) It incrementally computes answers and therefore efficiently adapts to changes – such as insertions and deletions of objects, as well as, dynamically changing values of *k*.

The V*-Diagram is based on the safe-region concept, but differs from any previous technique for M*k*NN queries in the following aspect: previous safe-region-based techniques compute safe regions purely based on the data (for example, you can compute the *k*VD without referring to the query point); *the V*-Diagram computes safe regions based on not only the data objects, but also the query point and the current knowledge of the search space.* This is one of the main novelties of the technique. By doing so, both computation and data retrieval of the V*-Diagram are more economical than those of the other techniques.

In the V*-Diagram, the $(k+x)$ NNs of the moving query point q are maintained, where x is the number of auxiliary objects to help the V*-Diagram work effectively. The known region bounds the $k + x$ NN objects of q. For each object, a safe region w.r.t a data point p is computed where the movement of the query object q will not cause p to be excluded from the $k + x$ NNs of q. Further, the fixed-rank region is computed to guarantee that the closeness order of the data objects to q does not change. The two types of regions together defines a region where the $k + x$ NN data objects of q and their closeness order to q do not change. As an experimental study shows, this region performs very well in reducing the computation and the V*-Diagram outperforms the best existing technique by two orders of magnitude.

3.3 Key Technique 3: Time Constraining

Time constraining is another technique for continuous query processing. A typical application of this technique is [8, 25], where the continuous intersection join query is studied.

In many previous studies, moving objects such as mobile phone users or vehicles have been modeled as points. The reason is that the objects' extents are negligible compared to the size of the whole region of interest. For example, ignoring the extents of vehicles does not hurt much if we want to have an idea of how many cars are in the central business district by performing a window query. However, there are also many scenarios where the extents of objects cannot be neglected. For example, in a scenario where we monitor the movements of vessels and storms on the sea, every vessel has an alert zone and there are regions in the sea covered by storms. Navigation systems on vessels should continuously report those vessels whose alert zones are intersected by the storm regions, so that the vessels can be alerted to the possible impact. For another example, in a massively multiplayer online game (MMOG), two teams of players are in a battle. Each player has a sector-shaped region in front of her as her attack range. The MMOG server needs to continuously keep track of the intersection among players' attack ranges at about the graphics frame rate, so that combats between players can be processed and then rendered in almost real time. The high frequency of intersection result updates brings in a

critical challenge to the server's performance and the immediate requirement of new query processing techniques. The current systems only allow dozens of players and can not handle hundreds of thousands players in a battle. Military simulations have similar requirements as MMOGs do. In a military simulation, there can be up to 100,000 objects that are moving and a primitive data management requirement is *interest management*, which is actually an intersection join of the interest ranges of objects.

The above applications represent the execution of *continuous intersection join query over moving objects (with nonzero extents) with updates*, which monitors two sets of moving objects and reports every pair of intersecting objects, one from each of the two sets, for every timestamp. Here, updates refer to changes in the spatial attributes (position or velocity) of the moving objects. They cause changes in the result of the query. In a moving object management system, the continuous intersection join result will update very frequently. It is this frequent answer update that differs continuous intersection join query from traditional spatial join queries.

3.3.1 A Naive Join Algorithm - NaiveJoin

Processing a continuous join (we omit "intersection" when the context is clear) consists of two phases: the initial join and the maintenance. For the initial join, we can use a naive algorithm described below to compute all the possible join pairs from now to the infinite timestamp. For the maintenance, whenever there is an object update, we need to perform an *answer update* as follows. First, we remove all the pairs containing the updated object from the current result; then we join the object with the other dataset (still using the naive algorithm) from the current timestamp to the infinite timestamp and the newly found pairs are added to the current join result. Next, we give the naive algorithm for computing join pairs.

Each dataset is indexed by a TPR-tree (tr_A and tr_B for A and B, respectively). The basic idea is to use the bounding relationship between a node of the TPR-tree and the entries inside it. Let N_A (N_B) be a node from tr_A (tr_B). If N_A does not intersect N_B, then none of the entries in the sub-tree rooted at N_A could intersect any of the entries in the sub-tree rooted at N_B, therefore we need not visit the sub-trees. Otherwise, there could be intersections between entries in the sub-trees and we should check the entries in them. This intersection-or-not checking is performed recursively on both trees in a top-down manner, until all possible intersections are explored. It is a synchronous traversal on both trees. This algorithm is named *NaiveJoin*.

3.3.2 An Adapted Join Algorithm - ETP-Join

An extended algorithm called *ETP-Join* may be used to process the continuously intersection join query as follows.

First, TP-Join is run to obtain the current answer and the next event. As time goes to the next event and the result changes, an answer update is performed by running TP-Join to get the new next event (no need to search for the new current answer since they can be computed from the previous answer and the event). When there is an update on object O, an answer update is also performed by traversing the tree to find the object's influence time $T_{INF}(O)$. If $T_{INF}(O)$ is before the current expiry time, then $T_{INF}(O)$ becomes the current expiry time and O becomes the next event; otherwise, the update is simply ignored (the tree has already been traversed). By this means, join pairs can be obtained for all the time.

3.3.3 Time Constraint Processing Algorithm - TC-Join

ETP-Join has to run TP-Join frequently because updates and changes of results are frequent. The problem of ETP-Join is computing the result for *too short* a time interval in each run, i.e., having a too short processing time interval. NaiveJoin has a high computation cost per run because it returns the answer up to the infinite timestamp. The problem of NaiveJoin is computing the result for *too long* a time interval in each run, i.e., having a too long processing time interval. This motivates us to optimize query processing in the time domain. The crux of the problem is to choose a "good" processing time interval for each join run. Next, we describe the concept of time-constrained (TC) processing, based on which the MTB-Join algorithm was proposed to achieve a similar per update cost to that of the ETP-Join algorithm while retain the same small number of answer updates as that of the NaiveJoin algorithm.

The key insight is that the join result between any two objects only needs to be valid until the next update on any of the two objects. Actually, if an object issues an update, all the predictions about this object's intersection with other objects in the future may become invalid immediately. We have to perform a join between the updated object with the other dataset anyway. In other words, an update of an object invalidates the object's join result starting from the update timestamp to the future. Therefore, an ideal time interval for computing join pairs for an object is from the current timestamp to the object's next-update timestamp. This ideal case is impossible in reality because we could not know in advance an object's next-update timestamp. However, fortunately we have an upper bound of an object's next-update timestamp, i.e., T_M from now. T_M is the maximum update interval of the objects. For an object, we only need to find its join pairs with the other dataset during the period $[t_c, t_c + T_M]$. Before $t_c + T_M$, this object will have to issue an update and we will then find its join pairs with the other dataset again for another T_M period. By this means, we can obtain correct answers for this object continuously. An algorithm that processes the join from t_c to $t_c + T_M$ is named TC-Join.

The TC-Join algorithm is further improved by grouping objects into time buckets based on their latest updates; therefore the set of objects in each time bucket (except the last one) has a smaller latest update time than that of the whole dataset. To group objects into time buckets for TPR-trees, a similar idea as used in the B^x-tree can be exploited. Particularly, we divide the time axis into equi-length time buckets; for each time bucket, a TPR-tree is used to index all the objects whose latest update time fall in the bucket. This results in a group of TPR-trees based on multiple time buckets, which we call the *MTB-tree*. Updates in the MTB-tree are handled as follows. When an object updates, we first identify which time bucket the object is currently indexed from its last update timestamp. We delete the object from the TPR-tree in that time bucket and insert it into the current TPR-tree. The cost of updating an object in the MTB-tree is almost the same as the cost of updating an object in a regular TPR-tree, because even if the objects are indexed by a regular TPR-tree, an object update still involves deleting an object from the tree first and then insert the updated object. The only overhead of an MTB-tree object update compared to a TPR-tree object update is identifying the time bucket in which the updated object is currently indexed, which is done by a simple modulus operation and hence the overhead is negligible. Typically the length of a time bucket can divide T_M exactly.

4 Conclusion

In summary, we have revisit three key techniques for streaming spatio-temporal queries. Besides the above problems, application of these three techniques are also found in the following papers [1-5, 13, 14].

In the last few years, studies in spatio-temporal queries have witnessed some new trends. Location optimization problems [26] and queries on continuous queries on moving objects have become popular [16, 23, 24]. Studies on temporal databases have turned to very large and streaming data [10, 15, 17]. Road networks and trajectories attract more and more research interests [6, 7, 12, 18, 19, 22].

References

[1] Glenn S. Iwerks, Hanan Samet, Kenneth P. Smith: Maintenance of Spatial Semijoin Queries on Moving Points. International Conference on Very Large Data Bases (VLDB) 2004.

[2] Hyung-Ju Cho, Chin-Wan Chung: An Efficient and Scalable Approach to CNN Queries in a Road Network. International Conference on Very Large Data Bases (VLDB) 2005.

[3] Mohamed F. Mokbel, Xiaopeng Xiong, Walid G. Aref: SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. ACM SIGMOD International Conference on Management of Data (SIGMOD) 2004.

[4] Haibo Hu, Jianliang Xu, Dik Lun Lee: A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. ACM SIGMOD International Conference on Management of Data (SIGMOD) 2005.

[5] Kyriakos Mouratidis, Marios Hadjieleftheriou, Dimitris Papadias: Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. ACM SIGMOD International Conference on Management of Data (SIGMOD) 2005.

[6] Jae-Gil Lee, Jiawei Han, Kyu-Young Whang: Trajectory clustering: a partition-and-group framework. ACM SIGMOD International Conference on Management of Data (SIGMOD) 2007.

[7] Hanan Samet, Jagan Sankaranarayanan, Houman Alborzi: Scalable network distance browsing in spatial databases. ACM SIGMOD International Conference on Management of Data (SIGMOD) 2008.

[8] Rui Zhang, Dan Lin, Kotagiri Ramamohanarao, Elisa Bertino. Continuous Intersection Joins over Moving Objects. Proceedings of the 24th International Conference on Data Engineering (ICDE) 2008.

[9] Mohammed Eunus Ali, Rui Zhang, Egemen Tanin, Lars Kulik. A Motion-Aware Approach to Continuous Retrieval of 3D Objects. Proceedings of the 24th International Conference on Data Engineering (ICDE) 2008.

[10] David Lomet, Mingsheng Hong, Rimma Nehme, Rui Zhang: Transaction Time Indexing with Version Compression. Proceedings of the VLDB Endowment (PVLDB), 1(1), 870-881, 2008.

[11] Sarana Nutanong, Rui Zhang, Egemen Tanin, Lars Kulik: The V*-Diagram: A Query Dependent Approach to Moving KNN Queries. Proceedings of the VLDB Endowment (PVLDB), 1(1), 1095-1106, 2008.

[12] Reza Sherkat, Davood Rafiei: On efficiently searching trajectories and archival data for historical similarities. Proceedings of the VLDB Endowment (PVLDB), 1(1), 896-908, 2008.

[13] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, Jeffrey Xu Yu: Monitoring path nearest neighbor in road networks. ACM SIGMOD International Conference on Management of Data (SIGMOD) 2009.

[14] Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Wei Wang, Wenjie Zhang. Lazy Updates: An Efficient Technique to Continuously Monitoring Reverse kNN Queries. Proceedings of the VLDB Endowment (PVLDB), 2(1): 1138-1149, 2009.

[15] Xiaoyan Liu, Xindong Wu, Huaiqing Wang, Rui Zhang, James Bailey, Kotagiri Ramamohanarao. Mining Distribution Change in Stock Order Streams. Proceedings of the 26th International Conference on Data Engineering (ICDE), pp. 105-108, 2010.

[16] Cui Yu, Rui Zhang, Yaochun Huang, Hui Xiong: High-dimensional kNN joins with incremental updates. GeoInformatica, 1 (14), 55-82, 2010.

[17] Rui Zhang, Martin Stradling. The HV-tree: a Memory Hierarchy Aware Version Index. Proceedings of the VLDB Endowment (PVLDB), 3(1), 397-408, 2010.

[18] Zhenhui Li, Bolin Ding, Jiawei Han, Roland Kays: Swarm: Mining Relaxed Temporal Moving Object Clusters. Proceedings of the VLDB Endowment (PVLDB), 3(1-2), 723-734 , 2010.

[19] Xin Cao, Gao Cong, Christian Jensen: Mining Significant Semantic Locations From GPS Data. Proceedings of the VLDB Endowment (PVLDB), 3(1-2), 1009-1020 , 2010.

[20] Sarana Nutanong, Rui Zhang, Egemen Tanin, Lars Kulik. Analysis and Evaluation of V*-kNN: An Efficient Algorithm for Moving kNN Queries. VLDB Journal, 19(3): 307-332, 2010.

[21] Mohammed Eunus Ali, Egemen Tanin, Rui Zhang, Lars Kulik. A Motion-Aware Approach for Efficient Evaluation of Continuous Queries on 3D Object Databases. VLDB Journal, 19(5): 603-632, 2010.

[22] Abdullah Mueen, Suman Nath, Jie Liu: Fast approximate correlation for massive time-series data. ACM SIGMOD International Conference on Management of Data (SIGMOD) 2010.

[23] Sarana Nutanong, Egemen Tanin, Rui Zhang. Incremental Evaluation of Visible Nearest Neighbor Queries. IEEE Transactions on Knowledge & Data Engineering (TKDE), 22(5): 665-681, 2010.

[24] Rui Zhang, H. V. Jagadish, Bing Tian Dai, Kotagiri Ramamohanarao. Optimized Algorithms for Predictive Range and KNN Queries on Moving Objects. Information Systems, 35(8): 911-932, 2010.

[25] Rui Zhang, Jianzhong Qi, Dan Lin, Wei Wang, Raymond Chi-Wing Wong. A Highly Optimized Algorithm for Continuous Intersection Join Queries over Moving Objects. VLDB Journal, 21(4): 561-586, 2012.

[26] Jin Huang, Zeyi Wen, Jianzhong Qi, Rui Zhang, Jian Chen, Zhen He. Top-k Most Influential Locations Selection. Proceedings of the 20th ACM international conference on Information and knowledge management (CIKM), Pages 2377-2380 , 2011.