2009

Continuous Monitoring of Spatial Queries

Kyriakos MOURATIDIS
Singapore Management University, kyriakos@smu.edu.sg

DOI: https://doi.org/10.1007/978-0-387-39940-9_82

Published in Encyclopedia of Database Systems. New York: Springer. Pages 479-484.
http://dx.doi.org/10.1007/978-0-387-39940-9_82

CONTINUOUS MONITORING OF SPATIAL QUERIES

Kyriakos Mouratidis
School of Information Systems
Singapore Management University
80 Stamford Road, 178902 Singapore
http://www.mysmu.edu/faculty/kyriakos/

SYNONYMS

Spatio-temporal stream processing

DEFINITION

A continuous spatial query runs over long periods of time and requests constant reporting of its result as the data dynamically change. Typically, the query type is range or nearest neighbor (NN), and the assumed distance metric is the Euclidean one. In general, there are multiple queries being processed simultaneously. The query points and the data objects move frequently and arbitrarily, i.e., their velocity vectors and motion patterns are unknown. They issue location updates to a central server, which processes them and continuously reports the current (i.e., updated) query results. Consider, for example, that the queries correspond to vacant cabs, and that the data objects are pedestrians that ask for a taxi. As cabs and pedestrians move, each free taxi driver wishes to know his/her closest client. This is an instance of continuous NN monitoring. Spatial monitoring systems aim at minimizing the processing time at the server and/or the communication cost incurred by location updates. Due to the time-critical nature of the problem, the data are usually stored in main memory to allow fast processing.

HISTORICAL BACKGROUND

The first algorithms in the spatial database literature process snapshot (i.e., one-time) queries over static objects. They assume disk-resident data and utilize an index (e.g., an R-tree) to restrict the search space and reduce the I/O cost. Subsequent research considered spatial queries in client-server architectures. The general idea is to provide the user with extra information (along with the result at query-time) in order to reduce the number of subsequent queries as he/she moves (see entry Nearest Neighbor Query). These methods assume that the data objects are either static or moving linearly with known velocities. Due to the wide availability of positioning devices and the need for improved location-based services, the research focus has recently shifted to continuous spatial queries. In contrast with earlier assumed contexts, in this setting (i) there are multiple queries being evaluated simultaneously, (ii) the query results are continuously updated, and (iii) both the query points and the data objects move unpredictably.

SCIENTIFIC FUNDAMENTALS

The first spatial monitoring method is called *Q-index* [13] and processes static range queries. Based on the observation that maintaining an index over frequently moving objects is very costly, *Q-index* indexes the queries instead of the objects. In particular, the monitored ranges are organized by an R-tree, and moving objects probe this tree to find the queries that they influence. Additionally, *Q-index* introduces the concept of *safe regions* to reduce the number of location updates. Specifically, each object p is assigned a circular or rectangular region, such that p needs to issue an update only if it exits this area (because, otherwise, it does not influence the result of any query). Figure 1 shows an example, where the current result of query q_1 contains object p_1, that of q_2 contains p_2, and the results of q_3, q_4, and q_5 are empty. The

safe regions for p_1 and p_4 are circular, while for p_2 and p_3 they are rectangular. Note that no query result can change unless some objects fall outside their assigned safe regions. Kalashnikov et al. [4] show that a grid implementation of *Q-index* is more efficient (than R-trees) for main memory evaluation.

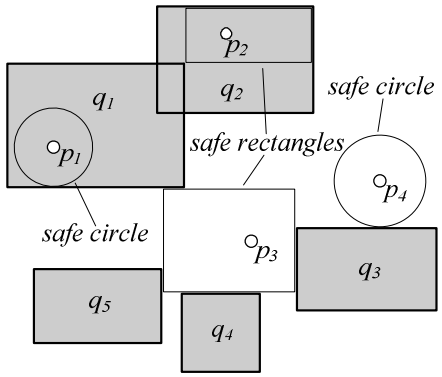


Figure 1 Circular and rectangular safe regions

Monitoring Query Management (MQM) [1] and *MobiEyes* [2] also monitor range queries. They further exploit the computational capabilities of the objects to reduce the number of updates and the processing load of the server. In both systems, the objects store locally the queries in their vicinity and issue updates to the server only when they cross the boundary of any of these queries. To save their limited computational capabilities, the objects store and monitor only the queries they may affect when they move. MQM and *MobiEyes* employ different strategies to identify these queries. The former applies only to static queries. The latter can also handle moving ones, making however the assumption that they move linearly with fixed velocity.

Mokbel et al. [7] present *Scalable INcremental hash-based Algorithm* (SINA), a system that monitors both static and moving ranges. In contrast with the aforementioned methods, in SINA the objects do not perform any local processing. Instead, they simply report their locations whenever they move, and the objective is to minimize the processing cost at the server. SINA is based on *shared execution* and *incremental evaluation*. Shared execution is achieved by implementing query evaluation as a spatial join between the objects and the queries. Incremental evaluation implies that the server computes only updates (i.e., object inclusions/exclusions) over the previously reported answers, as opposed to re-evaluating the queries from scratch.

The above algorithms focus on ranges, and their extension to NN queries is either impossible or non-trivial. In the following we discuss algorithms that target NN monitoring. Hu et al. [3] extend the safe region technique to NN queries; they describe a method that computes and maintains rectangular safe regions subject to the current query locations and *k*NN results. Mouratidis et al. [11] propose *Threshold-Based algorithm* (TB), also aiming at communication cost reduction. To suppress unnecessary location updates, in TB the objects monitor their distance from the queries (instead of safe regions). Consider the example in Figure 2, and assume that q is a continuous 3-NN query (i.e., $k = 3$). The initial result contains p_1, p_2, p_3. TB computes three thresholds (t_1, t_2, t_3) which define a range for each object. If every object's distance from q lies within its respective range, the result of the query is guaranteed to remain unchanged. Each threshold is set in the middle of the distances of two consecutive objects from the query. The distance range for p_1 is $[0, t_1)$, for p_2 is $[t_1, t_2)$, for p_3 is $[t_2, t_3)$, and for p_4, p_5 is $[t_3, \infty)$. Every object is aware of its distance range, and when there is a boundary violation, it informs the server about this event. For instance, assume that p_1, p_3, and p_5 move to positions p'_1, p'_3 and p'_5, respectively. Objects p_3 and p_5 compute their new distances from q, and avoid sending an update since they still lie in their permissible ranges. Object p_1, on the other hand, violates its threshold and updates its position to the server. Since the order between the first two NNs may have changed, the server requests for the current location of p_2, and

updates accordingly the result and threshold t_1. In general, TB processes all updates issued since the last result maintenance, and (if necessary) it decides which additional object positions to request for, updates the k NNs of q, and sends new thresholds to the involved objects.

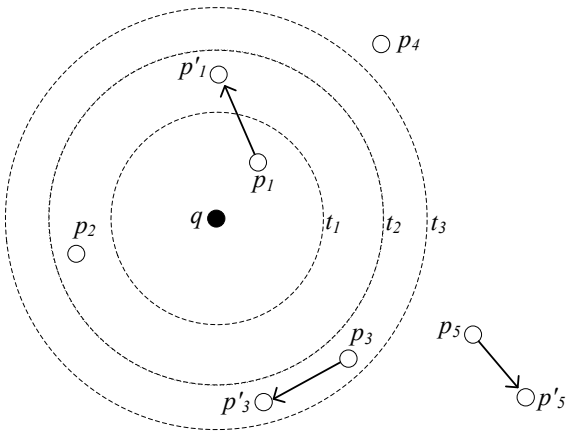
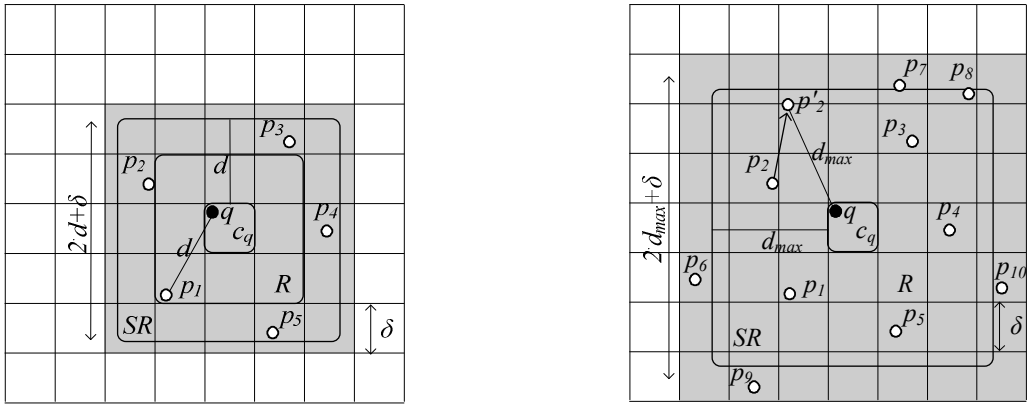


Figure 2 TB example (k = 3)

All the following methods aim at minimizing the processing time. Koudas et al. [6] describe *aDaptive Indexing on Streams by space-filling Curves* (DISC), a technique for e-approximate kNN queries over streams of multi-dimensional points. The returned (e-approximate) k^{th} NN lies at most e distance units farther from q than the actual k^{th} NN of q. DISC partitions the space with a regular grid of granularity such that the maximum distance between any pair of points in a cell is at most e. To avoid keeping all arriving data in the system, for each cell c it maintains only K points and discards the rest. It is proven that an exact kNN search in the retained points corresponds to a valid ekNN answer over the original dataset provided that $k \leq K$. DISC indexes the data points with a B-tree that uses a space filling curve mechanism to facilitate fast updates and query processing. The authors show how to adjust the index to: (i) use the minimum amount of memory in order to guarantee a given error bound e, or (ii) achieve the best possible accuracy, given a fixed amount of memory. DISC can process both snapshot and continuous ekNN queries.

Yu et al. [17] propose a method, hereafter referred to as YPK-CNN, for continuous monitoring of exact kNN queries. Objects are stored in main memory and indexed with a regular grid of cells with size $\delta \times \delta$. YPK-CNN does not process updates as they arrive, but directly applies them to the grid. Each NN query installed in the system is re-evaluated every T time units. When a query q is evaluated for the first time, a two-step NN search technique retrieves its result. The first step visits the cells in an iteratively enlarged square R around the cell c_q of q until k objects are found. Figure 3a shows an example of a single NN query where the first candidate NN is p_1 with distance d from q; p_1 is not necessarily the actual NN since there may be objects (e.g., p_2) in cells outside R with distance smaller than d. To retrieve such objects, the second step searches in the cells intersecting the square SR centered at c_q with side length $2 \cdot d + \delta$, and determines the actual kNN set of q therein. In Figure 3a, YPK-CNN processes p_1 up to p_5 and returns p_2 as the actual NN. The accessed cells appear shaded.

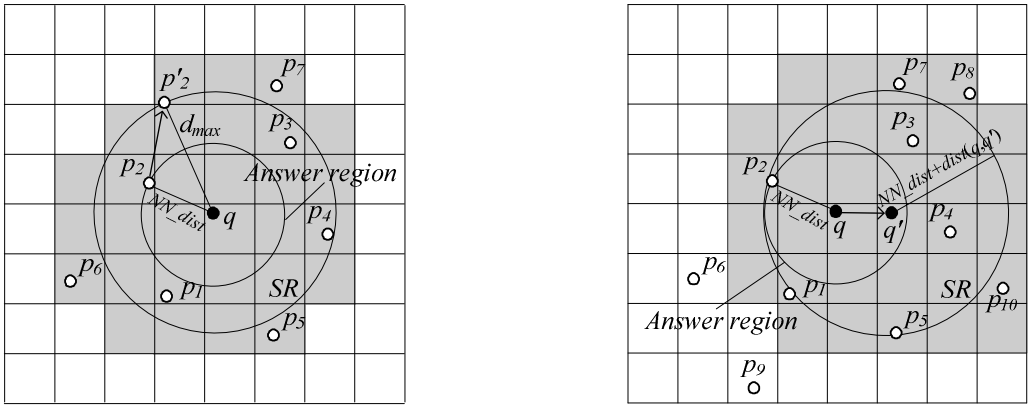


(a) NN computation (b) Update handling

Figure 3 YPK-CNN examples

When re-evaluating an existing query q, YPK-CNN makes use of its previous result in order to restrict the search space. In particular, it computes the maximum distance d_{max} among the current locations of the previous NNs (i.e., d_{max} is the distance of the previous neighbor that currently lies furthest from q). The new *SR* is a square centered at c_q with side length $2 \cdot d_{max} + \delta$. In Figure 3b, assume that the current NN p_2 of q moves to location p'_2. Then, the rectangle defined by $d_{max} = dist(p'_2,q)$ is guaranteed to contain at least one object (i.e., p_2). YPK-CNN collects all objects (p_1 up to p_{10}) in the cells intersecting *SR* and identifies p_1 as the new NN. Finally, when a query q changes location, it is handled as a new one (i.e., its NN set is computed from scratch).

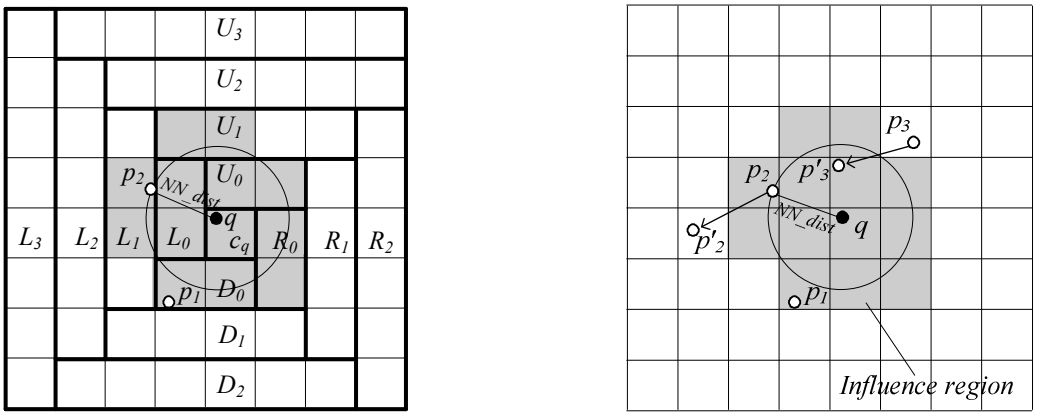
Xiong et al. [16] propose *Shared Execution Algorithm for Continuous NN queries* (SEA-CNN). SEA-CNN focuses exclusively on monitoring the NN changes, without including a module for the first-time evaluation of an arriving query q (i.e., it assumes that the initial result is available). Objects are stored in secondary memory, indexed with a regular grid. The *answer region* of a query q is defined as the circle with center q and radius *NN_dist* (where *NN_dist* is the distance of the current k^{th} NN). Book-keeping information is stored in the cells that intersect the answer region of q to indicate this fact. When updates arrive at the system, depending on which cells they affect and whether these cells intersect the answer region of the query, SEA-CNN determines a circular search region *SR* around q, and computes the new kNN set of q therein. To determine the radius r of *SR*, the algorithm distinguishes the following cases: (i) If some of the current NNs move within the answer region or some outer objects enter the answer region, SEA-CNN sets $r = NN\_dist$ and processes all objects falling in the answer region in order to retrieve the new NN set. (ii) If any of the current NNs moves out of the answer region, processing is similar to YPK-CNN; i.e., $r = d_{max}$ (where d_{max} is the distance of the previous NN that currently lies furthest from q), and the NN set is computed among the objects inside *SR*. Assume that in Figure 4a the current NN p_2 issues an update reporting its new location p'_2. SEA-CNN sets $r = d_{max} = dist(p'_2,q)$, determines the cells intersecting *SR* (these cells appear shaded), collects the corresponding objects (p_1 up to p_7), and retrieves p_1 as the new NN. (iii) Finally, if the query q moves to a new location q', then SEA-CNN sets $r = NN\_dist + dist(q,q')$, and computes the new kNN set of q by processing all the objects that lie in the circle centered at q' with radius r. For instance, in Figure 4b the algorithm considers the objects falling in the shaded cells (i.e., objects from p_1 up to p_{10} except for p_6 and p_9) in order to retrieve the new NN (p_4).



(a) NN movement (b) Query movement

Figure 4 SEA-CNN update handling examples

Mouratidis et al. [9] propose another NN monitoring method, termed *Conceptual Partitioning Monitoring* (CPM). CPM assumes the same system architecture and uses similar indexing and book-keeping structures as YPK-CNN and SEA-CNN. When a query q arrives at the system, the server computes its initial result by organizing the cells into conceptual rectangles based on their proximity to q. Each rectangle *rect* is defined by a *direction* and a *level number*. The direction is U, D, L, or R (for up, down, left and right), and the level number indicates how many rectangles are between *rect* and q. Figure 5a illustrates the conceptual space partitioning around the cell c_q of q. If $mindist(c,q)$ is the minimum possible distance between any object in cell c and q, the NN search considers the cells in ascending $mindist(c,q)$ order. In particular, CPM initializes an empty heap H and inserts (i) the cell of q with key equal to 0, and (ii) the level zero rectangles for each direction DIR with key $mindist(DIR_0,q)$. Then, it starts de-heaping entries iteratively. If the de-heaped entry is a cell, it examines the objects inside and updates accordingly the NN set (i.e., the list of the k closest objects found so far). If the de-heaped entry is a rectangle DIR_{lvl}, it inserts into H (i) each cell $c \in DIR_{lvl}$ with key $mindist(c,q)$ and (ii) the next level rectangle DIR_{lvl+1} with key $mindist(DIR_{lvl+1},q)$. The algorithm terminates when the next entry in H (corresponding either to a cell or a rectangle) has key greater than the distance NN_dist of the k^{th} NN found. It can be easily verified that the server processes only the cells that intersect the circle with center at q and radius equal to NN_dist. This is the minimal set of cells to visit in order to guarantee correctness. In Figure 5a, the search processes the shaded cells and returns p_2 as the result.



(a) NN computation (b) Update handling

Figure 5 CPM examples

The encountered cells constitute the *influence region* of q, and only updates therein can affect the current result. When updates arrive for these cells, CPM monitors how many objects enter or leave the circle

centered at q with radius *NN_dist*. If the *outgoing* objects are more than the *incoming* ones, the result is computed from scratch. Otherwise, the new NN set of q can be inferred by the previous result and the update information, without accessing the grid at all. Consider the example of Figure 5b, where p_2 and p_3 move to positions p'_2 and p'_3, respectively. Object p_3 moves closer to q than the previous *NN_dist* and, therefore, CPM replaces the outgoing NN p_2 with the incoming p_3. The experimental evaluation in [9] shows that CPM is significantly faster than YPK-CNN and SEA-CNN.

KEY APPLICATIONS

Location-based Services

The increasing trend of embedding positioning systems (e.g., GPS) in mobile phones and PDAs has given rise to a growing number of location-based services. Many of these services involve monitoring spatial relationships among mobile objects, facilities, landmarks, etc. Examples include location-aware advertising, enhanced 911 services, and mixed-reality games.

Traffic Monitoring

Continuous spatial queries find application in traffic monitoring and control systems, such as on-the-fly driver navigation, efficient congestion detection and avoidance, as well as dynamic traffic light scheduling and toll fee adjustment.

Security Systems

Intrusion detection and other security systems rely on monitoring moving objects (pedestrians, vehicles, etc.) around particular areas of interest or important people.

FUTURE DIRECTIONS

Future research directions include other types of spatial queries (e.g., reverse nearest neighbor monitoring [15, 5]), different settings (e.g., NN monitoring over sliding windows [10]), and alternative distance metrics (e.g., NN monitoring in road networks [12]). Similar techniques and geometric concepts to the ones presented above also apply to problems of a non-spatial nature, such as continuous skyline [14] and top-k queries [8].

EXPERIMENTAL RESULTS

The methods described above are experimentally evaluated and compared with alternative algorithms in the corresponding reference.

CROSS REFERENCES

B+-tree
Nearest Neighbor Query
R-tree (and family)
Reverse Nearest Neighbor Query
Road Networks
Space Filling Curves for Query Processing

RECOMMENDED READING

[1] Cai, Y., Hua, K., Cao, G. Processing Range-Monitoring Queries on Heterogeneous Mobile Objects. *MDM*, 2004.

[2] Gedik, B., Liu, L. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. *EDBT*, 2004.

[3] Hu, H., Xu, J., Lee, D. A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. *SIGMOD*, 2005.

[4] Kalashnikov, D., Prabhakar, S., Hambrusch, S. Main Memory Evaluation of Monitoring Queries Over Moving Objects. *Distributed and Parallel Databases*, (15)2: 117-135, 2004.

[5] Kang, J., Mokbel, M., Shekhar, S., Xia, T., Zhang, D. Continuous Evaluation of Monochromatic and Bichromatic Reverse Nearest Neighbors. *ICDE*, 2007.

[6] Koudas, N., Ooi, B., Tan, K., Zhang, R. Approximate NN queries on Streams with Guaranteed Error/performance Bounds. *VLDB*, 2004.

[7] Mokbel, M., Xiong, X., Aref, W. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. *SIGMOD*, 2004.

[8] Mouratidis, K., Bakiras, S., Papadias, D. Continuous Monitoring of Top-*k* Queries over Sliding Windows. *SIGMOD*, 2006.

[9] Mouratidis, K., Hadjieleftheriou, M., Papadias, D. Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. *SIGMOD*, 2005.

[10] Mouratidis, K., Papadias, D. Continuous Nearest Neighbor Queries over Sliding Windows. *IEEE TKDE*, 19(6): 789-803, 2007.

[11] Mouratidis, K., Papadias, D., Bakiras, S., Tao, Y. A Threshold-based Algorithm for Continuous Monitoring of *k* Nearest Neighbors. *IEEE TKDE*, 17(11): 1451-1464, 2005.

[12] Mouratidis, K., Yiu, M., Papadias, D., Mamoulis, N. Continuous Nearest Neighbor Monitoring in Road Networks. *VLDB*, 2006.

[13] Prabhakar, S., Xia, Y., Kalashnikov, D., Aref, W., Hambrusch, S. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Transactions on Computers*, 51(10): 1124-1140, 2002.

[14] Tao, Y., Papadias, D. Maintaining Sliding Window Skylines on Data Streams. *IEEE TKDE*, 18(3): 377–391, 2006.

[15] Xia, T., Zhang, D. Continuous Reverse Nearest Neighbor Monitoring. *ICDE*, 2006.

[16] Xiong, X., Mokbel, M., Aref, W. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. *ICDE*, 2005.

[17] Yu, X., Pu, K., Koudas, N. Monitoring K-Nearest Neighbor Queries Over Moving Objects. *ICDE*, 2005.

NOTICE: