CONTINUOUS RANGE QUERY PROCESSING FOR NETWORK CONSTRAINED MOBILE OBJECTS

Dragan Stojanovic, Slobodanka Djordjevic-Kajan
Department of Computer Science, University of Nis
Aleksandra Medvedeva 14, 18000 Nis, SERBIA and MONTENEGRO

Apostolos N. Papadopoulos, Alexandros Nanopoulos
Department of Informatics, Aristotle University
54124, Thessaloniki, GREECE

Keywords: Mobile objects, location based services, continuous range query, query processing.

Abstract: In contrast to regular queries that are evaluated only once, a continuous query remains active over a period of time and has to be continuously evaluated to provide up to date results. We propose a method for continuous range query processing for different types of queries, characterized by mobility of objects and/or queries which follow paths in an underlying spatial network. The method assumes an available 2D indexing scheme for indexing spatial network data. An appropriately extended R*-tree provides matching of queries and objects according to their locations on the network or their network routes. The method introduces an additional pre-refinement step which generates main memory data structures to support efficient, incremental reevaluation of continuous range queries in periodically performed refinement steps.

1 INTRODUCTION

Advances in wireless communication technologies, mobile positioning and Internet-enabled mobile devices have given rise to a new class of mobile applications and services. Location-based services (LBS) deliver geo-information and geo-processing services to the users according to their current location, or locations of the objects of their interests. Such services, like automatic vehicle location, fleet management, tourist services, transport management, traffic control and digital battlefield are all based on mobile objects and the management of their continuously changing location data.

In several applications, the object movement is constrained by an underlying spatial network, i.e., objects can not move freely in space, and their position must satisfy the network constraints. Network connectivity is usually modelled by a graph representation, comprising a set of nodes (intersections) and a set of edges (segments). Depending on the application, the graph may be *weighted* (a cost is assigned to each edge) and *directed* (each edge has an orientation).

Several types of location-dependent queries are significant in LBS, such as range queries, k-nearest neighbor (k-NN) queries, reverse neighbor queries, distance joins, closest pair queries and skyline queries. In this paper, we address the problem of processing continuous range queries over mobile objects, whose motion is constrained by a spatial network. The query range represents the user-selected area, the map window, the polygonal feature, or the area specified by the distance from a reference point of interest. In contrast to regular queries that are evaluated only once, a continuous query remains active over a period of time. A major challenge for this problem is how to provide efficient processing of continuous queries with respect of CPU time, I/O time and main memory utilization.

The rest of the article is organized as follows. The next section presents related work in the area and describes our contributions. Section 3 describes in detail the proposed framework, analyzing the methodology, the data structures and the query processing algorithms. Section 4 presents the performance evaluation results, whereas Section 5 concludes the article.

2 RELATED WORK AND CONTRIBUTION

Continuous spatiotemporal query processing in a location-aware environment is an active area of research, resulting in the proposal of many query processing methods, techniques and indexing schemes. In (Prabhakar et al., 2002), velocity con-

Stojanovic D., Djordjevic-Kajan S., N. Papadopoulos A. and Nanopoulos A. (2006).
CONTINUOUS RANGE QUERY PROCESSING FOR NETWORK CONSTRAINED MOBILE OBJECTS.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - DISI*, pages 63-70
DOI: 10.5220/0002455300630070
Copyright © SciTePress

strained indexing and query indexing (Q-index) has been proposed for efficient evaluation of stationary continuous range queries. According to the proposed method in-memory data structures and algorithms are developed and presented in (Kalashnikov et al., 2004). By indexing queries, and not mobile objects, the Q-index method avoids frequent updates of the index structure and thus expensive maintenance of this structure.

The MQM method presented in (Cai et al., 2004) focuses on stationary continuous range queries. It is based on the partitioning the query space into rectangular sub-domains, and the assignment of the resident domain to each mobile object. A mobile object is aware only of the range queries intersecting its resident domain, and reports its current location to the server only if it crosses the boundary of any of these queries.

Gedik and Liu in (Gedik and Liu, 2004) propose a method and a system for distributed query processing, called Mobieyes. Mobieyes ships some part of the query processing to the mobile clients while the server mainly acts as a mediator between mobile objects. The method tries to reduce the load on the server and save communication costs between mobile objects and the server. In the paper (Gedik et al., 2004) the authors propose a scheme called Motion Adaptive Indexing (MAI) which enables optimization of continuous query evaluation according to the dynamic motion behavior of the objects. They use the concept of motion sensitive bounding boxes (MSB) to model and index both moving objects and moving queries.

Mokbel et al. in (Mokbel et al., 2004) present SINA, a server-side method based on shared execution and incremental evaluation of continuous queries. Shared execution is achieved by implementing query evaluation as a spatial join between the mobile objects and the queries. Incremental evaluation means that the query processing system produce only the positive or negative updates of the previously reported answer, not the complete answer for every evaluation of the query. Both the object and query indexes are implemented as disk based regular grids.

Hu et al. (Hu et al., 2005) propose a generic framework for monitoring continuous spatial queries over moving objects, both range and k-NN queries. Two index structures for indexing the past trajectories of mobile objects in networks have been proposed.

The Fixed Network R-tree (FNR-tree) (Frentzos, 2003) consists of a top level 2D R-tree to index the edges of the network, whose leaf nodes contains pointers to 1D R-trees indexing the time interval of each objects movement within the line segments of the network. Almeida and Guting in (de Almeida and Guting, 2005) propose the MON-tree, to manage objects moving in a spatial network. They describe two network models (edge and route oriented) that can be indexed by the MON-tree. Both approaches deal with past positions of objects.

In this work, we propose a framework for continuous range query processing for objects moving on network paths. The framework introduces the methodology, the data structures and the query processing algorithms for processing continuous range queries over mobile objects, when queries may be both stationary and mobile. Our methodology is based on an extension of R*-tree index, that indexes network data in main memory. It introduces an additional step in traditional spatiotemporal query processing strategy (filter refinement), namely, the pre-refinement step. The filter step selects the candidate objects according to fulfillment of the spatial query condition using an extended R*-tree index on the spatial network. The pre-refinement step is performed after the filter step, to further refine the mobile objects obtained by the filter step and to build the main memory data structures to support periodical and incremental refinement steps. The filter and pre-refinement steps are performed only once, unless the reference query object changes its underlying network segment. The refinement step is performed periodically by processing in-memory data structures generated by the pre-refinement step. To the best of our knowledge, there is no reported work on query processing of continuous range queries over mobile objects whose motion is constrained by a spatial network.

3 PROPOSED FRAMEWORK

The methodology for processing continuous range queries in a mobile environment is developed as a part of ARGONAUT, a service framework for mobile object data management (Predic and Stojanovic, 2005). We base our approach on the application scenario appropriate in LBS for monitoring and tracking mobile objects. In this scenario, users have wireless devices (e.g., mobile phones or PDAs) that are online via some form of wireless communication network. We assume that users can obtain their positions using Global Positioning System (GPS) technology. A setting is assumed in which a central database at the LBS server stores a representation of each mobile object's current position. Each mobile object stores locally its position assumed by the server. Then, an object updates the database whenever the deviation between its actual position (as obtained from a GPS device) and the local copy of the position that the server assumes exceeds the uncertainty threshold. The mobile objects move along the paths in an underlaying spatial network. In order to perform tracking with as few updates as possible, the LBS server matches the position received from the mobile object to the network segment, by a map matching technique. Knowing the mobile object's speed and the time of the position update, the server determines the current position of the object till the next intersection (node) assuming that it moves at a constant speed. Reduction of updates reduces communication between clients and the server, as well as server side update processing.

3.1 Methodology Overview

The ARGONAUT methodology employs an incremental continuous query evaluation paradigm. The methodology is based on a main memory R*-tree index structure and network connectivity graph structure, which support the filter step of query processing algorithm and the map matching procedure. The methodology introduces an additional step in query processing scheme, a pre-refinement step, that creates additional main memory data structures and support subsequent incremental refinement steps. Since both objects and queries move on a spatial network, their spatial properties (location or partial route) are indexed within the same index structure. The leaf nodes of the R*-tree is appropriately modified to enable indexing of both mobile objects and queries. The leaf node entry of the general R* tree (Beckmann et al., 1990)has the form ($IDseg$, MBR), where $IDseg$ is an unique identifier of the network segment, and MBR is its minimal bounding rectangle. The entry is extended by two new elements and is represented as a quadruple ($IDseg$, MBR, $Olist$, $Qlist$). They contains the pointers to the list of objects IDs ($Olist$) and queries IDs ($Qlist$) that move along or reside on that network segment.

The connectivity graph of the spatial network is maintained by the Network Connectivity Table (NCT) which stores information about the connectivity of network segments. A NCT entry is described as ($IDseg$, $ptrRTEntry$, $segLength$, $startCon$, $endCon$), where $ptrRTEntry$ is the pointer to the R*-tree leaf node entry for this segment and $segLength$ is the length of the segment. The table is indexed on the $IDseg$ attribute. The elements $startCon$ and $endCon$ are pointers to the lists of records described as ($IDseg$, dir), where $IDseg$ represents the ID of the segment connected to the start/end node and dir is the direction of that segment in the connection. The NCT is maintained to improve the updating of the index structure when mobile object/query issue a location update after changing its network segment. The NCT also improves map matching performed at the server in order to find network segment according to the current location of a mobile object. Also, the connectivity structure maintained in NCT is more effective for LBS queries (range, k-NN, etc.) over objects on the spatial network, where distance between two objects is not determined in Euclidean space (Euclidean distance), but as network distance.

The insertion of the object into the index structure and the generation of the object list attached to the leaf node entries are performed according to the following rules:

- A stationary object ID is inserted in the $Olist$ attached to the leaf node entry of the R*-tree index that corresponds to the network segment at which such object resides,
- A mobile object ID is inserted in the $Olist$ attached to the leaf node entry of the R*-tree index that correspond to the network segment along which the object currently moves (the segment is the known route of the mobile object).

The insertion of the query in the same index structure is performed according to following rules:

- A stationary range query is inserted in each $Qlist$ attached to the leaf node entry of the R*-tree index if its range overlaps the MBR of the corresponding network segment.
- A mobile range query is inserted in each $Qlist$ attached to the leaf node entry of the R*-tree index if its known route overlaps with the MBR of the network segment. The known route of the mobile query is defined by the network segment of the reference mobile object and the query range.

The index structure performs the matching between mobile objects and mobile/stationary queries according to their spatial relations and fulfillment of the spatial condition of a query. It enables the calculation of the initial result set of the query (filter set). Since, the index structure maintains only the spatial properties of objects and queries, it is not selective enough, and the initial result set contains false positive results. The pre-refinement step refines the initial query result set with regard to temporal information of objects and queries motion, as well as their exact geometries. The pre-refinement step creates the data structures in main memory to support incremental refinement steps.

3.2 Access Methods

The two tables and associated lists are created in main memory by the pre-refinement step. Continuous Range Query Table (CRQT) stores information regarding continuous queries. A CRQT entry is described as (QID, OID, $range$, $resultSet$) and stores information regarding continuous queries. The table is indexed on the QID attribute which is the unique query identifier. OID is the identifier of the reference object of the query and $range$ defines the shape of the spatial query range around the reference query object. $resultSet$ is the initial query result set obtained by the filter step with additional, temporal information about satisfaction of a query condition. The initial result is a list of elements $CQResult$ defined as (RID, OID, $resPeriod$, $status$), where RID is the result identifier, OID is the unique identifier of the object which is the result of the query during period $resPeriod$, while its status in the query result is described by the $status$ attribute. The values of the $status$ attribute are INIT_RESULT, NEW_RESULT, OLD_RESULT, NO_RESULT. In the simplified case, when the resulting period is single time period, the resulting object, during its motion and/or query motion, change all status values sequentially. Thus, an object has INIT_RESULT status when it will be the result of the query in some period(s) in the future. The status NEW_RESULT is associated to an object, when it becomes the new result of the query. The status OLD_RESULT is associated to an object when it is the member of the current query result, as well as was a member of the query result in the previous evaluation of the query. An object has status NO_RESULT when it is not a member of the current query result, nor will be in the future, but was the member of previous query results. Due to time progress, any resulting object changes its status in a strict sequence [INIT_RESULT→ NEW_RESULT→ OLD_RESULT]n→ NO_RESULT, where [...]n indicates that the status sequence can be repeated if the resulting period of an object is a multi period.

For each mobile object in the system, an in-memory Mobile Object Table (MOT) is created and maintained. The mobile object entry is described as (OID, loc, $time$, $speed$, $route$, $querySet$), where OID is the unique mobile object identifier, loc is the last received location, $time$ is the time instant of the location update and $speed$ is the last received speed. The $route$ represents the pointer to the ordered list of records described as ($IDseg$, dir) which is the known route of the mobile object. When the route (whole or partial) is not known in advance, the route list contains only one element denoting the current network segment of the mobile object. The $querySet$ attribute represents the list of queries in which such object participates, either in a query result, or as a reference object of a query. Each query in this list is represented by a $QueryRef$ element which contains two attributes: QID, the query identifier and $resID$, the reference to the appropriate $CQResult$ element of this query maintained for the mobile object. If a mobile object is the reference object of a query, this reference is set to NULL.

3.3 Algorithms

The periodic, incremental evaluation of continuous range queries is performed by scanning and examining the CRQT, while location update for each mobile object requires scanning and updating both tables. The algorithms for creating these tables are slightly different for the cases of stationary and mobile continuous queries. The input argument of the algorithms is the set of mobile object OIDs that represent the potential query results obtained by the filter step, by the examination of the R*-tree index. The pre-refinement algorithm is performed on exact spatial and temporal geometries of mobile objects and mobile/stationary queries and creates necessary data structures. The algorithm for stationary queries over mobile objects is given in Figure 1.

Algorithm Pre-refinement step for stationary queries over mobile objects
Input: $filterSet$: a set of OIDs

```
1.  add new entry $smquery$ in CQRT;
2.  for each $OID$ in $filterSet$
3.      if (not exist $mo$ in MOT with such OID) then
4.          add new entry $mo$ in MOT;
5.      else
6.          find entry $mo$ in MOT;
7.      end if
8.      if $mo.route()$ intersects $smquery.range$ then
9.          $rp \in TimePeriod$ and $rp$ = when ($mo.loc$
            within $smquery.range$);
10.         if $rp.end > currentTime$ then
11.             $cqres \leftarrow$ new CQresult($RID$, $mo.OID$, $rp$,
                INIT_RESULT);
12.             $smquery.resultSet$.add($cqres$);
13.             $qr \leftarrow$ new QueryRef($smquery.QID$, $cqres.RID$);
14.             $mo.querySet$.add($qr$);
15.         end if
16.     end if
17. end for
```

Figure 1: Pre-refinement step for stationary query - mobile objects.

After updating the CQRT and MOT tables, the algorithm examines the spatial relation $intersects$ on a mobile objects route (the geometry of the current network segment) and the stationary query range. This step enables removing false spatial positives obtained by the filter step. For those objects satisfying the spatial relation $intersects$, the time period (multi time period) in which the mobile object is (was, will be) $within$ the query range is calculated, based on current motion parameters (speed, route) of the mobile object. The operators for spatiotemporal geometric calculations and topological relations in space+time (such as $intersects$, $within$, $when$, etc.) are developed and integrated in the ARGONAUT mobile object data management framework (Stojanovic and DjordjevicKajan, 2003). For objects whose resulting period ends somewhere in the future, the new

$CQResult$ and $QueryRef$ elements are created and added to the lists of corresponding CQRT and MOT entries.

For the mobile query over mobile objects the algorithm for the pre-refinement step is slightly different, because the relationship between two moving objects is not linear and it is impossible to exactly determine the resulting period for each mobile object a potential result of the mobile query. Therefore, for mobile queries, the spatial condition given in line 8 of the previous algorithm is changed. Instead of examining the query range in lines 8 and 9, the query route is examined, which is determined as a buffer around the query's network segment defined by the range.

The refinement step is performed periodically and evaluates the temporal query condition. It determines the final result set and the incremental result in regard to the previous evaluation (refinement step), which is sent to the mobile client issuing a continuous query. The incremental result represents the set of $IncResult$ elements containing the OID of the object and the Boolean attribute resUpdate indicating that the object becomes the part of the result of the query ($true$ value), or that it is not the result any longer ($false$ value). The refinement step is performed for all continuous queries active in the system, according to the algorithm shown in Figure 2.

If a mobile object enters the query range several times during its motion (the result period is a set of time periods), when one period from the set is expired, the objects status is INIT_RESULT again, until the beginning of the next time period. If the result period of an object expires, the negative query result update is generated and the corresponding element is removed from the query result set. The refinement step for the case of mobile queries over mobile objects must include an additional test of the spatial condition on exact mobile object and query location, for those objects that already satisfy the temporal condition. Thus, the line 4 of the algorithm in (Figure 2) introduces an additional condition and becomes:

```
4.   if $cqres.resPeriod$ contains $currentTime$ and
          $mo$.currentLoc() within $cq$.currentRange() then
```

The functions $currentLoc()$ and $currentRange()$ calculate the location/range of the mobile object/query at the current time, given the last received location, time and speed of the object/query reference object, as well as its current route (network segment). As mentioned previously, a mobile object moves on its network segment with the last reported speed. When its predicted location (obtained by the currentLoc() function) differs form the exact location by the specified threshold, the object must send location, time and speed updates. Upon receiving updates, the server must determine if the mobile object changes its network segment using map matching techniques on segments that are connected to the previous one using NCT. If the mobile object remains on its network segment, the update algorithm scans the pre-refinement data structures and updates the corresponding MOT entry, as well as all CQRT entries of the affected queries and their result sets using the simple expressions (Figure 3). If the mobile object leaves its network segment and starts moving along a new one, the pre-refinement data structures ($querySets$, $resultSets$) related to the mobile object and the new set of queries obtained by the new R*-tree $Qlist$, should be updated or new entries should be added according to the algorithms for the pre-refinement steps depicted in Figure 1. If a mobile object is also a reference object of the mobile query, the new R*-tree $Olists$ must be examined and the corresponding $querySets$ and $resultSets$ should be updated.

The update of the result period of an object issuing location and speed updates at a certain time instance is based on the threshold value u_t, previous speed v_o, the new speed v_n, and the time t_n of the new update. The $update$ function (lines 31 and 34 of Figure 3) updates the resulting period rp of a mobile object (or

Algorithm Refinement step for stationary queries over mobile objects
Output: $incResultSet$, a set of $IncResult$

```
1.  for each $cq$ in $CQRT$
2.      if $cq.period$ contains $currentTime$ then
3.          for each $cqres$ in $cq.resultSet$
4.              if $cqres.resPeriod$ contains $currentTime$ then
5.                  if $cqres.status$ == INIT_RESULT then
6.                      $cqres.status \leftarrow$ NEW_RESULT;
7.                      $ir \leftarrow$ new IncResult ($cqres.OID$, true);
8.                      $incResultSet$.add($ir$);
9.                  else
10.                     $cqres.status \leftarrow$ OLD_RESULT;
11.                 end if
12.             else if $cqres.status$ == OLD_RESULT; then
13.                 if $cqres.period$ is a $TimePeriodSet$ and
                            $hasperiodsinthefuture$ then
14.                     $cqres.status \leftarrow$ INIT_RESULT;
15.                 else
16.                     $cqres.status \leftarrow$ NO_RESULT;
17.                     $ir \leftarrow$ new IncResult ($cqres.OID$, false);
18.                     $incResultSet$.add($ir$)
19.                     remove $cqres$ from $cq.resultSet$;
20.                 end if
21.             end if
22.         end for
23.     end if
24. end for
```

Figure 2: Refinement step for the set of stationary continuous queries over mobile objects.

Algorithm Location/speed/network segment update of a mobile object
Input: $newloc$, $newspeed$, $newtime$ of the mobile object and eventually $newseg$ obtained using map matching and NCT

```
1.  update $MOT$ entry for $mo.OID$;
2.  $mo.loc \leftarrow newloc$;
3.  $mo.speed \leftarrow newspeed$;
4.  $mo.time \leftarrow newtime$;
5.  if $mo$ change network segment then
6.      $mo.route.IDseg \leftarrow newseg$;
7.      remove $mo.OID$ from the old $Olist$;
8.      add $mo.OID$ to the new $Olist$;
9.      for each $cq$ in new $Qlist$
10.         if $cq.QID$ exist in $mo.querySet$ then
11.             Update corresponding CQResult and QueryRef elements;
12.         else
13.             Add new CQResult and QueryRef elements;
14.         end if
15.     end for
16.     if $mo$ is a reference object for query $cq.QID$ then
17.         remove $cq.QID$ from the old $Qlist$;
18.         add $cq.QID$ to the new $Qlist$;
19.         for each $mo$ in new $Olist$
20.             if $mo.OID$ exist in $cq.resultSet$ then
21.                 Update corresponding CQResult and QueryRef elements;
22.             else
23.                 Add new CQResult and QueryRef elements;
24.             end if
25.         end for
26.     end if
27. else
28.     for each $qr$ in $mo.querySet$
29.         $cq \leftarrow$ CRQ with $qr.qid$;
30.         if $qr.resEntry$ != NULL then
31.             update($cq.resulSet$, $qr.resEntry$);
32.         else
33.             for each $res$ in $cq.resultSet$
34.                 update($cq.resultSet$, all);
35.             end for
36.         end if
37.     end for
38. end if
```

Figure 3: Location/speed/network segment update of a mobile object.

every time period in the set of time periods) according to the following formulae:

$$rp.start = \frac{v_0 \cdot (rp.start - t_c) \pm u_t)}{v_n} + t_c$$

$$rp.end = \frac{v_0 \cdot (rp.end - t_c) \pm u_t)}{v_n} + t_c$$

The uncertainty threshold value in this formula is added if the mobile object is advanced in regard to its predicted location, and it is subtracted if it is late in regard to its predicted location. Thus, the system provides an up to date and accurate result set for every continuous range query it maintains, according to location/speed updates of mobile objects.

4 PERFORMANCE STUDY

We have used the Network-based Generator of Moving Objects (Brinkhoff, 2002) to generate a set of 10000 mobile objects and 1000 mobile queries. The input to the generator is the road map of Oldenburg (a city in Germany). The road network consists of 2873 intersections and 3803 segments. The output of the generator is a set of moving objects that move on the road network of the given city. We choose some objects randomly and consider them as reference objects and centers of square range queries. All the experiments have been conducted on an Intel Pentium IV CPU 3.0 GHz with 512 MB RAM running Windows XP Professional. The page size is set to 4KB. We have implemented the extended R*-tree based on the original implementation of (Beckmann et al., 1990). ARGONAUT continuous query processing algorithms and main memory data structures have been implemented using Microsoft Visual Studio C++ and the STL library. Our performance measures are: i) the CPU time required for the pre-refinement step and the creation of in-memory data structures, ii) the CPU time for the refinement step and the generation of incremental results and iii) the CPU time for updating the main memory data structures upon receiving location/speed/network segment update of a mobile object. Since the performance for access and update of R*-tree which index network segments highly depends on size and characteristics of the underlying spatial network, we do not present the results of those experiments.

We start our performance consideration using the set of objects produced by the filter step, which are the candidates for the final result set according to their spatial characteristics, i.e., location and range for the stationary queries and routes for mobile objects/queries. We perform the experiments for 10000 mobile objects and 1000 mobile (stationary) queries and measure the average CPU time (in milliseconds) necessary for the pre-refinement step of a continuous query and the generation of the necessary data structures in main memory for refinement steps which are performed periodically and produce incremental query answers (Figure 4). We vary the query range from 0.004 to 0.8 of each dimension of the data space.

The experiments show that the pre-refinement step of the query requires a very small amount of CPU time (milliseconds). More specifically, for 10000

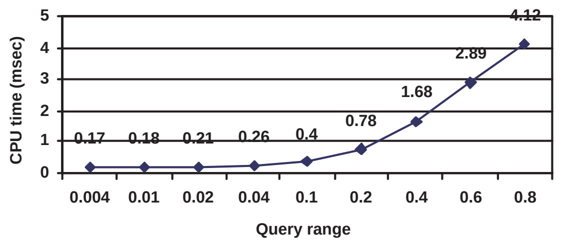

Figure 4: The CPU time for the pre-refinement step per query for 10000 MO / 1000 queries.

mobile objects and 1000 continuous range queries, the pre-refinement step needs 4.12 milliseconds for queries whose query range represents 0.64 of the data space. The average number of mobile objects per query obtained in the filter step for different query ranges is given in Figure 5.

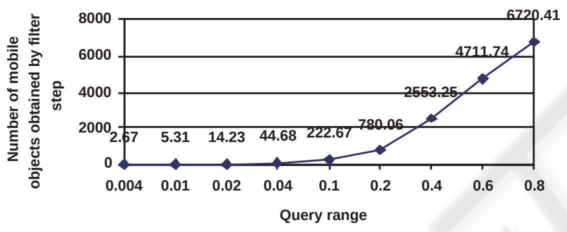

Figure 5: The average number of mobile objects obtained in the filter step for 10000 MO / 1000 queries.

During the refinement step an access to the main memory data structures and the examination of the temporal part of the query condition are performed. The refinement step is performed periodically, i.e., once every T seconds. For each object in the result set of the continuous query, the refinement step should select those objects that constitute the current result of the query as well as to generate the incremental result in regard to the previous refinement step and evaluation of the query. We examine the duration of the refinement step for 1000 continuous queries over 10000 objects, using different query ranges. Figure 6 depicts the CPU time required for the incremental evaluation of 1000 stationary and mobile continuous queries over mobile objects depending on query range, according to the algorithm shown in Figure 2 and its extension for mobile queries.

When the mobile object (query) issues a location/speed/segment update, the main memory data structures should be updated appropriately, to reflect the new location and the motion parameters of the mobile object/query, as well as its new route (net-

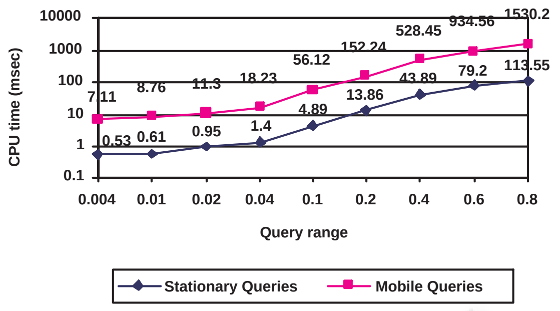

Figure 6: The CPU time needed for refinement of 1000 mobile/stationary queries over 10000 mobile objects.

work segment). According to the algorithm shown in Figure 3, a mobile object should update the resulting periods for all continuous queries in which it participates, according to its new location and new reported speed. When a mobile object is the reference object of a query (queries) the resulting periods of all results in the set should be updated accordingly. The CPU time required for updating the main memory data structures, for 10000 mobile objects and 1000 continuous queries is presented in Figure 7. The CPU time shown represents the average time per object needed for the update of main memory data structures that preserves the consistency of the result sets. If a mobile object changes its underlying network segment, it is necessary to update the main memory data structures related to this object, and the queries which may be affected, according to the pre-refinement algorithm. In this case it is necessary to perform the filter and the pre-refinement step for each such object and generate the new $querySets$ and $resultSets$ structures.

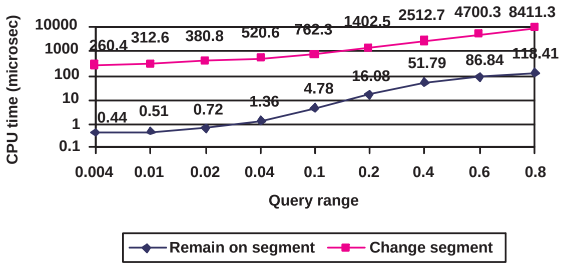

Figure 7: The average CPU time needed for data structures update upon location/speed/segment update of a mobile object.

With respect to the above results, the ARGONAUT methodology offers acceptable performance in updating the resulting periods of the queries result sets, when receiving new values for the location or speed of a mobile object. Therefore, the proposed methodology can be used to solve real-life problems and aid real-life applications which require the storage and manipulation of mobile objects moving on a spatial network.

5 CONCLUDING REMARKS

This paper introduces the ARGONAUT framework and methodology for evaluating continuous range queries over mobile objects moving on a spatial network. We have performed comprehensive experiments, measuring the required CPU time for the query processing algorithms. The experimental results show that the performance of the ARGONAUT query processing methodology is satisfactory for real world settings in LBS applications for monitoring and tracking mobile objects.

We plan to continue working on continuous range query processing, to further exploit indexing schemes and the network connectivity graph (NCT) for range queries whose range is not based on the Euclidean distance. Also, we plan to consider the issues of distributed and mobile query processing techniques which ship some part of query processing to the mobile objects which have the computational and storage capabilities to perform some part of the query processing algorithms.

ACKNOWLEDGEMENTS

Research supported by the 2003-2005 Serbian-Greek joint research and technology program and by ARCHIMEDES project 2.2.14, “Management of Moving Objects and the WWW”, of the Technological Educational Institute of Thessaloniki (EPEAEK II).

REFERENCES

Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger., B. (1990). The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD, pp.322-331*.

Brinkhoff, T. (2002). A framework for generating network-based moving objects. In *GeoInformatica,Vol.6, No.2, pp.153-180*.

Cai, Y., Hua, K., and Cao, G. (2004). Processing range-monitoring queries on heterogeneous mobile objects. In *Mobile Data Management, pp.27-38*.

de Almeida, V. and Guting, R. (2005). Indexing the trajectories of moving objects in networks. In *GeoInformatica, Vol.9, No.1, pp.33-60*.

Frentzos, E. (2003). Indexing objects moving on fixed networks. In *8th International Symposium on Spatial and Temporal Databases, pp.289-305*.

Gedik, B. and Liu, L. (2004). Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In *EDBT, pp.67-87*.

Gedik, B., Wu, K., Yu, P., and Liu, L. (2004). Motion adaptive indexing for moving continual queries over moving objects. In *CIKM, pp.427-436*.

Hu, H., Xu, J., and Lee, D. (2005). A generic framework for monitoring continuous spatial queries over moving objects. In *ACM SIGMOD Conference*.

Kalashnikov, D., Prabhakar, S., and Hambrusch, S. (2004). Main memory evaluation of monitoring queries over moving objects. In *Distributed and Parallel Databases, Vol.15, No.2, pp.117135*.

Mokbel, M., Xiong, X., and Aref, W. (2004). Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *ACM SIGMOD Conference, pp.623-634*.

Prabhakar, S., Xia, Y., Kalashnikov, D., Aref, W., and Hambrusch, S. (2002). Query indexing and velocity constrained indexing scalable techniques for continuous queries on moving objects. In *IEEE Transaction on Computers,Special Issue on DBMS and Mobile Computing, Vol.51, No.10, pp.1124-1140*.

Predic, B. and Stojanovic, D. (2005). A framework for handling mobile objects in location based services. In *AGILE, pp.419-427*.

Stojanovic, D. and DjordjevicKajan, S. (2003). Modeling and querying mobile objects in location based services. In *Scientific Journal Facta Universitatis Series Mathematics and Informatics, Vol.18, No.1, pp.59-80*.