Strategies for Continuous Region Query Processing in Moving Point Sets

Shauli Sarmin Sumi
Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Alberta, Canada
shauli.sumi@uleth.ca

Wendy Osborn
Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Alberta, Canada
wendy.osborn@uleth.ca

***Abstract*—Nowadays, many mobile applications provide location-based services that allow users to access location-related information from anywhere, whenever they desire. A moving user can issue queries to access information about moving or static objects. Continuous spatial query processing systems are used for this type of application. We propose two continuous query processing strategies moving point sets. The objectives of our strategies are to reduce: (1) the server workload, (2) the data transmission cost and (3) the query response time, for location-based services while providing an answer for a continuous region query. We compare our strategies with a brute-force strategy. Results show that our strategies can significantly reduce the server workload, data transmission cost, and response time over the brute-force method in many cases.**

***Index Terms*—location based services, continuous region queries, spatial access methods, performance**

I. Introduction

Many mobile applications provide location-based services [1] that make location-related information highly accessible anywhere. One such location-based service handles a continuous region query where both the query issuer and the target objects are moving. For example, a taxi driver asks for the locations of customers within a radius of 2 miles who are looking for a taxi. The result of such a query changes with the movement of objects and requires continuous updates. The challenges of location-based services that handle continuous queries are [2]: (1) processing and sending the query results to the issuer requires a chain of server processes since the query issuer object is moving, and (2) the region of interest for the query is also changing due to the object(s) movement which adds more complexity to processing.

For a successful location-based service, the server workload, data transmission cost, and response time should be minimized. An increased server workload may affect and delay the overall query processing task. Also, any delay in query processing can make the query result outdated or useless since the objects are continuously changing their location. To address these issues, we propose two efficient processing strategies for continuous region queries over moving point sets where both the query issuer object (i.e., client) and the target objects are moving. Our goal is to reduce the server workload, data transmission cost and query response time. In our environment, a moving user issue a query to request information from a central server through a wireless network. The server processes the spatial queries and sends the answer to the user. Since the result of a continuous query changes with the movement of the user and the objects, continuous updates are required. Therefore, the server will use a spatial index to assist in query processing. The server will also monitor any changes to the location of object(s) so that the server workload, response time, and data transmission cost are all minimized.

The rest of this paper is organized as follows. Section 2 provides related work for continuous spatial query processing in mobile information systems. Section 3 presents our proposed original strategy and the improved strategy. In Section 4, we present the experimental results of our strategies over the brute-force (BF) method. Finally, Section 5 presents the conclusion and also provides future research directions.

II. Related Work

For processing continuous queries, such as region queries, over moving objects, various research has been undertaken. Some proposed strategies are proposed solely for region queries while others also work for additional query types. IIarri *et al.* [3] propose a system that supports distributed processing of continuous location dependent queries in a mobile environment, using mobile agents to carry out the necessary processing tasks. Their work dividies significantly large areas into proxies and utilizes a query monitor to track query movement between the areas. Liu and Hua [4] propose a distributed framework to process moving queries, such as the moving range query and the moving kNN query, over moving objects in a spatial network environment. Using partitioning of a road network, objects only update their location when moving to a new area. Lin and Wu [5] propose a system architecture for a road network so that direction of travel is considered when processing a query. They also provide an approach to determine the safe period for updating the query result for both the centralized and distributed processing of directional continuous range queries. Gupta *et al.* [6] propose a hierarchical database framework and algorithm based on a tree architecture, which can be applied to both single-zone and multiple-zone queries. Wang and Zimmermann [7] propose

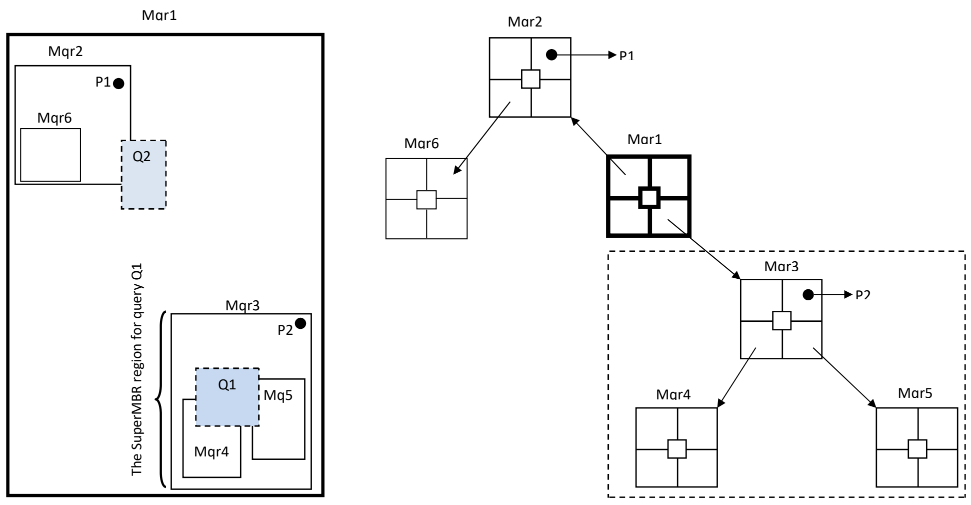


Fig. 1. An example mqr-tree and superMBR for a query

an index structure for tracking distances of moving objects, and related algorithms that exploit the index structure for processing a continuous range query in road networks. Ku *et al.* [8] proposal a peer-to-peer approach where cached results are shared among neighbouring mobile peers. Tracking is used to ensure that the shared results are valid. Mouratidis *et al.* [9] proposed a compact grid-style index for continuous query processing in broadcast environments, and associated algorithms for handling updates submitted by clients. Park [10] also propose a hierarchical structure for continuous query processing in broadcast systems.

Most existing work on continuous queries over moving point data do not use a spatial access method. Some existing works use spatial access method which contain overlapping. However, the existence of overlapping can affect the query processing ad response time due to the unnecessary searching in the overlapped region. Moreover, validity regions for a query are not obtained directly from those spatial indices which adds extra computational cost since they must be computing on-the-fly every time a new one is needed. Other existing works, especially in broadcast networks, use monitoring for continuous location updates, but this monitoring is performed by the client and not the server, and therefore can affect the response time of a query. Therefore, using both a spatial index for obtaining validity regions, and server-side monitoring of point movement will lead to improvements in server workload, data transmission cost and response time.

A. mqr-tree

The mqr-tree [11] [12] is a two-dimensional index structure that organizes spatial objects in two-dimensional nodes based on different spatial relationships, and helps make the search process and data retrieval more efficient. An object, or subregion of space that contains objects, is represented using a minimum bounding rectangle (MBR), which is the minimum two-dimensional rectangular range that contains objects or subregions. A performance comparison with other strategies shows that the mqr-tree achieves a lower average number of node accesses, which can reduce the query execution time for the region search. In addition, for point data, the mqr-tree has no overlap of subregions and therefore the point search will only proceed along one path and a region search will proceed along significantly fewer paths that may contain no points for the result. We can also modify the region search to also follow only one path. For these advantages, in our strategies, the server uses the mqr-tree spatial index and a modified mqr-tree region search process in order to perform the query processing task. An example of a mqr-tree for a point set is shown in Figure 1. In the figure, the root node MBR, Mqr1 which is highlighted in bold lines, represents a region that encompasses the whole set of point MBRs as well as the subtree MBRs.

B. Identification of superMBR

We define a **superMBR** (Super Minimum Bounding Rectangle) as the last nodeMBR along a query path which fully encloses the query. This superMBR will serve as the validity region mentioned above. In the Figure 1, suppose, a client issues a region query Q1 (a shaded rectangle). For this query, the Mqr3 MBR is the superMBR because this is the last node MBR along the query path in the tree that wholly contains the query region Q1. Also note that Mqr4 overlaps the query region Q1 but does not fully contain it and so it can not be the superMBR for Q1. If we consider another region query Q2 in Figure 1, the superMBR is the root MBR, which is Mqr1. This is because only Mqr1 fully contains the query region Q2.

Figure 2 shows the process for identifying the superMBR for a query region. If the query region is fully contained in the root node, then it is tested to see if it is fully contained in one of the NW, NE, SW and SE child nodes of the root node. If the query region is not fully contained in any of the four child nodes (NW, NE, SW and SE) of the root node, then the search process identifies the MBR of the root node as the superMBR. If the query region is fully contained in the MBR of one of the

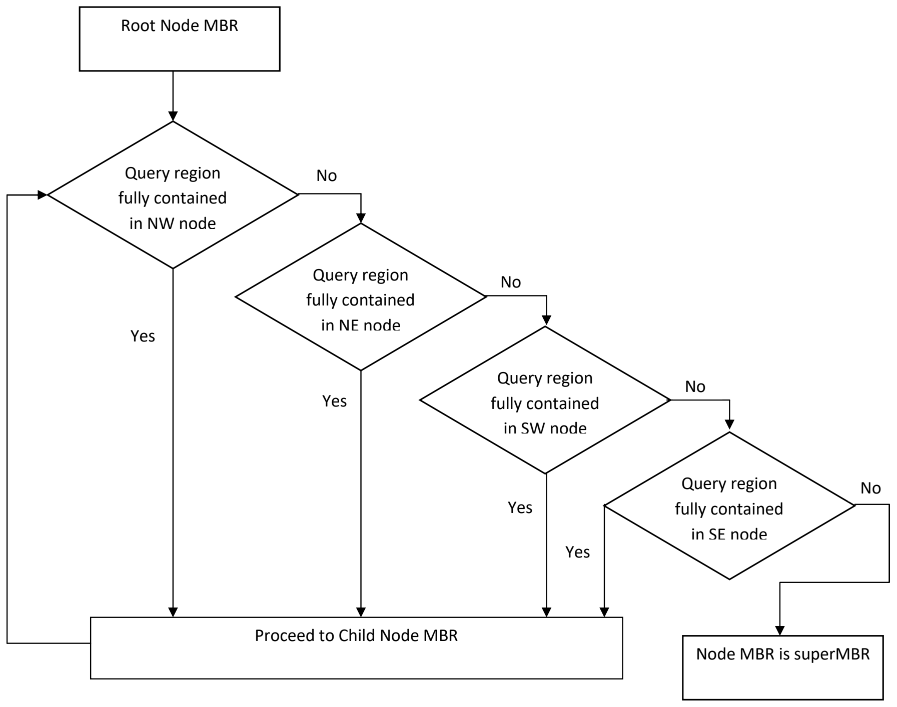

Root Node MBR
Query region fully contained in NW node
No
Yes
Query region fully contained in NE node
No
Yes
Query region fully contained in SW node
No
Yes
Query region fully contained in SE node
No
Yes
Proceed to Child Node MBR
Node MBR is superMBR

Fig. 2. superMBR Identification Process

child nodes of the root node, then the search process continues in the chosen subtree. This process is repeated until the region query is not fully contained in any child node MBRs. After identifying the superMBR, the residing point set is generated by traversing the subtree pointed to by the superMBR node. An example is shown in Figure 1, where the Mqr3 MBR is the superMBR for region query Q1. In that figure, the three nodes inside the dashed rectangle are within the superMBR region. The server fetches all of the pointMBRs that are contained within these three nodes as a result of query Q1.

C. *Original Strategy*

For our Original (first) Strategy (OS), no monitoring is used to identify points that have moved between queries. The strategy begins when the server receives a query from the client, which starts the search process. For the first query from a client, the server locates a superMBR, fetches all of the points within the superMBR, and sends the superMBR and resulting point set to the client. The server also saves this superMBR and point set for future comparison with subsequent queries.

When this client issues all subsequent queries, the server locates a new superMBR and corresponding point set to compare to the previous result. One of two things happens: only the old and new superMBRs need to be compared, or both superMBRs as well as the point sets need to be compared. First, we compare the new and previous superMBRs. If they are different, the new data set and superMBR are sent to the client. If they are the same, then the point sets also need to be compared. This is because although the superMBRs are same, the residing point set can be different. For example, when some point moves within or outside of the region identified by the superMBR but the points on the border region remain unchanged, then the superMBR remains unchanged but the point set changes. If the new and previous point sets need to be compared and are different (e.g. there is at least one point that has moved), then the new data set and superMBR are sent to the client. If the point sets are the same, only a message indicating this needs to be sent to the client.

D. *Example*

We now give an example that illustrates the proposed Original Strategy. Each point is represented here as: *lx : ly : hx : hy : 0 : ID*, where *(lx,ly)* is the lower left corner and *(hx,hy)* is the upper right corner of the MBR for a data. Since we are working with point data, the lower and upper corners are the same for each point MBR. Our sample data is as follows:

P1 →	0.11	68.64	0.11	68.64
P2 →	0.30	35.99	0.30	35.99
P3 →	1.37	47.97	1.37	47.97
P4 →	1.92	22.64	1.92	22.64
P5 →	2.40	32.73	2.40	32.73
P6 →	3.71	185.36	3.71	185.36
P7 →	3.77	139.74	3.77	139.74
P8 →	4.08	115.77	4.08	115.77
P9 →	4.26	20.97	4.26	20.97
P10 →	4.97	196.34	4.97	196.34

E. *mqr-tree for the Sample Data*

The mqr-tree for the sample data is shown in Figure 3. Here Mqr1 is the root node which points to two nodeMBRs

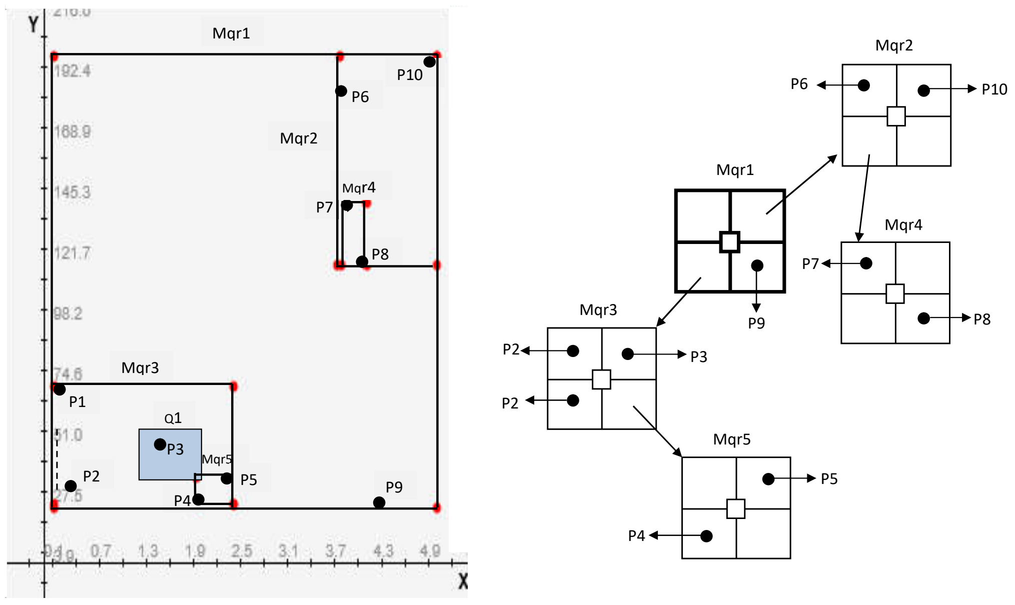
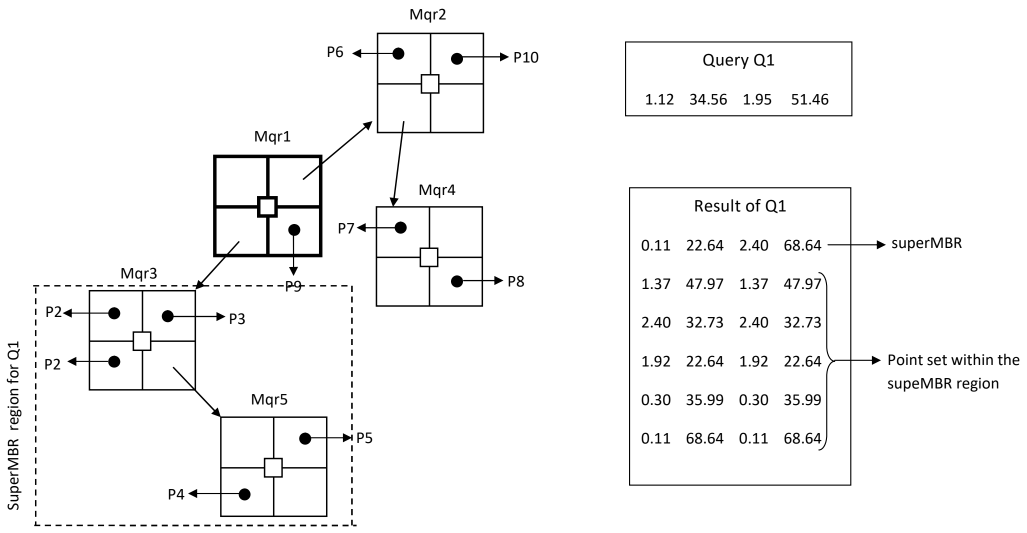


Fig. 3. The mqr-tree and a query Q1 for sample data



Query Q1

1.12	34.56	1.95	51.46

Result of Q1

0.11	22.64	2.40	68.64
1.37	47.97	1.37	47.97
2.40	32.73	2.40	32.73
1.92	22.64	1.92	22.64
0.30	35.99	0.30	35.99
0.11	68.64	0.11	68.64

Fig. 4. Result of the Region Query Q1

with their associated subtree: Mqr2 in its NE region and Mqr3 in its SW region. Mqr1 also references a point P9 in its SE location. Mqr2 references a nodeMBR and subtree Mqr4 in its SW region. It also contains two points: P6 and P10 in the NW and NE locations respectively. Mqr3 points to a subtree (and nodeMBR) Mqr5 in its SE region. It also references three points P1, P3 and P2 in the NW, NE and SW locations respectively. Mqr4 references two points: P7 in the NW and P8 in the SE locations. Mqr5 references two points P5 and P4 in the NE and SW locations respectively. Therefore, the coordinates *(lx ly hx hy)* of each node MBR are as follows:

Mqr1 →	0.11	20.97	4.97	196.34
Mqr2 →	3.71	115.77	4.97	196.34
Mqr3 →	0.11	22.64	2.40	68.64
Mqr4 →	3.77	115.77	4.08	139.74
Mqr5 →	1.92	22.64	2.40	32.73

F. Query Processing

A client sends a region query, suppose Q1 (as shown in the shaded rectangle in Figure 3) to the server. The coordinate *(lx ly hx hy)* of Q1 is: Q1 → (1.12, 34.56,1.95, 51.46)

After receiving the query Q1, the server starts the search process. Starting from the root node Mqr1, it checks to see whether Q1 exists in the NW, NE, SW or SE region of Mqr1. It is seen that Q1 exists in the SW region of Mqr1 as in Figure 3. So, the search process proceeds along the subtree Mqr3. It is

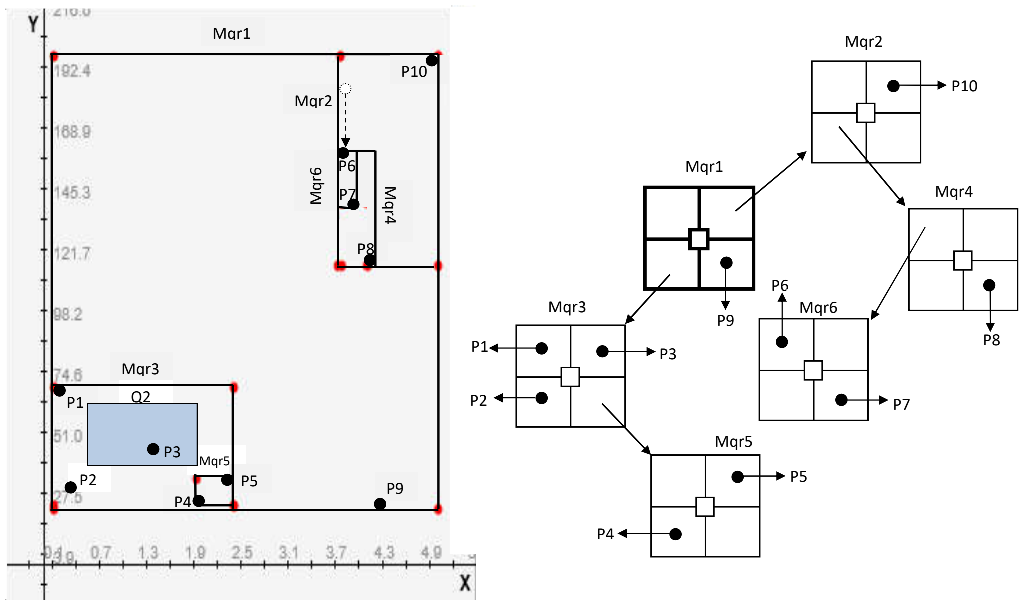


Fig. 5. The mqr-tree and the second query Q2

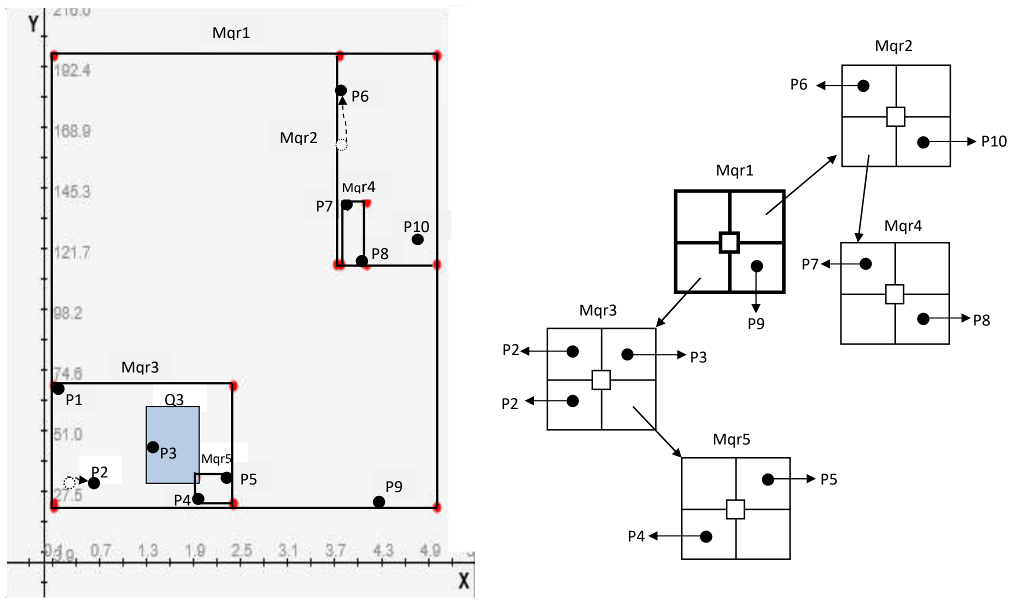


Fig. 6. The mqr-tree and the third query Q3

found that Mqr3 is the last node which fully contains Q1. So, Mqr3 is identified as the superMBR for Q1. Now the server traverses the subtree Mqr3 to generate the set of points that are residing within the superMBR region. The server fetches all of the pointMBRs within Mqr3, as well as fetches all of the pointMBRs within Mqr5, which is a subtree also pointed to by the Mqr3. The server now sends the superMBR Mqr3 and the generated point set to the client as the result of query Q1. The result of the region query Q1 is shown in Figure 4.

The client sends a second query Q2 as shown in Figure 5. The coordinate *(lx ly hx hy)* of Q2 is: (0.45, 39.61, 1.94, 64.22). The server then identifies Mqr3 as the superMBR for Q2, and fetches all of the pointMBRs within this Mqr3 region. The server now compares the new superMBR (i.e., Mqr3) with the previous superMBR, which is the superMBR for Q1, and finds that they are same. So, the server now compares the new point set with the previous point set of Q1. It is found that the point sets are also the same. In Figure 5, we can see that the point P6 has moved (as shown by a dashed arrow) to a new location of coordinate (3.71, 153.36, 3.71, 153.36). This change of P6's location also changes the tree structure which we can see in Figure 5 (i.e., Mqr4 is splits into two: Mqr4, and

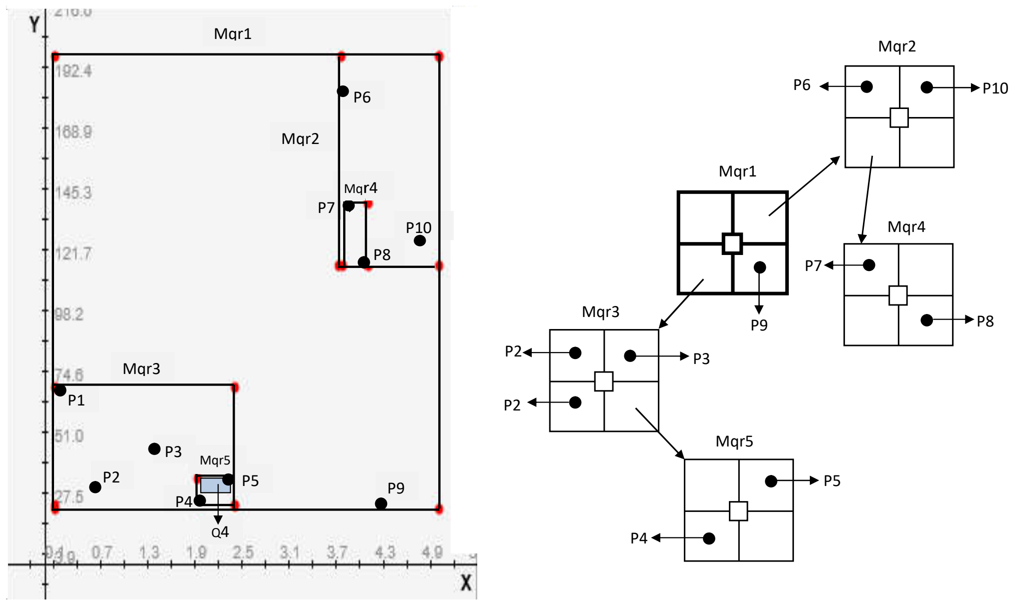


Fig. 7. The mqr-tree and the forth query Q4

Mqr6). Since the point P6 resides and moves outside of the superMBR region, it does not affect the new result set. The superMBR and the point set of Q2 is the same as the previous result, so the server only sends a message indicating the result is same as before, instead of sending the entire result file.

The client sends a third query Q3 identified by (1.30, 35.99, 1.97, 63.2) is shown in Figure 6. The server again identify the Mqr3 as the superMBR of Q3, and fetches all the pointMBRs within this region. After finding the same superMBR, the server compares the new point set with the point set of the previous query Q2. The server finds that the new point set is different. From Figure 6, we see that point P2 has moved to a new location (0.60, 35.99, 0.60, 35.99), and also that point P6 has moved back to its first location. The point P2 is inside the superMBR region and has changed the result set. So, the server sends the new point set with the superMBR to the client.

Figure 7 shows a fourth query Q4 identified by (1.98, 22.00, 2.37, 31.80), sent by the client. The server identifies Mqr5 as the superMBR for Q4 (because Q4 is fully contained in the Mqr5 region), and then fetches all the pointMBRs within Mqr5. The server now compares the new superMBR with the superMBR of previous query Q3 and finds that they are different. So the server sends the new point set with its superMBR to the client without comparing the new point set with the point set of Q3.

G. Improved Strategy

A SuperMBR can be the same for different queries. In spite of the same superMBRs, the point sets can be different. For this reason, the old and new point sets must be compared when the superMBRs are the same, as we have done in our first strategy. When the superMBR for a query identifies a large region, then the residing point set can be large too. In this case, comparing all points between two large point sets results in significant overhead for the server. Therefore, we propose another strategy called our Improved Strategy, in order to reduce the overhead of comparing large point sets.

Our Improved Strategy (IS) uses monitoring to track any points that have moved between the transmission of two query regions. The server keeps track of both the new location and the old location of any points that have moved. The first part of this strategy - the generation of the first superMBR and point set - is identical to that for OS. The differences occur from the second query and onwards.

After generating the result for the next query, the server still compares the new superMBR with the previous one, and sends them to the client if the new and previous superMBRs are different. If the superMBRs are the same, then the server checks the monitoring information to see if any points have moved with respect to the new and previous superMBR. If any moved point (which is new to the current data set) resides in the superMBR, then the new point set and superMBR are sent to the client. If a moved point does not reside in the superMBR, the server checks the old points (which have been moved to a new location) to see whether any of these points resided in the superMBR. If any old point (which has been moved to a new location) resided in the superMBR, the server sends the new data set and its superMBR to the client. If no old point (which has been moved to a new location) resides in the superMBR, then the server only sends a message to the client indicating that the result is same as the previous one.

III. Experiments and Results

We compare our Original (OS) and Improved (IS) strategies with a brute-force (BF) method, which always sends the entire result back to the client, without comparing the new query

TABLE I
AVERAGE DATA TRANSMISSION COST OVER STATIC DATA

Data Sets	Data Transmission Cost		
	OS	BF	BF/OS
500	1932.4	2300.8	16.01182
1000	3801.2	7318.4	48.05969
5000	16975.6	17442.2	2.675121
10000	32477.3	64764.8	49.85347
50000	160428.4	320761.6	49.98516
100000	320489.2	320825.6	0.104854

TABLE II
AVERAGE DATA TRANSMISSION COST OVER MOVING DATA

Data Sets	Data transmission cost		
	OS	BF	BF/OS
500	2274.3	2304	1.289062
1000	7093.5	7318.4	3.073076
5000	16975.6	17436.8	2.644981
10000	64478.2	64619.9	0.219282
50000	320435.7	320787.2	0.109574
100000	320582.9	320822.4	0.074652

TABLE III
AVERAGE DATA TRANSMISSION COST OVER VARYING MOVING POINTS

# Moving Points	Data transmission cost		
	OS	BF	BF/OS
1	7093.5	7318.4	3.073076
2	7002.1	7318.4	4.321983
3	7008.5	7328	4.359989
4	7008.5	7328	4.359989
5	7008.5	7328	4.359989

TABLE IV
AVERAGE RESPONSE TIME OVER STATIC DATA

Data Sets	Response Time (ms)				
	OS	IS	BF	BF/OS	BF/IS
500	291300	258900	294700	1.154	12.148
1000	1139000	1102800	1167300	2.424	5.526
5000	5402900	5277500	5396900	-0.111	2.212
10000	19693000	17140400	19950000	1.288	14.083
50000	70260500	66483400	76978700	8.727	13.634
100000	73084600	71379800	80552100	9.27	11.387

result with the previous result for changes. Due to space limitations, we report the result of the data transmission and response time evaluation here.

We simulate a typical client server application (e.g., Location-based service) for our experiments. We evaluated our query processing strategies in different scenarios: (1) Moving client issues queries on static data (2) Moving client issues queries on moving data (3) Moving client issues queries on moving data, where the number of moving points on the data set varies. We implemented the server side processing using the C programming language and the client using Java for Android programming. All the experiments were carried out on an Intel(R) Core(TM) i7-5500U CPU@2.40 GHz with 8 GB RAM computer running the Windows 10 Home (Version 1703) operating system.

We use six randomly-generated data sets that contain between 500 to 100000 points. For the first scenario experiment, the data sets contain static points, while for the second and third scenario experiments, the data sets contain randomly moving points. For each data set size, a user trajectory consisting of ten region queries of size (10x10) is created, which are used on both the static and moving point data sets.

A. Data transmission cost results

We recorded the cost of transmitting the result of each query to the client from the server. We calculate this cost by using 8 bytes per floating point number, 4 bytes per integer, and 1 byte per character. Although we report the results from the OS strategy, note that the data transmission costs from both the OS and IS strategies are the same.

Tables I, II and III shows the average data transmission cost result for the three scenarios respectively. The result shows that OS has a lower average data transmission cost in all three scenarios. This is because the server does not send the entire result file when the current results are the same as the previous results, but the BF strategy does.

The data transmission cost depends on the query result. If the result file contains a large data set, that means that a larger data transmission cost is incurred. Similarly, if the result set contains a smaller file then the data transmission cost will be small. OS does not send the entire result file when the current results are the same as the previous results, but if those result files are smaller, it makes a small reduction to the average data transmission cost from the BF strategy (though BF strategy sends the entire result files for all queries). Table II shows that the data transmission cost of OS for moving data is lower than the BF strategy, but not as low as for the static data set. For the moving data sets, the results of two queries may be different even though the superMBRs are the same. In this situation, the server using our strategy needs to send the entire result file to the client even though the superMBRs are the same. So, for some individual queries, the data transmission cost of our strategy is the same as the BF strategy. Nonetheless, OS has a lower average data transmission cost in comparison with BF because there still exist some queries that have the same superMBRs and no moving points within that region. From the Table III, we see that the data transmission cost is almost the same for different numbers of moving points and still lower than the BF strategy. If any number of points move, it does not significantly affect the data transmission cost. This is because once one point causes change in the result, the server must send the entire result, no matter if other points change or not.

B. Response time results

The query response time measured in microseconds and it is the time from receiving the query at the server until identifying and sending the result back to the client. Table IV, V and VI shows the average query response time results for the three scenarios respectively. For OS, for each query the server compares the generated result with the previous result before sending an outcome to the client. The query result may contain a large set of points and in those cases

TABLE VII
RESULT SENDING WORKLOAD OVER STATIC AND MOVING DATA

Data	**Result Sending Workload**	
Sets	static data	moving data
500	6	9
1000	6	9
5000	6	6
10000	7	8
50000	6	7
100000	6	7

TABLE V
AVERAGE RESPONSE TIME OVER MOVING DATA

Data	**Response Time (ms)**				
Sets	OS	IS	BF	BF/OS	BF/IS
500	626700	535100	566900	-10.549	5.609
1000	1795100	1743700	1946200	7.764	10.405
5000	4952200	4771900	4810900	-2.937	0.811
10000	19566100	19159800	19167200	-2.081	0.039
50000	50834800	49923800	50467300	-0.728	1.077
100000	83817000	73528800	103953200	19.371	29.267

TABLE VI
AVERAGE RESPONSE TIME OVER DIFFERENT MOVING POINTS

#Moving	Response time (ms)				
Points	OS	IS	BF	BF/OS	BF/IS
1	1677900	1631500	1711800	1.98	4.691
2	1855100	1488100	1609300	-15.274	7.531
3	1793600	1443500	1673000	-7.209	13.718
4	1730200	1616200	1658100	-4.348	2.527
5	1899600	1652100	1784300	-6.462	7.409

the comparison time adds additional overhead. On the other hand, for BF, the server generates the result and then sends the result to the client. In this way, sometimes the BF strategy shows lower response times because it sends the result while OS must compare the result first. IS significantly reduces the file comparison overhead, which can reduce the response time. The experiment results show that the query response time of IS is lower than the other two strategies. This is because IS compares only the changed points in order to identify any changes between two result files which is generally much less than the entire result file.

C. Result sending workload results

We measure the result sending workload by the number of times the entire result set is sent to the client over the entire trajectory. This is recorded for both the static and moving 1000 point sets. Table VII shows the result sending workload of OS over static and moving data scenarios. OS has lower result sending workload than the BF method. The BF method sends the entire result every time that is the result sending workload is 10 for all three scenario, though in some cases different queries can have the same result. In case of the third scenario experiment where we vary the number of moving points in 1000 points data sets, the result shows that the result sending workload does not significantly increase with the increasing number of moving points. Because once one point causes a change in the result, the server has to send the entire result, it does not matter if other points change or not.

IV. CONCLUSION

We propose two query processing strategies for location-based services. Both strategies are for continuous region query processing over moving point sets, where the query issuer object or client is also moving. The experimental results show that our strategies achieve significant improvements over the brute-force strategy. Our Original Strategy achieves up to 49.98% of data transmission cost performance improvement for static data and up to 3.07% improvements for moving data, over the Brute-Force strategy. Our Improved Strategy achieves up to 14.08% of response time performance improvement for static data and up to 29.26% improvement for moving data, as compared to the brute-force strategy. Finally, our strategies achieve a server workload reduction of between 10% and 40%.

In the future, we will compare our strategies with other query processing strategies which use other spatial index method instead of the mqr-tree. We are also planning to propose a client side processing strategy so that the client can locally process a query while moving within the superMBR region.

REFERENCES

[1] S. Ilarri, E. Mena, and A. Illarramendi, "Location-dependent query processing: Where we are and where we are heading," *ACM Computing Surveys (CSUR)*, vol. 42, no. 3, p. 12, 2010.

[2] F. Liu, "Query processing in location-based services," Master's thesis, University of Central Florida, 2010.

[3] S. Ilarri, E. Mena, and A. Illarramendi, "Location-dependent queries in mobile contexts: Distributed processing using mobile agents," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 8, pp. 1029–1043, 2006.

[4] F. Liu and K. A. Hua, "Moving query monitoring in spatial network environments," *Mobile Networks and Applications*, vol. 17, no. 2, pp. 234–254, 2012.

[5] C.-S. Lin and S.-Y. Wu, "Processing directional continuous range queries for mobile objects on road networks," in *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on*. IEEE, 2014, pp. 330–335.

[6] M. Gupta, M. Tu, L. Khan, F. Bastani, and I.-L. Yen, "A study of the model and algorithms for handling location-dependent continuous queries," *Knowledge and information systems*, vol. 8, no. 4, pp. 414–437, 2005.

[7] H. Wang and R. Zimmermann, "Processing of continuous location-based range queries on moving objects in road networks," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 23, no. 7, pp. 1065–1078, 2011.

[8] W.-S. Ku, R. Zimmermann, and H. Wang, "Location-based spatial query processing with data sharing in wireless broadcast environments," *Mobile Computing, IEEE Transactions on*, vol. 7, no. 6, pp. 778–791, 2008.

[9] K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous monitoring of spatial queries in wireless broadcast environments," *Mobile Computing, IEEE Transactions on*, vol. 8, no. 10, pp. 1297–1311, 2009.

[10] K. Park, "Efficient data access for location-dependent spatial queries," *Journal of Computer Science and Technology*, vol. 29, no. 3, pp. 449–469, 2014.

[11] M. Moreau and W. Osborn, "mqr-tree: A 2-dimensional spatial access method," *arXiv preprint arXiv:1212.1469*, 2012.

[12] W. Osborn and A. Hinze, "Tip-tree: A spatial index for traversing locations in context-aware mobile access to digital libraries," *Pervasive and Mobile Computing*, vol. 15, pp. 26–47, 2014.