Semantic Caching in Location-Dependent Query Processing*

Baihua Zheng Dik Lun Lee

baihua@cs.ust.hk *dlee@cs.ust.hk*

Department of Computer Science, Hong Kong University of Science and Technology

April 3, 2001

Abstract

A method is presented in this paper for answering location-dependent queries in a mobile computing environment. We investigate a common scenario where data objects (e.g., restaurants and gas stations) are stationary while clients that issue queries about the data objects are mobile. Our proposed technique constructs a Voronoi Diagram (VD) on the data objects to serve as an index for them. A VD defines, for each data object d, the region within which d is the nearest point to any mobile client within that region. As such, the VD can be used to answer nearest-neighbor queries directly. Furthermore, the area within which the answer is valid can be computed. Based on the VD, we develop a semantic caching scheme that records a cached item as well as its valid range. A simulation is conducted to study the performance of the proposed semantic cache in comparison with the traditional cache and the baseline case where no cache is used. We show that the semantic cache has a much better performance than the other two methods.

Keywords: mobile computing, location-dependent query, Voronoi Diagrams.

1 Introduction

With the advance of wireless networks and the popularity of portable electronic devices, mobile computing has been one of the hottest topics in computer science research in the past several years. Mobility has created new challenges to the existing computing infrastructures, such as databases, networks and so on. In the database research area, for example, data models must support the notion of user and data mobility as first-class data types. Furthermore, database systems must be able to represent location information efficiently to support complex location-dependent queries.

*Research supported by the Research Grants Council of Hong Kong, China (HKUST6241/00E).

The fact that clients in a mobile environment can change locations opens up the possibility of answering queries in a way that is dependent on the current position of the client [2]. These kinds of queries are called *location-dependent queries*. Examples of location-dependent queries are "find the nearest gas station to my current location," "find all the cinemas within a 1 km radius," "which buses will pass by me in the next 10 minutes?", and so on. While the data objects in the first two examples are stationary, they are mobile in the last example.

The provision of location-dependent information for the same user at different locations is a challenging problem. In addition, queries should be processed in such a way that consumption of wireless network bandwidth and battery power of the portable client is kept to a minimum. Techniques such as broadcast, caching, and indexing have been developed for this purpose. The focus of this paper is on location-dependent queries. Since users are mobile in a mobile computing environment, location-dependent queries must be answered according to the user's current location. For example, "find the nearest restaurant" would return totally different answers to the same user when the query is issued at different locations. If a user keeps moving after he/she submits a query, the problem becomes more complicated because the user's location is changing continuously and thus the results would change accordingly. How to answer a continuous query and provide an accurate answer is an open question.

Although a lot of research work been done in this area, methods for answering location-dependent queries efficiently and also guaranteeing the validation of the answer are not available. Our method makes use of Voronoi Diagrams to preprocess the data in order to answer location-dependent queries quickly, and a semantic cache to validate the answer.

Section 2 of this paper gives the definition of location-dependent queries, describes the difficulties of solving them, and outlines some existing work done in this field. Our method is described in Section 3. Section 4 gives the simulation and performance evaluation results. Finally, the summary and future work of this project are given in Section 5.

2 Related Work

In order to support location dependent queries, some basic requirements must be met:

1. Locating the user: The prerequisite of answering location-dependent queries is to locate the user. Available technologies that can provide user locations include GPS (Global Positioning System) and cellular wireless networks. Since these technologies have inherent errors, applications demanding high precision must take into account the location errors during query processing.

2. Maintaining the mobility of moving objects: Since the location information of moving objects keeps changing all the time, storing the location in the database and updating it whenever it changes is not an acceptable solution because of the frequent updates required. Some dynamic, intelligent representation should be adopted in order to reduce the number of updating to the database.

3. Predicting future movement trends: There is a time interval between the submission of the query and the return of the answer to the user. As such, prediction and validation must be done in order to guarantee that the answer is correct at the time when it is received by the user (not just at the time when the answer is computed). Also, there are queries about future situations such as forecasting traffic conditions of particular areas. In order to answer these kinds of queries, some future information should be predicted from current information.

4. Processing queries efficiently: Due to the expensive bandwidth of the wireless network and the limitations of computing resources for portable devices, query processing must be done efficiently. Technologies such as caching, data replication, and indexing can be used to improve efficiency.

5. Guaranteeing the boundaries of the precision: Neither the position of a current location nor the prediction of a future location is 100% accurate, so some mechanisms should be put in place to provide a bound to the error.

It is clear that many research problems have to be addressed and resolved before these requirements can be met satisfactorily. In this paper, we focus on the query processing aspect of location-dependent queries. A brief summary of work related to query processing is given in the following subsections.

2.1 Caching

The client cache stores frequently used information on the client so that queries can be answered without connecting to the server. In addition to providing answers quickly, the cache can also provide a limited level of services when a connection is not available.

- ***Data caches:*** Just like a traditional cache in a database, a data cache stores recently accessed data in the mobile client in order to save wireless bandwidth and improve access time. For location-dependent information such as local traffic information, cached data should also be validated when the client changes location. Xu et al. proposed a bit-vector approach to identify the valid scope of the data, and investigated a couple of advanced methods of validating caches based on this approach [15].

- ***Semantic caches:*** A semantic cache stores data and a semantic description of the data in the mobile client [10]. The semantic description enables the cache to provide partial answers to queries which don't match the cache data exactly. As such, wireless traffic can be reduced and queries may be answered even in the case of disconnections. This characteristic makes a semantic cache an ideal scheme for location-dependent queries. A cache method was proposed in [11]. A tuple $S = \langle S_R, S_A, S_P, S_L, S_C \rangle$ was used to record data in the local client. S_R and S_A are, respectively, the relationships and the attributes in S; S_P is the selection conditions that data in S satisfy; S_L is the bound of the location; and S_C represents the actual content of S. When a query is received by the client, it is trimmed into two disjointed parts: a *probe query* that can be answered by some cached data in the client, and a *remainder query* that has to be transmitted to the server for evaluation.

2.2 Continuous Queries

A *location-dependent query* becomes more difficult to answer when it is submitted as a continuous query. For example, a client in a moving car may submit the query: "Tell me the room rate of all the hotels within a 500 meter radius from me" continuously in order to find a cheap hotel. Since the client keeps moving, the query result becomes time-sensitive in that each result corresponds to one particular position and has a valid duration because of location dependency. The representation of this duration and how to transmit it to the client are the major focuses of *Continuous Queries (CQ)*. Sistla et al. employed a tuple $(S, begin, end)$ to bound the valid time duration of the query result [13, 14]. Based on this method, they also developed two approaches to transmitting the results to the client: an *immediate approach* and a *delayed approach*. The former transmits the results immediately after they are computed. Thus, some later updates may cause changes to the results. The latter transmits S only at time *begin*, so the results will be returned to the client periodically, thus increasing the wireless network burden. To alleviate the limitations of both approaches, new approaches, such as the Periodic Transmission (PT) Approach, the Adaptive Periodic Transmission (APT) approach and the Mixed Transmission (MT) Approach, were proposed [5, 6].

2.3 Query Types

According to the mobility of the clients and the data objects to be queried by the clients, location-dependent queries can be classified into three types:

- ***Mobile clients querying static objects:*** Queries like: "Tell me where the nearest gas station is" and "Where is the nearest restaurant?" are popular queries in real-world applications. In general, the clients submitting this kind of queries are mobile and the data objects are fixed. The main challenge of this type of queries is how to get the locations of the clients

and also guarantee the validation of the results when the client keeps moving during the query evaluation process. Queries such as "Report all the available hospitals within a 500-meter radius" are an extension of this type of query.

- ***Stationary clients querying moving objects:*** An example of this type of query is "Report all the cars that pass gas station A in the next 10 minutes." Here, gas station A is static and the moving cars are the objects being queried. The result only depends on the location of the moving cars. Actually, this is an extension of traditional database queries with dynamic answers to the same query. This query is considered as a location-dependent query because the data objects are all mobile and the answer to the query depends on the locations of the data objects. Consequently, this type of query requires good representation of moving objects and efficient indexing techniques. Usually, this type of query is interested in information about the future, so later updates will cause some objects to become unqualified for the query condition. As such, the real-time guarantee of the validity of the answer is also a challenge. In an earlier paper [3], the authors mentioned the invalidation of results containing location-dependent data caused by later updates, and considered that objects that satisfy the query at one moment might become disqualified after being updated.

- ***Mobile clients querying mobile objects:*** That both the clients submitting the queries and the data objects are continuously moving is the main characteristics of this kind of query. For example, a query of this type:"Tell me all the cars that will pass me after 20 minutes" while the client is driving on a highway, is very complicated since it is a combination of the first two types. Even a simple query like: "Tell me all the cars that will pass me within the next 20 minutes" bears the characteristics of CQs.

Of course, all the queries listed above can also contain conditions querying about location-independent attributes, such as: "Tell me the nearest restaurant providing Chinese food." Since these queries can be broken down into two parts: one for location-dependent information and the other for location-independent attributes, we only consider queries on location-dependent information as the others can be retrieved by traditional query-processing methods.

Most, if not all, of the location-dependent queries can be categorized as one of the three types described above. We can analyze each type separately in order to define the scenario clearly and simplify the problem. The rest of this paper will focus on the first type since some research on the second type has already been done [7, 8], and research on the third type can be simplified if solutions to the first two types have been found. However, to the best of our knowledge, no previous research has been done on type one queries. The remainder of this paper will introduce a technique in order to solve the first type of query.

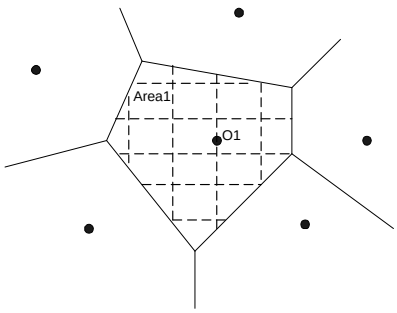

Figure 1: Voronoi Diagrams used in nearest neighbor queries

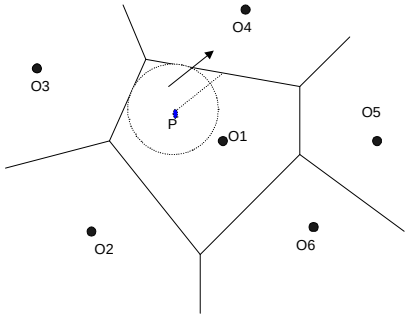

Figure 2: Semantic Caches in Voronoi Diagrams

3 Voronoi-Diagram-Based Indexing

From the example queries in the previous section, we can see that queries to find the nearest service are very useful and popular. In this section, we present a method for retrieving the nearest service efficiently in a mobile environment. An assumption of our work is that the location of a client is known from GPS. When a user submits a query, its current location and speed together with the timestamp are also submitted.

3.1 Basic Data Structure

Our method makes use of Voronoi Diagrams (VDs), which, by nature, are suitable for nearest-neighbor queries. A Voronoi Diagram records information about what is close to what. Let $P = \{p_1, p_2, \cdots, p_n\}$ be a set of points in the plane (or in any n-dimensional space), each of which we call a *site*. Define $\mathcal{V}(p_i)$, the *Voronoi cell* for p_i, to be the set of points q in the plane such that $dist(q, p_i) < dist(q, p_j)$. That is, the Voronoi cell for p_i consists of the set of points for which p_i is the unique nearest neighbor of q:

$$\mathcal{V}(p_i) = \{q | dist(q, p_i) < dist(q, p_j), \forall j \neq i\}. \quad (1)$$

As shown in Figure 1, all the points in the shadowed region $Area_1$ have the same nearest fixed point, namely, O_1.

Since a VD is a traditional data structure in computational geometry, adequate structures for storing a VD and efficient point location methods for locating a point in a region are available [4]. As such, we assume without further description that a standard structure for representing the information in a VD and a corresponding location method are available.

In the scenario of finding the nearest restaurant, the restaurants are the fixed sites and the mobile clients are the points needing to be located. Once the mobile client is located in one area, the unique fixed site of this region is its nearest neighbour. The good thing about this type of

query is that all the services are fixed and only change occasionally. Furthermore, the location information of the services is available before query submission. Thus, preprocessing can be done to construct the corresponding VD of different services. Three data structures are used to record the preprocessed data. The first one is *edge*, denoted by $\langle id, x_1, y_1, x_2, y_2 \rangle$, which is used to record the segment *id* and the endpoints of a segment. The second one is *service object*; it records the position of the service object (i.e., the site in the definition of Voronoi Diagrams) and its bounding edges. It is a tuple $\langle id, x, y, number, list \rangle$, where x and y are the coordinates of the site, *number* is the number of edges bounding this site, and *list* is the list that records the *id*s of all the edges. The last one is *edge_service*, which records the information between the service objects and the edges using a tuple $\langle segment\_id, serv\_object\_id_1, serv\_object\_id_2 \rangle$. Since the return set of *point location* algorithm that we use is the edge that is just below that point, the mapping relationship between the edge and corresponding site should be maintained. id_1 and id_2 are the sites above and below this edge respectively. Currently, this method is just like the traditional static data cache for nearest neighbor queries, but there are two major differences:

1. The client submitting the query is always mobile, so the query processing method should handle the validation of the answers while the location of the client changes. It should also predict the possible future answer according to the direction and speed of the mobile client, i.e., the result of our method is dynamic while the traditional answer is static.

2. Our method supports the semantic cache that is not provided in a traditional database.

3.2 Data Preprocessing

Although VD is the most suitable mechanism to find the nearest neighbor, it is seldom used in real applications because of the expensive maintenance of the structure when updating occurs. Fortunately, in the context of this paper, the chance of having to change a real service (moving or rebuilding) is very small. Furthermore, a large geographic area is likely to be divided into small regions, each of which is usually a cell covered by a base station in the wireless system. Within each region, we can classify the services into different kinds such as restaurants, hospitals, gas stations, etc. For each kind of service, a VD index is constructed based on the data structures described above. Overall, the maintenance cost of a VD index is reasonable considering the gain in query performance.

3.3 Query Processing and Semantic Caching

Now we consider how to answer the query: "Tell me where the nearest restaurant is." When answering this query, we report the nearest restaurant from the VD index. Based on the known

speed of the client, the next nearest restaurant can also be predicted. As denoted in Figure 2, point P is the mobile client and the arrow indicates the direction of the client's movement. Currently, the nearest restaurant to P is O_1 and after P crosses the line between O_1 and O_4, the nearest restaurant should be O_4. Knowing the length of the dash line and the speed of P, we can estimate the time when P crosses the line, say, two minutes later. Then the answer should be presented as $\langle O_1, 2, O_4 \rangle$. The first object of this tuple is the answer according to the current location of client P, the second element is the time interval during which the first answer is valid, and the last one is the predicted new answer after the valid time interval.

One thing to notice is that the client may change its speed and direction. Thus, the valid duration of the first answer cannot be guaranteed using this mechanism. One way to prevent any false answers is to resubmit the query when the client changes its speed or direction. Here we propose a better alternative that uses the maximum speed and the shortest distance to guarantee a valid duration. The shortest vertical distance between the current position of the client and the bounding edges of this site divided by the maximum speed of the mobile client can be guaranteed to be valid. This method may make the valid duration of the answer shorter than the real one, but it will not produce any invalid results.

A circle in Figure 2 should also be noted. This is the maximal circle having P as its center and the whole circle is within the region of O_1. Actually, the radius of this circle is the shortest distance that we used in our scheme to get the valid duration of the first answer. We store the location of P, the radius of the circle, and also O_1 in the client cache as $\langle P.x, P.y, radius, O_1 \rangle$. From the above example, we can see that the cache actually contains information about many circles. If the position of the client submitting the query is within one of the circles, then the nearest restaurant within this circle is the answer to the query, which can be answered without connecting to the server. For simplicity, the predicted answer is omitted for clients who can get the result from a local cache. In other cases , we should transmit the whole query to the server and create one new record in the cache after we get the result.

In summary, the following steps are taken by both the client and the server to answer a query:

1. The local cache is checked to see whether the information is available. If there are no suitable cache records corresponding to the location of the client, one should proceed to the third step. Otherwise, one should continue.

2. If there is some related information, the most appropriate piece can be retrieved from the cache and then this query is finished.

3. The current location of the client and also the speed of it are transmitted to the server. The server will first locate this client in the VD index and find the nearest restaurant, and then compute the maximal circle around it within this region.

4. Based on the region of the nearest restaurant and the speed of the client, we can identify the time when this client moves to another region.

5. The result is returned to the client.

6. After the client gets the result from the server, a new cache record is inserted into the local cache.

In Step 4, we use the first scheme (which assumes a constant speed and direction). If the second scheme that can guarantee the valid duration described above is used, the maximum speed rather than the current speed of the client should be used.

4 Simulation Model

In order to evaluate the performance of the proposed method, a simple simulation model has been established using CSIM [12]. In the following section, the model setting is described first and then the performance evaluation result is presented.

4.1 Model Setting

In our simulation model, we simulate one cell, i..e., one base station covering a number of clients. The cell has a limited geographical coverage and the clients have a speed limit. When a client crosses the border of the cell, it is deleted and a new client lying within the cell is produced randomly. In other words, the number of clients within one cell is constant. The server answers queries according to the order of which the requests arrived. It takes $ServiceTime$ to finish one query. The client sends out a new query only after the current one has been answered. It can send out the new query immediately or after some time. The longest duration is denoted as $Max\_Thinktime$. $ServNum$ means the number of the facilities available within this cell. Assuming that we are interested in the nearest restaurant, the facility here is the restaurants and $ServNum$ is the number of restaurants within the cell. Table 1 is a summary of the setting. In the following experiments, the setting is used unless otherwise stated.

Given $ServNum$, the positions of the service objects are produced randomly, and the corresponding Voronoi diagrams are produced using an available program *Triangle* [1], which uses $O(n \times log(n))$ space to build a VD of n service points.

Figure 3 shows a set of 20 nodes (i.e., $ServNum = 20$) produced randomly by the system. Figure 4 shows the corresponding Voronoi Diagram produced. Based on the VD produced, all the necessary data can be precomputed and stored in the memory for the simulation program to use.

Parameter	Description	Setting
SimNum	No. of queryies answered by the server in one simulation. When the server answers SimNum queries, the simulation is finished.	$5000 - 45000$
ClientNum	No. of clients	1000
ServNum	No. of service objects	$10, 20, 30$
UpperBound_Y	the maximal value of the y_coordinate	1000
LowerBound_Y	the minimal value of the y_coordinate	0
RightBound_X	the maximal value of the x_coordinate	1000
LeftBound_X	the minimal value of the x_coordinate	0
Max_Speed	the maximal speed of the client	10
ServiceTime	the time the server used to answer one query and also send the request	1.0
Max_Thinktime	the maximal time duration between one client's receiving the answer to the original query and its sending out the next request	20.0
CacheSize	the size of the client's cache	10

Table 1: Simulation model setting

4.2 Simulation Results

In our simulation, a large number of tests have been carried out. However, due to limitations of space, only some of the performance graphs are shown. For those which are not shown, we describe the observations whenever necessary.

4.2.1 *Client cache types*

The cache has a significant impact on the performance of this method. We tested the average waiting time for the clients using different cache schemes: *no cache* means that the client submits every query to the server and does not store any answers for later use; *normal data cache* means that the client just stores the answer to a query in the local cache and the cached answer can be reused when it submits the same query at the same location, and *semantic cache* means that the client stores not only the answer in the local cache but also the description of the valid area of this answer.

The performance of these three cache schemes is given in Figure 5 and Figure 6 with 10 and 50 service objects, respectively. We also performed a simulation using 20 and 30 service objects and found that the results were similar. We observed that for clients using no cache, the performance

Figure 3: The Location of the restaurants

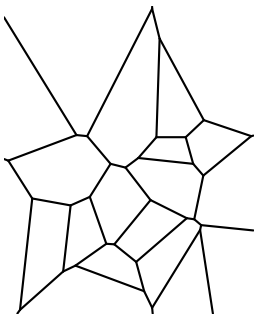

Figure 4: Corresponding Voronoi diagrams

is the worst, whereas the semantic cache produces the best result. This result is consistent with common sense because for *no cache*, every query must be submitted to the server; for a *normal data cache*, the chance that a client submits the same query at the same location is nearly zero, and so is the probability of reusing cached answers; and for a *semantic cache*, the client has a much higher probability of remaining in the area where the cached answer is valid and thus experiencing the best performance.

In the simulation, the positions and the speeds of the clients are produced by *CSIM* randomly. At the beginning of the simulation, the whole system is not in a steady state since there are fewer requests. Thus, the average waiting time during the initial period is shorter than the average in the stable state. In Figures 5 and 6, the difference between the first and the second points is very obvious. For a cell with a different number of service objects, the warm-up duration is also different. Basically, the average waiting time of a semantic cache decreases with the increase of the simulation time as long as it has not arrived at the maximum cache hit rate.

We also conducted the simulation using another benchmark program called *GSTD*, which generates sets of moving points or rectangular data following an extensive set of distributions [9]. Figure 7 is the result. A large number of snapshots indicate that GSTD regenerates the set of moving clients at a higher frequency (i.e., a higher temporal resolution).

4.2.2 *Number of service objects*

From the results of the above experiments, the conclusion is that the number of service objects can also affect the average waiting time. For clients with no cache, the impact is negligible, but for those with a semantic cache, the impact is obvious. This is because in one fixed-size cell, the more service objects there are, the smaller is the area in which cached data remain valid. This means a lower cache hit rate and a longer average waiting time. Figure 8 illustrates the negative impact of the number of objects on the cache hit rate. It can also be observed that the cache hit rate increases with the rise of the simulation time.

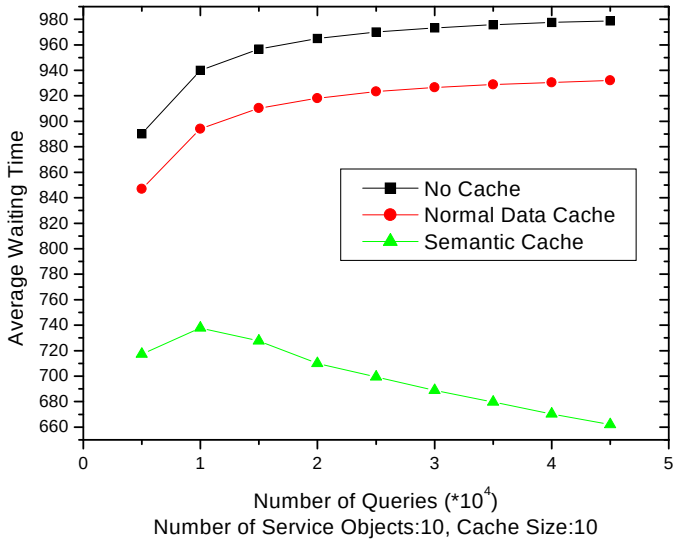

Figure 5: Average waiting time given 10 objects

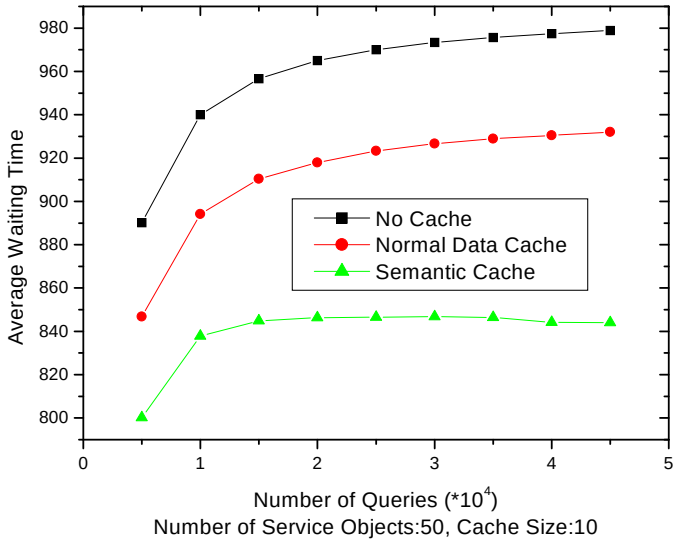

Figure 6: Average waiting time given 50 objects

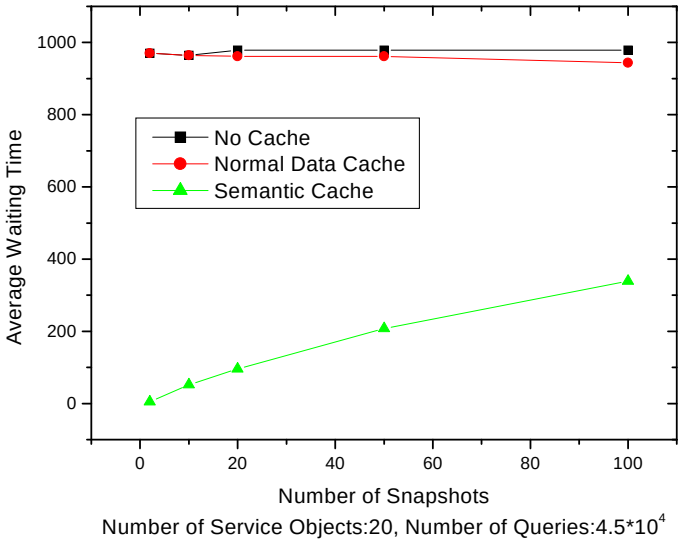

Figure 7: Average waiting time given 20 objects

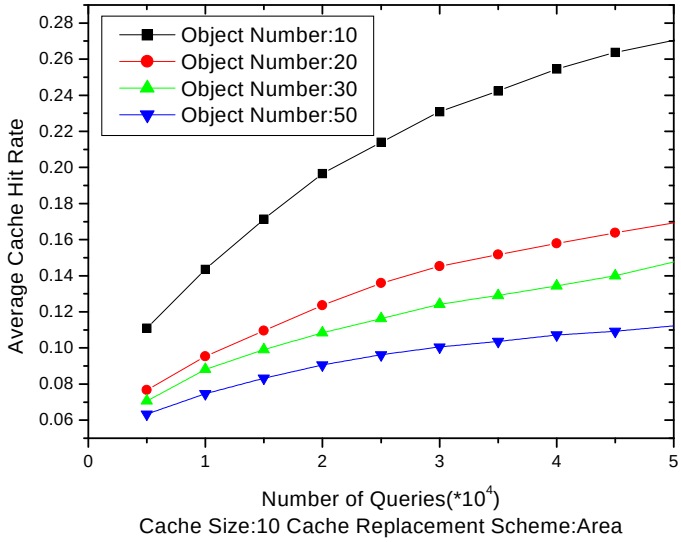

Figure 8: Average cache hit rate

In summary, when the number of service objects is limited, a semantic cache and a VD index are very efficient. When the number becomes very large, the semantic cache can only contain a very limited area and the cache hit rate is reduced. As such, the advantages of a semantic cache are not obvious.

4.2.3 *A cache replacement scheme*

Since the size of a local cache is usually limited, a cache replacement scheme is needed. For our method, three different cache replacement schemes have been devised.

- *Area:* Since the cached data are represented by many circles, this scheme is based on the areas of the circles. The newly-added data always replaces cached data corresponding to the smallest area, with the aim of making the total area corresponding to the cached data the largest.
- *Dist:* Based on the distance between any two centers of the cached data, this scheme replaces the cache item that has the shortest distance from the item to be inserted into the cache. This is based on the assumption that the client will soon cross the nearest voronoi cell.
- *ComA:* This scheme replaces the circle that has the biggest common area with the newly-added one. The objective of this scheme is to make the relatively changed area less, this is a modification of the *Area* scheme.

From the simulation results (Figure 9 and Figure 10), *Area* and *ComA* have almost identical performance whereas *Dist* does not work as well as the other two. In the simulation result presented in the following subsections, we adopt *Area* as the cache replacement method.

4.2.4 *Frequency of updating speed*

Using the benchmarking algorithm *GSTD*, we can control the number of snapshots. The larger the number, the more frequent the client changes its speed. Observing the simulation result presented in Figure 11, this also has a significant impact on the cache hit rate: the more frequent the change, the lower the cache hit rate.

4.2.5 *Range of the think time*

The *Max_Thinktime* indicates the frequency that a client submits queries. The higher the frequency, the more queries need to be answered, so the longer the waiting time is. If nearly all the clients submit a query after a long duration, the number of queries that one server needs to answer will be less, so the average waiting time will be shorter. This can be observed in Figure 11.

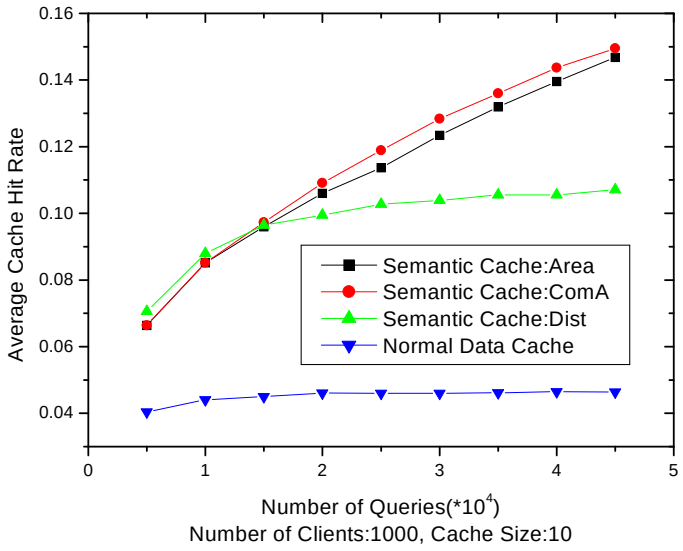

Figure 9: Average cache hit rate given 30 objects

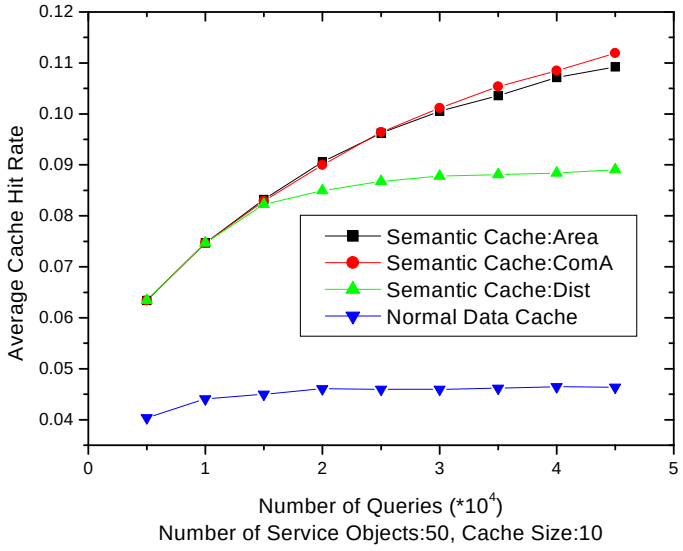

Figure 10: Average cache hit rate given 50 objects

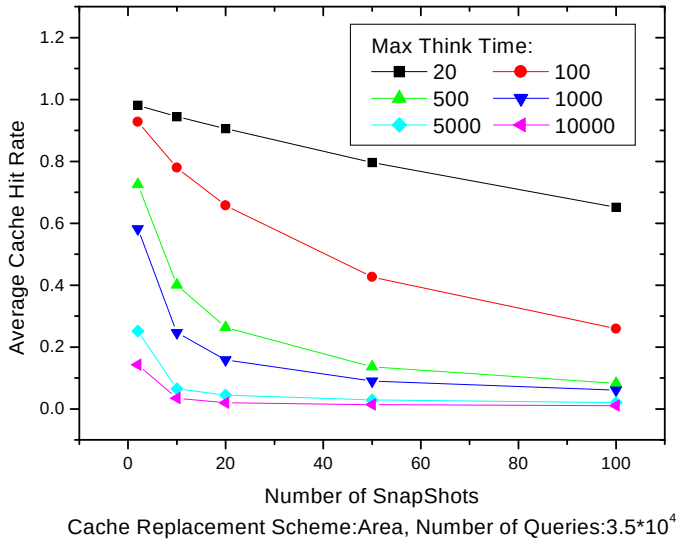

Figure 11: Average cache hit rate given 20 objects

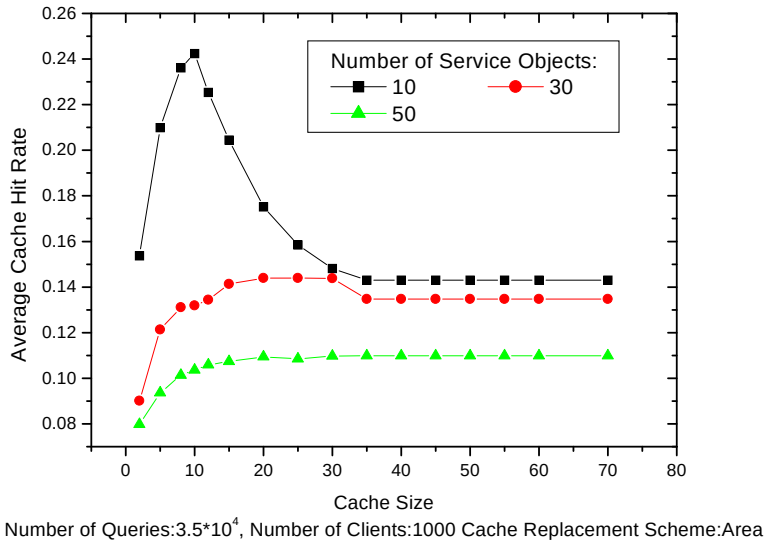

Figure 12: Cache size vs performance

4.2.6 *Cache size*

The size of the cache plays an important role in its performance. Usually, the larger its size, the better its performance. The results show that the performance does not increase with the size of the cache monotonously. Figure 12 shows the simulation result. Here, we can notice that given the number of service sites, an optimized cache size can be obtained. How to get this optimized one is a question we will address in future research.

5 Conclusion

In this paper, we presented an indexing and semantic cache method for location-dependent queries based on the Voronoi Diagrams. Various cache replacement strategies were proposed and evaluated. We conducted a simulation to evaluate the performance of the proposed methods under different parameter settings. We concluded that the semantic cache method performs much better than the normal data cache method.

In this paper, we only considered a single cell where the clients and objects are produced randomly. In future research, we will consider the handoff problem and cache invalidation methods in a multiple cell environment. Furthermore, since the positions of the clients and data objects as well as the speed of the clients are produced randomly, we will investigate the performance of the proposed schemes using different distributions.

References

[1] Triangle: A two-dimensional quality mesh generator and delaunay triangulator. http://www.cs.cmu.edu/~quake/triangle.html.

[2] D. Barbara. Mobile computing and databases-a survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), January -February 1999.

[3] Budiarto, K. Harumoto, M. Tsukamoto, and S. Nishio. Position locking: Handling location dependent queries in mobile computing environment. In *Proceeding of the Wordwide Computing and Its Applications*, 1997.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*, chapter 7. Springer-Verlag, 2000.

[5] H. G. Gök. Processing of continuous queries from moving objects in mobile computing systems. Master's thesis, Bilkent University, 1999.

[6] H. G. Gök and Ö. Ulusoy. Transmission of continuous query results in mobile computing systems. *Information Sciences*, 125, 2000.

[7] G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest neighbor queries in a mobile environment. In *International Workshop on Spatio-Temporal Database Management*, September 1999.

[8] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *18th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*, May 1999.

[9] M. Nascimento and Y. Theodoridis. Benchmarking spatial-temporal databases: The gstd software. http://www.cti.gr/RD3/GSTD/.

[10] Q. Ren and M. H. Dunham. Semantic caching in mobile computing. Submitted, under revision, 2000.

[11] Q. Ren and M. H. Dunham. Using semantic caching to manage location dependent data in mobile computing. In *The Sixth Annual International Conference on Mobile Computing and Networking*, August 2000.

[12] H. Schwetman. *Csim User's Guide (version 17).* MCC Corporation, 1992.

[13] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Thirteenth International Conference on Data Engineering*, April 1997.

[14] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. *Querying the Uncertain Position of Moving Objects*, pages 310–337. Springer Verlag, 1998.

[15] J. L. Xu, X. Y. Tang, D. L. Lee, and Q. L. Hu. Cache coherency in location-dependent information services for mobile environment. In *The First Conference on Mobile Data Management*, December 1999.