# Defining Scope of Query for Location-Dependent Information Services

James Jayaputera and David Taniar

School of Business Systems
Monash University, Clayton
Vic 3800 Australia
{James.Jayaputera,David.Taniar}@infotech.monash.edu.au

**Abstract.** The demand of information services has fueled in recent years. However, the requested of correct answer in a mobile environment needs to have more attentions. This is due to the scope of query depends to the user location. In this paper, we propose an algorithm by using a square as a query scope. The aim is to increase chances to find rare targets. We assume that the client is at the corner boundary of the square when it receives the answer. Our results show that using a square is more efficient in finding the target if the target is rare instead of using other shapes, especially circle.

## 1 Introduction

The appearance of mobile devices has became popular and fueled almost every human life unrestrictly. This situation enable mobile user to use their devices anywhere while imposing or maintaining network connection compare to traditional computing model. Therefore, a Base station needs to maintain their movement by identifying mobile user devices. A Base station is a host that does an address translation from a static network to wireless devices [1]. Low resources (eg. processing speed, memory and power), frequent disconnections and slow speed transmission [2,3], are the most common problems in a mobile environment. The capability of mobile users to change their locations brings some information services, called *Location-Dependent Information Services* (LDIS) [4]. This means that if a user sends a query and then changes his/her location, the answer of that query has to be based on the location of the user issuing the query [5,6,7].

In this paper, we propose a new approach to define a valid scope by using a square in order to always provide with a valid query result. The valid scope is defined as a user boundary taken from the distance of the user query. The aim of this paper is to evaluate if the performance of square as a valid scope to find objects inside the scope feasible.

To simplify our discussion, a geometric location is represented as two dimensional coordinate, the velocity of user is always constant towards x and y coordinate, the velocity value of x and y is same. We assume that a user has not moved to a location of a given query upon receiving the answer and the user always does not travel further than the the scope boundary.

The correct query result should be bounded in the current user location. The query result may become invalid when a user changed their location. For example, a user is located in the country side area at night. He/she thinks that there will be a restaurant opened at night within 1 km. Therefore, He/she sends a query to find a restaurant within 1 km from his/her mobile device. There is only one restaurant which is opened at that time and located 1.1 km away from the user location. If we use a circle, the user will get a message "no restaurant found" from a server. Therefore, the user needs to resubmit a new query again to the server. On the other hand, if the square is used, then the user will get a message "there is one restaurant found within 1.1 km from your location" without a query resubmission again.

To generate a correct query result for the query above, we need to define a shape as a valid scope. Some works have been done to define a shape as a valid scope [8,9,10]. They represent Polygonal Endpoints (PE) [9] and Approximate Circle (AC) [8] to define a valid scope for cache invalidation [8,9]. The valid scopes are represented with different levels of fidelity and liability. A rectangular was used to find a nearer object, which is a static or moving object, to a given query [10].

The rest of this paper is organized as follows. In next section, some related works of this paper are presented. In section 3, our proposed algorithm will be discussed. In section 4, we show the performance of our proposed algorithm. Finally, the last section will summarize the contents of this paper.

## 2 Related Work

Defining a valid scope for a mobile client is important to generate a correct answer to a given query since the mobile user has moved to a new location [10, 4]. In this section, we analyze previous studies on defining a valid scope. The existing works focussed on defining a valid scope using polygon, rectangle and circle. None of them are using square to define a valid scope. According to Zheng et al [4], a valid scope can be defined by using Polygonal Endpoints(PE) and Approximate Circle(AC). The PE scheme will be discussed first, followed by the AC scheme.

A direct way to explain the valid scope of data value is using PE scheme. All endpoints of the polygon were recorded to define a valid scope. However, it is hard to define the boundary of polygon when there are a large number of endpoints.

Another way to define a valid scope is using the AC scheme. The AC scheme is one of the most convinient ways to generate a valid scope, if we know how far the user would like to find an object. Otherwise stated, a valid scope can be defined by the center of the circle and the radius of value. However, it has the same problem as polygon to find the boundary of circle unless we calculate the distance of the object to the user location. Another problem of using a circle, if an object location is a bit further from the circle boundary, the server could not categorize the object as a valid object, especially if the object is rare.

In addition, the maximum size of circle [11] can be defined as the current velocity of the user. The advantage is to predict the valid scope at the current speed in a time interval. However, if the user moves to a new BS boundary in two seconds, this prediction will be invalid for the old BS after two seconds. Unless the user resubmit a new query.

On the other hand, Stanoi et al [12] addressed a solution to answer Reverse Nearest Neighbor queries in two-dimensional space. They divide a space around the client location into six equal regions by a straightline intersecting the client location.

The TPR-tree [10] uses a rectangle to enclose moving objects at all times in the future. In this scheme, the size of rectangle is extended based on the velocity and time. Therefore, the number of targets remained inside the rectangle will increase as the size of rectangle is increased.

## 3 Proposed Algorithm

In this section, we propose an approach for location-dependent query that a valid scope of user to get a correct answer from a server is presented by a square. The scope is generated by a server based on the requesters's location. There are a number of shapes to represent to define a scope of user query, such as rectangles, triangle, polygon, circle and so on. In early section, we discuss why we choose to use a square instead of other scopes. Followed by our proposed algorithms and some examples to support our algorithm. We choose a square, because it is more accurate and easy to find a target closest to the user compared to other shapes and it has same length and width. Targets are defined as objects probed by a user. The dimension of a square will be represented by the distance of the user query to the left, right, top and bottom. If an input is an area, then, the dimension of the square can be found by taking a square root of the area ($\sqrt{area}$). Therefore, a distance from a user as center point can be found and then, a square can be formed.

Other shapes can be used to define a valid scope, however, it is hard to know whether a target is valid in a situation. Let us consider to use a triangle to represent a valid scope. Assume that the distances from the center to left, right and top are same. These distances tell us that base is twice than its height. If we calculate the area of triangle, then, the area of triangle is same as the area of square. However, it is hard to decide whether a target is inside the boundary. On the other hand, if we consider to use a circle as a valid scope, then, the circle area is still smaller than the square. We will not consider to use a rectangle, since the horizontal and the vertical distance is not the same.

Figure 1 shows all locations of vending machines, a restaurant and a user within the BS boundary. Assuming a user would like to find the nearest restaurant within $n$ kilometres or $m$ square kilometres, where $n$ and $m$ is a number which represent a distance or area where the target will be probed. The BS will

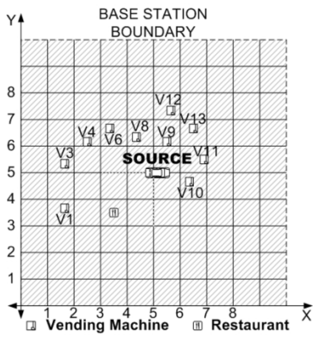


**Fig. 1.** Init Situation

give all locations requested within the boundary of Base Station (BS) which is presented by an outer shape. In order to get a valid answer, the BS need to keep track the current location of a user. Otherwise, the answer will be invalid when the user has moved, even though, its movement is still in the same boundary of the BS.

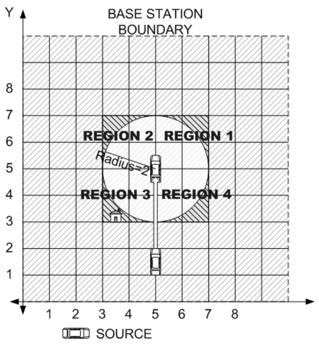


**Fig. 2.** The proposed approach

Let us consider if a user would like to find a restaurant within a distance of 2 km. The query result will be *null* when we define a circle as a scope of query, unless the user resubmit a new query again to the server. On the other hand, the restaurant will be found if we use a square as a valid scope, because the square has a greater scope compared to a circle. Hence, the user only needs to submit a query once. Figure 2 shows our proposed approach. The inner circle shows if we define a circle as a valid scope. The possibility to find the restaurant is smaller than using a square.

Figure 3 shows our proposed algorithm. We explain our algorithm by using the example above. The user sends a request which contains the current position,

```
Let $time_{start}$ <- Receiving Time
Let Queryrequest <- (location, query, speed, direction)
Let forwarded <- false
Let all_target_found <- empty
Let current_location <- $Query_{request}$->location
Let length <- distance from the query multiply by 2

Create a scope with dimension length multiply by length for current_location
Divide scope into 4 equal area where the user is a center point

If direction == 0 then
  check all targets in 4 regions

While (forwarded != true)
 If direction == down then
   all target found <- add all target found in (region 3+4) at $time_{start}$
 If direction == left then
   all target found <- add all target found in (region 2+3) at $time_{start}$
 If direction == up then
   all target found <- add all targets found in (region 1+2) at $time_{start}$
 If direction == right then
   all target found <- add all targets found in (region 1+4) at $time_{start}$
 If the source is moving to NE, NW, SW, SE directions, then
   all target found <- add all targets found in region 1, region 2, region 3,
                       region 4 respectively at $time_{start}$
 Forward all target found to user
 If the answer is not received then
    $time_{start}$ <- $time_{start}$ + 1
    Update current_location <- current location at $time_{start}$
    Create a new scope for current_location
 Else
    forwarded = true
 End if
End loop
```

**Fig. 3.** The Proposed Algorithm

speed and movement direction to the server. The server gets the client request and set initial value. The current_location parameter is set to the current position of user. Then, the server generates a square as a scope to search the targets. Since a square has same height and width, therefore, dimension of the square is presented by the *length* parameter. The *length* parameter is the searching distance from the client request and multiply the distance by two. We multiply by two, because the length is a distance from the user to the left and the right sides.

After we defined a scope, we divide the scope four equal areas which is shown in figure 2. The aim of this division is to speed up the searching process on server side. Then, the server will check if the user is moving or stopping. If the user is stopping, the server will search all regions. Then, the server will forward the query result to the user. The user receives the query without changing his/her location and the server set the value of *forwarded* parameter to be *true*.

If the user is moving, the server will search targets within certain regions based on the movement direction of the source. If the user is moving down, the server will only check region 1 and 2 of the generated scope. If the user is moving diagonally to North East direction, the region 1 will be searched. We assume that the user will only move to the middle or the corner of the scope boundary for present time. When the server got the result, it will forward the query result the user. If the result is not received at time $t_{start}$, the server will generate a new query result for next location at $t_{start+1}$. Otherwise, the server will set the value of *forwarded* parameter to be *true*.

### 3.1 Examples

There are some situations where the mobile client is not moving, moved to close destination or moved to a far destination from its original location when it receives an answer from the server. This will result some target in the answer invalid, therefore the server needs to regenerate a new answer based on the current location. The examples below illustrate the situations mentioned above.

1. *The mobile user stopped*
   Consider a mobile user is located at point (5,5) and sends a query to a server. The query is *Find a closest restaurant within 2 km.* This user will stay at same location when the answer is given from server which is shown in Figure 1.

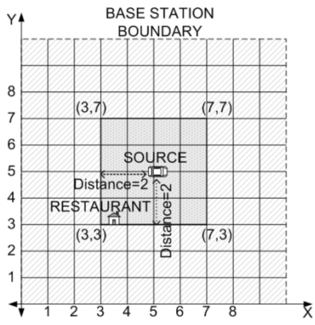


**Fig. 4.** The mobile user stopped

   The server will generate a valid scope by adding and substracting the distance towards the mobile user position. Therefore, we have a square that is formed by the following coordinate: Top Right: (7,7), Bottom Right: (7,3), Top Left : (3,7), Bottom Left: (3,3), shown in Figure 4.

   After, the valid scope is produced, it will search a restaurant within range, $3 < x < 7$ and $3 < y < 7$. In other words, all regions will be searched. Once, the server find a restaurant within that range, then the server will generate an answer for that query. The answer will match the description of the target, the position of the target and the position of the query requested. Once the answer is ready, the server will check about the current position of the mobile user.

2. *The mobile user is moving down*
   Let consider, "Find all vending machines within distance 2 km" and the user is moving down as shown in figure 5. The server will generate a new scope for the location at $t_{start}$. The server will search targets within region

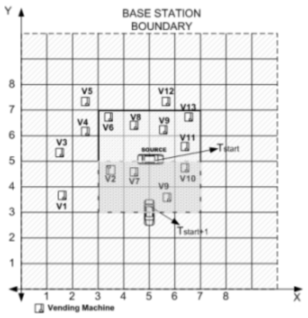


**Fig. 5.** The User is moving down

3 and 4 instead of all regions, because we assume the user will receive the query result at $t_{start+1}$. We assume that the user will arrive at point (5,3) at time $t_{start+1}$. If the server produces the query result by searching all regions, it will waste the server resources. Because the vending machines; (v6, v8, v9, v11 and v13), will become invalid answer and only vending machines; (v2, v7, v9, v10), will only become a valid answer.

3. *The mobile user is moving to the corner of scope boundary*

   Let consider, "Find all vending machines within distance 2 km" and the user is moving to the corner of scope boundary as shown in figure 6. In the beginning of process, the server will generate a new scope for the location at t_start. The server will search targets within region 1 instead of all regions, because we assume the user will receive the query result at $t_{start+1}$. We assume that the user will arrive at point (7,7) at time $t_{start+1}$. The only valid vending machines will be vending machines (v9, v11, v13). Therefore, the server will save the resources by only search one region instead of all regions.

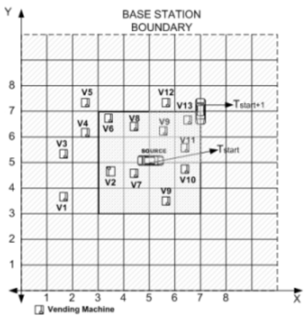
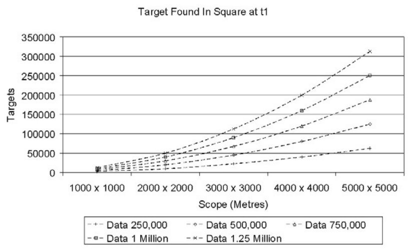


**Fig. 6.** The user is moving to the corner of scope boundary

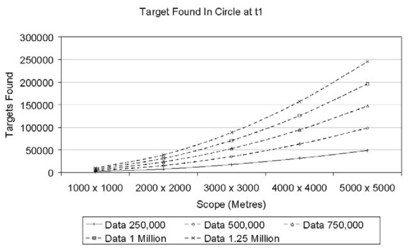
## 4 Performance Evaluation

In this section, we will examine the performance of our proposed algorithm. Our simulator is implemented in Java, running under Linux Fedora Operating System. The simulation database contains various number records of the random number of x,y. The number of records in database are varied from 250,000 up to more than 1 million records. The experiments presented are designed for two objectives. First, we examine the performance differences between square and circle. Second, we evaluate the performance of our algorithm to find a various number of objects in various size of scopes.

Figure 7a shows the number of targets found within scope 1000 x 1000, 2000 x 2000, 3000 x 3000, 4000 x 4000 and 5000 x 5000. From the figure we can see the number of targets found within area in different databases. The numbers are getting bigger as the area is getting bigger. It shows a exponential number as the database and scope is getting bigger.



(a)



(b)

**Fig. 7.** Number of Targets Found in (a) Square, (b) Circle

Figure 7b shows the number of targets found within radius 500, 1000, 1500, 2000 and 2500 metres. The number of targets found within a circle from various size of database is about same as the square. It shows an exponential number as well. However, if we notice carefully, the number of targets found in the circle is a bit smaller compared to the one in the square.

Figure 8 shows the differences of the total targets found between square and circle. The figure shows that all targets can be found in a square, but, there are only around 21 to 22 percents of targets cannot be found by using a circle. This percentage does not depend on the number of records in database. Therefore, a square will have more chance to find the target around 21 to 22 percents compared to a circle to find a number of rare places.

After we examined the efficiency of using square as a valid scope, the efficiency of our proposed algorithm will be discussed next. In our proposed algorithm, we will search the region based on the client's direction. We assume that distance of client request is always same as the direction of the client. For example, the user sends a request to find a place within 500 metres. Assume that user will be

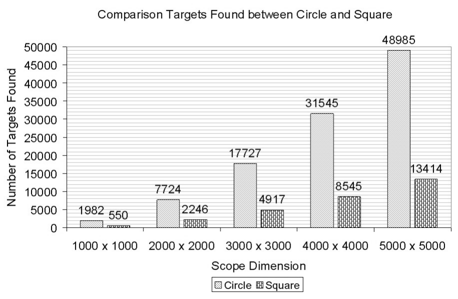


**Fig. 8.** Comparison Number of Targets Found in Circle and Square

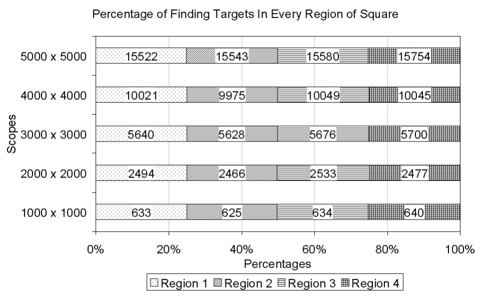


**Fig. 9.** Comparison Number of Targets Found In Each Region.

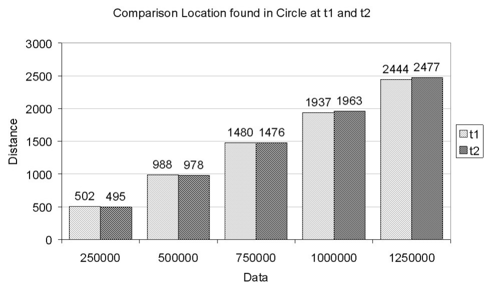


**Fig. 10.** Comparison Number of Targets Found In Circle at time $t_1$ and $t_2$.

at location (500,500) when he/she received the request. The server just needs to find the target in a specific region based on the direction to the new location.

Figure 9 shows the comparison of number of targets found in every region. If the server explore the whole regions, it will waste the resources of the server. This is because some targets found in the answer from the server will become invalid due to the current location of the server. We suggest that the server only chase after the region based on the direction of the client by assuming that client always go to the corner of scope square boundary. Searching in one or two regions is very efficient, because there will have no invalid answer when the client received the answer. It is about 25-50 percents faster than explorer the whole regions.

If we use a circle as the valid scope for the condition above, there will be some invalid targets that need to be evicted when the client reached the corner of scope square boundary. This is not efficient since the server needs to chase after the new targets in the new locations. A larger database will give more new data and evict more invalid number of data, shown in figure 10.

## 5 Conclusion and Future Work

In this paper we propose an approach to use a square to define a valid scope in order to find objects within the square for Location-Dependent. To simplify our explanation, we assume the geometric location is specified as two-dimensional coordinate. Dimension of a square is defined by the distance of a given query to the left, right, top and bottom from the client. In our algorithm, we divide the square into 4 regions in order to speed up the searching process. When a user move diagonally to the one of corner of square, the server needs to evaluate only first region. If the user move horizontally or vertically, the server only needs to find 2 vertical or horizontal regions respectively. We assumed that the user will be at the corner or the midlle of the square upon receiving the answer. We have evaluated the effective of using square compared to other shapes, such as circle, polygon and rectangles. Using a square is effective to find rare objects that can not be reached by a circle. Based on our evaluation, it give more chances to find the rare objects and invalidate less objects compared to a circle.

In the future, we would like to evaluate if the user has vary movement and speeds towards the two dimensional coordinate.

## References

1. Goodman, D.J.: Wireless Personal Communications Systems. Addison-Wesley Wireless Communications Series (1998)
2. Barbara, D., Imielinski, T.: Sleepers and workaholics: Caching strategies for mobile environments. In: Proceedings of the 1994 ACM SIGMOD international conference on Management of data, ACM Press, New York, USA (1994) 1–12 http://www.acm.org/pubs/.
3. Baihua, Z.e.a.: Data management in location-dependent information services. IEEE Pervasive Computing **1** (2002) 65–72
4. Zheng, B., Xu, J., Lee, D.: Cache invalidation and replacement strategies for location-dependent data in mobile environments. IEEE Transactions on Computers **51** (2002) 1141–1153
5. Cheverst, K., Davies, N., Mitchell, K., A., F.: Experiences of developing and deploying a context-aware tourist guide. Proceedings of the sixth annual International Conference on Mobile Computing and Networking (2000) 20–31

6. Dunham, M., Kumar, V.: Location dependent data and its management in mobile databases. Proceedings Ninth annual Int'l Workshop Database and Expert Systems Applications (1998) 414–419
7. Dunham, M., Kumar, V.: Using semantic caching to manage location dependent data in mobile computing. Proceedings of the sixth annual International Conference on Mobile Computing and Networking (2000) 210–221
8. Tang, X., Xu, J., Lee, D.: Performance analysis of location dependent cache invalidation schemes for mobile environments. IEEE transactions on Knowledge and Data Engineering **15** (2003) 474–488
9. Tang, X., Xu, J., Lee, D., Hu, Q.: Cache coherency in location-dependent information services for mobile environments. Proceedings First International Conference Mobile Data Access **1748** (1999) 182–193
10. Benetis, R., C.S, J., Karciauskas, G., Åaltenis, S.: Nearest neighbor and reverse nearest neighbor queries for moving objects. International Database Engineering and Applications Symposium (2002) 44–53
11. Zheng, B., Lee, D.: Processing location-dependent queries in a multi-cell wireless enviroment. Proceedings of the ACM international workshop on Data engineering for wireless and mobile access (2001) 54–65
12. Stanoi, I., Agrawal, D., Abbadi, A.: Reverse nearest neighbor queries for dynamic databases. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 2000) (2000) 44–53