Classification of Spatiotemporal Queries in LBS applications

Yong WANG[1] **Hiroyuki KAWANO**[2]

The server functions of Location-Based Services (LBS) are realized by integration of GPS and advanced database systems. In this paper, we focus on two characteristics of queries in spatiotemporal database systems based on the requirements of LBS, namely, coordinate-based queries and trajectory-based queries. We propose a general framework of LBS applications and apply it to a simulator of a modern navigation system, using the real navigation data collected from Internet Taxi in Nagoya.

1. Introduction

Location-Based Services (LBS) are growing up remarkably in the mobile service field. A common framework of a LBS application can be divided into four parts: advanced database server, LBS server, communication networks, and LBS clients [6]. A framework of a modern navigation system is proposed based on a spatial database and a LBS server called as *GeoMobility Server (GMS)* (Fig. 1)[6]. But the application interface of spatial database is insufficient to manage data of moving objects such as trajectories that almost all of data processing is performed in GMS.

In this paper, we propose an advanced framework for LBS using spatiotemporal database. In our framework, we do not only change the database type, but also categorize two sets of advanced queries in the spatiotemporal database such as *k shortest-path query with constrained ranges*, *time-parameterized range query*, *time-parameterized join query*, and others. Using such advanced queries, we can derive simpler result data from database server to GMS. We also evaluate the efficiency of our framework by applying it to a simulator of a modern navigation system, using the real navigation data collected from Internet Taxi in Nagoya[3].

[1]Student Member. Master's Program, Graduate School of Informatics, Kyoto University. wyong@sys.i.kyoto-u.ac.jp

[2]Regular Member. Faculty of Mathematical Sciences and Information Engineering, Nanzan University. kawano@it.nanzan-u.ac.jp

2. Spatiotemporal Data in LBS

Navigation system, a simple but typical application in LBS, guides and tracks moving objects from one place to another using positioning, communication, spatial data storage and data processing technologies. A framework of a modern navigation system is proposed as a general framework of LBS applications[6].

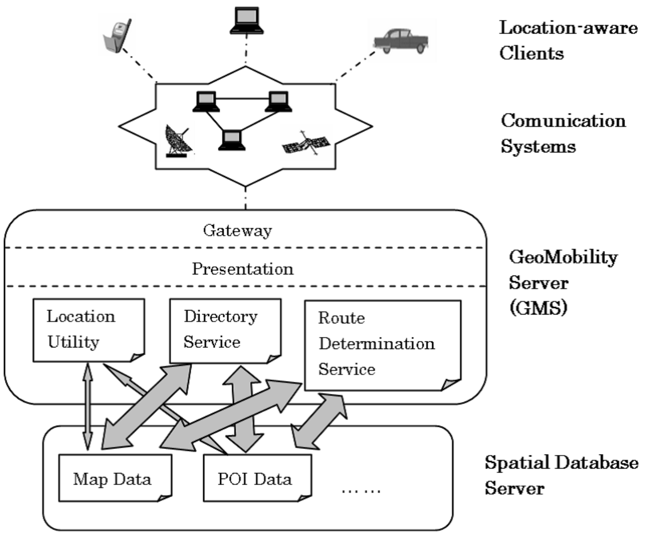

Fig. 1: A framework of a modern navigation system

In this framework, *GMS* is a core component which works as following. The *gateway* receives service requirements through the *communication systems* sent by the *location-aware clients*. The *location utility* analyzes the location information from each service requirement. The *directory service* completes spatial query functions using the map data fetched from the *spatial database server*. And the *route determination service* generates routes for the clients. At last a result formatted by the *presentation* component will be sent to the clients through the *communication systems* via the *gateway*.

On the other hand, the data used in LBS could be divided into two types: the map information and the data of moving objects. The map information is spatial data, including not only the spatial attributes but also some non-spatial attributes such as POI (Point of Interest). The data of moving objects, including both spatial information and temporal information, could be stored into spatiotemporal database. So that the spatiotemporal database is more appropriate for handling moving objects in LBS. Many valuable researches [1] [2] [4] [5] have been carried out on the issues of spatiotemporal databases, such as spatiotemporal database scheme, data representation and queries, which are the fundamentals of our research.

3. Advanced Queries in LBS

The framework proposed in [6] has several problems that may limit the functions of a modern navigation system or other LBS applications.

1. The interface of the framework is not appropriate because that almost all of the LBS processing is concentrated in GMS.
2. The spatial database server handles only the static data, not including dynamic data of moving objects which could be used widely in many fields.
3. The framework only operates the relationships among moving objects and static map locations. Spatiotemporal relationships among moving objects or trajectories are rarely considered, which are important for the expansion of service content in LBS.

In order to resolve above problems, we propose an improved LBS framework shown in Fig. 2. In our framework, the functions of some LBS services in GMS are shifted to the spatiotemporal database, realized by *coordinate-based query processing* and same kind of queries. On the other hand, we make different type queries, *trajectory-based queries* which deal with the spatial relationships among moving objects.

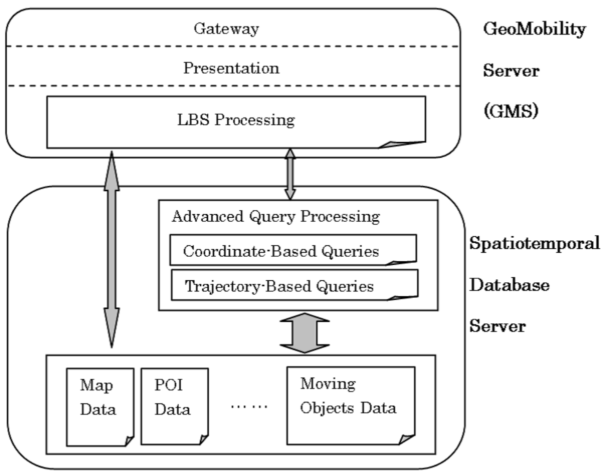

Fig. 2: An improved LBS framework

We propose our framework using three spatial data types: *point*, *line*, and *polygon*. A *point* object represents a 2-dimensional point in a map space. A *line* object, representing a segment of a directed road or a trajectory, is composed by start point, end point and several points at which the direction of the line is changed obviously or it is connected to other lines. A *polygon* object represents a spatial area closed by several lines.

In Table 1, we define several variables and functions to implement the advanced query processing used in LBS applications.

Table 1: Variables and functions

Variables	Functions
P: point	f_d: distance function
L: line	f_r: spatial relationship function
R: range	f_j: moving trajectory function
T_j: trajectory	f_t: moving time function
$O_{(n)}$: moving object (n: object Id)	f_p: moving object positioning function
T_s: start time	
T_w: time-window	
D: distance	
C_r: spatial relationship constraint	

- **Coordinate-Based Queries:**
  1. **Line Query**
     $LQ(L_1,\ \{L_{2(n)}\},\ C_r)$
     From a line set $\{L_{2(n)}\}$, *line query* retrieves lines which satisfy some spatial relationship C_r with a specified line L_1.
  2. **Range Query**
     $RQ(R_1,\ \{R_{2(n)}\},\ C_r)$
     From a range set $\{R_{2(n)}\}$, *range query* retrieves ranges which satisfy some spatial relationship C_r with a specified range R_1.
  3. ***k* Shortest-Path Query with Constrained Ranges**
     $kSPQ\_CR(P_1,\ P_2,\ \{R_{(n)}\},\ k)$
     k shortest-path query with Constrained Ranges retrieves the first k shortest paths from point P_1 to point P_2 which must touch or cross those constrained ranges $\{R_{(n)}\}$.
  4. ***k* Nearest-Neighbor Query**
     $kNNQ(P_1,\ \{P_{2(n)}\},\ k)$
     k nearest-neighbor query is used to find out the first k nearest points around a specified point P_1 from a point set $\{P_{2(n)}\}$.

- **Trajectory-Based Queries:**
  1. **Time-Parameterized Line Query**
     $TPLQ(L,\ \{O_{(n)}\},\ T_s,\ T_w,\ C_{rs},\ C_{rw})$
     Time-parameterized line query is used to search for objects from an object set $\{O_{(n)}\}$ which has some spatial relationships C_{rs} and C_{rw} with a specified line L in a time interval $[T_s,\ T_s+T_w]$.
  2. **Time-Parameterized Range Query**
     $TPRQ(R,\ \{O_{(n)}\},\ T_s,\ T_w,\ C_{rs},\ C_{rw})$
     Time-parameterized range query is used to search for objects from an object set $\{O_{(n)}\}$ which has some spatial relationships C_{rs} and C_{rw} with a specified range R in a time interval $[T_s,\ T_s+T_w]$.
  3. **Time-Parameterized Join Query**
     Time-parameterized join query fetches result

by operating spatial relationships among trajectories of moving objects.

(a) **Intercept Query**

$TPJQ\_IQ(O_1,\ O_2,\ T_s,\ T_w)$

Intercept query is to discover the smallest distance of two specified moving objects O_1 and O_2 between a time interval $[T_s,\ T_s+T_w]$ and the time when the smallest distance occurs.

(b) **ϵ-Distance Join Query**

$TPJQ\_DJQ(\{O_{1(m)}\},\ \{O_{2(n)}\},\ T_s,\ T_w, \epsilon)$

ϵ-distance join query fetches all pairs of moving objects from two object sets $\{O_{1(m)}\}$ and $\{O_{2(n)}\}$ whose distance will not over ϵ after a time interval T_w from start time T_s.

4. ***k* Fastest-Path Query with Constrained Ranges**

$kFPQ\_CR(P_1,\ P_2,\ \{R_{(n)}\},\ k)$

k Fastest-Path Query with Constrained Ranges retrieves the first k fastest paths from point P_1 to point P_2 which must touch or cross those constrained ranges $\{R_{(n)}\}$.

5. **Continuous Nearest-Neighbor Query**

$CNNQ(\{P_{1(m)}\},\ \ldots,\ \{P_{(k)(n)}\},\ T_j)$

Continuous nearest-neighbor query is to search for a set of different points from some point sets $\{P_{1(m)}\},\ \ldots,\ \{P_{(k)(n)}\}$ which have the smallest distance from a specified a trajectory T_j.

According to the interdependency of above queries and the distance functions, we can make hierarchy of these queries presented in Fig. 3. The left branch is based on *Range Query* using Euclidean distance function, while the right branch begins with *k Shortest-Path Query with Constrained Ranges* using path distance function. And *Continuous Nearest-Neighbor Query* utilizes both of the two distance functions.

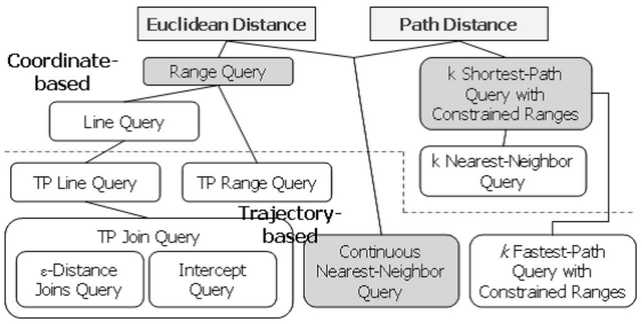

Fig. 3: Relationship among advanced queries in LBS

By implementing SQL functions to realize these queries, we can execute queries smoothly in LBS. Here are two examples in a modern navigation system.

Q1: For a beforehand reservation service of taxi, searching an optimal path from P_1 to P_2, and take up several persons on its way at place $R_1,\ \ldots,\ R_{(n)}$.

Answer: $SELECT\ kSPQ\_CR(P_1, P_2, \{R_{(n)}\}, 1)$

Q2: For a temporary reservation service of taxi, dispatch a taxi to a passenger at P as quickly as possible.

Answer:

1) P_1 = $SELECT\ kNNQ(P, \{P_{(n)}\}, 1)$; ($\{P_{(n)}\}$ is a set of taxi stops)

2) O_1 = $SELECT\ kNNQ(P, \{O_{(n)}\}, 1)$; ($\{O_{(n)}\}$ is a set of taxies which are out of service)

3) $SELECT\ kSPQ\_CR(O, P, \phi, 1)$; (O is taxi O_1 or a taxi dispatched from taxi stop P_1, which has the nearest distance to P)

The detailed discussion of other examples is described in [7].

4. Experimental Evaluation

We evaluate the effectiveness of our LBS framework combined with spatiotemporal queries. The test machine has a Pentium IV 3.00GHz CPU, 1024M memory, 250GB hard disk, FreeBSD 5.2.1 and PostgreSQL 7.4.5. (PostgreSQL is equipped with a powerful spatial extension: PostGIS.) The road map is Digital Road Map (DRM) version 3.1 provided by Japan Digital Road Map Association [3]. And the real navigation data are collected from Internet Taxi in Nagoya.

We design a car navigation system simulator, which can use real navigation data to evaluate the efficiency of our framework and advanced queries. The road data covers a rectangular area with corner points longitude and latitude (136.00, 34.67) and (138.00, 35.33), containing 227105 nodes and 374150 road segments. The average number of taxies is over 1500 drifting around 1000 in one day.

Our simulator has following two modes.

- **Monitor:** This mode only shows how the taxies moved at the specified date. This is the simplest mode that the simulator only fetches history data, and then shows it on the screen.

- **Selected taxi in LBS:** In this mode, the selected taxi receives LBS messages, while other taxies move according to the history data. A service request can occur in two ways. Firstly, if there is any request in the real history data, it will be requested at the same time and the same place automatically. Secondly, a service request can be added at the place selected by the mouse device. For each service request, the database server searches for the closest taxi from the taxies which are in LBS setting in startup parameters (Nearest-Neighbor

Query), and calculates the shortest path to its destination (k Shortest-Path Query).

Our experimental results show that it is very easy to develop LBS applications with these advanced queries implemented in spatiotemporal database server because the application interface is powerful that a service method could be constructed easily. Our framework also decreases the maintenance cost when service contents need to be modified frequently.

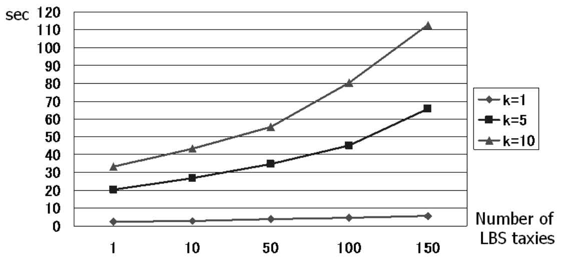

Fig. 4: Average time for executing "Nearest-Neighbor Query + k Shortest-Path Query"

But under our test environment listed above, the performance is still not good enough. As shown in Fig. 4, even in the pattern of "Nearest- Neighbor Query + Shortest-Path Query" (k = 1), the maximum number of the taxies to be computed simultaneously in LBS of our simulator is about 150. If more taxies are put into LBS, the process speed cannot satisfy the taxi requests gotten from the real navigation data. We consider the performance is greatly influenced by many factors, such as architecture of hardware, indexing, computing algorithms (exhausting or approximate) and others.

5. Conclusion and Future Work

With the expansion of LBS contents, spatiotemporal database has many advantages than spatial database especially in dealing with spatial relationships among moving objects. We proposed two categories of advanced spatiotemporal queries for LBS based on their spatial and spatiotemporal characteristics. We also proposed an improved framework for LBS applications. We presented that more complex LBS processing could be realized by combining different query categories.

In this paper, we only evaluate the performance of spatiotemporal queries. More accurate evaluation and performance analysis should be done firstly in our future work. We will try to propose other techniques such as new indexing methods and approximate algorithms to improve the performance of computing. We will also expand our framework to perform data mining on moving objects for possible service contents of LBS in the near future such as optimum positioning problems.

[Acknowledgment]

We are particularly grateful to Dr. Eiji Hato (Ehime University) for discussion and offering data used in our experiment. This project is partially supported by the grant-in-aid for scientific research (16016248, 13680482) in Ministry of Education, Science, Sports and Culture and Japan Society for the Promotion of Science.

[References]

[1] R. H. Güting, V. T. de Almeida, and Z. Ding, "Modeling and querying moving objects in networks," *Informatik-Report*, vol. 308, Apr. 2004.

[2] S. Gupta, S. Kopparty, and C. Ravishankar, "Roads, codes, and spatiotemporal queries," in *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Paris, France, 2004, pp. 115–224.

[3] *Standards of National Digital Road Map Database*, 3rd ed., Japan Digital Road Map Association, Mar. 1999, (In Japanese).

[4] H. Kawano and E. Hato, "Architecture of spatial data warehouse for traffic management," in *Proceedings of the Second International Workshop on Active Mining*, 2003, pp. 183–192.

[5] H. Mokhtar, J. Su, and O. H. Ibarra, "On moving object queries," in *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Madison, Wisconsin, USA, 2002, pp. 188–198.

[6] J. Schiller and A. Voisard, *Location-Based Services*. Morgan Kaufmann Publishers, 2004.

[7] Y. Wang and H. Kawano, "Advanced query processings for location-based services," in *Proceedings of DBWeb2004*, 2004, pp. 81–88.

Yong Wang
Student in Master's Program, Graduate School of Informatics, Kyoto University. Graduated from Donghua University in China, received the Bachelor Degree in Automatic Control in 1994. Student Member of DBSJ.

Hiroyuki KAWANO
2004 Professor, Faculty of Mathematical Sciences and Information Engineering, Nanzan University. 1997 Associate Professor, Graduate School of Informatics, Kyoto University. 1990 Graduated Ph.D program from Graduate School of Engineering, Kyoto University. Doctor of Engineering. Regular Member of IEICE, IPSJ, JSAI, etc.