Towards a Taxonomy of Location Based Services

Kostas Gratsias[1,2], Elias Frentzos[1], Vasilis Delis[2], and Yannis Theodoridis[1,2]

[1] Department of Informatics, University of Piraeus,
80 Karaoli-Dimitriou St, GR-18534 Piraeus, Greece
{gratsias, efrentzo, ytheod}@unipi.gr
http://isl.cs.unipi.gr/db
[2] Research and Academic Computer Technology Institute,
11 Akteou St & Poulopoulou St, GR-11851 Athens, Greece
{gratsias, delis, ytheod}@cti.gr

Abstract. Location-based services (LBS) constitute an emerging application domain involving spatio-temporal databases. In this paper, i) we propose a classification of LBS, depending on whether the user (query object) and the data objects are moving or not and ii) we provide algorithms for the efficient support of real applications, for every class. We also survey recent work in query processing for the proposed LBS algorithms and sketch open issues for future research.

Keywords: location based services, moving objects, spatiotemporal databases.

1 Introduction

The rapid growth of mobile devices, such as mobile phones and Personal Digital Assistants (PDAs) has contributed to the development of an emerging class of e-services, the so-called Location-Based Services (LBS). An LBS provides information relevant to the spatial location of a receiver. LBS constitute an innovative technological field which influences the way that people organize their activities, promising great business opportunities for telecommunications, advertising, tourism, etc. [10].

From a database perspective, LBS have combined most of the existing research results in the spatial and spatiotemporal database domain regarding indexing [4, 14] and query processing [1, 5, 13, 17, 18]. They have also indicated some open research issues such as the processing of forecasting queries (determining future positions of moving points), the support for continuous location change in query processing techniques [8], knowledge discovery from data collected via LBS, etc.

In this paper, we propose a novel classification of LBS based on whether the involved objects are stationary or mobile (Section 2) and provide indicative examples of applications as well as sketch algorithms for their efficient support (Section 3), for each class. In Section 4 we review the related literature and briefly discuss implementation issues for such query processing. Relevant open problems are also mentioned. Finally, Section 5 concludes the paper giving hints for future work.

K.-J. Li and C. Vangenot (Eds.): W2GIS 2005, LNCS 3833, pp. 19 – 30, 2005.

2 LBS Taxonomy

From a query processing point of view, the data structures and algorithms required to support queries involved in LBS depend on the mobility of the user. For a stationary user, the execution of a nearest neighbor query over a set of spatial objects (without taking into consideration any network constraints) could be sufficiently supported by a simple R-tree [4] and a classic nearest neighbor algorithm [13, 5]. However, such algorithms are not applicable in the case of a mobile user who wishes to be informed about the nearest point of interest at any time of his/her movement. Continuous nearest neighbor searching techniques that can efficiently support this type of service are presented in [18].

Likewise, let us consider services where the user asks for information about continuously moving objects; finding a cab is a good example. The efficient support of such queries entails indexing the moving objects (including future positions) by an appropriate structure (for example the TPR-tree [14]) and the use of a suitable nearest neighbor algorithm, such as the one presented in [1].

So, one could extend these thoughts and fully classify LBS according to the mobility of the user (query object) and the data objects. Four classes are identified which are presented in Table 1 along with indicative application examples. These examples are further described in the following section.

Table 1. LBS Classification

Data Object \ Query Object	Stationary (S)	Mobile (M)
Stationary (S)	*What-is-around* *Routing* *Find-the-nearest*	*Guide-me*
Mobile (M)	*Find-me*	*Get-together*

The novelty of the proposed classification stands on the fact that database query processing is the driving wheel behind our approach. As we will further discuss in subsequent sections, whether involving mobility or not in a service offered by an LBS system adds new and very interesting processing issues. On the other hand, other classification approaches that can be found in the literature either target on kinds of LBS applications [9] or present core services with respect to interoperability [10].

3 Fundamental LBS and Algorithms

In this section we describe six representative services that cover all S and M combinations that appear in Table 1: the former three address the (naïve) S-S category while the latter three uncover interesting issues of the (more advanced) S-M, M-S and M-M categories. Due to space constraints, we provide the corresponding high-level algorithmic descriptions of the latter three services only.

Some definitions: Let $G(V, E)$ be the directed graph that represents the underlying road network on which objects are moving. So, the vertices (nodes) V of G correspond to junctions while its edges E represent road segments. We can apply movement constraints for some (blocked) edges. Let S denote this subset of the edges. Each edge is associated with two weights (distance metrics), the length of the corresponding road segment and the average time required to travel through that segment, respectively. Finally let L denote all the landmark (point of interest) types and Q denote the set of all objects whose type belongs to L (i.e. pairs of the form <*object_id*, *object_type*>).

3.1 Both (Query and Data) Objects Are Stationary

The first three services belong to the S-S category, where both query and data objects are stationary. Although they are algorithmically simple, they constitute an LBS core upon which most of the rest are based.

What-is-around
The simplest service is the one that retrieves and displays the location of every landmark being in a circular[1] area (P, d), where P is the location of the user (or simply a point of interest – POI) and d is a selected distance. This service assumes a storage containing all the existing types of landmarks (POIs). The input of the corresponding algorithm for "*what-is-around*" consists of the point P, the radius d and the set LT (indicating the preference on object types, e.g. 'gas-station' and 'ATM') which is a subset of L. The algorithm returns the set $Q' \subseteq Q$ of all objects inside the circular area (P, d) whose type belongs in LT.

Routing
This service provides the optimal route between a departure and a destination point, P and Q, respectively, using the minimal network distance or minimal traveled time as the optimization criterion. The input of this service includes the graph G, the set of constraints S, the coordinates[2] of P and Q and a flag f that specifies the optimization criterion. The algorithm is based on the *Routing* function, which will be described in Section 4.

Find-the-Nearest
This service retrieves the k nearest landmarks (POIs). For example, "*find the two restaurants that are closest to my current location*" or "*find the nearest café to the railway station*". The underlying algorithm takes as input the graph G, the subset of landmarks of interest LT and a query point P (for example, calling user's current

[1] Alternative to circle, a rectangular or a more complex shape could be used to be the 'window' of the query.

[2] Finding the shortest path between two arbitrary points that lie on two edges, rather than between two nodes of a graph, is a slightly more involved operation. It depends on whether the particular road segments are bidirectional and the data structure that implements the graph. In the worst case it requires the calculation of four shortest paths, for the four pair combinations among the nodes of the corresponding edges. The distance between the point lying on the edge and the end-nodes must also be computed.

location). Using the *Net_k_Nearest_Neighbor* routine (described in Section 4), the system returns the set of points $Q' \subseteq Q$, which are the k nearest to P members of Q and their type belongs to LT. Once more, any of the two distance metrics can be applied.

3.2 Involving Mobility

The services presented in this section belong to the S-M, M-S or M-M category, where at least one part (query and/or data objects) is mobile. In contrast to the previously discussed services, these ones involve advanced graph and trajectory management routines, a fact that makes their development really attractive from the perspective of moving object database management.

Find-me

The provision of the *Find-me* type of services is fundamental for a commercial LBS platform. The scenario is the following (Fig.1): after receiving an emergency call, the system routes the k (in Fig.1, $k = 3$) nearest mobile first-aid units (U_2, U_3, U_4) to the location of the calling user (point P), who is periodically informed about the condition of his request (for example, he/she receives messages of the form "*The ambulance is 1 Km away*").

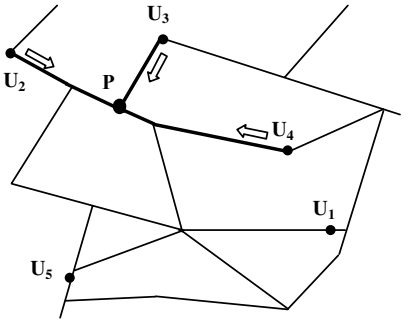

Fig. 1. Find-me example

In this service the query object (user) remains stationary, while the data objects (first-aid units) are mobile and their locations are automatically being recorded as they approach the calling user.

Obviously, this service can be implemented as a combination of *Find_the_nearest* and *Routing* presented in Section 3.1. The input of the proposed algorithm, which is illustrated in Fig.2, includes:

- the graph G along with the set of movement constraints S,
- the location P of the calling user,
- the set of ids of the mobile units $U = \{U_1, \ldots, U_N\}$,
- a distance threshold $\delta \geq 0$, which defines the minimum distance between the user and a mobile unit below which the unit is supposed to have reached the user,

- a time delay *dt* between subsequent notifications of a user about the state of his request, and finally,
- the parameter *k*.

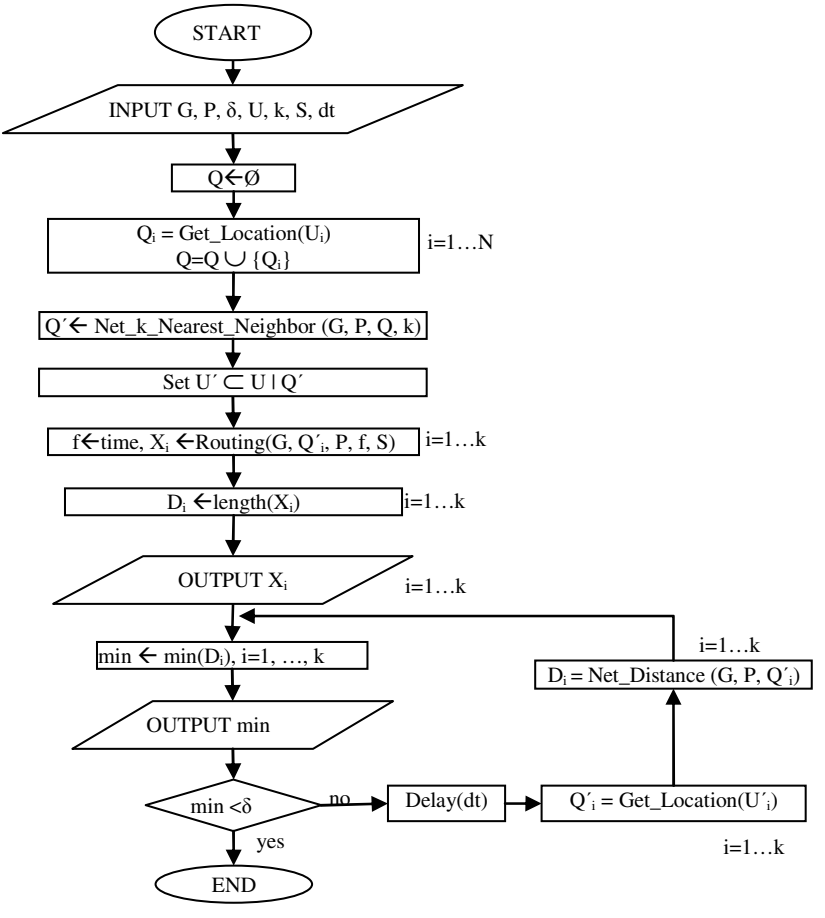

Fig. 2. Find-me algorithm

Guide-me

An extension of the (static) *Routing* described above is the so-called *Guide-me* service (an example of which is illustrated in Fig. 3): Likewise, the system determines the best route between a departure and a destination point (*P* and *Q* respectively). Simultaneously, the system keeps track of the user's movement towards the destination point and allows him/her to deviate from the 'optimal' route, as long as the user's location does not fall out of a predefined safe area (buffer) built around this route. The user is notified of his/her deviation every time he/she crosses out of the buffer's border and he/she is given the option of re-routing from that current location (point *P´*).

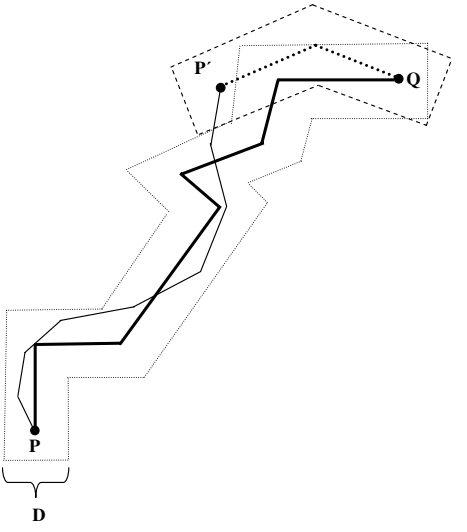

Fig. 3. Guide-me example

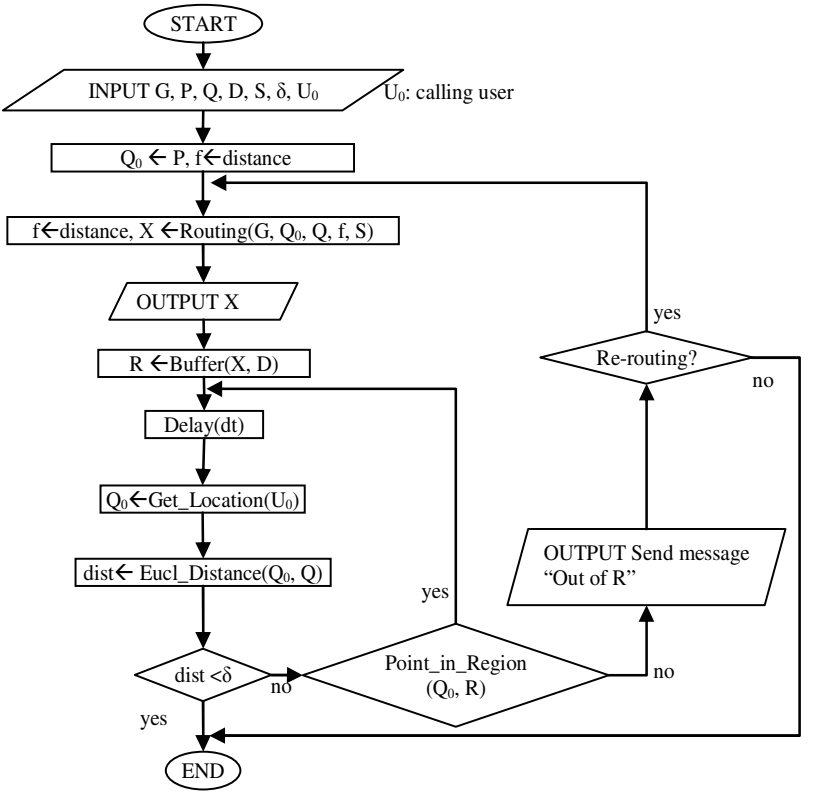

Fig. 4. Guide-me algorithm

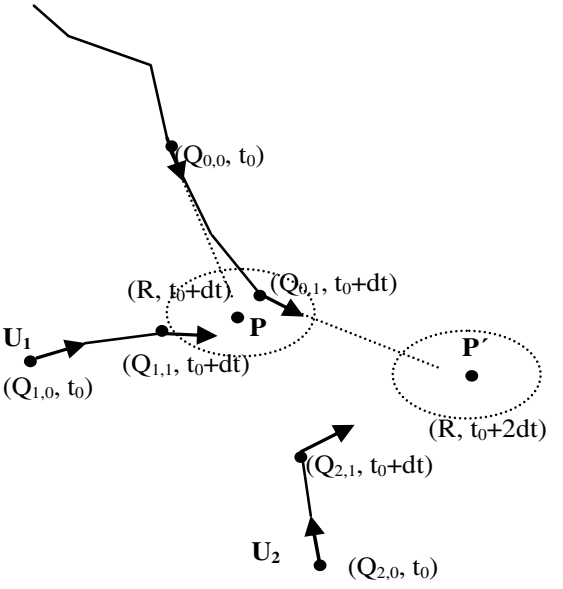

Fig. 5. Get-together example

The input of *Guide-me* algorithm is the following:

- the graph G along with the set of movement constraints S,
- the departure and destination points, respectively P and Q
- the id of the calling user, U_0
- the distance D, which defines the buffer width
- the distance threshold δ (as in the previous algorithm)

As illustrated in Fig.4, *Guide-me* computes the (dynamically updated) path between the user's current location and Q.

Get-together

With this service a moving user U_0 invites other users U_i, $i = 1, 2, \ldots, N$ (let us assume, members of a community), also moving in the same area, to converge at a meeting point not-known in advance. This point is periodically (every dt seconds) calculated by the system based on the future projection of the calling user's trajectory. This query falls in the M-M category, which is algorithmically the most complicated one.[3]

As an example of *Get-together*, consider Fig.5, where the calling user U_0 is at (the spatiotemporal) location $(Q_{0,0}, t_0)$ and moves towards the direction shown. The invited users U_1 and U_2 are located at $(Q_{1,0}, t_0)$ and $(Q_{2,0}, t_0)$ respectively. The system predicts that by the time t_0+dt U_0 will most probably be at location P and therefore routes U_1 and U_2 towards P. Eventually, at t_0+dt the (recorded) location of U_0 is $(Q_{0,1}, t_0+dt)$. Likewise, the system projects the trajectory of U_0 to the future time t_0+2dt, predicting that at that time U_0 will be at P', so it routes the rest of the users accordingly. The

[3] Ericsson offers a simplified and static version of this service, called '*Seek-your-friends*', where all U_0 and U_i are stationary [15].

algorithm terminates when the distance between U_0 and each U_i becomes smaller than a predefined threshold δ; then, the system informs all that they have practically converged to the same location.

The input of the *Get-together* algorithm, which is illustrated in Fig.6, consists of:

- the graph G along with the set of movement constraints S and the distance/time flag f,
- the distance D, which defines a proximity are of users to be invited,
- the set $U = \{U_1, \ldots, U_N\}$ of ids of the called users,
- a distance threshold δ which defines the minimum distance (i.e. the convergence criterion) between U_0 and each U_i, and
- a time delay dt

The algorithm returns a set of (dynamically updated) paths between each U_i and the periodically predicted location of U_0.

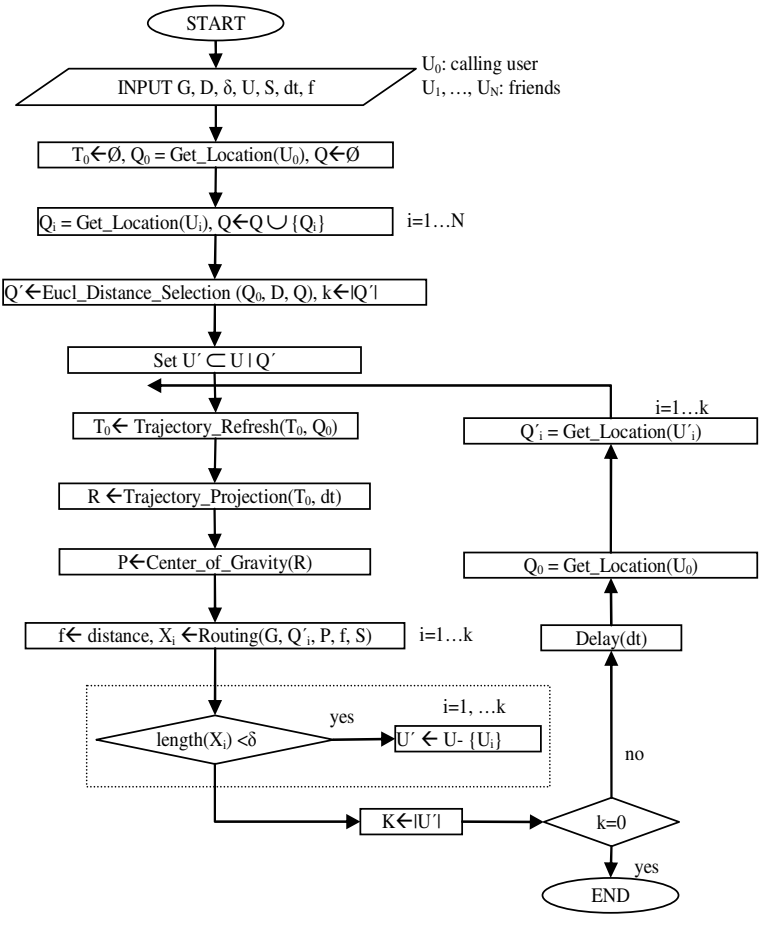

Fig. 6. Get-together algorithm

4 Query Processing Issues

In this section we summarize the routines involved in the implementation of the algorithms of Section 3. These routines can be classified into four main types: computational geometry operations, graph operations, trajectory operations and adaptive LBS operations; they are summarized in Table 2.

Computational geometry operations provide solutions to simple problems and are usually included in the relevant functionality of GIS or DBMS providing spatial data handling. More details on computational geometry algorithms can be found in [12].

Graph operations can be further classified into i) routing queries and ii) nearest neighbor (NN) queries. Calculating shortest paths in graphs is a well studied problem (see for example the Dijkstra [2] and the Bellman Ford [6] algorithms). The *k*-NN problem has been in the core of spatial and spatiotemporal database research during the last decade. Initially, the proposed *k*-NN algorithms focused on Euclidean space [13, 5]. Papadias et al. [11] presented the first comprehensive approach for query processing in spatial network databases in which data and query points are stationary and proposed several novel algorithms for *k*-NN, range, closest pairs and distance join queries. The proposed in [11] *k*-NN algorithm can be utilized for the implementation of the *Net_k_Nearest_Neighbor* operation used in the *Find-the-nearest* and *Find-me* services. A *k*-NN algorithm for the case of moving query and data points, the movement of which is constrained on a road network, is presented in [7].

Among the trajectory operations, *Trajectory_Projection* appears to be the most interesting to be further discussed. This routine estimates the future location of a moving object by projecting its trajectory to a future time t_0+dt. Such a computation requires some extra knowledge such as the velocity and the direction of each object. Although several methods have been proposed for processing and indexing the current and future location of objects moving without network constraints [14, 19], the problem of predicting the position of an object moving in a network (Fig.7) still remains open. The *Trajectory_Projection* routine is a step towards this direction, as it calculates a set of possible future locations (points Q_1, Q_2). Furthermore, instead of considering only object's current position (point *P*) (as the proposed in the literature methods do), an efficient *Trajectory_Projection* makes use of a trace (of dynamic length) of the trajectory for the estimation of the object's future location (thus the point Q_3 is excluded due to the direction of the trajectory of the moving object).

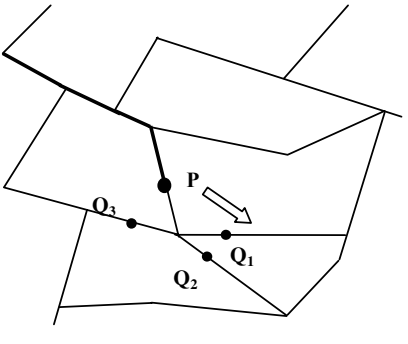

Fig. 7. Example of Trajectory Projection

Table 2. LBS Routines

Notations: *Qs*: Set of Points, *D*: Distance, R: Polygonal Region, *G*: Network Graph, *P*,*Q*: Points, *f*: Minimization criterion, *S*: Set of Network Constraints, *X*: Path	
Computational geometry operations	
Eucl_Distance_Selection (Qs, P, D)	Returns the set of points *Qs´*, *Qs´*⊆*Qs*, located inside the circular area (*P*, *D*)
Length(X)	Returns the length of a path *X*
Eucl_Distance(P, Q)	Calculates the Euclidean distance *D* between the points *P* and *Q*
Point_in_Region(R, P)	Checks whether a point *P* is located inside a region *R*
Center_of_Gravity(Qs)	Calculates the centre of gravity of a set of points *Qs*
Buffer(X, D)	Builds a buffer of width *D* around a path *X*
Graph operations	
Routing(G, P, Q, f, S)	Finds the best route on a graph *G* between a departure point *P* and a destination point *Q*, considering the set of constraints *S* and the minimization criterion *f* (minimal network distance or minimal traveled time)
Net_Distance(G, P, Q)	Calculates the network distance on a graph *G* between the points *P* and *Q*
Traveled_Time(G, X)	Calculates the time required to move through a path *X* of graph *G*
Net_k_Nearest_Neighbor (G, Q, P, k)	*R*eturns the *k* points of the set *Q* nearest (with respect to network distance on *G*) to a point *P*.
Trajectory operations	
Get_Trajectory(U)	Returns the trajectory of user *U* stored in a data file
Trajectory_Refresh(T, Q)	Adds (spatiotemporal) point *Q* to the existing trajectory *T* of an object
Trajectory_Projection (T, dt)	Projects the trajectory *T* of a moving object to the future time *now*+*dt*. In particular, considering the existing trajectory of the object as well as the direction in which it moves, this operation computes the set *R* of all the potential point locations of that object at *now*+*dt*.
Adaptive LBS operations	
Get_Location(U)	Returns the coordinates (*x*, *y*) of the current location of object *U* (assumes some positioning technology)

Finally, regarding the last type of LBS operations, it involves *Get_Location*, which returns the current location of a moving object. This can be easily developed with the utilization of current or emerging positioning technologies, such as TDOA (Time Difference Of Arrival) and GPS (Global Positioning System) [15, 3].

5 Conclusions

In this paper we present the first, to the best of our knowledge, attempt of classifying LBS from a stationary vs. mobile object perspective. For each LBS class, we describe representative examples of potentially useful services, focusing on the query processing requirements that arise.

It has become evident that several of the involved operations can be supported by existing commercial DBMS and GIS while some others require special data structures and query processing algorithms. Another important conclusion is that, as indicated in [16], the efficient handling of a new 'first-class' datatype, namely, the *trajectory*, poses new functional requirements for GIS and DBMS in terms of storage, indexing and query processing. It seems that commercial systems have still a long way to go.

Future continuation of this work focuses on open research issues, such as future trajectory projections.

Acknowledgements

Research partially supported by FP6/IST Programme of the European Union under the GeoPKDD project (2005-08) and EPAN Programme of the General Secretariat for Research and Technology of Greece under the NGLBS project (2004-06).

References

1. Benetis, R., Jensen, C., Karciauskas, G., and Saltenis, S., Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. *Proceedings of IDEAS Symposium*, 2002.
2. Dijkstra, E. W., A note on two problems in connexion with graphs, *Numerische Mathematik*, vol 1, pp. 269-271, 1959.
3. Davies, N., et al., Using and Determining Location in a Context-Sensitive Tour Guide, IEEE Computer, vol. 34(8), pp. 35-41, 2001.
4. Guttman, A.: R-Trees: a dynamic index structure for spatial searching. *Proceedings of ACM SIGMOD Conference*, 1984.
5. Hjaltason, G., and Samet, H., Distance Browsing in Spatial Databases, *ACM Transactions in Database Systems*, vol. 24(2), pp. 265-318, 1999.
6. Johnson, D., B., Efficient algorithms for shortest paths in sparse networks, *Journal of the ACM*, vol. 24(1), pp. 1-13, 1977.
7. Jensen, C., Kolar, J., Pedersen, T., and Timko, I., Nearest Neighbor Queries in Road Networks, *Proceedings of ACM-GIS Symposium,* 2003.
8. Jensen, C., Christensen, A., Pedersen, T., Pfoser, D., Saltenis, S. and Tryfona, N., Location-Based Services: A Database Perspective. *Proceedings of Scandinavian GIS*, 2001.

9. Lopez, X., The Future of GIS: Real-time, Mission Critical, Location Services. *Proceedings of Cambridge Conference*, 2003.
10. Open GIS Consortium, OpenGIS® Location Services (OpenLS): Core Services. Available at http://www.opengis.org.
11. Papadias, D., Zhang, J., Mamoulis, N., and Tao, Y., Query Processing in Spatial Network Databases, *Proceedings of VLDB Conference*, 2003.
12. Preparata, F., Shamos, M. *Computational Geometry, An Introduction*, Springer-Verlag, New York, 1985.
13. Roussopoulos, N., Kelley, S., and Vincent, F., Nearest Neighbor Queries, *Proceedings of ACM SIGMOD Conference,* 1995.
14. Saltenis, S., Jensen, C. S., Leutenegger, S., and Lopez, M., Indexing the Positions of Continuously Moving Objects, *Proceedings of ACM SIGMOD Conference*, 2000.
15. Swedberg, G., Ericsson's Mobile Location Solution, *Ericsson Review*, vol. 4, 1999.
16. Theodoridis, Y., Ten Benchmark Queries for Location Based Services, *The Computer Journal*, vol. 46(6), pp.713-725, 2003.
17. Tayeb, J., Ulusoy, O., and Wolfson, O., A Quadtree-Based Dynamic Attribute Indexing Method, *The Computer Journal*, vol. 41(3), pp. 185-200, 1998.
18. Tao, Y., Papadias, D., and Shen, Q., Continuous Nearest Neighbor Search, *Proceedings of VLDB Conference*, 2002.
19. Tao Y., Papadias D., Sun J., The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries, *Proceedings of VLDB Conference*, 2003.