Towards the Next Generation of Location-based Services

Elias Frentzos, Kostas Gratsias and Yannis Theodoridis[†]

Department of Informatics, University of Piraeus,
80 Karaoli-Dimitriou St, GR-18534 Piraeus, Greece
{efrentzo, gratsias, ytheod}@unipi.gr
and
Research Academic Computer Technology Institute,
Patras University Campus, GR-26500 Rio, Greece

Abstract. Location-based services (LBS) constitute an emerging application domain rapidly introduced in modern life habits. However, given that LBS already count a few years of commercial life, the services provided are rather naïve, not exploiting the current software capabilities and the recent research advances in the fields of spatial and spatio-temporal data management. The goal of this paper is to fill this gap by first presenting the next generation of location-based services and then, demonstrating their implementation which takes advantage of both modern commercial software and state-of-the-art spatial network and spatio-temporal databases techniques. Novel techniques are also proposed in fields non-thoroughly addressed by the research community, such as the prediction of future position of objects moving on a road network.

Keywords: Location Based Services, Spatial Network Databases, Spatio-temporal databases

1 Introduction

The rapid introduction of location-aware mobile devices in modern life, such as GPS-enabled mobile phones and Personal Digital Assistants (PDAs), has triggered the development of an emerging class of e-services, the so-called Location-Based Services (LBS), which provide information relevant to the location of a receiver [15]. LBS are rapidly introduced in modern life habits, influencing the way that people organize their activities, promising great business opportunities for telecommunications, advertising, tourism, etc. [11], also setting the research agenda in several technological fields including spatial and spatio-temporal data management.

From the database management perspective, efficient LBS support request the integration of several research advances in indexing [13], [17] and query processing techniques [5], [12], [14], [16]. The development of such services has also indicated several open research issues such as the processing of forecasting queries (i.e., determining future positions of moving points [4]), the support of continuous location

† Contact author's address: 80 Karaoli-Dimitriou St., GR-18534 Piraeus, Greece. Tel: +30-2104142449, Fax: +30-2104142264.

change in query processing techniques [3], knowledge discovery from data collected via LBS [10], etc.

On the other hand, although LBS already count some years of commercial life (i-area by NTT DoCoMo was launched in 2001), the services currently provided are rather naïve, not exploiting the current software capabilities and the recent advances in the research fields of spatial and spatio-temporal databases. The goal of this paper is to present the next generation of LBS and, then, demonstrate their implementation taking advantage of modern GIS and DBMS software as well as recent advances in spatial network and spatio-temporal databases.

Investing on the taxonomy of LBS proposed in [1], we present a set of advanced LBS and then sketch up the respective algorithms along with a description of their implementation details. The taxonomy classes provided in [1] are based on the discrimination of mobile vs. stationary reference (query) object, on the one hand, and data objects (e.g., landmarks), on the other hand, resulting in four classes of services (i.e., both being static, *S-S*, one being static and one being mobile, *S-M* and *M-S*, both being mobile, *M-M*), summarized in Table 1.

Table 1. LBS Classification and example services, according to [1]

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

The driving force behind the LBS classification presented in [1] is about query processing issues: the query processing techniques behind a service involving mobile objects, clearly fall into the domain of Moving Object Databases (MOD), while processing of services concerning stationary objects is likely to be an issue related to the area of Spatial (or Spatial Network) Databases (SDB or SNDB). The services presented in this paper cover the *S-S*, *M-S* and *M-M* classes introduced in [1], employing respective query processing techniques, while, their majority, to the best of our knowledge, is not currently supported by commercial LBS providers.

Outlining the rest of the paper, Section 2 presents the framework on top of which LBS are developed as well as a set of fundamental LBS of the *S-S* LBS class. Section 3 is the core of the paper where a set of novel services is presented, constituting our proposal regarding the next generation of LBS, as well as the algorithmic and implementation issues raised during their development. Section 4 provides technical details regarding the development platforms. Finally, Section 5 summarizes our work providing the conclusions and some interesting directions for future work.

2 Background

In this section, we first introduce the framework used in the rest of the paper, and then describe a set of services already provided by current LBS solutions. We include this set in our discussion since they are fundamental and, thus, used as a basis for the (more advanced) novel LBS set that will follow in Section 3.

Table 2. Table of notations

Symbol	Description
$V = \{V_i\}$	the set of vertices corresponding to road network junctions on a road network
$E = \{E_{i,j}\}$	the set of edges connecting vertices V_i and V_j, corresponding to road segments on a road network
$G(V, E)$	the directed graph that represents the underlying road network on which objects are moving
$L = \{L_i\}$	the set of points of interest (POIs) or Landmarks
T_i , $T_{i,j}$, $T^{\dagger}_{i,j}$	the trajectory of a mobile user, its actual (sampled) and its predicted spatio-temporal position (i.e., time-stamped spatial point) at timestamp t_j
$T^x_{i,j}, T^y_{i,j}, T^t_{i,j}$	the x-, y-, t- components of the spatio-temporal position $T_{i,j}$. Also note that $T^t_{i,j} \equiv t_j$
$\overrightarrow{V_{i,j}}, \overrightarrow{V^x_{i,j}}, \overrightarrow{V^y_{i,j}}$	the (estimated) velocity vector of object T_i at timestamp t_j and its projection along the x and y axes
$T = \{T_i\}$	the set of trajectories of mobile users
$D_{Eucl}(P, Q)$	the Euclidean distance between the two dimensional points P and Q
$D_{Net}(P, Q)$	the network distance on the graph G between the two dimensional points P and Q
Buffer(X, D)	A method that builds a buffer of width D around a path X
Route(P, Q)	A method that retrieves a set of bi-connected line segments $\{E_i\}$ of the network graph forming a single path between points P and Q; usually, the result of a routing operation

2.1 The Framework

Before proceeding into describing the LBS, we set the framework on which the services will be based. Specifically, this framework contains:

– A directed graph $G(V, E)$ that represents the underlying road network on which objects are moving. The set of vertices (nodes) $V = \{V_i\}$ of G correspond to road junctions while its edges $E=\{E_{i,j}\}$ (connecting nodes V_i and $V_j\}$ represent road segments. Each edge $E_{i,j}$ 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. Movement constraints (one-ways etc.) can be also applied in the graph $G(V, E)$ by appropriately setting the weights of each edge.

– The set $L = \{L_i\}$ containing the points of interest (POIs) or landmarks. POIs can be also categorized into classes (e.g., restaurants, gas stations, ATMs etc); in fact, in our implementation POIs are divided in such categories. We choose however, to restrict our discussion in the case of one class for sake of clarity in presentation.

– The set $T = \{T_i\}$ of trajectories of mobile users. Each trajectory is represented as a set of time-stamped sampled locations $<x, y, t>$, applying linear interpolation in-between them.

The above definitions along with the notation used in the rest of the paper are summarized in Table 2.

The services presented in this paper are part of an LBS platform following the system architecture illustrated in Fig.1. Among the components contained in the system architecture, we focus on the LBS suite, which encloses the logic and a number of advanced algorithms behind the developed services. Regarding the other components, the architecture includes an extensible Database Management System (*DBMS*) with a *Spatial Extension* communicating with the LBS suite via OLE DB protocol; the *Routing Server* maintains the network graph G and provides routing between two points on the network, while its communication with the LBS suite is achieved via the Routing Client middleware. Exploiting the simple functionality provided by these components, we expand it towards many directions. Among others, the developed software supports nearest neighbor search using network (rather than Euclidean) distance, in-route nearest neighbour search, and predictive location queries which are supported by novel techniques.

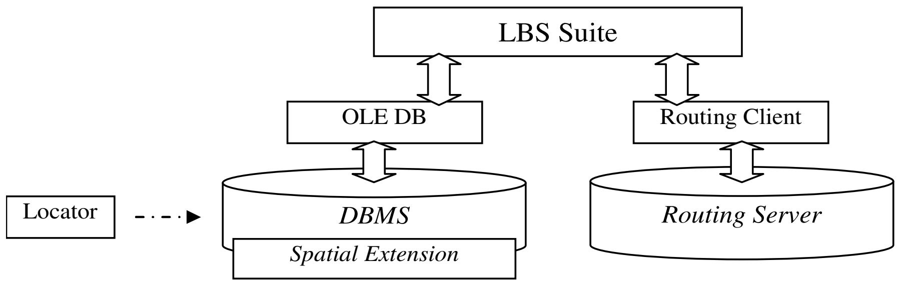

Fig. 1. LBS Suite Architecture

The database maintained in the DBMS of Fig.1 contains, among others, a set of tables fundamental for the advanced services that will be proposed later. Specifically, *Landmarks*(*object_id, object_name, object_geometry*), contains the set L of POIs while *Trajectories*(*User_id, Trajectory_id, sampled_position_geometry, timestamp*) contains the set T of trajectories of the tracked LBS users. For performance issues, we build a view *Current_Positions*(*User_id, last_position_geometry*) on top of *Trajectories*, which contains the current positions of all tracked LBS users. The table *Trajectories* (and its view *Current_Positions*) are assumed to be periodically updated from an external Locator component. We also point out that *geometry* type columns in tables assume the Spatial Extension component in the involved DBMS illustrated in Fig.1.

2.2 A Set of Fundamental LBS

The basic set of LBS usually operated by current LBS solutions consists of three fundamental services, namely *What-is-around*, *Routing*, and *Find-the-Nearest*, which evidently fall into the *S-S* class of the taxonomy presented in [1]. All three services (and the respective algorithms) assume the presence of a graph *G* and/or a set of POIs *L*. Moreover, all services involving graph operations (e.g., routing between two points), can be evaluated with any of the two optimization criteria presented, either traveled distance or traveling time, choosing between the two weights set for each edge of the graph (length and time, respectively); in the following sections, for sake of simplicity, we restrict our discussion to the distance (rather than time) optimization. The following paragraphs describe the functionality of each of the above fundamental LBS:

- *What-is-around*: The simplest service is the one that retrieves and displays the location of every POI being located inside a rectangular area (Q, d), where Q is the location of the user (or simply a user-defined point) and d is a selected distance (i.e., the half-side of the query rectangle). The input of the corresponding algorithm consists of the point Q and the distance d, while it returns the set $L' \subseteq L$ containing all POIs inside the rectangular area (Q, d). This LBS is fundamental for the user in order to know where he/she is located and what he/she can find nearby, while in terms of spatial database operations it involves a simple *spatial range query*.
- *Routing*: This service provides the optimal route between a departure and a destination point, P and Q, respectively. The input of the respective algorithm is the departure and destination points P and Q; the service returns a route on the graph connecting the two points. Apart from other applications, this LBS gives the user the tool to make use of the previous service and find the way to the landmark of interest. This service is implemented by performing a simple request to the routing component (i.e. the *Routing Server* in Fig.1).
- *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 query point Q (for example, calling user's current location), and returns the set of points $L' \subseteq L$, which are the k nearest to Q members of L.

Regarding the third service, it is important to note that the conventional nearest neighbor (NN) search supported by current LBS solutions retrieves the k-NN objects based on the Euclidean distance between the reference (query) and the data objects stored in the database. However, the proper functionality of this service requires finding the nearest neighbor based on the *network distance* between the two points (i.e., the distance traveled by an object constrained to move on the network edges). On the other hand, a recent solution in the field of SNDB includes the "*Euclidean Restriction*" algorithm described in [12]. The algorithm is based on the observation that $D_{Eucl}(Q, P) \leq D_{Net}(Q, P)$ holds for each pair of two-dimensional points Q and P; as such, every object P' with Euclidean distance from Q greater than the respective network distance of another object P can be safely pruned without further considering its network distance, which by definition is greater than the respective Euclidean distance.

Based on the above three fundamental LBS, in Section 3 we propose a set of advanced LBS, covering also the *M-S* and *M-M* classes of services, according to the taxonomy introduced in [1].

3 Next Generation LBS

In this section we describe a set of novel services constituting our proposal regarding the next generation of LBS, which can be also considered as extensions of the three fundamental services discussed in Section 2.2. In particular, we focus on the following three services, named *Guide-me* (or, *Dynamic Routing*), *In-Route-Find-the-Nearest*, and *Get-together*. Once again, all services (and the respective algorithms) assume the presence of a graph G and/or a set of POIs L:

- *Guide-me (Dynamic Routing)*: A first extension of the (static) *Routing* described in Section 2.2 is the so-called *Guide-me* service, illustrated in Fig. 2(a). Likewise, the system determines the best route between the calling user's current location (point P) and a destination point Q, and, then, keeps track of the user's movement (by simply updating its position) towards the destination point, allowing him/her to deviate from the 'optimal' route, as long as his/her 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 R). The input of *Guide-me* algorithm contains the *id* of the calling user, the destination point Q, and the distance D, which defines the buffer width.
- *In-Route-Find-the-Nearest*: It is a combination of *Routing* and *Find-the-Nearest* services which, given a departure and a destination point, P and Q, respectively, finds the best route between them, constrained also to pass through one among the specified set of candidate points (e.g., one of the points contained in Landmarks). For example, a request for this service could be, "*provide me the best route from my current location to city A constrained to pass from a gas station*". Once again, the input of the respective algorithm contains the departure and destination points, P and Q respectively. This problem can also be considered as a special case of the so-called *Trip Planning Query* (*TPQ*) [5], with the number of different classes requested set to one.
- *Get-together*: With this service a moving user T_k 'attracts' a set of other users $S_k \subseteq T$ (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 Δt seconds) calculated by the system based on the future projection of the calling user's trajectory. As an example, consider Fig.2(b), where the calling user T_k is at (the spatiotemporal) location ($T_{k,j0}$) and moves towards the direction shown. The 'attracted' users T_{i1} and T_{i2} are located at ($T_{i1,j0}$) and ($T_{i2,j0}$) respectively. The system predicts that by the time $t_{\mathrm{j}}+\Delta t$ user T_k will most probably be located at location $T^{\dagger}_{k,j+\Delta t}$ (point P in Fig.2(b)) and, therefore, routes T_{i1} and T_{i2} towards this point. Eventually, at $t_{\mathrm{j}}+\Delta t$ the (recorded) location of T_k is $T_{k,j0+\Delta t}$. Likewise, the system projects the trajectory of T_k to the future time $t_{j0}+2\Delta t$, predicting that at that

time T_k will be at $T^{\dagger}_{k,j+2\Delta t}$ (point P' in Fig.2(b)), so it routes the rest of the users accordingly. The service terminates when at least one or all the members of S_k reach T_k. The input of the respective algorithm is the id of the 'master' user T_k, the set of the ids of the 'attracted' users S_k, and a time delay Δt. The algorithm returns a set of (dynamically updated) paths between each $T_i \in S_k$ and the periodically predicted location of T_k.

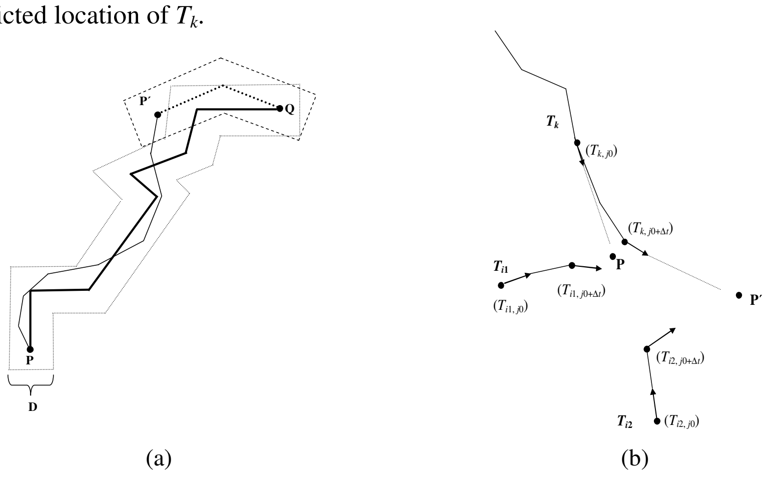

Fig. 2. (a) Guide-me and (b) Get-together examples

Among these services, the first evidently falls into the *M-S* class since it involves a moving user and several static data objects, while, the second is classified as *S-S*. Finally, the third service belongs to *M-M* class since both the reference (query) and the data objects are moving. The algorithms supporting the above three services are presented in the sections that follow.

3.1 Guide-me (Dynamic Routing)

As already discussed (and illustrated in Fig. 2(a)), this service requires the DBMS to keep track of the user's current position. As such, it exploits the previously introduced view *Current_Positions* over the table *Trajectories* (containing the current position of each user T_i). The algorithm developed to support the *Dynamic Routing* service is illustrated in the pseudo-code of Fig.3.

Apparently the first step of the `Dynamic_Routing` algorithm, involving the retrieval of the current position of object T_i, is a simple selection on *Current_Positions*. Regarding the second step, `Route` is performed by a simple request to the routing component (i.e., the Routing Server of Fig.1). Finally, in step 9, it is requested to check whether the object's current location $T_{i,j}$ lies on a buffer of the route R with distance D; this operation is performed by compiling the `Buffer` and `Contains` spatial methods provided by the Spatial DBMS extensions:

```
Algorithm Dynamic_Routing(User Id $T_i$, destination point $Q$,
distance $D$, time period $\Delta t$)
    $T_{i,j}$ = Retrieve current position of $T_i$
    $R$ = Route($T_{i,j}$,$Q$)
    DO WHILE $T_{i,j}$ has not reached $Q$
        Wait $\Delta t$; $j$ = $j$+$\Delta$t
        $T_{i,j}$ = Retrieve current position of $T_i$
        IF NOT $T_{i,j}$ lies in the buffer Buffer($R$,$D$) THEN
            $R$ = Route($T_{i,j}$,$Q$)
        ENDIF
    LOOP
```

Fig. 3. Algorithm Dynamic_Routing

```
SELECT * FROM CurentPositions
WHERE User_id=UId AND
Contains(Buffer(R,D),last_position_geometry)
```

The Contains(*A*,*B*) function returns true when the spatial object *A* contains the spatial object *B*, while the Buffer(*A*,*D*) function constructs a spatial object representing the buffer of the *A* with distance *D*.

3.2 In-Route-Find-the-Nearest

This service retrieves the best route one has to follow in order to travel from a departure to a destination point, *P* and *Q*, respectively, also constrained to pass via a landmark among the ones contained in the Landmarks table. The developed algorithm, based on the TPQ solutions provided in [5], is illustrated in the pseudo-code of Fig.4.

```
Algorithm In_Route_Find_the_Nearest(departure point $P$,
destination point $Q$)
Nearest.Dist=$\infty$
Retrieve route $R$=Route($P$,$Q$)
Find the Euclidean nearest object $N$ to the route object $R$
Calculate $D_{Net}(P,N)$ and $D_{Net}(N,Q)$
Retrieve all POIs $N_i$ having
                 $D_{Eucl}(P,N_i)+D_{Eucl}(N_i,Q) < D_{Net}(P,N)+D_{Net}(N,Q)$ and
sort them incrementally according to $D_{Eucl}(P,N_i)+D_{Eucl}(N_i,Q)$
DO WHILE $D_{Eucl}(P,N_i)+D_{Eucl}(N_i,Q) \leq$ Nearest.Dist
  IF Nearest.Dist > $D_{Net}(P,N_i)+D_{Net}(N_i,Q)$ THEN
    Nearest.Point = $P_i$
    Nearest.Dist = $D_{Net}(P,N_i)+D_{Net}(N_i,Q)$
  ENDIF
NEXT $N_i$
Return Nearest
```

Fig. 4. Algorithm In_Route_Find_the_Nearest

The `In_Route_Find_the_Nearest` algorithm is based on the same principle with the Euclidean restriction algorithm [12], that is, the Euclidean distance between two points serves as a lower bound for their network distance. As such, the algorithm initially produces the optimal route *R* between *P* and *Q* by performing a request to the Routing Server, while subsequently, uses *R* as a query object in order to retrieve the Euclidean *NN* among the records contained in the Landmarks table (Lines 2-3 in pseudo-code, also illustrated in Fig.5(a)). In this step, we exploit the R-tree-based nearest neighbor operator provided from the DBMS Spatial Extension, which retrieves the nearest to the query object *R* among those that are contained in the Landmarks table.

Then, the algorithm performs requests to the Routing Server in order to retrieve the best route between *P*, *N* and *Q* calculating the network distance between them (Line 4 in pseudo-code, also illustrated in Fig.5(b)), while afterwards uses their sum in order to retrieve candidate objects with a total distance from both *P* and *Q* upper bounded by it (Line 5 in pseudo-code, also illustrated in Fig.5(c)). It is also important to note that these objects are contained inside an elliptical region with *P* and *Q* as foci.

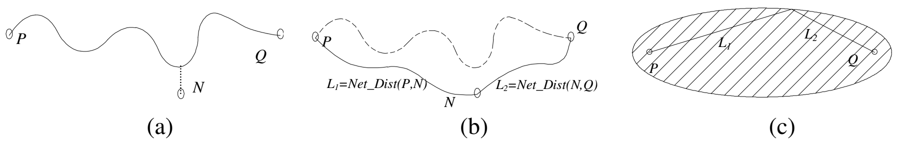

Fig. 5. An illustration of the In-Route-Find-the-Net-Nearest functionality

In its final step, the algorithm calculates the network distances between *P*, *Q* and the candidate points N_i, until the sum of the Euclidean distances of N_i from *P* and *Q* is greater than the respective network distance of the candidate nearest (lines 6-11). Finally, the algorithm reports the candidate nearest as the answer to the query.

3.3 Get-together

This service requires the DBMS to keep track of a user's current position (we name him/her 'master'), along with the positions of the users in the set S_k 'attracted' by user T_k; recall that all current positions are maintained in *Current_Positions* view. Moreover, in order to 'project' the master user's trajectory in a future position, it also employs the *Trajectories* table containing the history of each tracked object. The algorithm developed to support the *Get-together* service is illustrated in the pseudo-code of Fig.6.

The `Get_together` algorithm iterates until the position of all users contained in set S_k have converged to the master user's current position $T_{k,j}$ at timestamp t_j. Inside each iteration, the algorithm projects the trajectory of object T_k into the future timestamp $t_j + \Delta t$ calculating the respective position $T^{\dagger}_{k,j}$ on which T_k is estimated to be found at this future timestamp (Line 2). Subsequently, the algorithm computes and reports the routes needed for all objects of set S_k, in order to reach position $T^{\dagger}_{k,j}$ (Line 3), and finally, updates the actual positions of all objects involved in the algorithm at timestamp $t_j + \Delta t$ (Lines 5-6) - these positions are used in order to determine whether the algorithm may or not continue with the next iteration (Line 1).

```
Algorithm Get_together(User Id $T_k$, destination point $Q$,
distance $D$, time period $\Delta t$)
1   DO WHILE $T_{i,j}$ has not reached $T_{k,j}$ ($\forall T_i \in S_k$)
2       $T^{\dagger}_{k,j}$ =Project_trajectory($T_k$,$\Delta t$)
3       FOR EACH $T_i \in S_k$ Report Route($T_{i,j}$, $T^{\dagger}_{k,j}$ )
4       Wait $\Delta t$; $j = j+\Delta t$
5       FOR EACH $T_i \in S_k$ Update current position $T_{i,j}$
6       Update current position $T_{k,j}$ of $T_k$
7   LOOP
```

Fig. 6. Algorithm `Get_together`

Evidently, the most interesting operation involved in the algorithm is the `Project_trajectory` function. This routine estimates the future location of a moving object by projecting its trajectory to a future timestamp $t_j+\Delta t$. Such a computation requires some extra knowledge about the objects movement, 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, including [13] and [16], the problem of predicting the position of an object moving in a network, is very challenging by its nature, and therefore, still remains open. Specifically, the majority of the proposed strategies consider only the object's current position [4], while an efficient `Project_trajectory` function should make use of a trace (of dynamic length) of the trajectory for the estimation of the object's future location. Moreover, since our need is to estimate a single point on which the members of S_k will be routed, the `Project_trajectory` function should produce a 'winner' (i.e., single) projected point. As such, existing approaches which produce matrices with probabilities for each candidate network edge [4] cannot be used; instead we must estimate a single destination point for all 'attracted' objects.

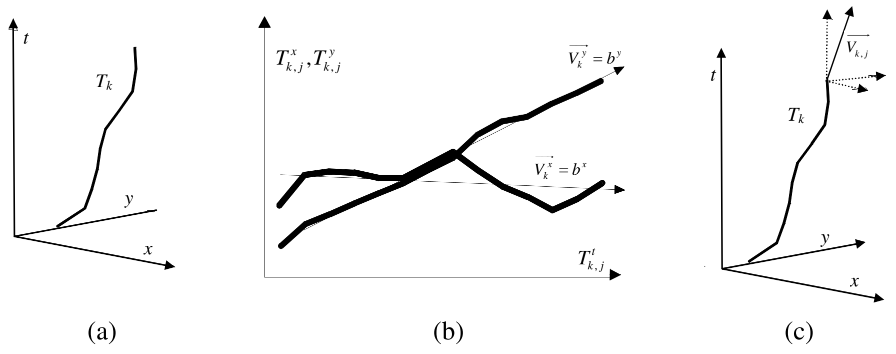
t
T_k
y
x
$T^x_{k,j}, T^y_{k,j}$
$\overrightarrow{V^y_k} = b^y$
$\overrightarrow{V^x_k} = b^x$
$T^t_{k,j}$
t
$\overrightarrow{V_{k,j}}$
T_k
y
x
(a)
(b)
(c)

Fig. 7. The Project_trajectory function: (a) a trajectory in the 3-dimensional space, (b) the x and y trajectory coordinates expressed as functions of time, and, (c) the estimated vector $\overrightarrow{V_{k,j}}$

We address this problem based on a simple regression model, which calculates the projection of the velocity vector along the *x*- and *y*- axes. Specifically, assuming that the object follows a 'general' direction, our goal is to determine this direction; we therefore employ the *simple linear regression* along each axis, assuming that the values of $T_{k,j}^{x}$ and $T_{k,j}^{y}$ at each timestamp $t_j = T_{k,j}^{t}$, linearly depend on time, i.e.,

$$T_{k,j}^{x} = a^{x} + b^{x} \cdot T_{k,j}^{t}\,, \text{ and } T_{k,j}^{y} = a^{y} + b^{y} \cdot T_{k,j}^{t}\,, \tag{1}$$

and, the velocity vector of this 'general' direction at timestamp t_j will be determined as the vector $\overrightarrow{V_{k,j}} = \overrightarrow{V_{k,j}^{x}} + \overrightarrow{V_{k,j}^{y}}$. The magnitude for both $\overrightarrow{V_{k}^{x}} = b^{x}$ and $\overrightarrow{V_{k}^{y}} = b^{y}$ can be calculated using *linear regression analysis techniques* given the values of time-stamped sampled positions of T_k. Consider, for example, Fig.7(a) illustrating a trajectory T_k in the 3-dimensional space; then, Fig.7(b) demonstrates the corresponding *x* and *y* trajectory coordinates as functions of time, along with the values of $\overrightarrow{V_{k}^{x}} = b^{x}$ and $\overrightarrow{V_{k}^{y}} = b^{y}$ calculated by the 'mean' trajectory direction along each axis. Then, their combination in Fig.7(c) produces the vector $\overrightarrow{V_{k,j}}$ standing for the general trajectory direction.

Finally, a first approximation of the projected location $T_{k,j+\Delta t}^{\dagger}$ can be calculated by the simple formula $T_{k,j+\Delta t}^{\dagger} = T_{k,j} + \overrightarrow{V_{k,j}} \cdot \Delta t$, which leads to

$$T_{k,j+\Delta t}^{\dagger x} = T_{k,j}^{x} + \left|V_{k,j}^{x}\right| \cdot \Delta t \text{ and } T_{k,j+\Delta t}^{\dagger y} = T_{k,j}^{y} + \left|V_{k,j}^{y}\right| \cdot \Delta t \tag{2}$$

It is also worth to note that the velocity vectors produced by linear regression already include the mean object speed along the two axes, which, consequently, is incorporated in the above formulas. Anyway, its value $\overline{V_{k,j}}$ can be easily calculated as the magnitude of vector $\overrightarrow{V_{k,j}}$, thus:

$$\overline{V_{k,j}} = \left|V_{k,j}\right| = \sqrt{\left|V_{k,j}^{x}\right|^{2} + \left|V_{k,j}^{y}\right|^{2}} \tag{3}$$

There is one more issue that has to be clarified regarding the calculation of $\left|V_{k,j}^{x}\right|$ and $\left|V_{k,j}^{y}\right|$. Specifically, one first approach on their computation via the regression analysis, is to proceed with it based on the entire moving object trajectory history (i.e., using the complete set of time-stamped moving object positions $T_{i,j}$ for j = 0, .., *now*). Although, this approach may sound reasonable, it can lead to false conclusions regarding their estimated values since, moving objects may change travelling direction arbitrarily, perform U-turns, vary their speed etc. Consider, for example, a truck delivering several commercial products around a metropolitan area; the truck starts its trip from a departure point (e.g., the company warehouse), delivers the products to several locations around the city (stopping at each one of them), and returns back to its original location (i.e., the company warehouse). Thus, the values of

$\left|V_{k,j}^x\right|$ and $\left|V_{k,j}^y\right|$ calculated from the regression model will include all this arbitrarily directed trajectory history.

On the other hand, the direction criterion applied in *local circumstances* can provide more accurate results; for example, when this delivery truck moves between two delivery points, it actually follows a general direction. As such, we propose that *the calculation of* $\left|V_{k,j}^x\right|$ *and* $\left|V_{k,j}^y\right|$ *could be safely performed based on the* Δt *last timestamps*. Formally, in our implementation, $\left|V_{k,j}^x\right|$ and $\left|V_{k,j}^y\right|$ are determined by the following formulas (which are directly derived from the linear regression analysis of the last Δt time-stamped trajectory sampled positions):

$$\left|V_{k,j}^x\right| = \frac{\sum_{l=j-\Delta t}^{j}\left(T_{k,l}^x - \overline{T_{k,j}^x}\right)\cdot\left(T_{k,l}^t - \overline{T_{k,j}^t}\right)}{\sum_{l=j-\Delta t}^{j}\left(T_{k,l}^x - \overline{T_{k,j}^x}\right)^2}, \text{ and } \left|V_{k,j}^y\right| = \frac{\sum_{l=j-\Delta t}^{j}\left(T_{k,l}^y - \overline{T_{k,j}^y}\right)\cdot\left(T_{k,l}^t - \overline{T_{k,j}^t}\right)}{\sum_{l=j-\Delta t}^{j}\left(T_{k,l}^y - \overline{T_{k,j}^y}\right)^2} \qquad \textbf{(4)}$$

where $\overline{T_{k,j}^x} = \frac{1}{\Delta t+1}\cdot\sum_{l=j-\Delta t}^{j} T_{k,l}^x$, $\overline{T_{k,j}^y} = \frac{1}{\Delta t+1}\cdot\sum_{l=j-\Delta t}^{j} T_{k,l}^y$ and $\overline{T_{k,j}^t} = \frac{1}{\Delta t+1}\cdot\sum_{l=j-\Delta t}^{j} T_{k,l}^t$.

Subsequently, in order to support more realistic, network-constrained projected location, in the proposed algorithm we search for the nearest network link based on $T_{k,j+\Delta t}^{\dagger}$, and we route object T_k to it, thus producing route R. Finally, we determine the point on this route, on which the object will be found after the Δt time period, i.e., after traversing a distance of $\overline{V_k}\cdot\Delta t$ (which is performed by simply traversing it and counting the distance so far). The developed algorithm is illustrated in the pseudo-code of Fig.8.

```
Algorithm Project_Trajectory(User Id $T_k$, time period $\Delta t$)
1    Calculate $\left|V_k^x\right|$ and $\left|V_k^y\right|$ based on Eq.(4)
2    $T_{k,j+\Delta t}^{\dagger} = T_{k,j} + \overrightarrow{V_k} \times \Delta t$
3    $R$ = Route($T_{k,j}$ , $T_{k,j+\Delta t}^{\dagger}$ )
4    Determine point $P$ on $R$ with $D_{Net}(T_{k,j}, P) = \left|V_k\right|\cdot\Delta t$
5    Return $P$
```

Fig. 8. Algorithm `In_Route_Find_the_Nearest`

A final point to be discussed is that the calculation of $\left|V_{k,j}^x\right|$ and $\left|V_{k,j}^y\right|$ through Eq.(4) (i.e., the first algorithm step) requires only to query the *Trajectories* table based on the master user's id and the last Δt timestamps; the results of this query will retrieve the set of all time-stamped trajectory positions involved in Eq.(4) which will be subsequently used to produce $\left|V_{k,j}^x\right|$ and $\left|V_{k,j}^y\right|$.

4 Implementation Details

All the services presented in Sections 2 and 3 have been implemented on top of three basic components. The first one is the *Microsoft SQL Server* [9], which is a relational database management system (RDBMS) that does not natively support spatial objects such as points, lines etc; consequently, the employment of an extension component, which enables SQL Server to support spatial data is an obligatory action, in order for the LBS suite to be properly developed.

The Spatial Extension component is the *MapInfo SpatialWare* [6], which enables the DBMS to store, manage, and manipulate location-based data. It allows therefore spatial data to be stored in the same place as traditional data, ensuring data accessibility, integrity, reliability and security through the mechanisms of the SQL Server. SpatialWare includes a variety of non-traditional data types, such as points, lines, polyline, regions (polygons), supports numerous spatial functions,and it is Open GIS compliant [11]. However, the most important SpatialWare feature is the support it provides for R-tree indexing [2], making it able to support huge volumes of spatial data; R-tree indexing allows pruning the search space when a spatial query is executed. Otherwise (i.e., in the case where no spatial index is present), the execution of each spatial query would lead to linear scans over the entire dataset, which is a very expensive operation.

The *Routing Server* component used in our implementation is the *MapInfo Routing J Server* (*RJS*) [7], which is a street network analysis tool for finding a route between two points, the optimal or ordered path between many points, the creation of drive time matrices, and the creation of drive time polygons. RJS calculates either the shortest distance or quickest timed route between any two points, returning text-based driving directions and spatial points to the parent application. This functionality is achieved by *XML requests* over a continuously running server: the client queries the RJS with an XML file containing information such as, the departure and the destination point, and after processing the request, RJS returns another XML file containing the optimal route in terms of its lines segments (i.e., edges of the respective directed graph). Mapinfo provides also the *RJS .NET client* middleware, which undertakes the tasks of composing the XML file used for making the request, and subsequently, interpreting the server's answer to a set of comprehensive objects implemented in the form of .NET objects.

Finally, the LBS suite, requesting data from all the above components, is implemented with Microsoft Visual Studio .NET [8], while the connection to the DBMS is realized by an *OLE DB connection* [9].

5 Summary

In this paper we presented typical examples of the next generation of location based services, sketched up the respective algorithms and provided some interesting details on their implementation. The majority of the presented services are not currently supported by commercial LBS providers, or are available in an inefficient and inaccurate manner. Among others, the developed services involve nearest neighbor queries using network (rather than Euclidean) distance, optimal route finding between a set of user-defined landmarks, and in-route nearest neighbor queries. The developed algorithms and their implementation are supported by recent advances in the field of Spatial Network Databases [5],[12],[14], and novel techniques originally proposed in this paper.

The solutions provided are not only focused on LBS; actually, many of them are directly applicable in the context of route planning, which is a task usually performed via web services. Therefore, our plan is to employ our LBS suite in the framework of a web-based application providing users with advanced functionality regarding their travelling needs.

Acknowledgments

Research partially supported by FP6/IST Programme of the European Union under GeoPKDD project (2005-08) and EPAN Programme of the General Secretariat for Research and Technology of Greece under NGLBS project (2004-06). The services proposed in this paper are part of a commercial LBS platform; they designed and developed in cooperation with Telenavis S.A. in the context of NGLBS project. Telenavis S.A. is a commercial LBS provider providing both GSM-based and web-based private and corporate solutions (see for example http://www.navigation.gr).

References

1. Gratsias, K., Frentzos, E., Delis, V. and Theodoridis, Y., 2005: *Towards a Taxonomy of Location-Based Services*. Proceedings of Web and Wireless GIS (W2GIS), 2005
2. Guttman, A., 1984: *R-Trees: a dynamic index structure for spatial searching*. Proceedings of ACM SIGMOD Conference, 1984.
3. Jensen, C.S.; Christensen, A.; Pedersen, T.; Pfoser, D.; Saltenis, S. and Tryfona, N., 2001: *Location-Based Services: A Database Perspective*. Proceedings of Scandinavian GIS, 2001.
4. Karimi H. and Liu, X., *A Predictive Location Model for Location-Based Services*, Proceedings of ACM-GIS, 2003
5. Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G. and Teng, S.-H., 2005: *On Trip Planning Queries in Spatial Databases*. Proceedings of SSTD, 2005
6. MapInfo Corporation, 2007a: *MapInfo SpatialWare*, Available at http://extranet.mapinfo.com/products/ Overview.cfm?productid=1141, (accessed 18 May 2007)
7. MapInfo Corporation, 2007b: *MapInfo Routing J Server*, Available at http://extranet.mapinfo.com/ products/Overview.cfm?productid=1144, (accessed 18 May 2007)
8. Microsoft Corporation, 2007a: *Microsoft Visual Studio .NET*, Available at http:// msdn.microsoft.com/ vstudio/, accessed 18 May 2007.
9. Microsoft Corporation, 2007b: *Microsoft SQL Server*, Available at http://www.microsoft.com/sql/, accessed 18 May 2007.
10. Nanni, M., Pedreschi, D.: Time-focused clustering of trajectories of moving objects. *J. Intell. Inf. Syst*. 27(3): 267-289 (2006)

11. Open GIS Consortium, 2007: *OpenGIS® Location Services (OpenLS): Core Services*. Available at http://www.opengis.org, accessed 18 May 2007
12. Papadias, D., Zhang, J., Mamoulis, N., and Tao, Y., 2003: *Query Processing in Spatial Network Databases*. Proceedings of VLDB Conference, 2003.
13. Saltenis, S.; Jensen, C. S.; Leutenegger, S.; and Lopez, M., 2000: *Indexing the Positions of Continuously Moving Objects*. Proceedings of ACM SIGMOD Conference, 2000.
14. Sankaranarayanan, J.; Alborzi, H.; and Samet, H., 2005, *Efficient Query Processing on Spatial Networks*. Proceedings of ACM-GIS, 2005.
15. Theodoridis, Y., Ten Benchmark Queries for Location-based Services, *The Computer Journal*, vol. 46(6), pp. 713-725, 2003.
16. Tao, Y.; Papadias, D.; and Shen, Q., 2002.: *Continuous Nearest Neighbor Search.* Proceedings of VLDB Conference, 2002.
17. Tao Y., Papadias D., Sun J., *The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries*, Proceedings of VLDB Conference, 2003.