ΥΛΟΠΟΙΩΝΤΑΣ ΤΗΝ ΕΠΟΜΕΝΗ ΓΕΝΙΑ ΥΠΗΡΕΣΙΩΝ ΘΕΣΗΣ

Ηλίας Φρέντζος, Κώστας Γρατσίας και Γιάννης Θεοδωρίδης

Πανεπιστήμιο Πειραιώς, Τμήμα Πληροφορικής και
Ερευνητικό Ακαδημαϊκό Ινστιτούτο Τεχνολογίας Υπολογιστών (ΕΑ ΙΤΥ)

Περίληψη
Οι υπηρεσίες θέσης αποτελούν ένα αναδυόμενο πεδίο εφαρμογών, το οποίο βρίσκει όλο και περισσότερες εφαρμογές σε πολλές δραστηριότητες της σύγχρονης ζωής. Παρόλο που έχουν ήδη μερικά χρόνια εμπορικής ζωής, το σύνολο των υπηρεσιών θέσης και των λύσεων που εφαρμόζονται σε αυτές μπορούν να θεωρηθούν απλοϊκές, καθώς δεν εκμεταλλεύονται τις δυνατότητες ούτε του σύγχρονου λογισμικού, αλλά ούτε τα πρόσφατα ερευνητικά αποτελέσματα στο πεδίο των βάσεων χωρικών και χωροχρονικών δεδομένων. Ο σκοπός της παρούσας εργασίας είναι να συμπληρώσει αυτό το κενό, παρουσιάζοντας αρχικά την επόμενη γενιά των υπηρεσιών θέσης, και στη συνέχεια, επεξηγώντας τον τρόπο υλοποίησής τους με την αξιοποίηση σύγχρονων εμπορικών λογισμικών καθώς και πρόσφατων ερευνητικών εργασιών. Οι προτεινόμενες υπηρεσίες δεν είναι προσανατολισμένες μόνο προς τη κατεύθυνση των παραδοσιακών υπηρεσιών θέσης, οι οποίες παρέχονται σε έναν κινούμενο χρήστη μέσω ασύρματων δικτύων. Στη πραγματικότητα, πολλές από αυτές μπορούν να βρουν εφαρμογή στο καθορισμό – σχεδίαση βέλτιστων διαδρομών (πριν την έναρξη της μετακίνησης), μίας εργασίας η οποία πραγματοποιείται συνήθως μέσω διαδικτυακών εφαρμογών. Επιπλέον, από τη στιγμή που η υλοποίηση των υπηρεσιών είναι βασισμένη σε πλατφόρμες λογισμικού με υψηλές δυνατότητες κλιμάκωσης, μπορούν να εξυπηρετήσουν συγχρόνως αιτήσεις από ένα πολύ μεγάλο αριθμό χρηστών. Μπορούν επομένως πολύ εύκολα να συμπεριληφθούν στο πλαίσιο μία διαδικτυακής εφαρμογής που θα παρέχει στους χρήστες της προηγμένη λειτουργικότητα σε σχέση με τη θέση τους και τις ταξιδιωτικές τους ανάγκες.

IMPLEMENTING THE NEXT GENERATION OF LOCATION BASED SERVICES

Elias Frentzos, Kostas Gratsias and Yannis Theodoridis

University of Piraeus, Department of Informatics and
Research Academic Computer Technology Institute (RA CTI)

Abstract
Location-based services (LBS) constitute an emerging application domain rapidly introduced in modern life habits. However, given that LBS already have 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 databases. The goal of this paper therefore is to fill this gap by, presenting the next generation of location-based services and, then, demonstrating their implementation which takes advantage of both modern commercial software and recent advances in the research field of spatial and spatio-temporal databases. The solutions provided are not only focused on LBS; actually, many of them are easily applicable in the context of route planning, which is a task usually performed via web applications. Moreover, since the implementation is based on highly scalable platforms it can support requests from numerous users at the same time. Therefore, they can be easily employed in the framework of a web-based application providing users with advanced functionality regarding their location and travelling needs.

Λέξεις κλειδιά: Υπηρεσίες θέσης, Βάσεις Χωρικών και Χωροχρονικών Δεδομένων, Αλγόριθμοι
Key words: Location-based Services, Spatial and Spatiotemporal Databases, Algorithms

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), which provide information relevant to the spatial location of a receiver. LBS constitute an innovative technological field, rapidly introduced in modern life habits, influencing the way that people organize their activities, promising great business opportunities for telecommunications, advertising, tourism, etc. (Open Geospatial Consortium, 2007). On the other hand, although LBS already have a few years of commercial life, the services 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 thus of this paper is to fill this gap by presenting the next generation of location-based services and, then, to demonstrate their implementation taking advantage of both modern commercial software and recent advances in the research field of spatial and spatio-temporal databases.

More specifically, in this paper we present a set of LBS and then sketch up the respective algorithms along with a description of their implementation. The majority of the presented services are not currently supported by commercial LBS providers. The developed software is based on the *Microsoft.NET* (Microsoft Corp., 2007a) and *SQL Server* platforms (Microsoft Corp., 2007b), while it employs two MapInfo components, the first enabling SQL Server to support spatial objects and R-tree indexing (i.e., the *MapInfo SpatialWare* (MapInfo Corp., 2007)), and the other implementing the routing algorithm between two nodes along a given road network graph (i.e., *MapInfo Routing J Server* (MapInfo Corp., 2007b)). Exploiting the functionality provided by these components, we expand it towards many directions. Among others, the developed software supports 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 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 applications. Moreover, since our implementation is based on highly scalable platforms (e.g., SQL Server) it can support requests from numerous users at the same time. Therefore, they can be easily employed in the framework of a web-based application providing users with advanced functionality regarding their location and travelling needs.

Outlining the rest of the paper, Section 2 presents two sets of LBS (i.e., one with services currently supported by commercial LBS provider, and one with novel services, constituting our proposal regarding the next generation of LBS). Section 3 presents implementation issues (presenting the development platforms, and exemplifying the implemented algorithms used to support the proposed services). Finally, Section 4 closes the paper providing the conclusions and some interesting research directions. Table 1 summarizes the notation used in the rest of the paper.

Table 1. Table of notations

$V=\{V_i\}$, $i=1..n_1$	the set of vertices corresponding to road network junctions on a road network.
$E=\{E_i\}$, $i=1..n_2$	the set of edges connecting vertices V_i, corresponding to road segments on a road network[1].
$G(V, E)$	the directed graph that represents the underlying road network on which objects are moving[2].
$L=\{L_i\}$, $i=1..n_3$	the set of all points of interest (POIs) or Landmarks[3].
$T=\{T_i\}$, $i=1..n_4$	the set of all mobile users
$T_{i,j}$	the spatio-temporal point (i.e., time-stamped spatial point) of user T_i at timestamp t_j
Eucl_Dist (P, Q)	the Euclidean distance between points P and Q
Net_Dist (P, Q)	the network distance on the graph G between the points P and Q
Buffer(X, D)	builds a buffer of width D around a path X
Route(P, Q)	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. The Next Generation of LBS

In this section we describe a set of novel services constituting our proposal regarding the next generation of LBS; obviously, these services are not currently supported by commercial LBS providers. We also include in our discussion a set of already implemented services since: (a) they are fundamental and thus used as a basis for the (more advanced) novel LBS set, and, (b) existing solutions on these services (i.e., algorithms and implementation details) are rather naïve and based on approaches usually resulting in false results. The services were designed and developed on behalf of the Telenavis S.A. (Telenavis, 2007) in the context of the *Next Generation Location Based Services* (*NGLBS*) project funded by the General Secretarial of Research and Technology. 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).

[1] Each edge E_i 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.

[2] Movement constraints (such as user defined discretionary absence of tolls etc.) can be also applied in the graph $G(V, E)$ by simply modifying the weights of each edge.

[3] POIs can be categorized into classes (e.g., 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 due to clarity reasons.

2.1. A set of LBS

The first set of LBS which is already implemented by commercial service providers contains three fundamental services, named, *What-is-around*, *Routing*, and *Find-the-Nearest*. All services (and the respective algorithms) assume the presence of 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 (length and time); in the following sections, for clarity reasons, we restrict our discussion in the distance (rather than time) optimization, while the second criterion can be easily applied, by simply involving a maximum speed which converts any time period to a maximum distance. The following paragraphs describe the functionality of each LBS:

- *What-is-around*: The simplest service is the one that retrieves and displays the location of every POI being located in 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 for "*what-is-around*" 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).
- *Routing*: This service provides the optimal route between a departure and a destination point, P and Q, respectively.
- *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[4].

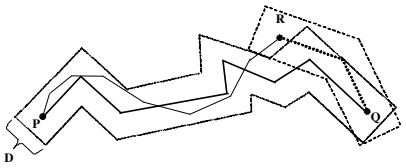

Figure 1: Guide-me example

2.2. A set of Novel LBS

The second set of LBS contains a number of advanced services which can be considered as extensions of the above three fundamental services. Among the ones that were designed and implemented during the NGLBS development, we will focus on three services, named *Guide-me (or Dynamic Routing)*, *Advanced Routing* and *In-Route-Find-the-Nearest*. The functionality of these services is described in the following paragraphs. For a more detailed description regarding all services, along with an interesting LBS taxonomy the interested reader is cited to Gratsias et al., 2005. Once again, all services (and the respective algorithms) assume the presence of graph G and/or a set of POIs L:

- *Guide-me (Dynamic Routing)*: A first extension of the (static) *Routing* described above is the so-called *Guide-me* service (Fig. 1): Likewise, the system determines the best route between the calling user's current location (point P) and a destination point Q, and then it 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 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 R). The input of *Guide-me* algorithm is the *id* of the calling user, the destination point Q, and the distance D, which defines the buffer width.
- *Advanced Routing*: The *Routing* service provided above can also be extended towards its "advanced" version, by requesting from the system to retrieve the best route between a departure and a destination point, P and Q respectively, requesting also to travel through a set of intermediate points $C=\{C_i\}$. The input of the respective algorithm is the departure and destination points, P and Q respectively and the set of intermediate points C.
- *In-Route-Find-the-Nearest*: It is a combination of the *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

[4] It is important to note here that the conventional nearest neighbor query supported by commercial SDBMS retrieves the nearest neighbor based on the Euclidean distance between the query and the data points stored in the SDBMS. 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)

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 is, "*provide me the best route from my current location to the city A constrained to pass from a gas station*". Once again, the input of the respective algorithm is the departure and destination points, *P* and *Q* respectively. This problem can also be seen as a special case of the so-called *Trip Planning Query* (*TPQ*) (Li *et al.,* 2005), with the number of different classes requested set to one.

3. Implementing the Services

In this section we describe the implementation of the proposed LBS suite. We will firstly introduce the development platforms, while we will subsequently illustrate the respective algorithms along with some interesting details on each service's implementation.

3.1 Development Platforms

All the services are implemented on top of three basic components. The first one is the *Microsoft SQL Server 2000* (Microsoft Corp., 2007b), which is a relational database management system (RDBMS) produced by Microsoft including standard RDBMS functionality (i.e., stored procedures, triggers etc.). SQL Server is commonly used in small to large enterprise databases. Here, we have to point out that Microsoft SQL Server does not natively support spatial objects such as points, lines etc.; consequently, the employment of a middle-ware component which enables SQL Server to support spatial data is an obligatory action, in order for the LBS suite to be properly developed.

This middle-ware component is the *MapInfo SpatialWare* (MapInfo Corp., 2007a), which enables the RDBMS (in our case, SQL Server) 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 (such as `ST_Buffer` generating buffers around spatial objects within a given tolerance etc.), and it is compliant with the *Open Geospatial Consortium* (*OGC*), 2007. However, the most important SpatialWare feature is its support for R-tree indexing (Guttman, 1984), making it able to support substantial quantities 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 third component used in the implementation of the LBS suite is the *MapInfo Routing J Server* (*RJS*) (MapInfo Corp., 2007b) 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 (i.e., the LBS suite) 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). In our implementation we used the *RJS .NET client* middleware, developed by Telenavis S.A. (Telenavis, 2007), 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. However, this comes for a cost, since this client only supports the simple operation of finding a route between two points, and not the more sophisticated ones that native RJS does (i.e., finding optimal or ordered path between many points).

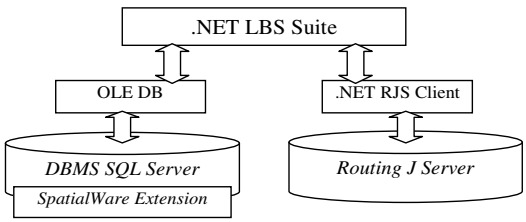

Figure 2: System Architecture

In top of all the above components, the LBS suite, is implemented using Microsoft Visual Studio .NET 2003 (Microsoft Corp., 2007a), while the connection to the DBMS is realized by an *OLE DB connection*

(Microsoft Corp., 2007a) to the SQL Server. Figure 2 summarizes the system architecture used for the developed LBS suite.

3.2 Implementation Details

In this section we will examine the implementation details of the above services. As previously discussed, since the first three services (i.e., What-is-around, Routing and Find-the-Nearest) are already included in the commercial – publicly available – Telenavis LBS suite, we will briefly describe both the existing solution along with our (more sophisticated) implementation. Then, we will proceed with the novel services, describing the algorithms and providing details about their implementation. In the rest of the paper we will refer to the existing LBS implementation as *TNLBS*, while ours will be called as *NGLBS*. Here, we have to point out that one of the core differences between TNLBS and NGLBS is that the former relies only on the SQL Server, without employing the SpatialWare component.

What-is-around

The first service on the TNLBS basically relies on the SQL Server DBMS, which contains, among others, a table of Landmarks (e.g., gas stations) in the form of [*object_id, object_name, x, y*]. Obviously, the spatial components of the landmark objects are represented in terms of their Cartesian coordinates in different table fields, resulting in a non-efficient scheme. We have to point out again however, that since the SQL Server does not natively support spatial objects, the above scheme is the only one suitable. The core of the service implementation is the following query (providing that the query point Q is given in terms of its coordinates Qx and Qy and d is a selected distance, that is, the half-side of the query rectangle):

```
SELECT * FROM Landmarks WHERE x>=Qx-d AND x<=Qx+d AND y>=Qy-d AND y<=Qy+d
```

On the other hand, exploiting the fact that NGLBS is based on a scheme which includes the SpatialWare component, the *x* and *y* fields can be substituted by a single *Geometry* field; as a result, the Landmarks table is reformulated in the structure [*object_id, object_name, object_geometry*], while the above query can be rewritten in terms of spatial database operations, employing the `HG_Box` function which returns a rectangle with the given coordinates:

```
SELECT * FROM Landmarks WHERE
ST_Overlaps(object_geometry,HG_Box(Qx-d,Qy-d,Qx+d,Qy+d)
```

Routing

The second service also included in the TNLBS, requires only querying the .NET RJS client with the appropriate inputs, i.e., the departure and destination points P and Q; the network graph $G(V, E)$ is already contained inside RJS. The results are provided in terms of text-based driving directions and spatial objects following the application model (i.e., points, lines, poly-lines). Regarding the algorithm used inside RJS in order to calculate the shortest path between the two points, RJS documentation does not provide any information about it. However, the problem of calculating shortest paths in graphs is well-known; therefore it can be solved by employing a variety of algorithms (see for example the Dijkstra (Dijkstra, 1959) algorithm). There is strong evidence however that the implementation of RJS in based on a variation of the A* algorithm (Hart et al, 1968) which is the one that is usually employed in real-world applications involving network graphs due to its computational optimality and straightforward implementation. This service is the only one that is implemented following exactly the same manner in both TNLBS and NGLBS implementations.

Find-the-Nearest

The third service, which retrieves the k nearest landmarks to the caller's location, is currently implemented in TNLBS based on the Landmarks stored in the SQL Server DBMS (using the (x, y) representation), and the RJS. The respective algorithm initially retrieves the $3 \cdot k$ nearest landmarks to the query location, which are subsequently treated as candidate nearest points. The task of retrieving the $3 \cdot k$ nearest landmarks is achieved by performing a SQL query calculating the Euclidean distance between the query point Q and all points contained inside the Landmarks table, and then, sorting the results according to the calculated distance:

```
SELECT TOP 3*k object_id,((Qx-x)^2+(Qy-y)^2) AS Dist FROM Landmarks
                        ORDER BY Dist DESC
```

The algorithm subsequently performs routing operations (using the .NET RJS client) to all candidate object retrieved by the previous query, sorts them according to the resulted *network* distance from the query point and finally reports the first $3 \cdot k$ objects.

However, there is strong controversy regarding this algorithm's performance and quality of output. For example, there is no concrete background behind the choice of retrieving the $3 \cdot k$ nearest points in order to treat them as candidates. As such, the approach of multiplying the number of k requested by 3 (or any other arbitrarily selected coefficient) may lead to false outputs; a case which is clearly illustrated in Fig.3(a), where the three nearest objects according to their Euclidean distance from the query point Q are points P_1, P_2 and P_3, while point P_4 is the actual nearest neighbor.

Moreover, the performance of this algorithm is far from being optimal, since it requires calculating the distance between the query and *all* POIs in the database during the execution of the above SQL statement. This happens due to the fact that there is no spatial index present, leading the database to perform a linear scan over Landmarks (calculating at the same time the requested distance expression); then the database performance deteriorates when the number of POIs exceeds a few hundreds of thousands.

On the other hand, in our implementation, we employed recent technological advances in the field of Spatial Network Databases (*SNDB*), and implemented the "*Euclidean Restriction*" algorithm described in (Papadias et al., 2003) in order to retrieve the nearest to a query point. This algorithm is illustrated in the following pseudo-code:

```
Algorithm Find_the_Net_Nearest(point Q)
1     Find the Euclidean nearest object P to the query object Q
2     Calculate Net_Dist(Q,P)
3     Retrieve all POIs Pi with Eucl_Dist(Q,Pi)<Net_Dist(Q,P)
4     For each Pi calculate Net_Dist(Q,Pi)
5     Return as nearest neighbor the object with the smaller Net_Dist(Q,Pi)
```

The algorithm is further exemplified in Fig,3(b) to Fig.3(d). Specifically, Fig.3(b) illustrates the first step (i.e., the algorithm retrieves object P_1 which is the nearest object to Q according to the Euclidean distance), then, its network distance D_1 is calculated (i.e., step 2), and finally object P_2 is retrieved since its Euclidean distance from Q is smaller than D_1 (i.e., step 3), and further examined as possible nearest neighbour in steps 4-5. This algorithm is based on the fact that Eucl_Dist(Q,P)≤Net_Dist(Q,P),∀ Q,P; as such, every object P' with Euclidean distance from Q greater than the respective network distance of another object P can be safely rejected (pruned) without further considering its network distance; formally, given that Eucl_Dist(Q,P')≥Net_Dist(Q,P), then also Net_Dist(Q,P')≥Net_Dist(Q,P') stands. It is proved therefore that the nearest neighbor *must* be found among the objects retrieved in the third step. For each network distance calculation requested, a routing operation is performed via the .NET RJS client, and the respective distance is calculated accordingly. Regarding the first algorithm's step which retrieves the Euclidean nearest object P to the query object Q, rather than using an approach similar to the one of TNLBS, we exploit the R-tree-based nearest neighbor operator provided from the SpatialWare, which improves the algorithm's performance:

```
SP_Nearest Landmarks, object_geometry, object_id, ST_Point(Qx,Qy),k
```

The `SP_Nearest` operator retrieves the nearest to the point `ST_Point(Qx,Qy)` among those that are contained inside the Landmarks table. Concluding, by employing the previously presented approach (followed in the NGLBS implementation), the Find-The-Nearest service, not only retrieves exact solutions, but since it employs spatial indexes, is much more efficient than TNLBS.

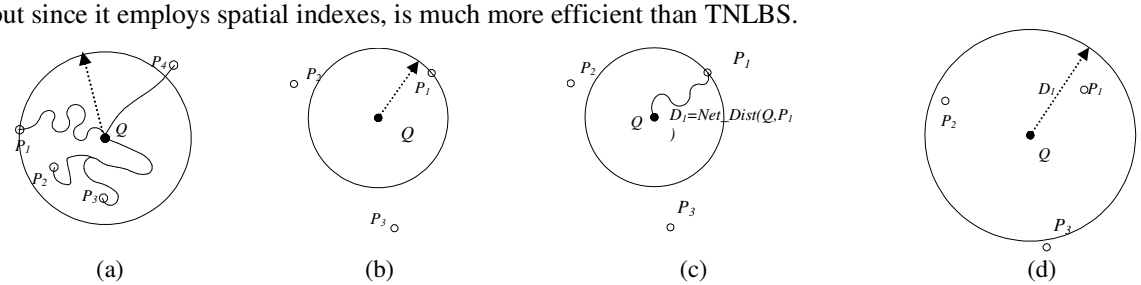

Figure 3: The Find-the-net-nearest service

Guide-me (Dynamic Routing)

This service requires the DBMS to keep track of the user's current position. As such, the SQL Server contains, among others, a relational table with each user's T_i current positions, which are updated from outside the developed suite. This table is named *Current_Positions* and has the form of [User_id, last_position_geometry]. The algorithm developed to support the *Dynamic Routing* service is illustrated in the following pseudo-code:

```
Algorithm Dynamic_Routing(User Id T, destination point Q, distance D, time period Δt)
1     Retrieve current position T_{i,j} of T_i
2     Retrieve route R=Route(T_{i,j},Q)
3     DO until T_{i,j} reaches Q
4        Wait Δt: j=j+Δt: Update T_{i,j}
5        IF NOT T_{i,j} lies on the buffer Buffer(R,D) go to Step 1
6     LOOP
```

Regarding the first step, it is performed by a simple request to the .NET RJS client. Probably, the most interesting operation of the algorithm is revealed in step 5, where 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 via the Spatialware by checking whether the following SQL statement returns any records:

```
SELECT * FROM CurentPosition WHERE User_id=UId AND
ST_Contains(HG_Buffer(ST_Spatial(R_String),D),last_position_geometry)
```

The `ST_Contains(A,B)` function returns `true` when the spatial object A contains the spatial object B, while the `HG_Buffer(A,D)` function constructs a spatial object representing the buffer of the A with distance D. Finally, the `ST_Spatial(string)` function converts a properly composed string (e.g., the route string) to a spatial object.

Advanced Routing

The *Advanced Routing* service, involving a departure and destination point, P and Q respectively, along with a set of predefined intermediate points P_i, can be seen as a variation of the well known *travelling salesman problem* (*TSP*); however in our case there are two special requirements

- the distance between P, Q and P_i is not Euclidean, thought it satisfies the triangle inequality, and
- the distances between points P_i (i.e., *Net_Distance*(P_i, P_j)) are not known in advance, rather than they are calculated during the algorithm's execution

As such, the algorithm recursively examines alternative solutions, until all possible routes have been checked. The algorithm prunes candidate routes by using the minimum network distance calculated so far; pruning is also performed using the Euclidean distance as a first approximation, and then, if the solution is not pruned by its Euclidean Distance, the network distance is calculated and the algorithm recursively proceeds with the remained objects until all alternative solution have been examined or pruned. The details of this algorithm are beyond the scope and this paper, and for this reason are omitted. It is however important to note that this algorithm utilizes only the .NET RJS client, without querying at all the DBMS.

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, constrained also to pass through a POI among the ones contained in the Landmarks table. The developed algorithm is illustrated in the following pseudo-code:

```
Algorithm In_Route_Find_the_Nearest(departure point P, destination point Q)
1  Retrieve route R=Route(P,Q)
2  Find the Euclidean nearest object N to the route object R
3  Calculate Net_Dist(P,N) and Net_Dist(N,Q)
4  Retrieve all POIs N_i with Eucl_Dist(P,N_i)+Eucl_Dist(N_i,Q)<Net_Dist(P,N)+Net_Dist(N,Q)
5  For each N_i calculate Net_Dist(P,N_i) and Net_Dist(N_i,Q)
6  Return as nearest neighbor the object N_i with the smallest
   Net_Dist(P,N_i)+Net_Dist(N_i,Q)
```

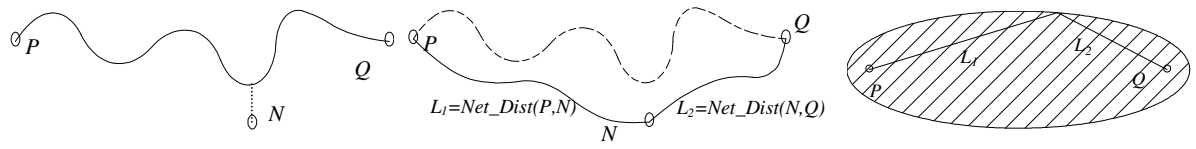

Figure 4: The In-route-find-the-net-nearest service

This algorithm is based on the same principle with the Find_the_Net_Nearest algorithm, that is, the network distance between two points is greater or equal to their Euclidean distance. As such, the algorithm initially produces the optimal route R between P and Q by performing request to the .NET RJS client, while subsequently, uses R as a query object on the `SP_Nearest` operator in order to retrieve the Euclidean

nearest neighbor *N* among the records contained in the Landmarks table (Lines 1-2 in pseudo-code and Fig.4(a)). Then it also performs requests to the .NET RJS client on order to calculate the network distance between *P*, *N* and *Q* (Line 3 in pseudo-code and Fig.4(b)), while afterwards uses their sum in order to retrieve candidate objects with total distance from both *P* and *Q* less than it (Line 4 in pseudo-code and Fig.4(c)). It is also important to note that these objects are contained inside an elliptical region with *P* and *Q* as foci. In its last step, the algorithm calculates the network distances between *P*, *Q* and all candidate points N_i, finally, reporting as answer the one that has the smallest sum of network distances.

4. Conclusions

In this paper we present the next generation of location based services, sketch up the respective algorithms and provide 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. Our implementation is based on the Microsoft.NET and Microsoft SQL Server platforms, employing two additional components, namely, the MapInfo SpatialWare and MapInfo Routing J Server in order to efficiently support spatial and graph-based (i.e., road network-based) operations. Among others, the software developed for the Next Generation Location Based Services, includes 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 (Papadias *et al.*, 2003, Li *et al.*, 2005, Kolahdouzan and Shahabi, 2004, Sankaranarayanan *et al.*, 2005).

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 applications (while the *Advanced Routing* and *In-route-find-the-nearest* are much more meaningful in the framework of route planning, rather than the GSM-based LBS). Moreover, since our implementation is based on highly scalable platforms (SQL Server along with the SpatialWare and RJS) it can support numerous concurrent requests. Therefore, our plan is to employ it in the framework of a web-based application providing users with advanced functionality regarding their travelling needs.

References

Dijkstra, E. W., 1959: A note on two problems in connexion with graphs, *Numerische Mathematik*, vol 1, pp. 269-271, 1959.

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

Guttman, A., 1984: *R-Trees: a dynamic index structure for spatial searching*. Proceedings of ACM SIGMOD Conference, 1984.

Kolahdouzan, M., Shahabi, C.: *Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases*. Proceedings of VLDB Conference, 2004

Hart, P. E., Nilsson, N. J., Raphael, B., 1968: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics SSC4* (2): pp. 100–107.

Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G. and Teng, S.-H., 2005: *On Trip Planning Queries in Spatial Databases*. Proceedings of SSTD, 2005

MapInfo Corporation, 2007a: *MapInfo SpatialWare*, Available at http://extranet.mapinfo.com/products/Overview.cfm?productid=1141, (accessed 18 May 2007)

MapInfo Corporation, 2007b: *MapInfo Routing J Server*, Available at http://extranet.mapinfo.com/products/Overview.cfm?productid=1144, (accessed 18 May 2007)

Microsoft Corporation, 2007a: *Microsoft Visual Studio .NET*, Available at http:// msdn.microsoft.com/vstudio/, accessed 18 May 2007.

Microsoft Corporation, 2007b: *Microsoft SQL Server*, Available at http://www.microsoft.com/sql/, accessed 18 May 2007.

Open Geospatial Consortium, 2007: *OpenGIS® Location Services (OpenLS): Core Services*. Available at http://www.opengis.org, accessed 18 May 2007

Papadias, D., Zhang, J., Mamoulis, N., and Tao, Y., 2003: *Query Processing in Spatial Network Databases*. Proceedings of VLDB Conference, 2003.

Sankaranarayanan, J., Alborzi, H., and Samet, H., 2005, *Efficient Query Processing on Spatial Networks*. Proceedings of ACM-GIS Workshop, 2005.

Telenavis S.A., 2007: http://www.telenavis.gr/, accessed 18 May 2007