Design and Implementation of Route Recording for Mobile Services

Agnė Brilingaitė Nora Zokaitė
agne@cs.auc.dk nora@cs.auc.dk
Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, 9220 Aalborg Øst, Denmark

June 6, 2003

Abstract

Several developments combine to enable a new class of on-line mobile services. Put simply, the performance/price ratio for consumer electronics is improving quickly, wireless communication technologies are becoming more and more widespread, and geo-positioning is becoming practical.

In comparison to desktop computing, mobile services use small screens, and they typically lack a keyboard. Further, they are often used in situation where the user's main focus of attention is not the service. For these reasons, it is important that the user receives the "right" information at the right time and with as little interaction as possible. These qualities may be obtained by making mobile services aware of the user's context. A mobile user's route towards her destination is an important aspect of her context.

This paper considers users traveling in road networks, and it presents a software component that builds routes for individual users based on traced coordinates. The paper presents the architecture and functionality of the route component. A database model that captures routes is described. The paper proposes the algorithms that solve the problems of route detection. These problems include travel position identification, formation of route parts, and route construction according to the structure of the road network. The algorithms, implemented using Java and Oracle's PL/SQL and Oracle Spatial, are also presented.

1 Introduction

Currently, wireless communication technologies are becoming more and more wide spread. Technical characteristics of mobile devices are improving constantly, but at the same time the price of mobile devices is becoming reasonable for the consumer. These facts cause result in a growing number of mobile users. The development and proliferation of mobile devices enable new applications in wireless communication. A user that has a mobile device is able to get route guidance, tourist services, or services based on her current location. However, in comparison to desktop computing, mobile services use small screens, and they typically lack a keyboard. Further, they are often used in situation where the user's main focus of attention is not the service. If a user is traveling on the road then driving is the first priority. If a user goes by foot in town then she has to be aware of her surroundings in order to not raise danger for herself and other people. For these reasons, it is important that the user receives only the relevant information and with as little interaction as possible. These qualities may be obtained by making mobile services aware of the user's context. A user's location is one possible context. Location context-awareness is important for the group of mobile service users that travel in a road network.

In this paper, we focus on a specific location context—routes. A route is a motion plan of a user traveling in a road network. Most of the time, people are traveling on routes they know in advance, and these routes can be predicted. When a location-based context-aware system knows where and when the mobile users are traveling, it can provide them with relevant information. For example, as that system knows that a user travels on her usual route from home to work every morning, it can caution her about traffic jams, streets closed for renovation, or speed changes in some parts of the streets. This is important for the user—she can re-plan her route or be aware about possible delays in advance. Sending alert messages is not the only use of routes. A route can be used when a user is interested in specific points of interest. The system that knows the user's route can give her information about objects that are based on her position on her route. For example, if the user wants to have a cup of coffee, she asks for the nearest cafeterias, and the system provides her with the cafeterias that are nearest on her route.

In this work, we design a software component that builds routes for individual users based on traces of coordinates. We assume that the mobile service user posses a device capable of providing the service with geo-positioning information. We propose a system architecture and specific functionality for such a component. We distinguish among four main functions that make the component usable. They include route recording, route renewal, collection of usage, and obtaining of routes. The client side is active in supporting the functionality. The client device performs information filtering and prepares information for sending it to the server. We propose a database model adjusted for route recording. The purpose of the database is to store the road network representation and information about the mobile service users and their routes. We use a linear-referencing framework to capture roads and routes. In our model, the user has her destination objects. Thus each route has a start and an end destination object. But the route is a sequence of road parts. We store geographical information about destination objects and base points of the road network in our model. Each usage of a route is also stored. In this paper, we present techniques that capture the routes of mobile service users. While detecting a route, we consider the user's geographical position and the road network structure. We implemented these techniques using Java and Oracle's PL/SQL and Oracle Spatial.

The paper is structured as follows. The next section presents a case study and requirements. An overview of related work is given in Section 3. The system architecture is described in Section 4. Section 5 analyzes system functionality. The definitions of the concepts and database model are given in Section 6.1. The algorithms are presented in Section 7. Section 8 presents practical part of our work. Section 9 summarizes the work and gives guidelines for future work.

2 Case Study and Requirements

In this section, we consider the overall design of a route recording component. At the beginning of this section we give an overview of the usage scenario, then discuss the requirements for the design.

The problem we solve in this paper is designing and implementing a route recording component. To begin with, we provide an example to give an intuition about the kinds of real-world situations in which the route recording component comes into play.

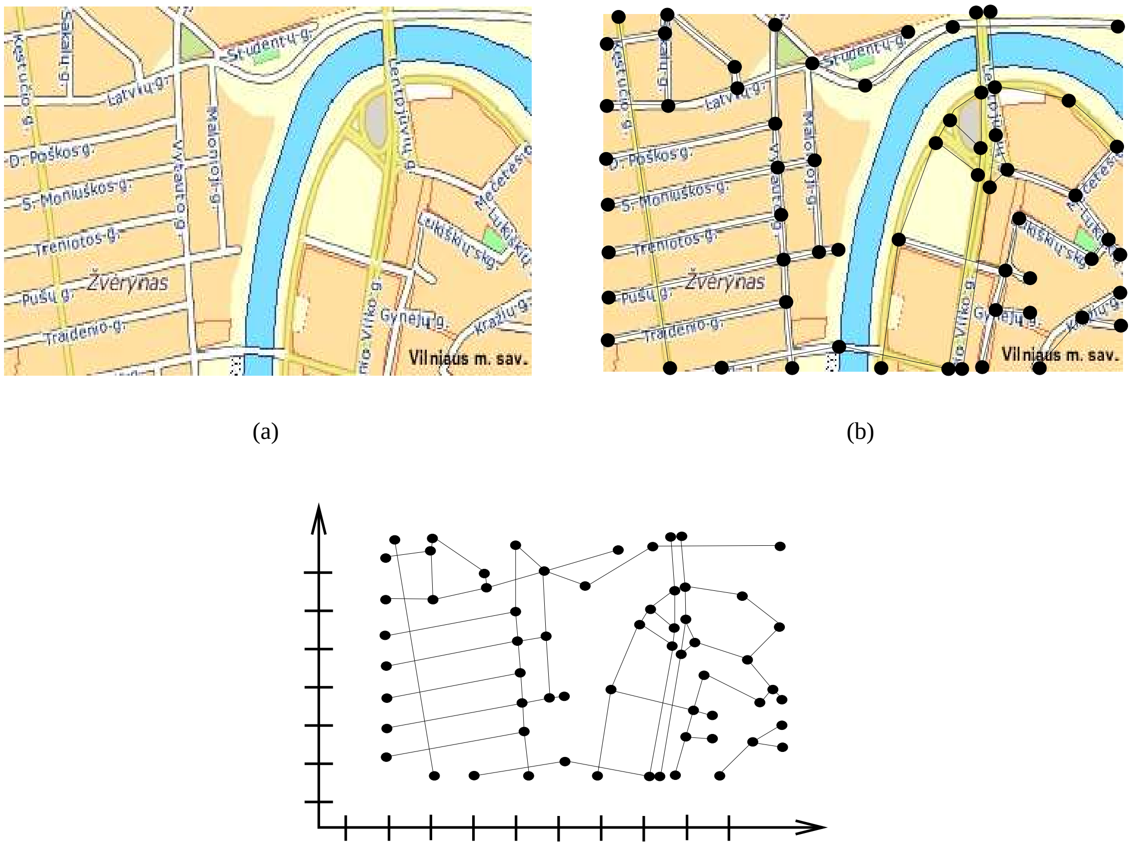
Example 2.1. Suppose we have a road network with car drivers traveling along it. Let us pick one of these drivers and analyze the situation where this driver, say John, is traveling from his home to the university. This is his usual route. It is important for John, as for any other employee, to be at work on time, so he wants to use a mobile service that informs him about road conditions, traffic jams ahead on the route, etc. In order to receive such information, John has to inform the service provider about exactly what route he is traveling on. Therefore John, with the help of a geo-positioning device (a GPS receiver) and a mobile phone records his travel coordinates all the way from the start of the route till the end of it in a log file, and he sends this information to the service provider. The mobile service provider in turn stores the information from the log file as "Johns route: Home–University." Now, every time John is traveling along this route, he is informed about road conditions and other relevant information sustaining to the route.

Example 2.1 illustrates the situation where we observe the exchange of information between a mobile service user and a service provider. The information that is involved in this exchange includes data about a real-world road network, mobile service users, and service users' routes. In our work, by integrating this kind of information into our system and enabling the functionality of that system, we create a so-called route recording component. Stated briefly, this is a software component that builds routes for individual mobile service users based on traces of their coordinates.

We model a route recording architecture that includes these components: a user's mobile device and a server.

The main task of the system is to record the routes of the mobile service users. In order to record the route of a user, the system has to be provided with information about where the route starts, where it leads to, and where it ends. In our paper, we analyze the situation where the information about the route of the user is obtained with the help of a GPS receiver. We assume that the mobile device collects this GPS information. Notice that "GPS sentences" are relatively long. Therefore, the longer the route, the bigger the file with the GPS information is. We allow different strategies for further maintenance of the GPS information, depending on the technical abilities of the mobile devices. One is to send this information to the server "online." Another strategy is to use buffering and send it in chunks of some defined size, or to send the whole file with GPS information. The server in turn maintains all the information received from all users.

In order to record and store the routes of the users on the server, we need to come up with a representation of the real-world road network. See Figure 1.



(a) (b) (c)

Figure 1: Modeling the road network (map from [8])

In our paper, the representation of a real-world road network is a projection of that network onto two-dimensional (2D) space. The way we do that is by sampling a set of *base-points* that hold the geo-information in x and y coordinates about the real-world road network, and by connecting these points into polylines. See Figures 1(b) and 1(c). The result of the projection is a network of approximated roads. In our work, we use a linear-referencing framework [1, 9, 13] in order not to loose information such as real road distances. This framework is also used for capturing routes as parts of the road network representation stored in the database. Each route is defined as a directed path leading from one destination object to another one. These destination objects have geo-information.

3 Related Work

We are not aware of any previous work on components that generate meanings for routes from the GPS information. But our work is related to a few research directions in mobile services. We also use already invented techniques in our work. Research challenges in mobile services are outlined in [11]. The challenges are grouped according to themes—data representation, indexing, querying, data modeling. Research issues in location management are presented in [20].

The building of our component involves the modeling of road network data. Our data model uses a linear referencing framework ([1, 4, 5, 7, 13, 16]. In particular we use linear referencing not only for capturing a road network, but also for capturing routes. A data model that integrates representations of geo-referenced content and transportation infrastructures is described in [9]. Our data model uses part of their model of road networks, but we supplement it in order to capture routes. Our data model integrates with any linear referencing model for road network ([1, 4, 5, 7, 9]. Linear referencing is not the only way to model a road network. Vazirgiannis et al. [18] model a road network as a directed graph. Their routes are sequences of edges.

Our paper uses already invented techniques that we do not develop further. We use the shortest path algorithm to fill the gaps in information when we construct routes. This part is related to works that consider shortest paths in graphs. Barrett et al. [2] study a generalized Dijkstra's algorithm for shortest paths in graphs on large transportation networks to do route planning. Vazirgiannis et al. [18] compute the lowest cost path.

To create a route, we do map matching of GPS coordinates to the road network. Bernstein and Kornhauser [3] explore map matching algorithms that can be used to reconcile inaccurate locational data with an inaccurate map. They analyze algorithms such as "Point-to-point," "Point-to-curve," and "Curve-to-curve." In contrast, we analyze more specialized situations, and we map match a GPS point to a position on a polyline. To do map matching, we also use the geographic location of the roads together with the structure of the road network, i.e., connections of the polylines. We focus on map matching in a specific data model, and we explore its properties, but the techniques proposed in [3] can be integrated into our work. Our map matching includes searching for nearest neighbors. We define the search range according to the allowed imprecision and the candidate polylines are within this range. That is related to the work of Roussopoulos et al. [15], in which they consider minimum and maximum distances from the query object while searching the nearest neighbors. We also choose the polyline according to how the previous GPS point was map matched. The nearest neighbors for the previous positions of the moving object are considered in [17]. In comparison to [15] and [17], we use nearest neighbor search for other purposes than they do. We search for nearest neighbors to define the movement of a user in the road network, as our users do not travel freely in 2-dimensional space. We construct a sequence of connected polyline elements, not a set of nearest objects for every step.

Our route component is a part of a context-aware system. That makes available the users' routes as context. We propose an architecture for a specific context-aware system and have requirements for concrete functionality, instead of focusing on a more abstract level of description ([10, 12]). Hohl et al. [10] examine the context concept, present a classification of context data, and give requirements of a mobile service platform for a context system. Data management issues and solutions in location-dependent information services are discussed in [12].

4 System Architecture

In this section we describe the architecture of the context-aware system for mobile services based on routes. The requirements for the parts of the system are presented.

4.1 Basic Requirements

We assume that our user has some mobile device that has the necessary technical properties to send information to the server, to get information from the server, and to store the information about the user and her routes. The device should include a GPS receiver and it also should be able to analyze the information that comes from the GPS receiver.

The server of the system must be able to take care of all the requests from the user. It has to analyze the information sent by the device and to record it into the database. Everything about a route (parts of the road network), its usage and user's personal information is stored on the server. This requirement is included in order to avoid information loss. This means that the user can change the device and subsequently obtain her routes from the server. The architecture of the system is presented in Figure 2.

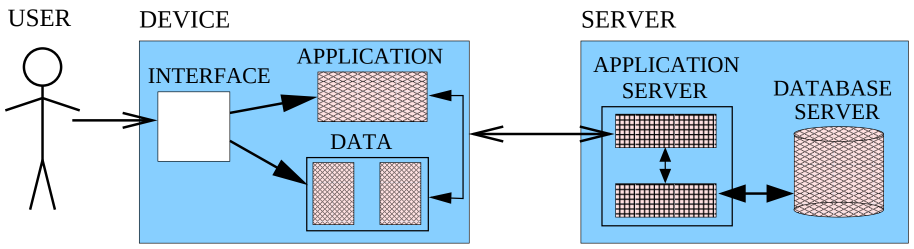
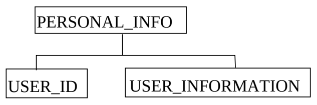


Figure 2: System architecture

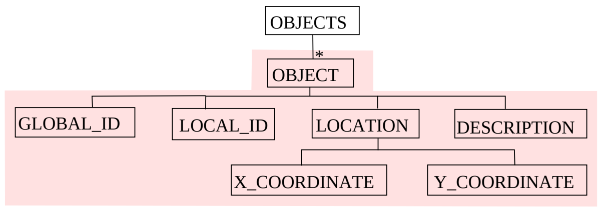
4.2 Client Side

The user uses the system functionality through the interface on the device. The interface interacts with the application that enables functionality of the system on the client side. The application uses stored data and records new data if it is necessary. The user can see what data is stored on the device using the interface. She does not need to know the technical details about the running application or stored data.

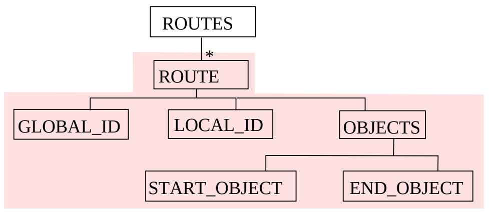
There are four data blocks that are stored on the device. XML ([19]) notation is used in Figure 3 to describe them. The first block contains user's personal information (Figure 3(a)). It contains the user's identification number and user's information. The identification number is a global parameter and it is unique for each system user. It is used on the server to record the route of a particular user. User's information includes the information the user wants the system to store. It can be a name, nickname and etc. The second data block contains the information about the user's destination objects (Figure 3(b)). Each object has a global identification number, local identification number, location information, and description. The global identification number is stored in order to have the same information on both client and server sides and to be able to make updates on both sides. Local identification makes route descriptions easy, because global parameters can be complex. Location of the object is described using geo-information. It is used when the application on the device analyzes the GPS information and detects if the destination object has already been defined. The description is needed for the user to have clearly defined destination objects, i.e. "home" or "work", because the user can forget what is his route from, let us say, the object 3 to the object 5. The third block of data is related to the route information (Figure 3(c)). Here we have global/local identification numbers, and objects. Objects define from which destination object to which one a particular route is. The objects are defined using local identification numbers from the second data block. The fourth block of data is so-called "log" data (Figure 3(d)). It includes the usage times of every route. The time is approximated to weekdays, hours, and quarters. These time intervals are individual for each route. The device stores the information how frequently the user traveled along the route on each approximated time.



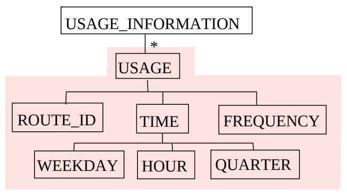
(a) User information



(b) Objects



(c) Routes



(d) Usage information

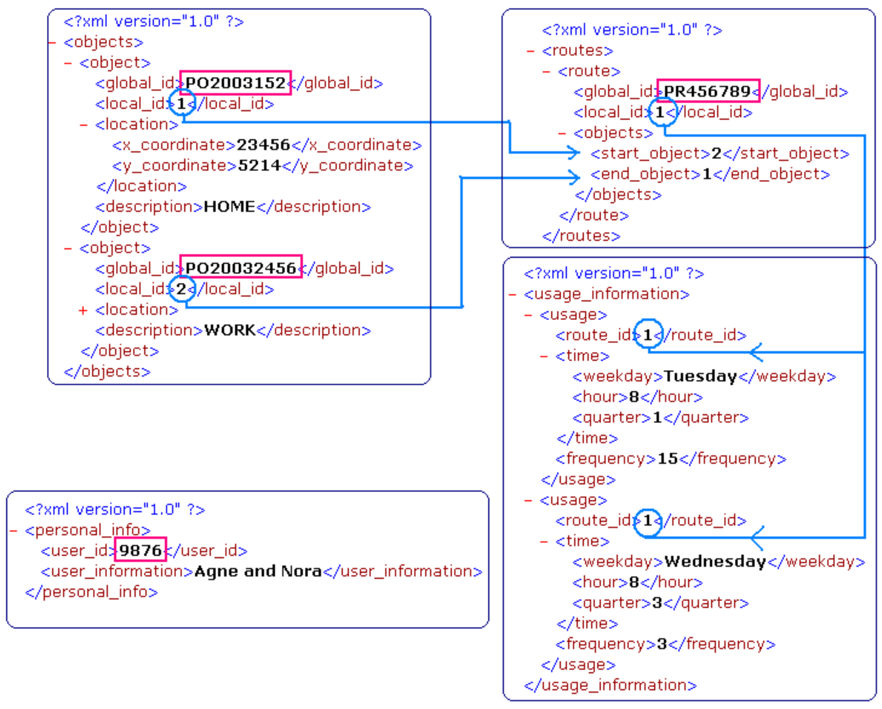
Figure 3: Data on the client side

Data on the device is stored in .xml or .txt files. The application is a Java or C program that can be run on the mobile device. The user inputs the information, i.e. personal information, names for destination objects, when she is asked by the program. As the device stores the information about the routes and the user it can predict what route is going to be used by the user at some particular time and there is no need to connect to the server. The application initiates interaction between the device and the server when it is necessary to unify the information on both sides.

4.3 Server Side

On server side (see Server in Figure 2) we have an application server and a database server. The application server interacts with the database server. The application server gets the requests from the clients. The requests can be different by the functionality. Thus the application server has to take care of all requests and at the same time to be able to interact with the database. There is a need for multi-client system based on multi-thread functioning. The application server analyzes the information from the client and makes calculations interacting with the database. The application server initiates the database updates when they are necessary, i.e. a new route is recorded or a route is used. As mentioned in the previous sections, the system uses global parameters that are stored on the client side and on the server side. The application server uses these parameters to distinguish among users and their routes.

Example 4.1. Figure 4 presents an example of data stored on the client side device. The data is stored in four XML files. Their structure follows the schema shown in Figure 3. The rectangles indicate data that is the same on the client and on the server sides, namely identification numbers for the user, destination objects, and routes. The circles indicate local data that is used in more than one data block. We have a user 9876. She has two destination objects. The user's first destination object "HOME" has global parameter PO2003152 and the second one, "WORK," has parameter PO20032456. The user has one route from the second destination object to the first destination object. The route is identified globally by number PR456789. This route was used during two time intervals—15 times on Tuesday approximately at 8:00-8:15 (the 1st quarter) and three times on Wednesday approximately at 8:30-8:45 (the 3rd quarter).



```
<?xml version="1.0" ?>
- <objects>
  - <object>
      <global_id>PO2003152</global_id>
      <local_id>1</local_id>
    - <location>
        <x_coordinate>23456</x_coordinate>
        <y_coordinate>5214</y_coordinate>
      </location>
      <description>HOME</description>
    </object>
  - <object>
      <global_id>PO20032456</global_id>
      <local_id>2</local_id>
    + <location>
      <description>WORK</description>
    </object>
  </objects>
```

```
<?xml version="1.0" ?>
- <routes>
  - <route>
      <global_id>PR456789</global_id>
      <local_id>1</local_id>
    - <objects>
        <start_object>2</start_object>
        <end_object>1</end_object>
      </objects>
    </route>
  </routes>
```

```
<?xml version="1.0" ?>
- <personal_info>
    <user_id>9876</user_id>
    <user_information>Agne and Nora</user_information>
  </personal_info>
```

```
<?xml version="1.0" ?>
- <usage_information>
  - <usage>
      <route_id>1</route_id>
    - <time>
        <weekday>Tuesday</weekday>
        <hour>8</hour>
        <quarter>1</quarter>
      </time>
      <frequency>15</frequency>
    </usage>
  - <usage>
      <route_id>1</route_id>
    - <time>
        <weekday>Wednesday</weekday>
        <hour>8</hour>
        <quarter>3</quarter>
      </time>
      <frequency>3</frequency>
    </usage>
  </usage_information>
```

Figure 4: Example of data on the client side

5 Functionality

This section concerns the functionality of the system. We distinguish among four main functions that support a route based context-aware system: route recording, collection of usage, renewal of routes, and obtaining of routes from the device. We present first the details of the route recording, as it is the main function in the system.

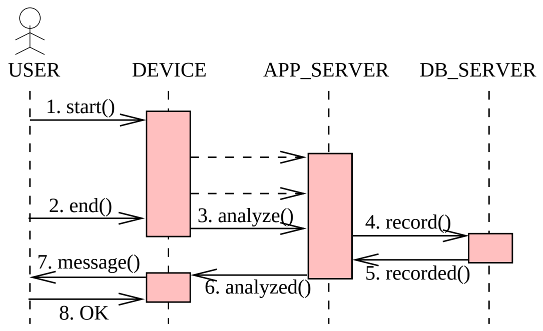
5.1 Route Recording

Route recording is the most complex function in our system. This section presents the description of the function. The interaction between the client side and the server side while supporting this function is explained. The algorithms for route recording on the client and server sides are presented in diagrams.

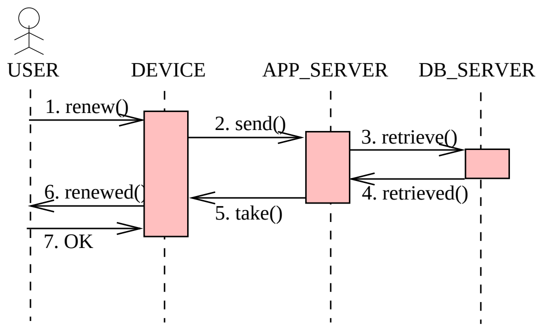
5.1.1 Description

The function sends buffer(s) of GPS data to the server, which then analyzes the data and records a route. *Arguments:* user identification number, information about destination objects, date, time, a set of the coordinates from the GPS receiver. *Result:* the route identification number and identified parts of the road network that belong to the route. If destination objects or the user herself are new, the function registers them/her in the system.

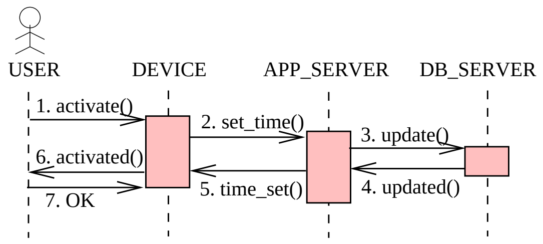
We assume that the user controls the process of her route recording (see Figure 5(a)). She activates the service (Step 1) and deactivates it (Step 2). During this time period, the device is active. The device filters and records the user location information that comes from the GPS receiver. It prepares the stream for the server by adding the parameters about the user and her destination objects to the GPS set. The information can be sent to the server as one buffer, or it can be sent from time to time when the buffer is large enough (dashed lines between Device and Application Server). The choice of strategy depends on the route length, the technical characteristics of the mobile device, and the connection quality. When the information or the last part of it is sent to the application server (Step 3), the application server can make calculations communicating with the database server—it generates a unique number for the route, defines the usage time and does map matching and route construction. The application prepares the sequence of updates and sends it to the database (Step 4). The database server informs the application server whether the update is successful or not (Step 5), and the application server can send the data stream to the mobile device of the user (Step 6). The device records the data from the stream and fixes the first usage time. After that, the mobile device informs the user about the end of the route recording (Step 7).



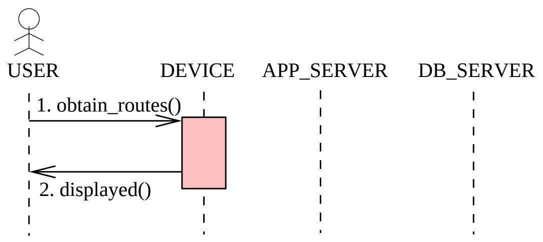
(a) Route recording



(b) Route renewal



(c) Collection of usage



(d) Obtaining routes

Figure 5: Sequence diagrams for client functions

The data stream that is sent to the server by the device consists of three parts: user information, object information, and standard information. The data stream can have different formats. The format depends on which data is already stored and on which data is new.

User information. If the user is already registered, this data block includes her identification number. As the route recording function also includes user registration, we can have additional data, namely a user description for a new user, in the data stream. Thus we have **[userId]** or **[undefined: description]** in this block.

Object information. This data block consists of information about the route destination objects if these objects are new. The destination objects can have been already used to define a start or an end for other routes. This means that the server knows the information about them, so that it, according to the GPS coordinates, can identify the objects itself. If the destination objects are new, the data stream includes their descriptions. If both objects are known, this data block is empty **[,]**. If we have one undefined object we have a data block that contains start description **[undefined:description,]** or end description **[,undefined:description]**. If start and end objects are undefined the block has descriptions for both of them: **[undefined:description, undefined: description]**.

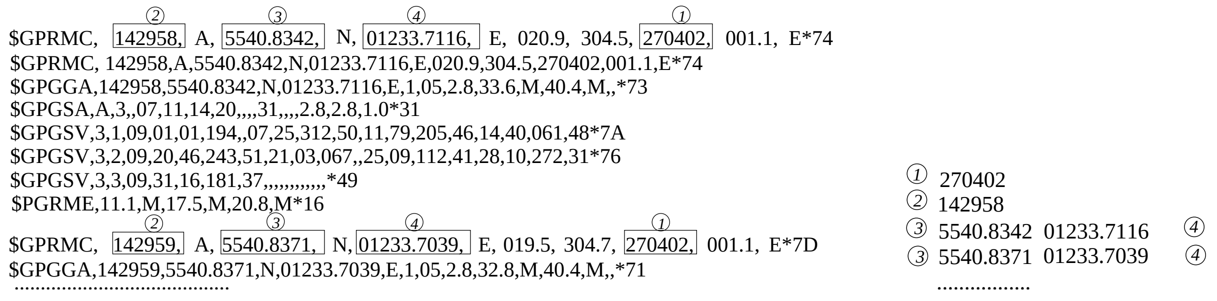
Standard information. Date, time, and GPS information are always included in the stream for the server. We call it standard information, and this block includes these three elements: **[date, time, GPS]**.

The server also sends a data stream to the client device. The server always returns the identification number for a newly recorded route. If any of the parameters are undefined, the device assumes that the data stream from the server will include the missing information. The server generates identification numbers for a user and the user's destination objects. These numbers are returned to the client. Thus, the format for the data stream from the server is **[userId, startObjectId, endObjectId, routeId]**, where **routeId** is the only parameter that is always included. The device gets the data stream from the server, analyzes it, and records its data.

5.1.2 Route Recording on the Client Side

Part of the route recording task is done on the client side. The client device prepares the data stream and sends it to the server, as it is described in the previous section. The first and second blocks of the data stream are constructed using data stored locally (see Section 4) on the device. The third data block is constructed analyzing the information from the GPS receiver. GPS receivers transmit NMEA sentences [6, 14]. These sentences include information about the position of the object, but also information about the GPS satellites. The device analyzes this data to retrieve the necessary information—date, time, and a set of GPS coordinates. The set of coordinates should include at least two coordinates to fix the start and the end position of the route.

Example 5.1. Figure 6(a) shows data from a GPS receiver. The block of GPS NMEA sentences—from $GPRMC to $PGRME—is formed for the position of an object every second. In our system, we use date (denoted by *1*), time (denoted by *2*), and a set of the coordinates (denoted by *3* and *4*). In each block of GPS sentences, the date is repeated, but we are interested just in the start time and date of the route usage. So the application on the device parses the stream of GPS sentences and forms a sequence (see Figure 6(b)) of data that includes only the necessary information with no repetitions.



```
                 ②           ③              ④                          ①
$GPRMC, 142958, A, 5540.8342, N, 01233.7116, E, 020.9, 304.5, 270402, 001.1, E*74
$GPRMC, 142958,A,5540.8342,N,01233.7116,E,020.9,304.5,270402,001.1,E*74
$GPGGA,142958,5540.8342,N,01233.7116,E,1,05,2.8,33.6,M,40.4,M,,*73
$GPGSA,A,3,,07,11,14,20,,,,31,,,,2.8,2.8,1.0*31
$GPGSV,3,1,09,01,01,194,,07,25,312,50,11,79,205,46,14,40,061,48*7A
$GPGSV,3,2,09,20,46,243,51,21,03,067,,25,09,112,41,28,10,272,31*76
$GPGSV,3,3,09,31,16,181,37,,,,,,,,,,,,*49
$PGRME,11.1,M,17.5,M,20.8,M*16
                 ②           ③              ④                          ①
$GPRMC, 142959, A, 5540.8371, N, 01233.7039, E, 019.5, 304.7, 270402, 001.1, E*7D
$GPGGA,142959,5540.8371,N,01233.7039,E,1,05,2.8,32.8,M,40.4,M,,*71
.........................................
```

```
① 270402
② 142958
③ 5540.8342  01233.7116  ④
③ 5540.8371  01233.7039  ④
.................
```

(a) Information from GPS (b) Filtered information

Figure 6: Filtering of GPS information

The order of the steps for route recording on the client device is presented in Figure 7. When the user activates her route recording process, the device starts getting GPS information from the GPS receiver. The device gets the first GPS message (1) and adds position information (2) to the data stream for the server. The first GPS block describes the start of the route; thus, the position information is noted (3) in order to use it later. Date and time are also retrieved from the block (4) to add (7) them to the data stream. All other transmitted GPS blocks (6) are analyzed to retrieve only the position information, which is added (5) to the data stream. After this, each block is checked (8) to determine if it is already the last block or not. If it is not the last block, the device gets another one. If the user has deactivated the service, then the block is the last, and the end of the route is noted (9) for further analysis. This part of the algorithm constructs the standard information block for the data stream that is sent to the server.

When no more GPS coordinates are transmitted, the device checks (10) if the user is already registered in the system. If the user is new, the device asks (11) the user to input her description. The device records

(12) the description locally, sets (14) the user as undefined in the data stream and adds (13) the description to the data stream. If the user is already registered, her identification number is added (15) to the data stream.

To build the block about the destination objects in the data stream, the device uses the noted location information about the start and the end of the route. If the start and end objects are undefined (17, 18) or the user is new, the device asks (19) the user to input descriptions for her destination objects. These descriptions are recorded (20) locally. But the objects are set (25) as undefined in the data stream, and the descriptions are added (24) to the data stream. If only one object is undefined, the same steps are done for one object—for the start (17, 18, 23, 22, 27, 28) or for the end (17, 16, 21, 26, 29, 30). If both objects are defined (17, 16), the block about destination objects in the data stream remains empty. When all three data blocks have been constructed, the route is recorded (31) locally using the local parameters and leaving the global parameters undefined. After that, the stream is sent (32) to the server.

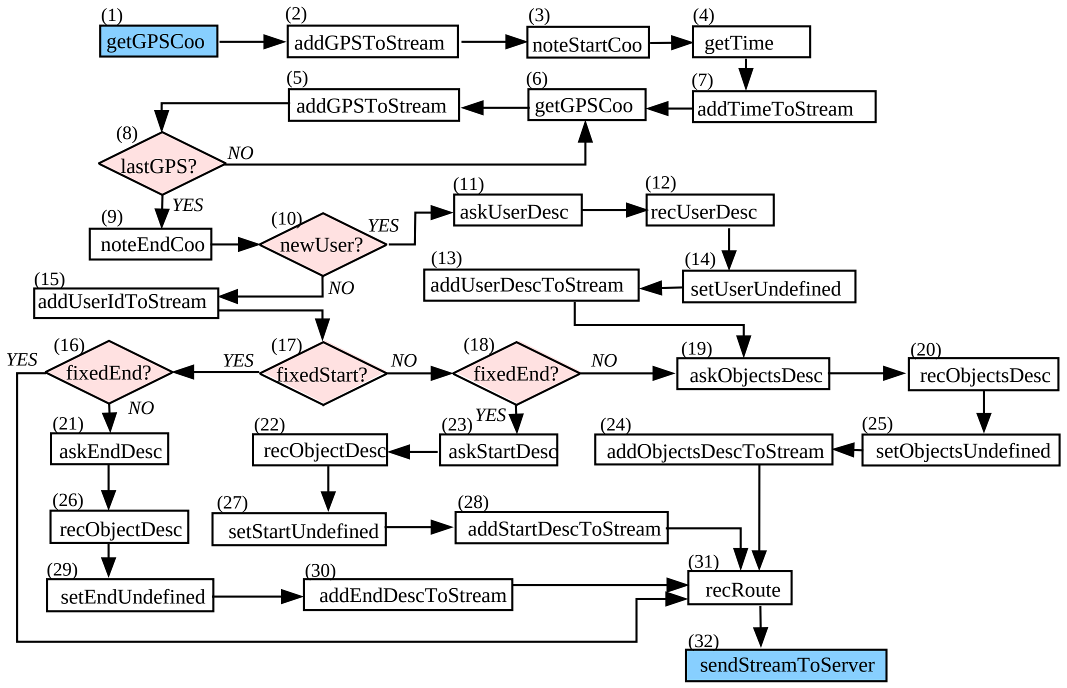


Figure 7: Route recording on the device

5.1.3 Route Recording on the Server Side

The main route recording is performed on the server side. As described in the previous sections, the server gets the data stream from the client and analyses it to define a route. The device only prepares the information that is necessary to define a route and to fix its first usage time. The server records the data concerning the new route. This data always includes route parts on the road and a route identification number that is generated. If the data stream from the device includes descriptions of destination objects or of the user herself, the new objects/user are registered. According to the information from the device, the server forms the data stream (see the format in Section 5.1.1).

The route recording process on the server is presented in Figure 8. The server gets (1) the stream from the device. The server checks (2) if the user is new. If so, the server gets (7) her description from the device data stream, generates (6) an identification number, records (11) the user's information, and adds (12) her identification number to the data stream for the device. Having identified a user, the server analyses the information about the destination objects. If the user is not new, the server checks if any destination objects come as undefined. If both destination objects are undefined (3, 8), i.e., the server gets their descriptions, or the user is new, the server takes (17) this information, generates (16) their identification numbers, records

(20) new objects, and adds (21) the identification numbers to the data stream. If only one object is undefined, the steps are done for one object. If the start is undefined (3, 8) then at the beginning, data about it is prepared (13, 14) and recorded (18, 23). After that, the end object is identified (22) using knowledge about the user's objects. If the start object is defined, but the end object is undefined (3, 4) according to the device stream, then the start is identified (5) at the beginning. After that, data about the end is prepared (10, 15) and recorded (19, 24). If both objects are defined (3, 4), they are identified (9) using stored data.

After the two first parts of the device stream are analyzed, the server analyses the third one that includes the standard data. The server detects (25) the route from the GPS information, generates (27) the identification number for the route, adds (26) this number to the data stream for the device, records (28) the route in the database, and records (29) the first usage time of the route. The constructed data stream is sent (30) to the device to end the route recording process.

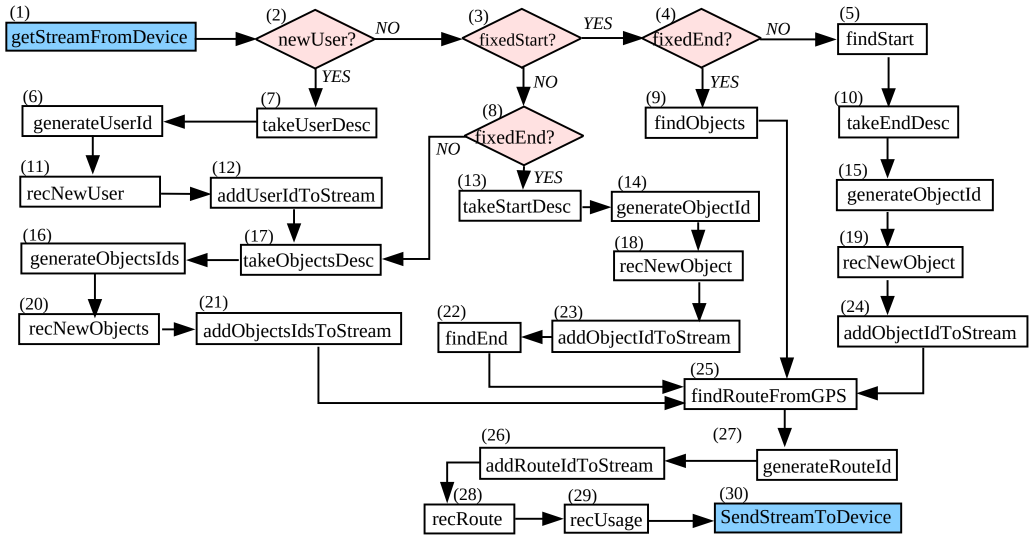


Figure 8: Route recording on the server

5.2 Collection of Usage

The collection of usage function enables the system to provide context-aware services. Its purpose is to store route statistics. *Arguments:* route identification number, usage date and time. *Result:* a fixed route usage time.

We assume that each user has more than one route and that the user uses some routes more than one time. The user controls the process of usage time recording (see Figure 5(c)). When the user starts traveling on a particular, already stored route, she uses the device to send information about the current date and time, when she activates the service (1). This means that the device sends a message (2) to the application server with information about the route, date, and time. The application server does the necessary calculations to approximate the time and it initiates an update of the database (3). Steps 4–7 are included to send messages about the result of the process. The database server informs the application server whether or not the usage date is fixed successfully, and the application server sends this message to the device the message. The device also records the usage time after it gets the message about a successfully fixed time on the server.

5.3 Renewal of Routes

The route renewal function returns a list of user routes from the server to the device. It may happen that the user looses the mobile device or she gets a new one. This function restores lost information on the client side. *Arguments:* user identification number. *Result:* the set of the user's routes, the set of the destination objects, and summarized usage. It may happen that a user has a lot of routes and that some of them were used once or long ago. Then the solution is to return “usable” routes and information about them.

The user can get her routes (see Figure 5(b)) from the server by just using the device (1). She only needs to input her identification number. When the device gets the number, it initiates the message (2) for the application server. The application server communicates with the database server to retrieve the information (3–4) about the routes of this particular user. Everything about routes together with destination objects and their summarized usage is prepared for sending to the device. The user description is also included. Information buffer(s) of a special format are constructed on the application server and are sent to the device (5). The device records data from the buffer(s) and informs the user about the result of the process.

5.4 Obtaining Routes

This function uses the history about the usage of routes. *Arguments:* location, date, time. *Result:* n routes ordered by their probability.

When the user wants her route usage recorded, she needs to select the route the usage applies to. The device can provide a service that guesses what route will be used. Then the user does not need to browse through all her routes. The user can ask for the routes (see Figure 5(d)) (1). The device calculates the probabilities for each route. Date, time, and location are used to make compute probabilities. The main parameter is location. If we have several routes starting at the same destination point, then time and date are also considered. The device presents the list (2) of the routes ordered by these probabilities.

5.5 Other Functions

We have present the four main functions for the route component. But there are other functions, i.e., route deletion and route re-recording, that complete the functionality.

Route Deletion The user may want to delete a route if she thinks she will never use it again. It is enough to send the route identification number of the route to be deleted from the database. The route identification number, route parts and usage times are to be deleted from the server and from the device. This may also cause the deletion of destination objects if they are not used in other routes. But they can be left, assuming that the user perhaps will use them in other routes. The strategy on both client and server sides must be the same.

Route Re-recording It may occur that the user changes some destination objects, but calls them by the same names. For example, the user moves to another apartment or changes her job. Every route related to these objects is to be re-recorded. The route parts and destination object(s) change, but the usage time intervals can be the same, because the usage depends on private habits and on certain circumstances. For example, the user has a longer route from home to the new work than she had previously, but the work starts later and the user starts traveling on this route on the same time. It follows from this that the usage times should not be deleted at once, but marked as “old” in order to track the new route. If the usage is not the same after some time, the old history is deleted.

6 Data Model

In this section we present the formal foundation for the information used by the route recording component. We also describe the database model that captures routes.

6.1 Data Structure

In our work the route recording component deals with the information about the real-world road network and mobile service users. Let us begin with describing the representation of the data about the real-world road network. As it was mentioned in the Section 2, we project that data into 2D space. We say that the result of such a projection is a set of points connected into polylines.

The smallest particles of the road network representation are points. The points relate the model of the road network with the real-world road network by holding the geo-information about the real-world road network. We assume that for any selected position of the real-world road network we can obtain the geo-measurements of that position and refer this information to a point in the 2D model. These points are called “base points.”

Definition 6.1. (Base Points) Let $B \subset \mathbb{R}^2$ be a finite set of *base points* (x, y) in 2D space.

When projecting the real-world road network into 2D space we select a set of base points and connect these points with line segments. To select such a set on the real-world road network we take into account characteristics of that road network, like sinuosity of roads (see Figure 9).

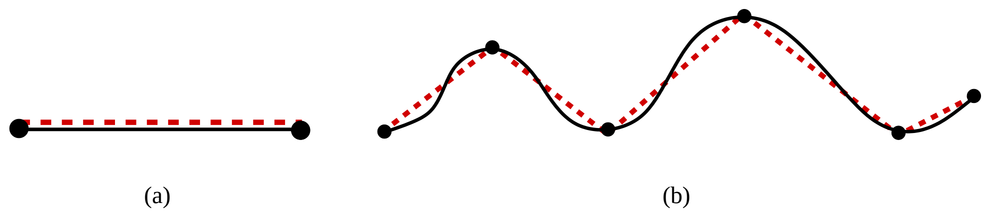


Figure 9: Example of road approximation

As a result we obtain a road network approximated with line segments. Notice, that for the road in Figure 9(a) it is enough to select two base points in order to obtain a good approximation of the road when connecting these two base points into a line segment, while it requires more base points to approximate the road in Figure 9(b). Therefore, the bigger the set of base points the more precise the representation of the real-world road network is. Also observe that the roads in Figure 9 are shown as a sequences of base points connected into line segments. In other words, the roads are represented as *polylines*:

Definition 6.2. (Polyline) Let $PL \subset B^{*2} = \{(b_1, ..., b_N) | b_i \in B \wedge N \geq 2\}$ be a finite set of *polylines*. Each polyline $pl = (b_1, b_2, \ldots, b_N)$ is a N-tuple, where

1) $(b_1, b_2, \ldots, b_N) \in PL$ is the sequence of base points that form the polyline;

2) $b_1 \in B$, $b_N \in B$ are, respectively, start and end base points of the polyline.

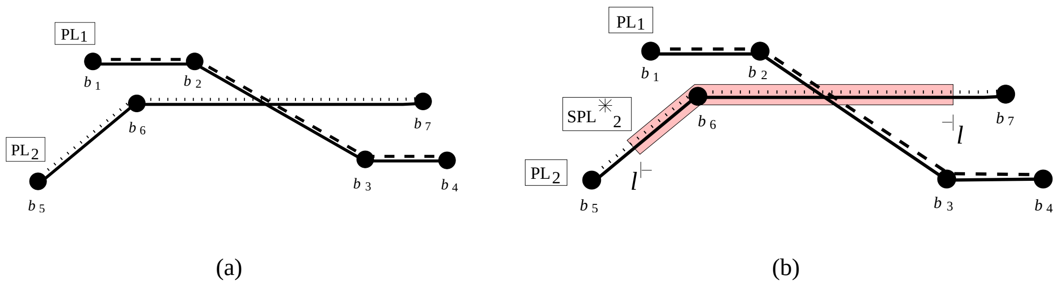


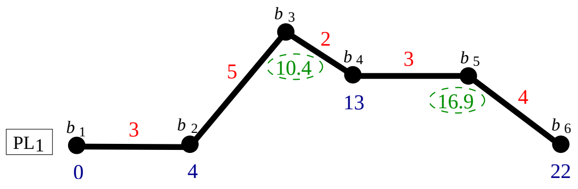
Figure 10: Example of polylines and a subpolyline

Example 6.1. Figure 10(a) illustrates two intersecting polylines PL_1 and PL_2. The polyline PL_1 is formed out of sequence of base points (b_1, b_2, b_3, b_4). The start base point is b_1 and the end base point is b_4. The polyline PL_2 is formed out of (b_5, b_6, b_7). The start base point of PL_2 is b_5, end—b_7.

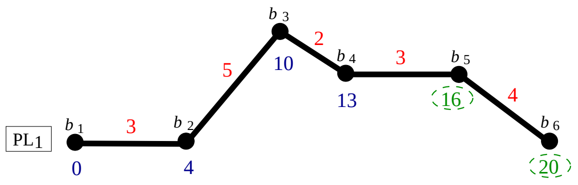
In our 2D road network model we say that each polyline represents a bidirectional road. Without reference to the directions of the roads the polylines have their "directions" leading from the start base points to the end base points. When simulating the movement of the car on our 2D road network model with the help of the direction of the polyline we can specify if the car is moving towards the start or end base point of that polyline.

The roads in our model also have lengths. We consider two cases for the obtaining the length values of the roads. One is to have the road network representation as it is shown in Figure 10(a), where the 2D base points are connected by line segments into polylines. Then the length of a polyline is calculated by summing up the Euclidean distances between the consequent points of the polyline. This approximation of curves into polylines makes calculations concerning the roads easy. But if we use the Euclidean distances for the length of the road, we get the imprecise lengths compared to the world situation. The other case for obtaining the lengths of the roads is to assume that we are able to get the real road distance measures for the base points (or some of them) from the road information providers. It means that for (some) base points of the polyline, we are provided with the road distances from the start of the polyline to these points. For example, the ideal case is when for any base point of the polyline, we have the road distance from the start of the polyline. The measure associated with the last base point of the polyline indicates the road length of the polyline. Then, the subtraction of distance measures of any two consequent base points is greater or equal to the Euclidean distance between these points.

There are situations where we have road distance information only for some base points of the polyline. If we have the measure for the last base point, then we distribute that distance among the intermediate non-measured points proportionally. If we do not have the distance measure for the last base point of the polyline, then we calculate the Euclidean distance between the couples of the remaining points, starting from the last base point with a known road distance measure to the last base point of the polyline.



(a)



(b)

Figure 11: Polyline length calculations

Example 6.2. Let us analyze the example in Figure 11(a). The polyline PL_1 is formed out of sequence of base points $(b_1, b_2, b_3, b_4, b_5, b_6)$. The numbers above the polyline indicate the Euclidean distance values between the couples of base points. The numbers below the polyline hold the road distance values given by the road information provider. The road distance from the base point b_1 to b_2 is equal to 4, from b_1 to b_4 — equal to 13. The road distance from b_1 to b_6 equals to 22, and it is the whole road length of the polyline PL_1.

But we are not provided with the road distances from b_1 to b_3 and from b_1 to b_5. In such a case we calculate approximate road distance values. First, we find the base points that have a known road distance values, and also are closest to b_3. These are b_2 and b_4. Then, from the distance value of the point b_4 we subtract the road distance value of the point b_2. The result is 9. Further, sum up the Euclidean distances between (b_2, b_3) and (b_3, b_4). The result is 7. Now, in order to obtain the approximated road value for the base point b_3, we say that the length of the line segment (b_2, b_3) takes $\frac{5}{7}$ of the road distance between b_2 and b_4. Therefore, we multiply 9 by $\frac{5}{7}$ and get 6.4. The value 6.4 is the road distance between b_2 and b_3. Finally, the number 10.4 inside the ellipse indicates the road distance between b_1 and b_3. For the base point b_5 we calculate the approximated road distance similarly as for b_3.

Figure 11(b) illustrates the situation when we are not provided with the road distance values for the last base points b_5 and b_6 of the polyline PL_1. In this case, we calculate the Euclidean distances for (b_4, b_5) and for (b_5, b_6). The results are, 3 and 4 respectively. The distance between b_1 and b_5 is calculated by summing up the Euclidean distance between b_4 and b_5, which is 3, with the road distance value of b_4, which is 13, and the result is 16. For the base point b_6 we take the calculated Euclidean distance between b_5 and b_6 and add with the polyline length value of b_5. The result is 20.

The operator that calculates the length measure for each base point of the polyline is defined as follows:

Definition 6.3. (Length) Let $\mathcal{L} : PL \times B \to \mathbb{R}$ be the length operator that returns the road distance from the start of the polyline to any base point of the polyline. The operator takes as arguments a polyline $pl = (b_1, ..., b_N) \in PL$ and a base point $b_i \in pl, 1 \leq i \leq N$.

For the first base point of a polyline $pl = (b_1, ..., b_N) \in PL$, we have a length measure that is equal to 0. For any other base point $b_i, i > 1$, the measure is greater than 0. It is also greater than the length of $\mathcal{L}(pl, b_{i-1})$ to the previous base point b_{i-1}. The difference $\mathcal{L}(pl, b_i) - \mathcal{L}(pl, b_{i-1})$ is at least the Euclidean distance value between b_{i-1} and b_i. $\mathcal{L}(pl, b_N)$ is the length of the entire polyline.

Further we define the notions of a part of a road network and a part of a polyline—a *subpolyline*:

Definition 6.4. (Subpolyline) Let $SPL \subset PL \times \mathbb{R}^2$ be a finite set of *subpolylines*. Each subpolyline $spl = (pl, l^{\vdash}, l^{\dashv})$ is a 3-tuple, where

1) $pl = (b_1, b_2, \ldots, b_N) \in PL$ is the polyline the subpolyline spl lies on;

2) $0 \leq l^{\vdash} < l^{\dashv} \leq \mathcal{L}(pl, b_N)$, where $l^{\vdash}$ and $l^{\dashv}$ are distances from the start base point b_1 of the polyline pl: $l^{\vdash}$ is the distance within the subpolyline starts on the polyline and $l^{\dashv}$ is the distance within the subpolyline ends on the polyline.

A subpolyline in our model can be seen as a "cutout" of a polyline. The length of a subpolyline is never 0, and it can be as long as the polyline it lies on. Let us look at Figure 10(b). It shows an accentuated part of polyline PL_2—subpolyline SPL_2. The subpolyline starts within the distance $l^{\vdash}$ from the start base point b_5 of the polyline PL_2, and it ends within the distance $l^{\dashv}$ from the same start base point b_5 of PL_2.

Having defined the representation of real-world roads as polylines, we also want to add into our model intersections of the roads. We assume that the information about the connectivity of the real-world roads is known in advance.

Definition 6.5. (Connection) Let $C \subset \{\{(pl_1, l_1^{\vdash}), \ldots, (pl_N, l_N^{\vdash})\} | (pl_i, l_i^{\vdash}) \in PL \times \mathbb{R} \wedge N \geq 2\}$ be a finite set of *connections*. Each connection $c_i = \{(pl_1, l_1^{\vdash}), (pl_2, l_2^{\vdash}), \ldots, (pl_N, l_N^{\vdash})\}$, $N \geq 2$.

Figure 12 illustrates the situations where two roads represented as polylines intersect at connections. Figure 12(a) describes the intersection of polylines PL_1 and PL_2. The connection point is within the distance $l_1^{\vdash}$ from the start of the polyline PL_1 and it also is within the distance $l_2^{\vdash}$ from the start of the polyline PL_2. Therefore, according to Definition 6.5, this point is defined as $c = \{(PL_1, l_1^{\vdash}), (PL_2, l_2^{\vdash})\}$. The connection points in Figures 12(b) and 12(c) are analogous to the one in Figure 12(a), but also show the situations where the connection point coincides with base points belonging to both of the polylines.

In our model we say that a *road network* is a set of polylines with connectivity rules. Further we define mobile service *users*:

Definition 6.6. (Users) Let U be a finite set of mobile service *users*.

Consider the situation with the user traveling on the road network. Let us say, that she started her trip from *home* and went to *work*. The objects *home*, *work* are the destination objects of the trip. In our model we call them *user objects*.

Definition 6.7. (User Objects) Let $UO \subset PL \times \mathbb{R} \times U$ be a finite set of *user objects*. Each user object $uo = (pl, l^{\perp}, u)$ is a 3-tuple, where

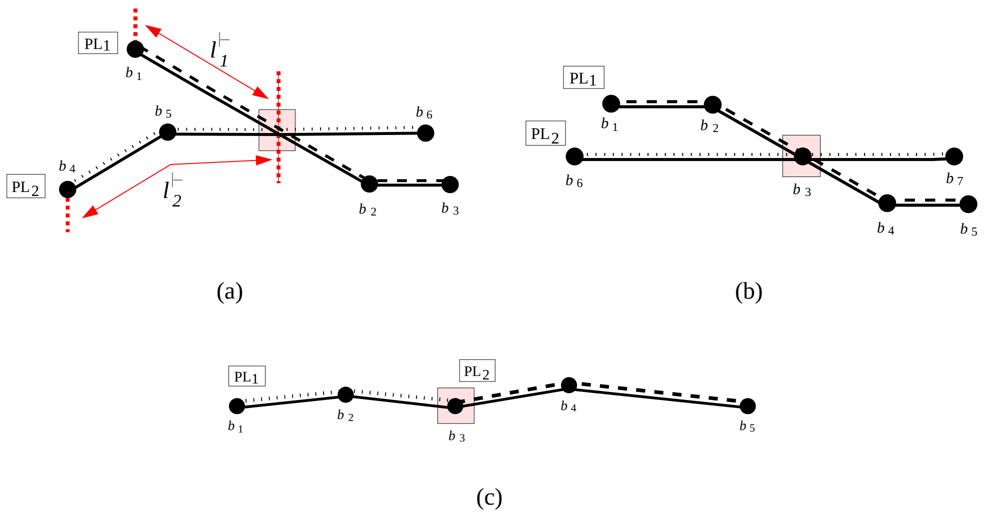


Figure 12: Connections of polylines

1) $pl \in PL$ is the polyline that user object is located on;

2) $0 \leq l^{\perp} \leq l$, where l is the length of the polyline pl, and $l^{\perp}$ denotes the distance within which the user object is located on the polyline;

3) $u \in U$ is the owner of the object.

The *timestamp* is used to fix the time the user travels on a particular *route* and to be able to approximate this time. As the users travel on their usual routes a weekday is one the factors to decide on the users' habits. To get a weekday we need a year, month and day. We also assume that users start to travel at inexactly the same time as they did on the previous days. Thus time is approximated to hours and their quarters using minutes and seconds. At last we have that a *timestamp* includes all the necessary information from which we can derive the necessary one:

Definition 6.8. (Timestamp) Let a *timestamp* T be a finite set of 6-tuples (y, m, d, h, mn, s), where

1) $y, m,$ and d denote *year, month,* and *day*;

2) $h, mn,$ and s denote *hours, minutes,* and *seconds.*

Now we have all the terms defined to have the description for the *route*. A *route* includes start and end objects, subpolylines that cover the paths between them together with directions according to the polyline's direction, and a set of timestamps to have usage information.

Definition 6.9. (Routes) Let R be a finite set of *routes*. Each route is a 4-tuple (RE, uo_s, uo_e, ST), where

1) RE is a sequence of pairs (spl_i, dir_i), where spl_i defines a subpolyline that forms the route and dir_i is the motion direction on this subpolyline:

$$RE = ((spl_1, dir_1), (spl_2, dir_2), \ldots, (spl_N, dir_N)),$$

where for each $spl_i = (pl_i, l_i^{\vdash}, l_i^{\dashv}) \in SPL$

$$dir_i = \begin{cases} 1 & \text{if motion direction on the subpolyline } spl_i \text{ coincides with direction of the polyline } pl_i \\ -1 & \text{if opposite} \end{cases}$$

2) $uo_s = (pl_1, l_s^{\perp}, u) \in UO$ is the start object of the route, and

$$l_s^{\perp} = \begin{cases} l_1^{\vdash} & \text{if } dir_1 = 1 \\ l_1^{\dashv} & \text{if opposite} \end{cases}$$

3) $uo_e = (pl_N, l_e^{\perp}, u) \in UO$ is the end object of the route, and

$$l_e^{\perp} = \begin{cases} l_N^{\dashv} & \text{if } dir_N = 1 \\ l_N^{\vdash} & \text{if opposite} \end{cases}$$

3) $ST \subset T$ denotes the time when the route was used by the user u.

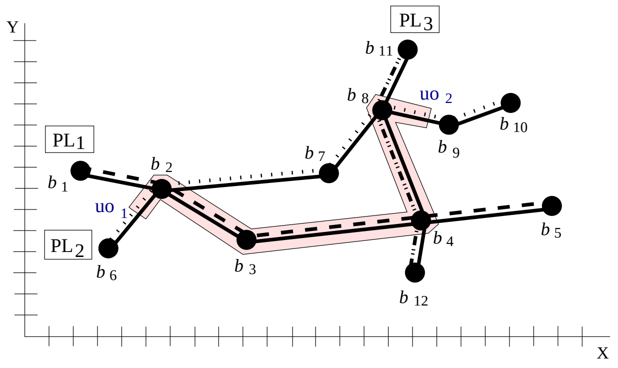


Figure 13: Route in 2D road network

Example 6.3. Figure 13 illustrates the fragment of the 2D road network with three polylines—$PL_1 = (b_1, b_2, b_3, b_4, b_5)$, $PL_2 = (b_6, b_2, b_7, b_8, b_9, b_{10})$, and $PL_3 = (b_{11}, b_8, b_4, b_{12})$. Let us analyze the example of a route *r*. The route *r* lies on the parts of all three polylines PL_1, PL_2 and PL_3. According to the Definition 6.9, *r* = (RE, uo_s, uo_e, ST), where uo_s is the start object of the route and corresponds to uo_1 in Figure 13. Similarly, the object uo_e is the end object of the route and corresponds to uo_2 in our example. ST denotes the times when the route was used by the route owner, and $RE = ((spl_i, dir_i)_N)$ is the set of subpolylines with directions that form the route r. Let us analyze the first subpolyline of the route r. The first subpolyline $spl_1 = (PL_2, l^{\vdash}, \mathcal{L}(PL_2, b_2))$ lies on the polyline PL_2. The start point of spl_1 is the object uo_1, which is located within the distance $l^{\vdash}$ from the start of the polyline PL_2. The end point of spl_1 is b_2 and it is located within the distance $\mathcal{L}(PL_2, b_2)$. In our example, the movement direction dir_1 on the subpolyline spl_1 coincides with the direction of the polyline PL_2. Also, the polyline PL_2 intersects with the polyline PL_1 at a connection point c. According to the Definition 12, this intersection is defined as $c = \{(PL_1, l_1^{\vdash}), (Pl_2, l_2^{\vdash})\}$. The value $l_1^{\vdash}$ denotes the distance from the start of the polyline Pl_1 to the connection point c, and $l_2^{\vdash}$ denotes the distance from the start of the polyline PL_2 to the same connection point c. In Figure 13 the connection point coincides with the base point b_2.

6.2 Database Schema

We create a model of the database (see Figure 14) to store the information defined in Section 6.1. To begin with, we create a table **LINEAR_ELEMENTS** for the main elements representing roads of the road network—polylines. Each row in the table **LINEAR_ELEMENTS** contains unique identification number of the polyline (attribute POL_ID) and also the length value of that polyline (attribute POL_LENGTH). The primary key of the table is the attribute POL_ID.

Our model of the real-world road network captures informati on about the intersections of the roads and describes that information in terms of intersection of the polylines at the connection points. Graphically it is illustrated in Figure 12. The information about the intersections of the polylines is stored in table **CONNECTIONS**. The record in this table describes the situation where a polyline (POL_ID) within the distance from it's start point (POL_FROM) intersects with other polyline(s) at the connection point (CONN_ID). The pair of attributes (POL_ID, CONN_ID) forms a primary key of the table **CONNECTIONS**. The attribute POL_ID is a foreign key referencing to the attribute POL_ID of the table **LINEAR_ELEMENTS**.

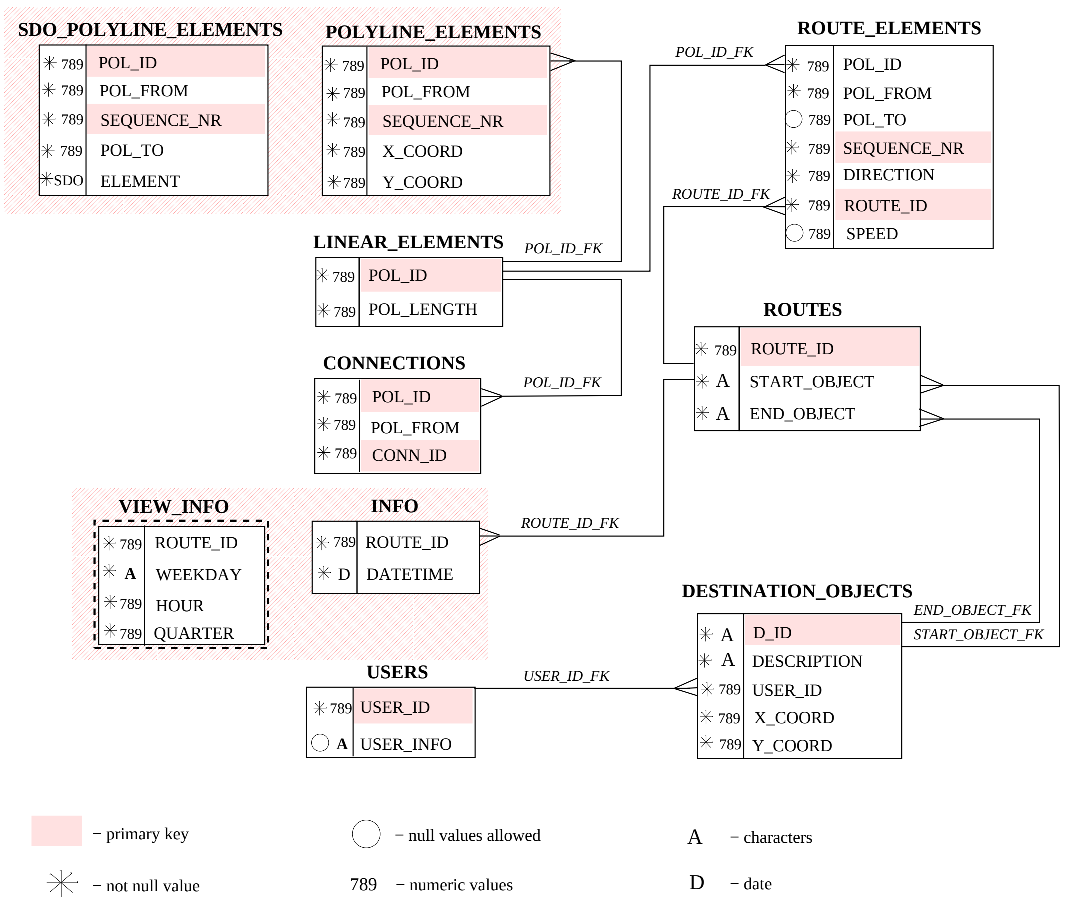


Figure 14: Tabular diagram

The table **POLYLINE_ELEMENTS** is created to store the geographical information on each polyline. As it was mentioned earlier in Section 6.1, the polylines are formed out of sequences of base point connected with the line segments. Therefore records of the table **POLYLINE_ELEMENTS** capture this representation of the polyline. The table contains five attributes: POL_ID, POL_FROM, SEQUENCE_NR, X_COORD and Y_COORD. Let us describe the record of the table **POLYLINE_ELEMENTS** more detailed. The part of the record captured by attribute POL_ID points out the concrete polyline. The attribute SEQUENCE_NR indicates the number of the base point in the sequence of the base points of that polyline. Further, the POL_FROM captures the distance from the start base point of the polyline to the one indicated by SEQUENCE_NR. The attributes X_COORD and Y_COORD hold the geographical location information of the base point. The primary key of the table **POLYLINE_ELEMENTS** is formed by the pair (POL_ID, SEQUENCE_NR). The attribute POL_ID is a foreign key, and it refers to the attribute POL_ID of the table **LINEAR_ELEMENTS**.

The table **SDO_POLYLINE_ELEMENTS** (see Figure 14) is created in order to use the possibilities provided by Oracle Spatial [13]. The attributes in this table are similar to the ones in the table **POLYLINE_ELEMENTS**, except the attribute ELEMENT. This attribute captures not the geo-information about a single base point, but the whole line segment with its start and end points.

Further, let us look at the table **USERS** in Figure 14. The record of this table contains the unique identification number of mobile service user (USER_ID) and also additional information about these users (USER_INFO). By "additional" information we mean that the user can provide service provider with her personal information (first name, last name, address or other), and store this information together with her route information. The attribute USER_ID of the table **USERS** is the primary key.

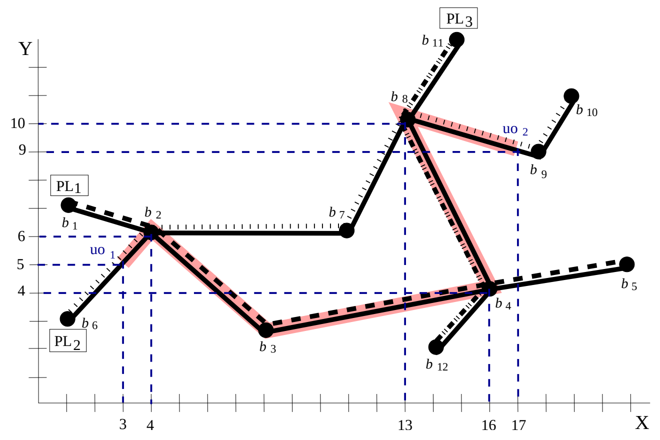
Before describing the tabular representation of the routes, we give the description of the table **DESTINATION_OBJECTS**. See Figure 14. The information stored in this table contain destination objects of the routes of the individual users. The attributes of this table are: D_ID, DESCRIPTION, USER_ID, X_COORD and Y_COORD. Each record of the table names the object (D_ID) of the user (USER_ID), gives the description of that object (DESCRIPTION), and also contains the information captured by the attributes X_COORD and Y_COORD. In our model, we assume that the descriptions of destination objects are provided by the user. The geo-location (X_COORD, Y_COORD) of the destination objects is obtained when analyzing the log file with the GPS points. The first GPS point from the log file is considered as a geo-location of the start object of the route. The last GPS point indicates location of the end object. The attribute D_ID of the table **DESTINATION_OBJECTS** is a primary key, while the attribute USER_ID is a foreign key referencing the attribute with the same name in table **USERS**.

The table **ROUTES** is created for storing the information about the routes of the mobile service users. In our road network representation each route starts and ends at the start and end destination objects of the route. This information is captured by the attributes ROUTE_ID, START_OBJECT and END_OBJECT of the table **ROUTES**. See Figure 14. Each record in this table contains the unique identification number of the route (ROUTE_ID), start object (START_OBJECT) and the end object (END_OBJECT) of that route. The primary key of the table **ROUTES** is ROUTE_ID. The foreign keys referencing the D_ID of the table **DESTINATION_OBJECTS** are the attributes START_OBJECT and END_OBJECT.

Records in table **ROUTE_ELEMENTS** (see Figure 14) describe routes in terms of subpolylines. In our road network representation the route can lie on several different polylines, as it is shown in Figure 13. Therefore, we modeled the route as a sequence of subpolylines. Each row in the table **ROUTE_ELEMENTS** describes a route element—subpolyline. The attribute POL_FROM captures the distance between the start of the polyline and the start of the subpolyline (see Definition 6.4). The attribute POL_TO captures the distance between the start base point of the polyline and the end point of that subpolyline. This attribute is created to make the calculations easier while querying the data. Following the route from its start to the end, we number the parts of the route (subpolylines) and store this sequence with the help of the attribute SEQUENCE_NR. Recall that the polylines in our road network representation have the start and end base points (see Definition 6.2), and we also assume that each polyline has a direction leading from the start of the polyline to the end of that polyline. Each route in our road network representation also has a direction leading from the start of the route to the end of it. The information we store in DIRECTION indicates whether the direction of the polyline coincides with the direction of the route on that polyline. The attribute SPEED captures the average speed of the user on each part of the route. The attributes ROUTE_ID and SEQUENCE_NR form the primary key of the **ROUTE_ELEMENTS**. The attributes POL_ID and ROUTE_ID are the foreign keys referencing respectively the POL_ID in the table **POLYLINE**, and the ROUTE_ID in the table **ROUTES**.

Further, we create the table **INFO** to store the statistical information about the usage of the routes (see Figure 14). The record in this table captures unique ID of the route (ROUTE_ID) and the time (DATETIME) when this route was used. Note that the attribute DATETIME is of a standard time format and captures the time with precision up to seconds. In our model, in order to calculate the usage of the route we find it more useful to operate with the terms such as weekdays, hours and quarters. This is obtained by creating a view table **VIEW_INFO** (see Figure 14). This table contains the attributes ROUTE_ID, WEEKDAY, HOUR and QUARTER (the PL/SQL function that creates the table **VIEW_INFO** can be found in Appendix A). The attribute WEEKDAY captures the days of the week when the route was used by the user. Similarly with the hours (HOUR). We approximate the exact route usage time with quarters (QUARTER). The values in

QUARTER are $1, 2, 3$ and 4, where each of them refer to the time interval, i.e., 1 quarter corresponds to the time interval from 0 to 14 minutes, 2—to the time interval from 15 to 29 minutes and so on.



(a) Fragment of the 2D road network

POLYLINE_ELEMENTS

POL_ID	POL_FROM	SEQUENCE_NR	X_COORD	Y_COORD
1	0	1	1	7
1	3.3	2	4	6
1	6	3	8	2
1	8.4	4	16	4
1	6.3	5	21	5
2	0	1	1	3
2	4.4	2	4	6
2	7	3	11	6
2	4.6	4	13	10
2	5.6	5	18	9
2	2.4	6	19	11
3	0	1	15	13
3	4	2	13	10
3	7	3	16	4
3	4	4	14	2

USERS

USER_ID	USER_INFO
1	Agnė

DESTINATION_OBJECTS

D_ID	DESCRIPTION	USER_ID	X_COORD	Y_COORD
1	*Home*	1	3	5
2	*University*	1	17	9

ROUTES

ROUTE_ID	START_OBJECT	END_OBJECT
1	1	2

LINEAR_ELEMENTS

POL_ID	POL_LENGTH
1	24
2	24
3	15

INFO

ROUTE_ID	DATETIME
1	2003:01:29 11:12:03
1	2003:02:06 12:34:15
1	2003:02:13 09:05:47

CONNECTIONS

POL_ID	POL_FROM	CONN_ID
1	3.3	1
1	17.7	2
2	4.4	1
2	16	3
3	4	3
3	11	2

ROUTE_ELEMENTS

POL_ID	POL_FROM	POL_TO	SEQUENCE_NR	DIRECTION	ROUTE_ID	SPEED
2	3	4.4	1	1	1	50
1	3.3	17.7	2	1	1	70
3	11	4	3	−1	1	50
2	16	20.6	4	1	1	60

(b) Example of the data stored in the database

Figure 15: Instance of the database

Example 6.4. Figure 15(b) illustrates the instance of the database build on the example of the 2D road network in Figure 15(a). Mobile service user Agnė (table **USERS**) travels every workday from her “HOME” to “UNIVERSITY”. These are descriptions of the destination objects of her usual route (see table **DESTINATION_OBJECTS**). This route has the identification number 1 and is stored in table **ROUTES**. The route 1 leads through the linear_elements with the identification numbers 1, 2 and 3, as it is shown in table **ROUTE_ELEMENTS**. The identification numbers of the are retrieved from the tables **LINEAR_ELEMENTS** and **POLYLINE_ELEMENTS**. The records in the table **INFO** describes the usage of the route.

7 High-level Algorithms

This section presents a set of high-level algorithms that solve the route recording problem. Polyline identification is covered. We also describe how we construct the subpolylines that make up a route, including the types of subpolylines. We give the algorithms that concern these topics. Finally, we give the route construction algorithm and describe how it solves the route finding problem.

7.1 Polyline Identification

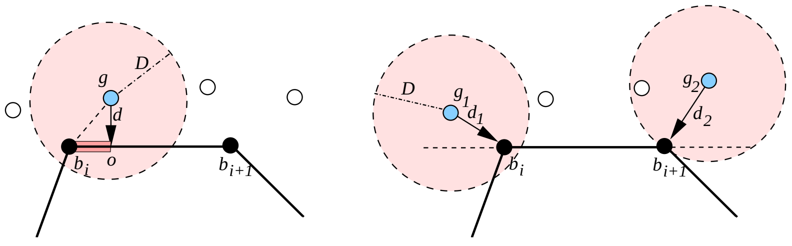
The first problem we address when recording a route is polyline identification. We get a set of GPS coordinates from the user device. Based on these, the polylines and positions on them have to be defined in order to obtain the subpolylines that form the route. We make a few assumptions about the GPS data that help solve this problem.

- We allow an imprecision D for the GPS coordinates. The information from the satellites is fairly accurate, but still not precise. Thus, the imprecision must be considered. D is the radius of a circle that has the GPS coordinate point as a center.
- We assume that the first GPS point is mapped to the correct polyline. This polyline is the nearest polyline to the first coordinate. This assumption is needed to have correct start information. We consider the previous polyline to define polylines for all other coordinates.
- We assume that a user moves on the road network according to the traffic regulations, i.e., she does not make illegal turns and she travels according to the allowed direction. The road properties such as driving directions are not considered to define the polyline in our work. The polyline identification is made according to the geographical coordinates and connections of the polylines.

Projection of the Point and Distance From the Start of the Polyline. The polyline is identified using the distance of the GPS coordinate point to the polyline. The GPS point is projected onto the polyline, and according to the projection, the distance from the start of the polyline is calculated. There are three cases of the projection. Figure 16 illustrates them. There is a polyline segment $b_i b_{i+1}$. Small circles indicate the sequence of the GPS points. The GPS points g, g_1, and g_2 are points of interest. Large circles have radius D, and they are areas of imprecision. The GPS point can be exactly "above" the polyline segment as g is in Figure 16(a). It is projected onto the polyline segment as point o, and Euclidean distance to the projection point is $d < D$. The coordinate can also be ahead of or behind the polyline segment as g_1 and g_2 are in Figure 16(b). Now we project the GPS points to the end points, i.e., b_i and b_{i+1}, of the polyline segment. We do not need the projected points in our algorithms. But we need the distance d from a GPS point to its projection, because based on this, we decide which polyline suits for us. We also need the distance $l^{\vdash}$ from the start of the polyline to the point projection for other calculations, like subpolyline formation and direction identification.

We calculate distance d using vector algebra. According to its definition (see Definition 6.2), a polyline has a direction. Thus, polyline segment $b_i b_{i+1}$ has a start and an end point. The start point b_i is the beginning of two vectors. The end of the first vector is the end point b_{i+1} of the segment. The end of the second vector is the GPS point g (see Figure 17). The angle α between these vectors is the argument to calculate the distance to the point projection. The angle is calculated using scalar multiplication:

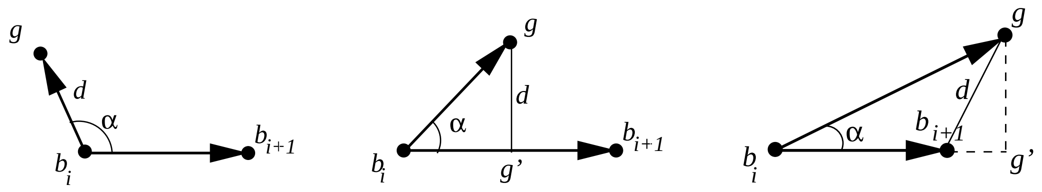
$$\overline{b_i b_{i+1}} \cdot \overline{b_i g} = |\overline{b_i b_{i+1}}| \cdot |\overline{b_i g}| \cdot \cos\alpha$$



(a) On the segment

(b) Ahead of or behind the segment

Figure 16: Position of the GPS coordinate due to the polyline segment



(a) Obtuse angle.

(b) Acute angle. The projection is shorter than the polyline segment.

(c) Acute angle. The projection is longer than the polyline segment.

Figure 17: Angles and projections

If the angle is obtuse (Figure 17(a)), the distance to the point projection is the Euclidean distance from the GPS point to the start of the polyline segment. The distance from the start of the polyline to the point projection is the distance from the start of the polyline to the start of the polyline segment (see Definition 6.3):

$$\text{if } 90^\circ < \alpha < 270^\circ \text{ then } d = |\overline{gb_i}|, l^\vdash = \mathcal{L}(pl, b_i), \text{ where } b_i \in pl = (b_1, ..., b_i, b_{i+1}, ..., b_N)$$

If the angle is acute, we have two possible situations (Figure 17(b) and Figure 17(c)). We project vector $\overline{b_i g}$ onto vector $\overline{b_i b_{i+1}}$. If the length $|b_i g'|$ of the projection is greater than the length of the segment $b_i b_{i+1}$, distance d is the distance between the end point b_{i+1} of the segment and the GPS point g. The distance from the start of the polyline is the distance from the start of the polyline to the end point b_{i+1} of the segment:

$$\text{if } -90^\circ \leq \alpha \leq 90^\circ \wedge |b_i g'| \geq (\mathcal{L}(pl, b_{i+1}) - \mathcal{L}(pl, b_i)) \text{ then } d = |gb_{i+1}|, l^\vdash = \mathcal{L}(pl, b_{i+1}),$$
$$\text{where } b_i, b_{i+1} \in pl = (b_1, ..., b_N)$$

If the projection length $|b_i g'|$ is less than the length of the polyline segment, then distance d is the distance from the GPS coordinate to the polyline segment. The distance from the start of the polyline is a sum of the projection length and the distance between the start of the segment and b_i:

$$\text{if } -90^\circ \leq \alpha \leq 90^\circ \wedge |b_i g'| < (\mathcal{L}(pl, b_{i+1}) - \mathcal{L}(pl, b_i)) \text{ then } d = |gg'|, l^\vdash = \mathcal{L}(pl, b_i) + |b_i g'|,$$
$$\text{where } b_i, b_{i+1} \in pl = (b_1, ..., b_N)$$

Using this strategy, function $calcParam$ (see Algorithm 7.1) calculates the distance d from the GPS point g to the segment pls of the polyline pl and the distance $l^\vdash$ from the start of the polyline to the projected point on it. In our paper, we find it convenient to use expression $a = b$ to let a denote b and as a logical expression. The meaning follows from the context in which it is used.

Algorithm 7.1 Calculation of the Parameters (function $calcParam$)

```
Require: INPUT: $g = (x, y) \in \mathbb{R}^2, pl \in PL, pls = ((x_1, y_1), (x_2, y_2)),$ where $(x_i, y_i) \in pl$. OUTPUT: $(d, l^{\vdash}) \in \mathbb{R} \times \mathbb{R}$.
 1: $\vec{v}_1 = \{vx_1; vy_1\} \leftarrow \{x_2 - x_1; y_2 - y_1\}$
 2: $\vec{v}_2 = \{vx_2; vy_2\} \leftarrow \{x - x_1; y - y_1\}$
 3: $|\vec{v}_1| \leftarrow \sqrt{vx_1^2 + vy_1^2}$
 4: $|\vec{v}_2| \leftarrow \sqrt{vx_2^2 + vy_2^2}$
 5: $\alpha \leftarrow \arccos((vx_1 \cdot vx_2 + vy_1 \cdot vy_2)/(|\vec{v}_1| \cdot |\vec{v}_2|))$
 6: if $90^\circ < \alpha < 270^\circ$ then
 7:     $d \leftarrow |\vec{v}_2|$
 8:     $l^{\vdash} \leftarrow \mathcal{L}(pl, (x_1, y_1))$
 9: else
10:     $projection \leftarrow |\vec{v}_2| \cdot \cos\alpha$
11:     $length \leftarrow \mathcal{L}(pl, (x_2, y_2)) - \mathcal{L}(pl, (x_1, y_1))$
12:     if $length \leq projection$ then
13:         $d \leftarrow \sqrt{(x - x_2)^2 + (y - y_2)^2}$
14:         $l^{\vdash} \leftarrow \mathcal{L}(pl, (x_2, y_2))$
15:     else
16:         $d \leftarrow |\vec{v}_2| \cdot \sin\alpha$
17:         $l^{\vdash} \leftarrow \mathcal{L}(pl, (x_1, y_1)) + projection$
18:     end if
19: end if
20: return $(d, l^{\vdash})$
```

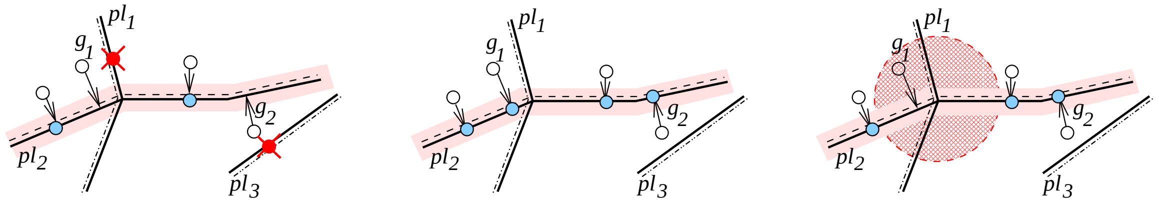
First GPS Point. As mentioned earlier, the first GPS point is mapped to the nearest polyline. The one parameter used to detect the polyline for the first GPS point is the distance from the polyline to the GPS point. The distance to the polyline is the distance from the GPS point to the polyline's nearest segment. Function $polyFirstId$ (see Algorithm 7.2) finds the nearest polyline for the GPS point g. It returns the polyline and the distance from the start of the polyline to the point projection. The function uses the parameters calculated by function $calcParam$. The search is performed analyzing all the polylines pl_i. The parameters $(currD, currL)$ for each polyline segment $(pl_{i_j} pl_{i_{j+1}})$ are calculated. If the distance $currD$ to the current polyline is less than the distance d to the previous nearest polyline, it becomes the nearest, and the distance $l^{\vdash}$ from the start of the polyline is $currL$.

Other GPS Coordinates. The GPS point other that the first one can also be mapped to the nearest polylines. But this strategy causes many faults. The GPS coordinates are not precise and are not exactly on the polyline segment. There is an imprecision D. There are a few cases of the road network when the nearest polyline is not the true polyline. The first case is when the GPS point is near a crossroads. It can be that the nearest polyline is one that crosses the true one. The second possible case is when the polyline segments are a small distance from each other, even if they do not intersect. In real world this can appear when there are two roads at different heights, for example, when one road passes under another. These problems are solved using our strategy. We identify polylines for GPS points considering the results of the mapping of the previous GPS point. If we have the point $g_j, j > 1,$ then we determine where the point g_{j-1} is mapped to. The point g_j should be mapped to the same polyline or the other polyline that has a connection point with the previous polyline. This strategy causes bad results in the case of a crossroads if the GPS point has to be mapped on the other polyline. The results that are in the areas of polyline connections can be eliminated.

Algorithm 7.2 First Polyline Identification (function $polyFirstId$)

```
Require: INPUT: $g \in \mathbb{R}^2$. OUTPUT: $(pl, l^{\vdash}) \in PL \times \mathbb{R}$.
1:  $(pl, l^{\vdash}) \leftarrow (\emptyset, \infty)$
2:  $d \leftarrow \infty$
3:  for all $pl_i = (b_{i_1}, ..., b_{i_n}) \in PL$ do
4:      for all $pl_{i_j} = (b_{i_j}, b_{i_{j+1}})$, such that $1 \leq j \leq n-1$, do
5:          $(currD, currL) \leftarrow calcParam(g, pl_i, pl_{i_j})$
6:          if $currD < d$ then
7:              $d \leftarrow currD$
8:              $(pl, l^{\vdash}) \leftarrow (pl_i, currL)$
9:          end if
10:     end for
11: end for
12: return $(pl, l^{\vdash})$
```

Figure 18 illustrates how the nearest polyline strategy can cause bad results (Figure 18(a)) and how other strategy can solve this problem (Figure 18(b) and Figure 18(c)).



(a) Nearest polylines (b) Polylines identified according to the previous results (c) Elimination of candidate polylines

Figure 18: Polyline identification

In Figure 18, we have three polylines. The route is on the polyline pl_2 with the traveling direction from left to right. Unfilled circles are GPS points. The arrows show where each point should be mapped to. Filled and crossed circles present bad projections. The GPS points g_1 and g_2 are points of interest because they are mapped to wrong polylines: g_1 is projected onto polyline pl_1, and g_2 is projected onto polyline pl_3. If we consider the previous result, then g_1 is mapped to pl_2 and g_2 (Figure 18(b)) is also mapped to pl_2. Polyline pl_3 cannot be the candidate polyline, because it does not even intersect with pl_2. If the GPS point is mapped to the connection area as g_1 (Figure 18(c)) its result can be eliminated.

Function $polyId$ (see Algorithm 7.3) identifies the polyline pl for the GPS point g according to the polyline $prevPl$ that the previous GPS point is mapped to. The function returns the polyline and the distance from the start of the polyline to the point projection. The function has two parts. In the first part (Steps 4–10), we check if the GPS point is on the same polyline as the previous GPS point. The distance $currD$ to every segment of the polyline $prevPl$ is calculated. The shortest one is chosen, as it was in function $polyFirstId$. But the distance has also be less than D, the a value of imprecision. If this search gives no results (Step 11), we assume that perhaps the GPS point should be mapped to the polyline that has a connection with the previous polyline. Thus, the second part (Steps 12–20) of the function searches for the polyline that is nearest among the polylines that intersect with $prevPl$. The distance to the polyline has to be less than the imprecision D. If this part of the algorithm also does not give an answer, the function returns the polyline and the distance undefined. Such a result means that there is a gap in the GPS data, which then has to be filled in.

To define if the GPS point is in the connection area, at first we need the result of function $polyId$. The

Algorithm 7.3 Polyline Identification (function $polyId$)

```
Require: INPUT: $g \in \mathbb{R}^2, prevPl \in PL$. OUTPUT: $(pl, l^{\vdash}) \in PL \times \mathbb{R}$.
 1: $(pl, l^{\vdash}) \leftarrow (\emptyset, \infty)$
 2: $d \leftarrow \infty$
 3: $pl \leftarrow prevPl = (b_1, ..., b_n)$
 4: for all $pl_j = (b_j, b_{j+1})$, such that $1 \leq j \leq n-1$, do
 5:     $(currD, currL) \leftarrow calcParam(g, prevPl, pl_j)$
 6:     if $currD < d \wedge currD \leq D$ then
 7:         $l^{\vdash} \leftarrow currL$
 8:         $d \leftarrow currD$
 9:     end if
10: end for
11: if $d = \infty$ then
12:     for all $pl_i = (b_{i_1}, ..., b_{i_n})$, such that $\exists c = (..., (pl_i, l_i), ..., (prevPl, prevL), ...) \in C$ do
13:         for all $pl_{i_j} = (b_{i_j}, b_{i_{j+1}})$, such that $1 \leq j \leq n-1$, do
14:             $(currD, currL) \leftarrow calcParam(g, pl_i, pl_{i_j})$
15:             if $currD < d \wedge currD \leq D$ then
16:                 $(pl, l^{\vdash}) \leftarrow (pl_i, currL)$
17:                 $d \leftarrow currD$
18:             end if
19:         end for
20:     end for
21: end if
22: return $(pl, l^{\vdash})$
```

function identifies the candidate polyline the GPS point is projected onto and gives the distance from the start of the polyline. The connection (see Definition 6.5) is defined using the distance from the start of the polyline. Thus, if the GPS point is mapped to the polyline and its distance to the connection on the polyline is less than imprecision D, we say that the GPS point is in the connection area. Later, we do not consider this result. We have function $possibleConnection$ (see Algorithm 7.4) that returns a Boolean value showing if the GPS point is in the connection area. All the connections related to the polyline are analyzed and the distance from the point projection to the connection is calculated. A distance less than imprecision D makes the result *true* and the function returns a result; otherwise, it remains *false*.

Algorithm 7.4 Detection of Connection Area (function $possibleConnection$)

```
Require: INPUT: $pl \in PL, l^{\vdash} \in \mathbb{R}$. INPUT: $connection \in \{\text{true}, \text{false}\}$.
 1: $connection \leftarrow \text{false}$
 2: for all $c_i = \{cc_1, ..., cc_n\} \in C$, such that $\exists cc_{i_j} = (pl, l^{\vdash}_{i_j}) \in c_i$ do
 3:     for all $cc_{i_j} \in c_i$, such that $cc_{i_j} = (pl, l^{\vdash}_{i_j})$ do
 4:         if $-D \leq l^{\vdash}_{i_j} - l^{\vdash} \leq D$ then
 5:             $connection \leftarrow \text{true}$
 6:             return $(connection)$
 7:         end if
 8:     end for
 9: end for
10: return $(connection)$
```

7.2 Formation of a Route Element

The next problem we deal with while detecting a route is the formation of subpolylines. A route is a sequence of subpolylines that are connected to the neighboring subpolylines. There cannot be gaps. According to our model, the user can change the polyline she is traveling on only at connections, but not anywhere else. The requirement to have a sequence of subpolylines that form an uninterrupted polyline makes the route construction include specific functions.

There are four main cases of subpolylines that form a route. Figure 19 illustrates them. The unfilled circles mark the GPS points. The are three polylines drawn in the figure: (b_1, b_2, b_3, b_4), (b_6, b_2), and (b_3, b_5). The route is emphasized.

Figure 19(a) illustrates the most simple case of a route, when only one subpolyline belongs to it. According to our model, we always fix the exact positions for the start and the end of the route and do not approximate them. That means that when we form such a polyline, we consider the first and the last GPS points, i.e., g_0 and g_N. The distances from the start of the polyline present which part of the polyline is the route and the movement direction indicates the start and the end.

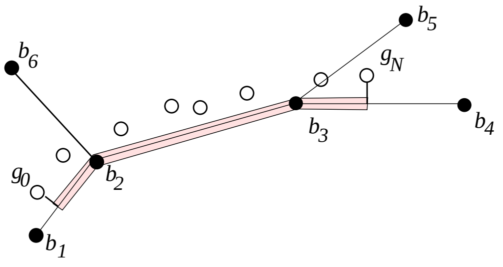
Figure 19(b) illustrates how the first subpolyline is formed. The feature of the first subpolyline is that one distance measure from the start of the polyline is the exact projection position, as it is the start of the route. Another measure usually has to be approximated to the value that shows the connection distance from the start of the polyline. As the figure shows, the GPS points $g_i, 0 \leq i \leq j$, are projected onto one polyline. The GPS point g_{j+1} is projected onto another polyline. The polyline can be changed only at connection position b_3; thus, the projection of g_j is approximated to the distance from the start of the polyline to the nearest intersection with the other polyline, i.e. to b_3. In case of the first subpolyline, the start is defined from the GPS point, and the end is calculated. According to the definition (see Definition 6.4), a subpolyline has start distance that is less than end distance. If the travel direction is opposite to the polylines's direction, these distances are interchanged and the direction is set to -1. This means that we have a measured end, but an approximated start.

Figure 19(c) illustrates how the last subpolyline is formed. This case is similar to the case of the first subpolyline, but it has the opposite way of formation. If the movement direction is the same as the direction of the polyline, the start of the last subpolyline is approximated, as for g_{j+1} in the figure. The end of the subpolyline is the end of the route, i.e., g_N. If the direction is opposite, the measures are formed in the opposite way.

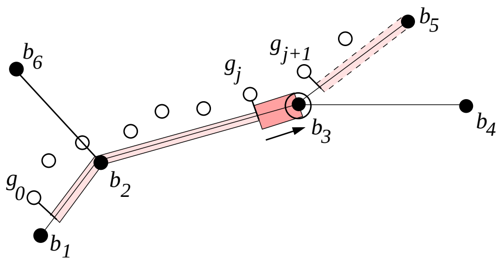
Figure 19(d) illustrates how the middle subpolyline is formed. Middle subpolylines are in the route if the GPS points are mapped to more than two polylines. These subpolylines are full. The start and the end distances for the subpolyline are not those from projections, but approximated to the distances of the connections at which the polylines are changed. As it is shown in the figure, GPS points g_{k+1} and g_j are the first and last GPS points that are projected onto the first polyline. Their distance values from the start of the polyline are approximated to values of the connections b_2 and b_3.

If there are no neighboring subpolylines that belong to the same polyline, only these four cases are used to form a subpolyline. There are situations when a route has two subpolylines belonging to the same polyline, but having the opposite direction. Figure 20 shows such a situation. The user can travel on the polyline and turn around at some point. In the real world, this situation may occur if it is not allowed to turn left at some crossroads. Then the user has to turn right and to turn around when it is possible, as in Figure 20(a). If the direction is changed on the same polyline, there are two neighboring subpolylines that share one common point: it is the end for one subpolyline, but the start point for another. Figure 20(b) shows how it looks in the real situation. The GPS points are marked by numbers, the greater number shows the later GPS point. The last point that is in the same direction as the current subpolyline is also the start of the new subpolyline. The real GPS point can be still on the same side of the polyline (before turning around) or already on the other (after turning around), as shown in Figure 20(b).

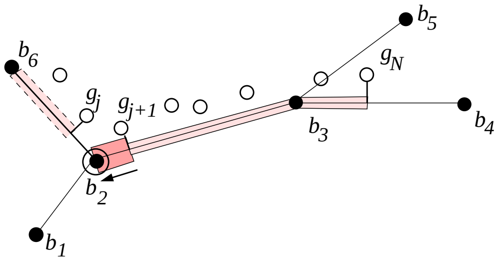
According to the described strategy, the movement direction is the parameter that is used to form a subpolyline. We have a function $defineDirection$ (see Algorithm 7.5) that defines the movement direction



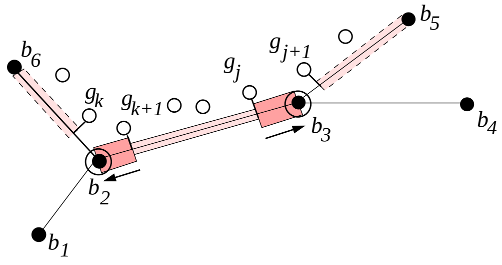
(a) Only one subpolyline



(b) First subpolyline

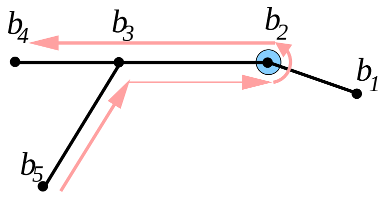


(c) Last subpolyline

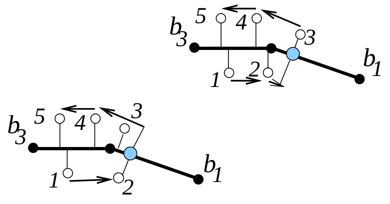


(d) Middle subpolyline

Figure 19: Cases of subpolylines



(a) Movement



(b) Formation

Figure 20: Subpolylines on the same polyline

on the polyline for two point projections. The function takes two distances from the start of the polyline: $prevDst$ is the distance to the previous point projection, and $currDst$ is the distance to the current point projection. The function also considers the movement direction $prevDir$ on the polyline until the current GPS point. The function returns the movement direction $direction$ that is on the polyline between the previous and the current points. If the previous distance is less than the current one, the direction coincides with the direction of the polyline— it is equal to 1. If the previous distance is greater than the current one, the direction is the opposite—it is equal to -1. If the previous distance is equal to the current distance, the direction cannot be defined and is equal to the previous direction. The last situation happens if the user is stuck in a traffic jam and moves so slowly that this causes the same GPS coordinates for a few points.

The movement direction is used to approximate distances from the start of the polyline to the point projections when we construct "uninterruptible" route from a sequence of subpolylines. Function $findEnd$ (see Algorithm 7.6) finds the distance $endDst$ from the start of the polyline $prevPl$ to the connection where the previous polyline $prevPl$ and the current polyline $currPl$ intersect. The function considers the distance $prevDst$ from the start of the polyline for the previous GPS point and also the direction $prevDir$ on the polyline $prevPl$. The function chooses the nearest connection if there are a few connections where the

Algorithm 7.5 Direction Identification (function $defineDirection$)

```
Require: INPUT: $prevDst, currDst \in \mathbb{R}, prevDir \in \{-1, 0, 1\}$. OUTPUT: $direction \in \{-1, 0, 1\}$.
1: $direction \leftarrow 0$
2: if $prevDst < currDst$ then
3:     $direction \leftarrow 1$
4: else if $prevDst > currDst$ then
5:     $direction \leftarrow -1$
6: else
7:     $direction \leftarrow prevDir$
8: end if
9: return $(direction)$
```

polylines intersect. The temporary variable $distToConn$ stores the value of the distance to the previous nearest connection. All the connections c_i where $prevPl$ and $currPl$ intersect are analyzed. The variable $currDistToConn$ is used to calculate the distance $prevDst$ from the previous projected point to each suitable connection. A connection is suitable if it is ahead of the projected point when the direction coincides with the polyline's direction, or if it is behind the projected point when the direction is the opposite (Step 4). Otherwise, the function returns an undefined value. If the distance to the connection is less than the distance to the previous connection then its distance from the start of the polyline is the candidate end distance for the subpolyline. The distance to it is noted in variable $currDistToConn$. If the direction on the subpolyline is undefined, the nearest connection is chosen as a candidate, and the direction to it does not matter.

Algorithm 7.6 End Position Identification for a Subpolyline (function $findEnd$)

```
Require: INPUT: $prevPl, currPl \in PL, prevDst \in \mathbb{R}, prevDir \in \{-1, 0, 1\}$. OUTPUT: $endDst \in \mathbb{R}$.
1: $distToConn, endDst \leftarrow \infty$
2: for all $c_i = \{cc_1, ..., cc_n\} \in C$, such that $\exists cc_{i_j}, cc_{i_k} \in c_i : cc_{i_j} = (prevPl, l^{\vdash}_{i_j}), cc_{i_j} = (currPl, l^{\vdash}_{i_k})$
   do
3:     $currDistToConn \leftarrow (l^{\vdash}_{i_j} - prevDst)$
4:     if $(prevDir = 1 \wedge currDistToConn > 0 \wedge currDistToConn < distToConn) \vee$
       $(prevDir = -1 \wedge currDistToConn < 0 \wedge currDistToConn > distToConn)$ then
5:         $endDst \leftarrow l^{\vdash}_{i_j}$
6:         $distToConn \leftarrow currDistToConn$
7:     else if $prevDir = 0$ then
8:         if $(currDistToConn > 0 \wedge currDistToConn < distToConn)$ then
9:             $endDst \leftarrow l^{\vdash}_{i_j}$
10:            $distToConn \leftarrow currDistToConn$
11:        else if $(currDistToConn < 0 \wedge currDistToConn \cdot (-1) < distToConn)$ then
12:            $endDst \leftarrow l^{\vdash}_{i_j}$
13:            $distToConn \leftarrow currDistToConn \cdot (-1)$
14:        end if
15:    end if
16: end for
17: return $(endDst)$
```

Function $findStart$ is closely related to function $findEnd$. The next subpolyline should always start at the same place where the previous subpolyline ended. Thus, function $findStart$ defines where the next subpolyline starts on the polyline $currPl$, according to the previous subpolyline that was on polyline $prevPl$ and ended at distance $prevDst$ from its start. The function returns the $startDst$ that is the distance at which the current subpolyline starts.

Algorithm 7.7 Start Position Identification for a Subpolyline (function $findStart$)

```
Require: INPUT: $prevPl, currPl \in PL, prevDst \in \mathbb{R}$. OUTPUT: $startDst \in \mathbb{R}$.
1: for $c = (cc_1, ..., c_n) \in C$, such that $\exists c_k, c_m : c_k = (prevPl, prevDst), c_m = (currPl, l_m^{\vdash})$ do
2:     $startDst \leftarrow l_m^{\vdash}$
3: end for
4: return $(startDst)$
```

Function $formSubPoly$ is used to create a subpolyline that satisfies the requirements for a subpolyline. According to the definition, the distance where the subpolyline starts should be less then the distance where it ends. While making calculations, the real start can be the end point of the "theoretical" subpolyline. Function $formSubPoly$ solves this problem. It takes the start and end distances as a parameters. If the movement direction coincides with the direction of the polyline, the parameters do not need to be exchanged. If the direction is opposite, then the start distance is greater, and it has to be exchanged with the end direction.

Algorithm 7.8 Subpolyline Formation (function $formSubPoly$)

```
Require: INPUT:$pl \in PL, l_s, l_e \in \mathbb{R}, dir \in \{-1, 0, 1\}$. OUTPUT:$spl = (pl, l^{\vdash}, l^{\dashv}) \in SPL$.
1: if $dir = 1$ then
2:     $spl = (pl, l^{\vdash}, l^{\dashv}) \leftarrow (pl, l_s, l_e)$
3: else
4:     $spl = (pl, l^{\vdash}, l^{\dashv}) \leftarrow (pl, l_e, l_s)$
5: end if
6: return $(spl)$
```

7.3 Route Construction

When constructing a route we deal with sequences of elements. We have a sequence of route elements, and a sequence of input GPS points. Thus, we need three operators to make operations on sequences that contain elements of the same type. The first operator is needed to add an element to the end of a sequence. The second operator is needed to return the first element from a sequence. The third operator removes the first element from a sequence.

Definition 7.1. **(Ω Operator)** Let $\Omega : S^* \times S \to S^*$, where S is a set of elements and S^* is the set of all finite sequences of elements from S, be a sequence operator that constructs a sequence by adding an element to the end of a sequence.

Definition 7.2. **(Υ Operator)** Let $\Upsilon : S^* \to S$, where S is a set of elements and S^* is the set of all finite sequences of elements from S, be a sequence operator that returns the first element of a sequence.

Definition 7.3. **(Ψ Operator)** Let $\Psi : S \to S^*$, where S is a set of elements and S^* is the set of all finite sequences of elements from S, be a sequence operator that constructs a sequence by removing the first element from the sequence.

Sequence $s \in S^*$ must have at least one element if we want to use operators Ψ and Υ. Thus, we have to check if s is empty. If the sequence has one element, then operator Ψ returns an empty sequence.

Example 7.1. Let us consider the route $r_c = (RE, uo_s, uo_e, ST)$ (see Definition 6.9), where $RE = (re_1, re_2, re_3)$ is a sequence of route elements. We can add the element re to the end of this sequence by using Ω operator: $\Omega\big((re_1, re_2, re_3), re\big) = (re_1, re_2, re_3, re)$. We can get the first element of the sequence by using Υ operator: $\Upsilon\big((re_1, re_2, re_3)\big) = re_1$. We can remove the first route element by using Ψ operator: $\Psi\big((re_1, re_2, re_3)\big) = (re_2, re_3)$.

These operators help to manipulate the sequences in the route finding algorithm (see Algorithm 7.9). The route finding algorithm is the step of route recording process on the server side described in Section 4. This algorithm constructs sequence RE of route subpolylines analyzing sequence G of GPS points. The algorithm uses two functions, $ValidateRoute$ and $FillGap$, that validate the route when other functions return undefined values.

FillGap As mentioned earlier, information gaps in the GPS sequence can cause situations where the polylines identified for neighboring GPS points are not neighboring themselves. Function $polyId$ cannot identify the polyline. In this case, function $polyFirstId$ is used to identify the polyline. If this polyline is not in the connection area, the gap between two polylines is to be filled. The strategy for how to do this is based on shortest path search in a graph. First of all, the polylines that intersect with the gap's first polyline are retrieved. For each of these polylines, we again retrieve all intersecting polylines. If the gap's second polyline is not among them, the procedure is repeated recursively for each polyline of this level. The search stops when the gap's second polyline is in the result set. When we have intersecting polylines, we take the set of connections and form subpolylines using the connection information. These constructed subpolylines fill the gap between two previously described polylines.

ValidateRoute This function eliminates bad results in projecting points. When we search for the end measure for a subpolyline, we consider its direction and make an approximation according to the direction. But when we do projections, we project onto the same polyline that the previous GPS point is projected onto if this is possible. These two requirements cause that we may jump through a connection, and function $findEnd$ does not turn back to get it. Function $ValidateRoute$ searches for the connection behind. Then it checks if the previous subpolylines are suitable for the current situation. If not then the subpolylines are validated.

The route finding algorithm starts by taking the first GPS point, removing it from the GPS sequence. The first polyline is identified using function $polyFirstId$. Several temporary parameters store values that are necessary to construct a route: parameters $(currPl, currDst)$ are the polyline, the current GPS point is mapped to and the distance from the start of the polyline; $(prevPl, prevDst)$ are the polyline and the distance for the previous GPS point; $(pl, l^{\vdash})$ store values for the start of the constructed subpolyline; dir and $prevDir$ are the current and the previous directions on the polyline. While the sequence of GPS points is not empty, each point is taken from it and analyzed. The polyline is identified for each point using function $polyId$. If this function returns an undefined polyline, it means that there is a gap in information and it is to be filled in. If the function returns the polyline, it is checked if the projection is in the connection area that can cause bad results. If the point projection is not in the connection area, i.e., the result of $possibleConnection$ is $false$ the other calculations can be done.

If the current polyline is not the same (Step 14) as for the previous GPS point, a new subpolyline is formed. First of all, the measure for the end of the polyline is calculated by function $findEnd$. This function may return the undefined end measure if there were bad projections. This fault is eliminated. Using the defined measure, we form a new subpolyline and add it to the sequence RE of route elements. The start measure is calculated for the next subpolyline. The temporary parameters get new values: the direction becomes undefined, the start of the polyline is a newly calculated measure.

If the polyline is the same (Step 26) as for the previous GPS point, we check if the movement direction is the same as until the previous point. If the previous direction was undefined, its value is set to a value of the current direction. If the direction is the same, no calculations are done; only the temporary variable $prevDst$ becomes equal to the distance of the current GPS point. If the direction is not the same, we have to form a subpolyline by taking the previous values. The previous distance becomes the start of the new subpolyline.

When the GPS sequence is empty, the last subpolyline of the route is formed, using the distance from the start of the polyline to the last GPS point.

Algorithm 7.9 Route Finding

```
Require: INPUT: $G = (g_1, ..., g_n), g_i \in \mathbb{R}, n > 1$. OUTPUT: $RE = ((spl_1, dir_1), ..., (spl_m, dir_m))$,
    $m \geq 1$, $spl_i = (pl_i, l_i^{\vdash}, l_i^{\dashv}) \in SPL$.
 1: $RE \leftarrow \emptyset$
 2: $g \leftarrow \Upsilon(G)$
 3: $G \leftarrow \Psi(G)$
 4: $(pl, l^{\vdash}),\ (currPl, currDst),\ (prevPl, prevDst) \leftarrow polyFirstId(g)$
 5: $dir,\ prevDir \leftarrow 0$
 6: while $Gisnotempty$ do
 7:   $g \leftarrow \Upsilon(G)$
 8:   $G \leftarrow \Psi(G)$
 9:   $(currPl, currDst) \leftarrow polyId(g, prevPl)$
10:   if $currPl = \emptyset$ then
11:     $FillGap()$
12:   else
13:     if $possibleConnection(currPl, currDst) =$ false then
14:       if $currPl \neq prevPl$ then
15:         $l^{\dashv} \leftarrow findEnd(pl, currPl, prevDst, prevDir)$
16:         if $l^{\dashv} = \infty$ then
17:           $ValidateRoute()$
18:         else
19:           $spl \leftarrow formSubPoly(pl, l^{\vdash}, l^{\dashv}, prevDir)$
20:           $RE \leftarrow \Omega(RE, (spl, prevDir))$
21:           $l^{\vdash} \leftarrow findStart(pl, currPl, l^{\dashv})$
22:           $pl \leftarrow currPl$
23:           $prevDir \leftarrow 0$
24:           $(prevPl, prevDst) \leftarrow (pl, l^{\vdash})$
25:         end if
26:       else
27:         $dir \leftarrow defineDirection(prevDst, currDst)$
28:         if $prevDir = 0$ then
29:           $prevDir \leftarrow dir$
30:         else if $prevDir = dir$ then
31:           $prevDst \leftarrow currDst$
32:         else
33:           $l^{\dashv} \leftarrow prevDst$
34:           $spl \leftarrow formSubPoly(pl, l^{\vdash}, l^{\dashv}, dir)$
35:           $RE \leftarrow \Omega(RE, (spl, prevDir))$
36:           $prevDir \leftarrow dir$
37:           $l^{\vdash} \leftarrow prevDst$
38:           $prevDst \leftarrow currDst$
39:         end if
40:       end if
41:     end if
42:   end if
43: end while
44: $l^{\dashv} \leftarrow currDst$
45: $spl \leftarrow formSubPoly(pl, l^{\vdash}, l^{\dashv}, prevDir)$
46: $RE \leftarrow \Omega(RE, (spl, prevDir))$
47: return $(RE)$
```

8 Practical Part

This section describes a practical part of our work. The data we tested our algorithms on is presented. The algorithms were implemented using Java and Oracle together with its PL/SQL and Oracle Spatial. The aspects of implementation are analyzed.

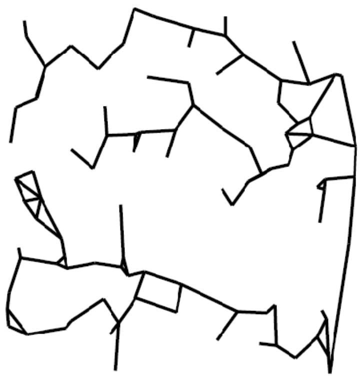
8.1 Data

We tested the algorithms on synthetic data. We wrote a generator (see Appendix D for more detailed explanations) to generate a road network and movement simulation on it. To test the algorithms we used Oracle Spatial operators and functions.

8.1.1 Generated Data

We generated a simple road network (see Figure 21(a)) to search for possible problems while detecting a route. This road network includes 100 randomly generated base points, and 117 polyline segments that use these points form 35 polylines. In our data, a number of segments for any polyline can vary from 1 to 9 segments. There are 49 connections where polylines intersect. The information about such a road network is stored in a database. The schema of the database is described in Section 6.2.

We also generated a number of GPS point sequences (see an example in Figure 21(b)) to simulate a movement of a user on this road network. To have destination points for a route we randomly chose two segments and positions on them. We searched for a path on segments between these two objects using breadth-first search strategy. For each segment on the path we generated GPS coordinates. A number of these GPS points depended on the segment length and on the step size. The points were not exactly on the segment in order to simulate real world situations. Also, the points could be on both sides of the segment and within different distances to that segment. We allowed the predefined imprecision of the GPS point to the segment.


(a) Road network

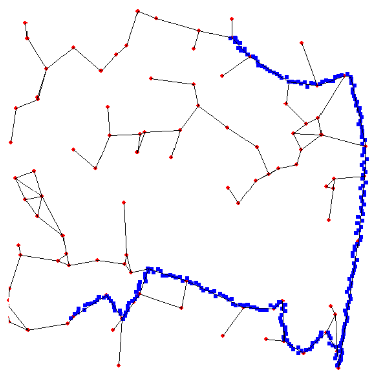

(b) Example of generated GPS coordinates

Figure 21: Generated data

8.1.2 Oracle Spatial

We used Oracle Spatial [13] while implementing our algorithms. As described in Section 6.2, segments of polylines are spatial data objects (see SDO_POLYLINE_ELEMENTS in Figure 14). Thus, we use spatial operators (see Table 1) and geometry functions (see Table 2) for calculations. Polyline segments are also linear referencing system (LRS) elements and we use LR functions (see Table 3).

Spatial operator	Short description
SDO_NN	Determines the nearest neighbor geometries to a geometry.
SDO_NN_DISTANCE	Returns the distance of an object returned by the SDO_NN operator.

Table 1: Spatial operators

Geometry function	Short description
SDO_GEOM.SDO_DISTANCE	Computes the distance between two geometry objects.

Table 2: Geometry functions

Linear referencing function	Short description
SDO_LRS.GEOM_SEGMENT_START_PT	Returns the start point of a geometric segment.
SDO_LRS.PROJECT_PT	Returns the projection point of a point on a geometric segment.

Table 3: Linear referencing functions

To use the Oracle Spatial functions we create an index on the spatial attribute. SQL script that forms a spatial element and inserts it into our database is shown below:

```
INSERT INTO POLYLINE_ELEMENTS VALUES
( 25, 0, 170, 1,
  MDSYS.SDO_GEOMETRY(3302, NULL,NULL,
  MDSYS.SDO_ELEM_INFO_ARRAY(1,2,1),
  MDSYS.SDO_ORDINATE_ARRAY(4130,2630,0,4280,2540,0))
);
```

This example shows how the first element of the polyline 25 is inserted into the database. This element starts at distance 0 and ends at distance 170 from the start of the polyline. A spatial attribute is constructed according to the syntax of the object MDSYS.SDO_GEOMETRY.

8.2 Route Finding with Oracle Spatial

The implemented route finding algorithm is different from the one described in the theoretical part. We use Oracle Spatial to identify the route, and for that we use the built-in functions or SQL statements. The main program is written in Java (see classes in Appendix C). The program uses JDBC to execute SQL queries.

8.2.1 Polyline identification

The first parameter according to which we choose the polyline is the distance to it. Oracle Spatial operator SDO_NN finds the nearest spatial objects and operator SDO_NN_DISTANCE returns the distances to these objects. For the first GPS point we need the nearest spatial object, i.e., polyline element, and also we need its distance from the start of the polyline to the projection of the GPS point. The example shows how the nearest polyline is identified for the first GPS point with coordinates (1570,2320):

```
SELECT a.pol_id, a.pol_from +
               SDO_GEOM.SDO_DISTANCE(SDO_LRS.PROJECT_PT(a.element,
               MDSYS.SDO_GEOMETRY(3001,NULL,MDSYS.SDO_POINT_TYPE(1570,2320,0),NULL,NULL)),
               SDO_LRS.GEOM_SEGMENT_START_PT(a.element),0.0001)  dist
FROM polyline_elements a
WHERE SDO_NN(a.element,MDSYS.SDO_GEOMETRY(3001,NULL,
           MDSYS.SDO_POINT_TYPE(1570,2320,0),NULL,NULL),'sdo_num_res=1',1)='TRUE';
```

The point is projected onto the element (SDO_LRS.PROJECT_PT) that is nearest (SDO_NN). The distance from this projection to the start of the element (SDO_LRS.GEOM_SEGMENT_START_PT) is calculated (SDO_GEOM.SDO_DISTANCE) and added to the distance from the start of the polyline to this element.

To identify polylines for other GPS points, we have function *polyId* (see Appendix B). This function analyzes the same polyline, where the previous GPS point was mapped to. If the distance to the polyline is greater than the imprecision, the function searches for the nearest polyline that intersects with the previous one. For generated GPS sequences we allowed the imprecision of 50 meters. Thus, the identified polyline should always be in less than 50 meters from the GPS point. We check every mapped point if it falls into a connection area. The points mapped to the area around a connection of polylines are ignored. We defined the imprecision of that area, and in our case it is the same as for GPS points. The query (see Appendix A) returns the connection if it is within distance of 50 meters from the point projection in any direction. If the point is not mapped to connection area the further procedure of route detection is done.

8.2.2 Formation of a Route Element

The end of the subpolyline is calculated using the strategy described in Section 7, but with one modification. In the theoretical part, we search for the connection that is ahead of the point projection. The distance from the start of the polyline to the connection becomes the end value for the subpolyline. This distance to the connection is greater than the distance to the point projection if the user travels in the polyline's direction, and less if the direction is opposite. In the implementation, we modify this requirement allowing the connection to be behind, but within less than 50 meters distance. The tests show that this strategy implies better results than the one with the strong requirement. And that is because the points are mapped to the same polyline after crossing the connection area even if the user has turned off to the other polyline. This situation occurs when after crossing the connection area the intersecting polylines are still within a distance less than 50 meters from each other for some time. We have two queries (see Appendix A) that calculate the end distance value for the subpolyline. The programming method controls its usage depending on the movement direction. The end distance value of the subpolyline is used to calculate the start distance value of the new subpolyline (see Appendix A).

8.2.3 Route Construction

The route finding algorithm (see Appendix C) is implemented in the same way as described in Section 7. We have sequences of route elements, i.e., subpolylines, that form the route. We use the built-in Java class *LinkedList* to store these elements. Java programming language supports operations to manipulate these elements in the list. We distinguish between the class that is responsible for the execution of SQL queries and a class that is responsible for route finding. But the second class includes the instance of the second one to be able to get results from the SQL queries.

9 Summary and Future Work

In part because of predominant tendency to develop small mobile devices, i.e., with small keyboards and screens, there is a need to have mobile services be aware of the users' contexts. It is important for mobile services to provide the users only with relevant information, with as little interaction as possible. For mobile service users that travel in road networks, the location context is important.

In this paper, we considered the routes of the users to be an interesting and useful context. We designed a route component that constructs routes based on a user's location information provided by the user's device. We proposed a system architecture for the route component. We introduced the main functions that were necessary in constructing such a component. A database model for road networks and for information about the users and their routes was presented.

The paper analyzed the specific problems that appear while trying to record user's routes. We proposed solutions for all these problems. For some of them, we used already invented techniques. Having the input information as a stream of location coordinates from a GPS receiver, we presented the main algorithms for detecting the route of the mobile user.

There are several possible directions in which to extend this work. In this paper, we assume that the user controls the process of her route recording. But this can be inconvenient for the user as she has to remember to start and stop recording. One of the extensions can be to make the system smart enough to decide if the current position of the user is already the end of the route. For example, if the user is at a particular position for some time without moving, the process of recording can be finished.

Another possible extension can be the creation of a system that is able to detect if the route is already recorded or to divide a long route into smaller ones when small parts of the route are used. Other possible extensions are related to the database schema. To add more functionality, the current database model can be extended by integrating into it some of road network features, like driving directions or turn restrictions. Currently, we do not consider these which may cause faults in the detection of routes. The driving time of the route can also be stored. This feature would add more context-awareness to the system.

10 Acknowledgments

We would like to thank the company Euman A/S for allowing us to use their road database, for sharing their insights into road data management with us, and for constructive comments.

References

[1] American National Standards Institute. *Geographic Information Framework—Data Content Standards For Transportation: Roads*, 2003.

[2] C. Barrett, K. Bisset, R. Jacob, G. Konjevod, and M. Marathe. Classical and Contemporary Shortest Path Problems in Road Networks: Implementation and Experimental Analysis of the TRANSIMS Router. In *Proc. of the 10th European Symposium on Algorithms, ESA*, volume 2461 of *Lecture Notes in Computer Science*, pages 126–138, 2002.

[3] D. Bernstein and A. Kornhauser. *An Introduction to Map Matching for Personal Navigation Assistants*. New Jersey TIDE Center, 1996. http://www.njtide.org/reports/mapmatchintro.pdf.

[4] J. A. Butler and K. J. Dueker. *A Primer on GIS-T Databases*. Center for Urban Studies, Portland State University, 2000. http://www.upa.pdx.edu/CUS/publications/discussionpapers.html.

[5] J. A. Butler and K. J. Dueker. Implementing the Enterprise GIS in Transportation Database Design. *Journal of the Urban and Regional Information Systems Association*, 13(1):17–28, 2001.

[6] CommLinx Solutions Pty Ltd. *Common NMEA Sentence Types*, 2002. http://www.commlinx.com.au/NMEA_sentences.htm.

[7] K. J. Dueker and J. A. Butler. GIS-T Enterprise Data Model with Suggested Implementation Choices. *Journal of the Urban and Regional Information Systems Association*, 10(1):12–36, 1998.

[8] HNIT-BALTIC Geoinfoservice. Digital maps. http://www.maps.lt/.

[9] C. Hage, C. S. Jensen, T. B. Pedersen, L. Speicys, and I. Timko. Integrated Data Management for Mobile Services in the Real World. To appear in VLDB, 2003.

[10] F. Hohl, L. Mehrmann, and A. Hamdan. A Context System for a Mobile Service Platform. In *Proc. of International Conference on Architecture of Computing Systems*, volume 2299 of *Lecture Notes in Computer Science*, pages 21–33, 2002.

[11] C. S. Jensen. Research Challenges in Location-Enabled M-Services. In *Proc. of the International Conference on Mobile Data Management*, pages 3–7, 2002.

[12] D. L. Lee, J. Xu, B. Zheng, and W.-C. Lee. Data Management in Location-Dependent Information Services. *Pervasive Computing, Mobile and Ubiquitous Systems. Context-Aware Computing*, 1(3):65–72, 2002.

[13] C. Murray. *Oracle Spatial User Guide and Reference, Release 9.2*. Oracle Corporation, 2002.

[14] NMEA. *NMEA 0183 Standard*, 2002. http://www.nmea.org/pub/0183/.

[15] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In *Proc. of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, 1995.

[16] P. Scarponcini. Generalized Model for Linear Referencing. In *Proc. of the 7th ACM-GIS International Symposium on Advances in Geographic Information Systems*, pages 53–59, 1999.

[17] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *Proc. of the SSTD International Symposium, Advances in Spatial and Temporal Databases*, volume 2121 of *Lecture Notes in Computer Science*, pages 79–96, 2001.

[18] M. Vazirgiannis and O. Wolfson. A Spatiotemporal Model and Language for Moving Objects on Road Networks. In *Proc. of the SSTD International Symposium, Advances in Spatial and Temporal Databases*, volume 2121 of *Lecture Notes in Computer Science*, pages 20–35, 2001.

[19] W3C. *Extensible Markup Language (XML)*. http://www.w3.org/XML/.

[20] O. Wolfson. Moving Objects Information Management: The Database Challenge. In *Proc. of the International Workshop, Next Generation Informat ion Technologies and Systems*, volume 2382 of *Lecture Notes in Computer Science*, pages 75–89, 2002.

A SQL Scripts

A.1 Creation of View VIEW_INFO

```
-- Creates view using data from table INFO. Weekday and Hour are
-- calculated using built in functions and Quarter is calculated using
-- our PL/SQL function getQuarter

CREATE VIEW VIEW_INFO AS
      SELECT Route_id, Weekday, Hour, Quarter, COUNT(*) datetime
      FROM
       (
        SELECT route_id, TO_CHAR(datetime, ’Day’) as Weekday,
               TO_CHAR(datetime, ’HH24’) as Hour,
               getQuarter(TO_NUMBER(TO_CHAR(datetime, ’MI’))) as Quarter
        FROM INFO
       )
      GROUP BY route_id, Weekday, Hour, Quarter;
```

A.2 Query to Define a Connection Area

```
-- Shows the example how it is checked if the GPS point is in the
-- connection area: some point was mapped to polyline 12 within distance
-- 960 from its start. The query returns any connection that is in
-- distance 50 from the point projection.

SELECT conn_id
FROM connections
WHERE pol_id = 12 AND
     (pol_from - 50 <= 960 AND 960 <= pol_from +50);
```

A.3 Queries to Find the End of a Subpolyline

```
-- Shows the examples how the end distance for the subpolyline is
-- defined. The previous point was mapped to polyline 12 within
-- distance 900. The current point is mapped to polyline 13. The first
-- example defines the end distance if the direction coincides with the
-- polyline direction. The second defines if the direction is
-- opposite.

SELECT  a.pol_from
FROM connections a, connections b
WHERE  a.pol_id = 12 AND b.pol_id = 13 AND a.conn_id = b.conn_id
       AND  (a.pol_from + 50 >= 900 OR 900 < a.pol_from);

SELECT  a.pol_from
FROM connections a, connections b
WHERE  a.pol_id = 12 AND b.pol_id = 13 AND a.conn_id = b.conn_id
       AND  (a.pol_from - 50 <= 900 OR 900 > a.pol_from);
```

A.4 Query to Find the Start of a Subpolyline

```
-- Shows the example how the start of the polyline is defined. The
-- previous subpolyline was on polyline 13 and ended within distance
-- 0. The current subpolyline is on polyline 12. The example
-- identifies the distance on the polyline where the previous
-- polyline intersects with the current one.

SELECT  b.pol_from
FROM connections a, connections b
WHERE a.pol_id = 13 AND a.pol_from = 0
AND b.pol_id = 12 and a.conn_id = b.conn_id;
```

B PL/SQL Functions

B.1 Quarter Calculation

```
-- Calculates a quarter for a value of minutes.

CREATE OR REPLACE FUNCTION getQuarter
 (
   minutes IN NUMBER
 )
RETURN NUMBER
AS
   quarter NUMBER;
BEGIN
  IF minutes BETWEEN 0 AND 14 THEN quarter := 1;
  ELSIF minutes BETWEEN 15 AND 29 THEN quarter := 2;
  ELSIF minutes BETWEEN 30 AND 44 THEN quarter := 3;
  ELSE quarter := 4;
  END if;
  RETURN quarter;
END getQuarter;
/
```

B.2 Polyline Identification

```
-- Identifies the polyline polId where the current point (x,y) is
-- mapped to. The previous polyline prevPl is taken into consideration. The polyline
-- and the distance to the mapped point on that polyline is returned.

CREATE OR REPLACE PROCEDURE polyId
(
prevPl   IN INTEGER,
x        IN NUMBER,
y        IN NUMBER,
polId    OUT INTEGER,
distance OUT NUMBER
)
IS

polFrom NUMBER := -1;
dist    NUMBER := -1;

CURSOR samePl IS
      SELECT pol_from, MDSYS.SDO_NN_DISTANCE(1) dist
      FROM
      (
        SELECT *
        FROM polyline_elements
        WHERE pol_id = prevPl
      )
      WHERE SDO_NN(element,mdsys.sdo_geometry(3001,NULL,
        MDSYS.SDO_POINT_TYPE(x,y,0),NULL,NULL),’sdo_num_res=10’,1) = ’TRUE’
      ORDER BY dist;

CURSOR connPl IS
      SELECT pol_id, pol_from, MDSYS.SDO_NN_DISTANCE(1) dist
      FROM
      (
        SELECT *
        FROM polyline_elements
        WHERE pol_id IN
             (
               SELECT DISTINCT b.pol_id
               FROM connections a, connections b
               WHERE a.pol_id = prevPl AND b.pol_id <> a.pol_id AND
                     a.conn_id = b.conn_id
             )
      )
      WHERE SDO_NN(element,MDSYS.SDO_GEOMETRY(3001,NULL,
             MDSYS.SDO_POINT_TYPE(1800,2040,0),NULL,NULL),’sdo_num_res=10’,1)=’TRUE’
      ORDER BY dist;

BEGIN

polId    := 0;
distance := -1;

OPEN samePl;
FETCH samePl INTO polFrom, dist;

IF ((samePl%FOUND) AND (dist <= 50)) THEN
   polId := prevPl;
   SELECT pol_from + SDO_GEOM.SDO_DISTANCE(SDO_LRS.PROJECT_PT
                (element,MDSYS.SDO_GEOMETRY
                (3001,NULL,MDSYS.SDO_POINT_TYPE(x,y,0),NULL,NULL)),
                SDO_LRS.GEOM_SEGMENT_START_PT(element),0.0001)
   INTO distance
   FROM polyline_elements
   WHERE pol_id = prevPl AND pol_from = polFrom;

   CLOSE samePl;

ELSE
   CLOSE samePl;
   OPEN connPl;
   FETCH connPl INTO polId, polFrom, dist;

   IF ((SQL%FOUND) AND (dist <= 50)) THEN
      SELECT pol_from + SDO_GEOM.SDO_DISTANCE(SDO_LRS.PROJECT_PT
                 (element,mdsys.sdo_geometry
                   (3001,NULL,MDSYS.SDO_POINT_TYPE(x,y,0),NULL,NULL)),
SDO_LRS.GEOM_SEGMENT_START_PT(element),0.0001)
      INTO distance
      FROM polyline_elements
      WHERE pol_id = polId AND pol_from = polFrom;

   CLOSE connPl;
   END IF;
END IF;

END polyId;
/
```

C Route Finder. Java Classes

We have four main classes that are written in order to implement route recording algorithm:

- Class **routeElement**. This class corresponds to the route element in data structures, i.e., a subpolyline with a direction. Thus, the class has four variables: *pol_id*, *pol_from*, *pol_to*, and *direction*. It has two constructors: for creation with default values, and for creation with predefined values.
- Class **tempValue**. This class is used to store values of projections of GPS points and direction according to the previous point. The main program has three instances of the class: for the start of the subpolyline, for the current point, and for the previous point. The class has variables *pol_id*, *dist_from_start*, and *direction*.
- Class **calcRoutes**. This class is responsible for the execution of SQL queries.
  - Methods *identifyPoly* and *identifyFirstPoly* correspond to the algorithms that deal with polyline identification and are described in Section 7. Method *identifyFirstPoly* uses the query described in Section 8 and method *identifyPoly* calls the procedure *polyId* given in Appendix B.
  - Method *possibleConnection* defines if the GPS point is the connection area. The method executes the query (see Appendix A) that returns the connections if it is so.

  - Methods *findEnd* and *findStart* deal with the calculation of the end distance and start distance values for subpolylines.

- Class **find**. The class is responsible for route construction and corresponds to the Algorithm 7.9. The class has the instances of classes **routeElement**, **tempValue**, and **calcRoutes**. The main methods are:

  - *getGps* forms a list of GPS coordinates from the input stream. Currently, it takes the coordinates from the file that is given to the method as a parameter.
  - *findRoute* constructs the route according to the route finding algorithm. This method uses other methods like *formElement* and *defineDirection* to calculate parameters for route elements. The result of the method is the list of route elements.

D Map Generator. Java Classes

We have two main classes that were used to generate our data. Class **mapGenerator** has methods to support map generation:

- Method *generateBasePoints* generates the base points for the road network. The number of base points and the ranges of two dimensional space depend on the input parameters.

- Method *findDensity* calculates the number of base points that fall into the square area around the particular base point.

- Method *generateSegments* defines how many segments for each point should be generated. Method *formSegments* generates the segments for each base point. Method *generatePolylines* generates polylines from the set of the segments. These polylines form our road network.

- Method *generateDstObject* generates a destination object choosing randomly a segment and then any distance on it.

- Method *generateGps* generates a sequence of the GPS points. In order not to have points on the segment, we randomize them using a maximum imprecision.

- Method *generatePath* generates a path in the road network. It calls *generateDstObject* to generate two destination objects. Then it finds the path between these two objects. Finally, the method calls *generateGps* to make a sequence of GPS points.

- Method *generateMap* generates a new map using the previously described methods and writes the results into the files. Method *generateMapFromFile* takes already generated GPS points from the file using *takeCooFromFile* and generates a map.

Class **makeData** has an instance of the class **mapGenerator** and gives the parameters to its public methods to generate a map.