doi: 10.1515/itms-2014-0011
Information Technology and Management Science
2014 / 17

STORN: Solution to Traversal of Road Networks

Janis Kampars [1], Elina Shmite [2], [1, 2] *Riga Technical University*

***Abstract* – The objective of the optimal traversal planning (OTP) is to calculate a route that would provide traversal of all streets in a predefined area. While solving the OTP problem, the total distance of the route should be minimized. This paper presents a solution that provides OTP and the execution of the corresponding plan using a series of loosely coupled web services. A mashup is created based on the provided web services and field tested using a data acquisition case in Riga, Latvia. Other possible application areas are street cleaning, package delivery and evacuation planning.**

***Keywords* – Chinese postman problem, cloud computing, travelling salesman problem, web services.**

I. Introduction

The purpose of this paper is the development of the Solution to Optimal Traversal of Road Networks (STORN). It will be implemented as a distributed system and comprise a series of web services that provide near optimal traversal of road networks using currently a popular Software as a Service (SaaS) delivery model. A mashup utilizing the mentioned web services will be created and tested by a company performing data acquisition in Riga, Latvia. The data acquisition process involves driving on all streets of Riga (bidirectional streets have to be traversed in both directions) and taking photos for later on-site processing. It is important to minimize the time needed for route generation and the total distance, while making sure that all streets have been covered.

The optimal traversal planning problem has other practical applications, some of which are snow plowing [1], police patrol planning [2], parcel delivery [3] and evacuation planning [4]. The STORN will be later adjusted to suit more use cases, promoting emergence of a wide range of optimal traversal planning mashups and development of thin clients for mobile computing platforms like iOS and Android.

The evolution of the technical paradigm called cloud computing (CC) has significantly affected the way information technology services are invented, developed, deployed and scaled [5], [6], and it strongly correlates with the chosen approach for implementing the STORN. Yang et al. have acknowledged the importance of CC in dealing with technical challenges in geospatial sciences and coined the term Spatial Cloud Computing (SCC) [7]. The STORN will follow the best practices of cloud application development and provide its functionality as a service. Study by Gartner [8] predicted that SaaS would become increasingly important and worldwide software revenues for SaaS delivery were said to grow by 19.4 % overall between 2008 and 2013. Web services provided by the STORN can be used to power mobile applications, following another technological trend and potential future of mobile application development – Mobile Cloud Computing (MCC) [9]. Although smartphones have become the devices of the future for computing and service access, they are still constrained by CPU performance, memory and battery capacity. The MCC is used for extending the power of CC to mobile devices by offloading computing or data intense application from mobile devices to scalable and virtually unlimited cloud resources [10].

The development of the STORN involves several challenges – acquisition of the necessary spatial data, identification of suitable algorithms for optimal traversal planning and plan execution, data model transformations to fit the algorithmic requirements, definition of a scalable architecture that can be used to provide optimal traversal of road networks as a service.

The organization of this paper is as follows. Section II reviews the algorithms in the area of OTP that could be used in development of the STORN. Current research in the area of cloud application design is also discussed in this section. Section III presents the algorithmic and architectural solutions for the STORN. Section IV experimentally compares two algorithms for OTP. Section V concludes final remarks and possible directions for the future research.

II. State of the Art

This section reviews related studies in order to develop the needed algorithms for the STORN and choose an appropriate architecture for deploying it in the cloud.

A. Existing Algorithmic Approaches

If the road network is converted into a graph, where street fragments correspond to arcs (directed links) and street intersections are transformed into nodes, we can formalize the development of the OTP as a type of Arc Routing Problem (ARP). ARP is a subtype of vehicle routing problems, in which the tasks to be performed are located on arcs. This problem has been studied less than the node routing problem; however, during the last decade there has been impressive development summarized by [11]. The graph that corresponds to a road network is more similar to one examined in the Mixed Chinese Postman Problem (MCPP) in terms that it contains arcs, edges and nodes; however, our use case requires that bidirectional streets (edges) have to be driven in both directions, while MCPP only requires edges to be traversed in one direction. If the road network is transformed into a graph, where bidirectional streets correspond to two arcs (one for each direction), then it can be solved as a Directed Chinese Postman Problem (DCPP). In DCPP each arc has a cost (e. g., distance of the street fragment) and to solve a DCPP one has to find a route (Chinese Postman Tour or CPT) containing all of the arcs. In order to find the CPT, graph must be strongly connected (any node can be reached from any other node). If the graph is an Eulerian, there is an Euler circuit traversing

each arc exactly once, which is also the optimal CPT. A theorem states that a graph is Eulerian only if all of its nodes are balanced (each node has to have an equal number of inbound and outbound arcs). Finding an Euler circle is a relatively easy task [12]; unfortunately, graphs derived from street networks are rarely Euler and thus getting the CPT is more complex.

There are approaches that search for the CPT in a graph that does not require nodes to be balanced [13], while others solve the Chinese Postman Problem (CPP) by converting the initial graph to an Eulerian one (e. g., performing an Eulerian extension) by adding virtual arcs to the graph so that all nodes are balanced [12], [14]. While performing the Eulerian extension, the total cost of added virtual arcs must be minimized. There are also studies that concentrate solely on Eulerian extension of weighted, directed graphs and it is known to be an NP-hard problem [15]. The general version of DCPP does not take into account that some of the turns in the road network may be prohibited by the traffic regulations (e.g., left turn) and some maneuvers are more complex than others (driving straight is easier than making a left turn).

To deal with this issue, there are studies that recommend transforming the CPP into an equivalent Asymmetric Travelling Salesman Problem (ATSP) [14], [16]. Other studies [17] favor a more direct approach and state that transformation to ATSP is inefficient. To solve the ATSP a route containing all the vertices (e. g., the Hamiltonian cycle) must be found. ATSP is an NP-complete problem [14] and there is a wide range of algorithms for finding near optimal solutions, including branch and bound [18], patching [19], [20] and genetic [21] algorithms.

B. Architecting Applications for the Cloud

Designing applications for the cloud is a relatively new research area and it certainly lacks a methodological support. One of the best and technologically detailed sources for cloud application design patterns is whitepapers by cloud service providers. Amazon has published a document [23] containing best practices for designing cloud native applications. It also provides a variety of reference architectures for various tasks (e.g., web applications, batch processing, media serving); however, these are tightly bound to Amazon AWS platform and might be hard to implement on other platforms.

Hamdaqa et al. [24] present a more abstract version of cloud application architecture, in which cloud application is made of `CloudFrontTask` (accessible by an end-user), `CloudRotorTask` (a background process), `CloudCrossCuttingTask` (provides logging services and monitors cloud resources, which is important for scaling), and `CloudPresistenceTask` (provides persistent storage). Tasks are loosely coupled and communication between them is provided using queues.

Christian Inzinger et al. [25] present a methodology that addresses the complete development lifecycle of cloud applications. The methodology is based on iterative refinement of the application architecture based on requirements, from abstract representation of the application that captures general business requirements to a concrete model of cloud-based application components.

Kwon et al. [26] present a method for transforming traditional applications into cloud-based services. The refactoring process consists of service extraction phase and service interface adaptation. The applicability of the approach is proven by transforming two third party Java applications into cloud-based applications.

Rimal et al. [27] classify architectural requirements for cloud computing systems according to the requirements of cloud providers, the enterprises that use the cloud and end-users. A relation between various cloud deployment models and requirements is also provided.

III. Proposed Solution

This section describes STORN services, provides a high level architecture, a data model and algorithms used in implementing the STORN. We use freely available data from OpenStreetMap as our spatial data source.

A. STORN Web Services

A total of six web services (WS) are provided by STORN. User ID and API ID are mandatory parameters at all times; therefore, they are omitted from the web service parameter list.

Routing region selection WS

It is used to obtain a region ID for the required traversal region that is used as an input in other WS. Required input parameter is a name of the obtainable area. Optional parameters include a country code and administrative level of the area to narrow the search.

Routing region definition WS

It is used to define a custom region to make it available for a route definition. Required parameters include area name and area boundaries. The STORN contains user generated regions, which are accessible only by the owner, and public regions, which are based on Open Street Maps (OSM) data [22] and are accessible by all users.

Route generation WS

It is used to generate the actual road network traversal route and to obtain a route ID to be used in other WS. The only required parameter is a region ID. Navigational costs for each turn type in seconds can be provided optionally. If no navigational costs are set, there are no maneuver preferences (driving straight or making a left turn is equally favorable). Other optional parameters include custom vertex and arc tolerance distances for the active arc detection (see Section C).

Route retrieval WS

It is used to obtain spatial information of the route. The only required input parameter is a route ID. Optionally a filter can be specified to retrieve only the visited arcs of the route or unvisited ones. If no filter is set, the complete route is returned. It is also possible to get corresponding street names and hints, which are not returned by default, by using parameters *showStreetName* and *showHints*.

Route traversal WS

It is used to perform the route traversal procedure. It obtains the list of next arcs (the number of arcs can be adjusted by using an optional parameter) to be visited and hints. Required input parameters include a route ID and current coordinates of the driver. Coordinates are used to update the progress of the route traversal. This WS can be used to develop clients for mobile devices. During the research an HTML5 based frontend was implemented based on this service and the Route retrieval WS.

Route progress tracking WS

It is used to monitor the route traversal process by obtaining driver's current location, speed, information about deviation from the generated route and the length of the traversed and remaining distances. The only input parameter required is a route ID.

B. STORN Architecture

STORN has been designed to employ the benefits provided by cloud computing, and its architecture is given in Fig. 1.

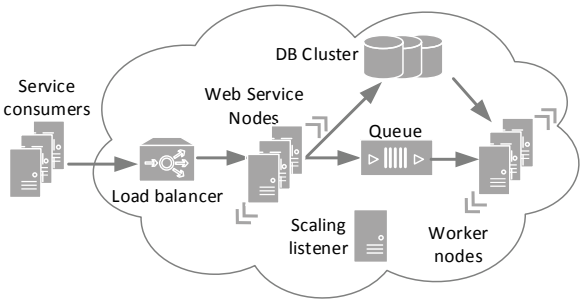

Fig. 1. High level architecture of the STORN.

Web services run on virtual machines that contain the necessary program logic and environment (e. g., Apache web server). The requests from service consumers are distributed to web services using a load balancer. Based on the nomenclature by [24] web service nodes are instances of `CloudFrontTask`. To provide high availability and performance, the minimum number of web service nodes is at least two at all times. The total number of web service nodes is being managed by a scaling listener (an instance of `CoudCrossCuttingTask`). When the average response time grows above a globally defined constant, new nodes are added. Similarly, the number of nodes is decreased when the average response time falls below a globally defined constant, but not lower than two nodes.

A Postgres-XL database cluster is used to provide a persistent data store. It is horizontally scalable, fully ACID and compatible with regular PostgreSQL and PostGIS. PostgreSQL/PostGis is community's preferred database management system in working with OpenStreetMap data, which made compatibility with PostgreSQL/PostGis an important criterion. Based on the nomenclature by [24] database cluster is an instance of `CloudPresistenceTask`.

Time and resource consuming tasks are performed by worker nodes (an instance of `CloudRotorTask`). Messages describing the tasks are added to a queue by web service nodes. These messages are later pulled from the queue by worker nodes and the required processing is performed. In our case, route generation (initiated by the Route generation WS) is implemented using worker nodes. The scaling listener monitors the length of the queue and adjusts the number of worker nodes accordingly. To provide high availability, there are at least two worker nodes at all times. Worker nodes are implemented as virtual machines, which contain the application logic and environment to execute the needed tasks. During the task execution data are retrieved from the persistent data store and at the end of the process results are written back. Worker nodes are stateless, which means that unrecoverable errors during task execution pose no danger. After a certain visibility period the message containing the instructions for task execution will reappear in the queue and will be processed by another worker node.

C. Data Model

A local copy of Riga, Latvia, highway data is obtained and loaded into a PostgreSQL database. Initial data are loaded in four tables, which serve as a base for further data model transformations. These tables are *planet_osm_lines* (streets, roundabouts, links etc.), *planet_osm_points* (traffic lights, intersection points etc.), *planet_osm_relations* (used as a source for navigational restrictions) and *planet_osm_polygon* (used as a source for public regions). Initial tables are adjusted and transformed according to STORN requirements. Resulting database model is given in Fig. 2.

Table *graph_arcs* corresponds to the Eulerian graph and contains relations between two records from *osm_points*, which form an arc. For each arc `[u;v]` a distance that equals to `Wuv` is stored. Each bidirectional arc is stored as two separate arcs – one for each direction. This table is used by Fleury's algorithm (See Section D). Tables *graph_e_nodes* and *graph_e_arcs* correspond to Expanded Line Graph and both are based on the *graph_arcs* table and used by Patched Cycle algorithm (See Section D).

Navigational restrictions are retrieved from *planet_osm_relations* and are available in two different forms. One of them defines only the allowed turns from a single or multiple ways (line geometry) via a specific node (point geometry) to a single or multiple ways. Restrictions are defined as string literals that start with a keyword "only" (e. g., "only_straight_turn"). The second approach for storing navigational constraints defines only prohibited turns and starts with a keyword "no" (e. g., "no_left_turn"). For our purposes, we transform every restriction record with many *from* or *to* entries into multiple restriction records with a single *from* and *to* entry.

Table *regions* contains OSM predefined set of administrative territories from *planet_osm_polygon*. Routing region definition WS users are able to add custom traversable regions to this table. If the region is user defined, user ID field contains the ID of the corresponding author (this field is left blank for publicly available regions retrieved from *planet_osm_polygon*).

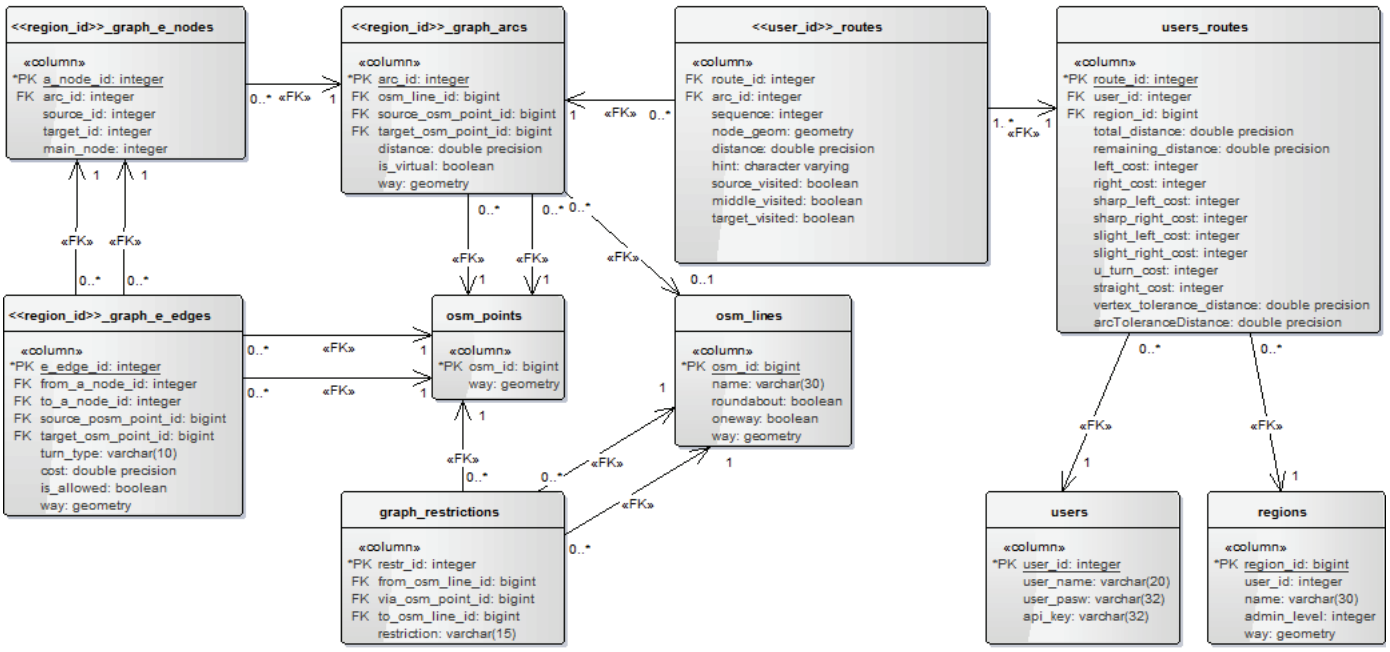

Fig. 2. Database entity relationship diagram of the STORN.

For each route a region ID, navigational costs and active arc detection tolerance parameters are stored in *users_routes*. Because of the different input parameters, tours created for the same regions are not the same and therefore cannot be reused as easily as graph tables.

Users can have more than one route each generated with different parameters. A separate *route* table is generated for each user. It contains the route definition and columns needed for storing route traversal progress.

D. Algorithms

We have based our solution on algorithms and graph models provided in [14]. In addition to navigational constrains, STORN encompasses prohibited turns, driving directions and route tracking.

If `[u;v]` is the previous traversed arc and `[v;n]` is the current traversable arc, we define a function `nav(u;v;n)` to get a type of the turn to be made. Possible values for the turn type are "straight", "left", "right", "sharp right", "sharp left", "slight right", "slight left"," u-turn" and "prohibited turn".

When the function checks if the turn is prohibited, the input of source arc and target arc are used to search for affecting restrictions. A restriction is found if *From* way ends with a source arc and the *via* node matches both the source arc's end point and target arc's start point. If a restriction is found, its type is further reviewed. If the type is "only_" and *to* way starts with a target arc or the type is "_no" and *to* way does not start with a source arc, turn is allowed, otherwise – turn is prohibited.

We define function `ncost(u;v;n)` to get a navigational cost for the turn made. Navigational costs used by the algorithms are set via the Route definition WS. Term path cost refers to both navigational costs and arc costs to traverse a given path.

Solving OTP is an optimization problem. Because of the presence of navigational constrains and prohibited turns in our solution we optimize time spent traversing the route. Two alternative algorithms for calculating OTP that work with two different graph models are described in this section – the greedy algorithm (GA) and the approximate algorithm (AA). We use Fleury's algorithm for the GA with added heuristics to minimize the cost of the Eulerian cycle and we use simplified Patched Cycle algorithm [19] for the AA with added heuristics to minimize the cost of the Hamiltonian cycle.

Graph balancing approach

Both GA and AA require a balanced graph, so first we check if the graph is strongly connected and remove any vertices, which are creating dead-ends. Vertice removal is done by searching for vertices that contain only inbound or outbound arcs. The procedure is repeated while all vertices have at least one inbound and outbound arc.

Further the initial graph `G` is converted to an Eulerian graph `Ge` by balancing all of its vertices. Path costs needed by the algorithm are calculated by implementing a shortest path Dijkstra-like algorithm [28]. The classical version of Dijkstra is not applicable [29] in the given graph with prohibited turns as there are possible situations when an already visited node must be included in the graph traversal route to reach the destination node (otherwise no cycle can be found).

The greedy algorithm

Fleury's algorithm removes a single arc at a time. If removing an arc from the graph makes it disconnected, the arc is called a bridge. Bridge is removed only if there is no other non-bridge arc to traverse. Heuristics are applied to make a minimum cost choice between two or more possible arc selections. The pseudocode of the GA is given below:

```
Algorithm(graph Ge, starting vertex s)
01. OTP = {s}; Bridge_children = {};
Nonbridge_children = {};
02. while there exists an unvisited arc
03.  for each unvisited arc [s;n]
04.   if [s;n] = bridge arc of Pss
05.    append n to Bridge_children;
06.   else
07.    append n to Nonbridge_children.
08.  if Nonbridge_children is not empty
09.    from all Nonbridge_children vertices x
10.    m = the vertex with the minimum ncost(p;s;m)
11.   else if Bridge_children is not empty
12.    from all Bridge_children vertices y
13.    m = the vertex with the minimum ncost(p;s;m)
14.   else
15.    replace any virtual arc [u;v] with the
       vertices of shortest path Puv
16.     for each arc pair ([u;v], [v;n])
17.      if nav(v;u;n) =  'restricted turn'
18.       replace [v;n] with the vertices of
          shortest path Pvn.
19.     return OTP.
20. mark [s;m] as visited;
21. p = s; //previous parent p
22. s = m; //new parent s
23. bridge_children = {};
24. non_bridge_children = {};
```

In a formed tour we try to replace any detected restricted turns [v;u;n] with the shortest path from v to n.

The approximate algorithm

To implement the AA graph Ge is converted to an Extended Line Graph (ELG) G*l. Instead of modeling intersections and the roads connecting them, ELG is used to model roads and turns from one to another [28]. For each arc [u;v] in graph Ge we add an arc [uuv;vuv] with a cost w_{uv} from Ge to G*l. After this step all the basic arcs are transferred. To add the turns for each vertex v_{uv} from G*l all the possible arcs [v_{uv};v_{vx}] with ncost[u;v;x] are added.

AA consists of two phases. At first, the AP is solved to match every vertex with its best neighbor. Finally, the resulting subcycles are patched with the minimum cost to make a single cycle using the Karp-Steel patching [19], [20]. Solving an AP in a large dataset can be a very time-consuming task. To minimize the execution time thus contributing to performance improvements, we create subcycles with Nearest Neighbor Algorithm using the same heuristics as Fleury's Algorithm. Optimization is done only at the second phase solving an AP iteratively patching the two largest subcycles at a time. The pseudocode of the AA is given below:

```
Algorithm(graph G*l, starting vertex s)
01. Subcycles={}; CurrentCycle={};
02. while there exists an unvisited vertex
03.  if CurrentCycle is empty
04.   v = the arbitrary chosen, unvisted vertex;
05.   append v to CurrentCycle;
06.   mark v as visited;
07.  else
08.   from all the unvisited vertices x such that
      there exists an arc [v;x]
09.   c = the vertex with minimum wvx.
10.    if c is null
11.     append CurrentCycle to Subcycles;
12.     CurrentCycle = {};
13.    else
14.     append c to CurrentCycle;
15.     v = c;
16.     mark c as visited.
17.     OTP={};
18. while subcycles count != 1
19.   replace any virtual edge [u;v] with the
      vertices of shortest path Puv
20.   SC1; SC2 = the two largest subcycles;
21.    [u;v] = the arc from SC1 and [n;m] = the arc
       from SC2 with the minimum sum of wum + wnv -
       wuv - wnm;
22.    delete [u;v] from SC1;
23.    delete [n;m] from SC2;
24.    append [u;m], [n,v] to SC1;
25.    join SC1 with SC2.
26.    for each arc [uvu,uun]
27.     if nav(v;u;n) = 'restricted turn'
28.      replace [uvu;uun] with the vertices of
         shortest path PuvuUun.
29. order SC1 starting from s;
30. OTP = SC1;
31. return OTP;
```

Hint generation approach

Hints are short keywords that show the next maneuver to be performed. They are used in combination with street names of the previous and current arcs, remaining route distance, distance to the next maneuver and other route related information. Hints are generated based on the following eight rules:

1. If there is no previous arc to determine a turn (first arc of the tour), the hint contains direction of the arc (e. g., west).

2. If the next arc is the only possible path from the current arc, no hints are generated.

3. If the next arc is included in a roundabout and current one is not, generate hint "enter roundabout".

4. If the last hint was "enter roundabout" it is determined which exit has to be taken. All allowed exits are counted starting from the roundabout entrance to the desired exit. A hint "exit-n" is generated, where n is the exit number.

5. If the current arc [u;v] is on the same road as the next arc [v;n] and nav(u;v;n) equals "straight", "slight left" or "slight right", no hint is generated. It is natural to continue driving on the same street straight forward or with a slight turn, therefore no guidance is needed.

7. If the current arc [u;v] is not on the same road as the next arc [n;v] and nav(u;v;n) equals "straight", "slight left" or "slight right", generate hint "continue".

8. If for the current arc [u;v] and next arc [v;n] nav(u;v;n) equals "left", "right", "sharp left", "sharp right" or "u-turn", the hint accordingly equals to nav(u;v;n).

Active arc detection approach

During the route traversal, the progress is saved in the *routes* table. To detect which arcs have been traversed, an active arc detection algorithm is defined. Active (current) arc is the arc which according to the previously generated route is being traversed at the moment and has not been marked as

finished. Three edges on the arc are identified – start, middle and end. Each of these can be marked as traversed, when visited by a driver. Active arc detection starts with checking whether the driver's distance to the traversable arc is less than a predefined constant – arc tolerance. If it is true, it is further evaluated if the driver's current distance to the next traversable edge (arc start, middle or end) is less than a predefined distance – vertex tolerance. If it is true, the edge (e. g., middle) is marked as finished. Edges can be marked as finished only by going from the start to the end. When all three edges are finished, the arc is marked as finished. Arc and vertex tolerance constants are specified by the route generation WS and used for dealing with problems caused by spatial data and GPS inconsistencies.

IV. Experimental Results

The experiments to compare the GA and the AA were conducted on a 1491 node large dataset with arc total distance of 72364.80 m, including virtual arcs – 85176.65 m. While calculating the path cost, the average driving speed is defined as 50 km/h.

The gathered results from using GA and AA on two different graph models are presented in Table I, the corresponding turn type counts are presented in Table II. The effectiveness of both approaches is evaluated in terms of path cost. Since both solutions use the same balanced graph, minimizing path cost also minimizes turn costs (chooses preferred maneuver types when possible).

TABLE I

Greedy and Approximate Solution Comparison

	Greedy solution		Approximate solution	
	Expanded route	Replaced restrictions	Applied patching	Replaced restrictions
Distance (m)	85176.65	101954.76	85176.65	103705.00
Turn costs (sec)	17075.00	17935.00	14020.00	12715.00
Prohibited turns	33	17	25	0
Path cost (h)	6.46	7.03	5.61	5.34

Total path distance for both approaches before restriction replacement is equal as both algorithms at this phase visit corresponding graph arcs or nodes (ELG nodes) at a single time. Turn costs are lower for the AA because of the applied optimization while patching subcycles.

Prohibited turns are not fully replaced in GA. Using the simple graph model inserting shortest path replacements or expanding virtual arcs can create new prohibited turns in the start and end of the path. Since the ELG model has the information about the previous turn made, it is possible to replace every restriction and expand virtual arcs without creating new restrictions.

Both algorithms favor driving straight, therefore straight driving has mainly been chosen in both cases. GA right and left turn count is almost the same, while AA has used more right turns. GA with Fleury's algorithm has more situations where there are possible better turns to make but you have to choose a more expensive non-bridge arc just to finish the algorithm execution.

TABLE II

Turn Count Comparison

Turn type	Turn cost	Greedy solution	Approximate solution
Straight	0	1746	1428
Right	5-15	203	232
Left	55-65	204	141
U-turn	80	3	8
Prohibited turn	240	17	0

AA takes more time to execute because solving the optimal assignment while patching is a time-consuming task; however, because of the transformed graph model and even using a partial optimization, AA path cost is lower than GA, thus the resulting route is considered to be more efficient.

V. Conclusion

This paper develops the STORN that provides OTP and the execution of the corresponding plan using a series of loosely coupled web services. The high level architecture and database model for the solution are defined. Two alternative OTP generation algorithms were tested and the AA was proved to be superior.

A mashup is created based on the provided web services and field tested using a data acquisition case in Riga, Latvia. Currently available feedback has shown that vertex and arc tolerance distances play an important role in city traversal and need to be adjusted by field testing (constants may vary for different cities). These are currently specified manually; however, a web service providing automatic calibration of these parameters should be developed. Another issue is dealing with road obstacles and construction works that are not considered during route planning and are discovered only while traversing the route. Currently the user interface of the mashup contains a fast-forward button to manually mark the next arcs as visited, but a more complex approach should be used (e. g., automatic route recalculation by using real-time data provided by the driver).

References

[1] G. Liu, Y. Ge, T. Z. Qiu, and H. R. Soleymani, "Optimization of snow plowing cost and time in an urban environment: A case study for the city of Edmonton," *Canadian Journal of Civil Engineering*, vol. 41, 2014, pp. 667–675. http://dx.doi.org/10.1139/cjce-2013-0409

[2] S. S. Chawathe, "Organizing Hot-Spot Police Patrol Routes," in *Intelligence and Security Informatics*, 2007 IEEE, 2007, pp. 79–86.

[3] L. Xintong and P. Feng, "The shortest path and spatial decision support system implementation in the context of parcel delivery," in *Geoinformatics, 2010 18th International Conference*, 2010, pp. 1–6.

[4] M. Saadatseresht, A. Mansourian, and M. Taleai, "Evacuation planning using multiobjective evolutionary optimization approach," *European Journal of Operational Research*, vol. 198, 2009, pp. 305–314. http://dx.doi.org/10.1016/j.ejor.2008.07.032

[5] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing – The business perspective," *Decision Support Systems*, vol. 51, Apr. 2011, pp. 176–189. http://dx.doi.org/10.1016/j.dss.2010.12.006

[6] J. Gao, P. Pattabhiraman, X. Bai, and W. T. Tsai, "SaaS performance and scalability evaluation in clouds," in *6th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2011, Irvine, CA*, 2011, pp. 61–71.

[7] C. Yang, M. Goodchild, Q. Huang, D. Nebert, R. Raskin, Y. Xu, et al., "Spatial cloud computing: How can the geospatial sciences use and help shape cloud computing?," *International Journal of Digital Earth*, vol. 4, 2011, pp. 305–329. http://dx.doi.org/10.1080/17538947.2011.587547

[8] A. Benlian and T. Hess, "Opportunities and risks of software-as-a-service: Findings from a survey of IT executives," *Decision Support Systems*, vol. 52, pp. 232–246, Dec. 2011. http://dx.doi.org/10.1016/j.dss.2011.07.007

[9] S. S. Qureshi, T. Ahmad, K. Rafique, and I. Shuja Ul, "Mobile cloud computingas future for mobile applications – Implementation methods and challenging issues," in *2011 IEEE International Conference on Cloud Computing and Intelligence Systems, CCIS2011, Beijing*, 2011, pp. 467–471.

[10] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," *IEEE Communications Surveys and Tutorials*, vol. 15, pp. 1294–1313, 2013. http://dx.doi.org/10.1109/SURV.2012.111412.00045

[11] A. Corbeŕn and C. Prins, "Recent results on arc routing problems: An annotated bibliography," *Networks*, vol. 56, pp. 50–69, 2010.

[12] H. Thimbleby, "The directed Chinese postman problem," *Software – Practice and Experience*, vol. 33, pp. 1081–1096, 2003. http://dx.doi.org/10.1002/spe.540

[13] E. Minieka, "Chinese postman problem for mixed networks", *Management Science*, vol. 25, pp. 643–648, 1979. http://dx.doi.org/10.1287/mnsc.25.7.643

[14] L. Kazemi, C. Shahabi, M. Sharifzadeh, and L. Vincent, "Optimal traversal planning in road networks with navigational constraints," in *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, 2007, pp. 138–145.

[15] F. Dorn, H. Moser, R. Niedermeier, and M. Weller, "Efficient algorithms for eulerian extension and rural postman," *SIAM Journal on Discrete Mathematics*, vol. 27, pp. 75–94, 2013. http://dx.doi.org/10.1137/110834810

[16] D. Soler, E. Martínez, and J. C. Micó, "A transformation for the mixed general routing problem with turn penalties," *Journal of the Operational Research Society*, vol. 59, pp. 540–547, 2008. http://dx.doi.org/10.1057/palgrave.jors.2602385

[17] J. Clossey, G. Laporte, and P. Soriano, "Solving arc routing problems with turn penalties," *Journal of the Operational Research Society*, vol. 52, pp. 433–439, 2001. http://dx.doi.org/10.1057/palgrave.jors.2601052

[18] S. Almoustafa, S. Hanafi, and N. Mladenović, "New exact method for large asymmetric distance-constrained vehicle routing problem," *European Journal of Operational Research*, vol. 226, pp. 386–394, 2013. http://dx.doi.org/10.1016/j.ejor.2012.11.040

[19] G. Jäger and W. Zhang, "A SAT based effective algorithm for the directed Hamiltonian cycle problem," in *5th International Computer Science Symposium in Russia, CSR 2010 vol. 6072 LNCS, ed. Kazan*, 2010, pp. 216–227.

[20] M. Turkensteen, D. Ghosh, B. Goldengorin, and G. Sierksma, "Iterative patching and the asymmetric traveling salesman problem," *Discrete Optimization*, vol. 3, pp. 63–77, 2006. http://dx.doi.org/10.1016/j.disopt.2005.10.005

[21] B. Freisleben and P. Merz, "Genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems," in *Proceedings of the IEEE Conference on Evolutionary Computation*, 1996, pp. 616–621. http://dx.doi.org/10.1109/ICEC.1996.542671

[22] "Downloading data," Aug. 9, 2014. [Online]. Available: http://wiki.openstreetmap.org/wiki/Downloading_data [Accessed: Oct. 5, 2014].

[23] J. Varia, "Architecting for the Cloud: Best Practices," 2001. [Online]. Available: http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf [Accessed: Sept. 20, 2014].

[24] M. Hamdaqa, T. Livogiannis, and L. Tahvildari, "A reference model for developing cloud applications," in *CLOSER 2011 – Proceedings of the 1st International Conference on Cloud Computing and Services Science*, 2011, pp. 98–103.

[25] C. Inzinger, S. Nastic, S. Sehic, M. Vogler, F. Li, and S. Dustdar, "MADCAT: A methodology for architecture and deployment of cloud application topologies," in *8th IEEE International Symposium on Service Oriented System Engineering, SOSE 2014, Oxford*, 2014, pp. 13–22.

[26] Y. W. Kwon and E. Tilevich, "Cloud refactoring: Automated transitioning to cloud-based services," *Automated Software Engineering*, vol. 21, pp. 345–372, 2014. http://dx.doi.org/10.1007/s10515-013-0136-9

[27] B. P. Rimal, A. Jukan, D. Katsaros, and Y. Goeleven, "Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach," *Journal of Grid Computing*, vol. 9, pp. 3–26, 2011. http://dx.doi.org/10.1007/s10723-010-9171-y

[28] R. Geisberger and C. Vetter, "Efficient routing in road networks with turn costs," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* vol. 6630 LNCS, ed, 2011, pp. 100–111.

[29] D. Luxen, "Flexible route guidance through turn instruction graphs," in *MapInteract 2013 – Proceedings of the 1st ACM SIGSPATIAL International Workshop on MapInteraction*, 2013, pp. 78–83.

Jānis Kampars obtained his Doctoral Degree in IT from RTU, Latvia in 2012. He works as an Assistant Professor at RTU and his areas of interest are application integration, web development and cloud computing.
E-mail: Janis.Kampars@rtu.lv.

Elīna Šmite has earned her B. Sc. (in the year 2013) degree in IT from RTU, Latvia. She has been working at RTU since 2013 as a Junior Researcher. Her areas of interest are web/mashup technologies.
E-mail: Elina.Smite@rtu.lv.

Jānis Kampars, Elīna Šmite. RCTI: izbraucama ceļu tīkla risinājums
Optimāla izbraukšanas plāna (OIP) mērķis ir noteikt maršrutu, kas nodrošinātu visu ielu izbraukšanu iepriekš noteiktā apgabalā. Risinot OIP problēmu, īpaša vērība jāpievērš kopējā maršruta izbraukšanas laika minimizēšanai. Svarīgs faktors ir maršruta ģenerēšanas laiks, kas strauji pieaug līdz ar ģeogrāfiskā apgabala palielināšanu. Ģenerējot maršrutu, ir jāņem vērā, ka daži pagriezieni var būt aizliegti un ka divvirzienu ielas ir nepieciešams izbraukt abos virzienos. Iespējamās jomas OBP izmantošanai ietver ielu tīrīšanu, pasūtījumu piegādi, evakuācijas plānošanu, policijas patrulēšanas plānošanu un citas. OIP noteikšanai no OpenStreetMaps tiek iegūti telpiskie dati, kas tiek pārveidoti grafa veidā (ielas segments atbilst grafa lokam, bet krustojums – virsotnei). Šāda grafa pilnīgai šķērsošanai ir nepieciešams apmeklēt visas tā malas vismaz vienu reizi. Rakstā tiek aplūkoti un eksperimentāli novērtēti divi atšķirīgi OIP izbraukšanas plāna iegūšanas algoritmi. Balstoties uz šiem algoritmiem, tiek definēts Risinājums ceļu tīkla izbraukšanai (RCTI). RCTI ir mākonī bāzēta sistēma, kurai piekļuve tiek nodrošināta ar vairāku tīmekļa servisu palīdzību. Tīmekļa servisus var izmantot jaunu izbraucamo reģionu definēšanai, maršrutu ģenerēšanai, maršrutu izbraukšanas atbalsta nodrošināšanai un maršruta izsekošanai. Rakstā ir definēta RCTI arhitektūra un datubāzes modelis. RCTI prototipa darbība ir tikusi testēta reālos apstākļos Rīgā, Latvijā, veicot maršruta ģenerēšanu un daļēju izbraukšanu.

Янис Кампарс, Элина Шмите. РПДС: Решение дорожных сетей для проезда
Цель плана оптимального проезда (ПОП) - обнаружить маршрут, который обеспечил бы проезд по всем улицам на заранее определённом участке. Решая проблему ПОП, особое внимание следует обратить на минимизацию времени проезда по всему маршруту. Ещё один важный фактор - время разработки маршрута, которое быстро возрастает вместе с увеличением соответствующего географического участка. Разрабатывая маршрут, нужно принимать во внимание, что некоторые повороты могут быть запрещены, и что на двусторонних улицах движение идёт в двух направлениях. Возможные области применения ПОП: чистка улиц, доставка посылок, планирование эвакуации, планирование маршрута полицейского патруля и другие. Для определения ПОП с OpenStreetMaps собираются пространственные данные, которые превращаются в графу (сегмент улицы соответствует дуге графы, а пересечение - вершине). Для полного обхода такой графы необходимо посетить все его стороны, по крайней мере, один раз. В статье рассмотрены и экспериментально оценены два разных алгоритма получения плана проезда ПОП. Основываясь на этих алгоритмах, определяется решение для проезда дорожных сетей (РПДС). РПДС - это система, расположенная в облаке, доступ к которой обеспечен с помощью множества Web-сервисов. Web-сервисы можно использовать для определения новых участков для проезда, разработки маршрута, обеспечения поддержки проезда и отслеживания маршрута. В статье определена РПДС архитектура и модель базы данных. Функционирование прототипа РПДС протестировано в реальных условиях - был разработан маршрут и частичный проезд в городе Рига (Латвия).