SAINT-PETERSBURG UNIVERSITY

Gershteyn Arkady Mikhail

Thesis: **Software tools for building safe transport routes**

Scientific specialty 2.3.5. Mathematical and software support of computer systems, complexes and computer networks

Translation from Russian

DISSERTATION

for the academic degree of Candidate of Physical and Mathematical Sciences

Scientific Supervisor:

Doctor of Physics and Mathematics, professor,

Andrey Nikolaevich Terekhov

Saint Petersburg

2024

Table of Contents

Introduction

"Yeah, bye," said Harry and instead of turning right, he turned left, taking a longer but safer route up to the Owlery.
J.K. Rowling, «Harry Potter and the Order of the Phoenix»

In this work, a software package has been created that allows one to construct safer routes for vehicles. The need to create such a software package arises from the fact that traffic accidents lead to immense human mortality and injuries. According to data from the WHO (World Health Organization) [https://ww.who.int/news-room/fact-sheets/detail/road-traffic-injuries] about 1.3 million people worldwide die annually in traffic collisions and about 50 million are injured to various degrees of severity.

Therefore, the task of reducing the number of road traffic accidents (RTAs) and, consequently, of reducing the number of deaths and injuries in road traffic accidents seems very important and consequential. This is also evidenced by the numerous works on the subject (see Chapter 1, devoted to the literature review). Here, in the Introduction, I will consider the most obvious approaches to solving this problem, applied in this work. The first task that comes to mind is to collect data on Traffic Accidents (TA) for several years (coordinates, severity, etc.), select clusters (i.e. places of substantially increased concentration of TA) from these data — and then bypass these clusters when building a safer route.

To move further, it is necessary to understand in general terms how vehicular routing is designed. Although in reality there are road segments and intersections, however all sorts of navigators use another, more abstract representation of roads in the city — a road network consisting of edges and vertices, and called a graph [1]. Edges replace road segments, and vertices indicate the transition from one segment to another (as a result, for example, of a change of direction). Vertexes are also present at road intersections, where the vertex marks the place where several road segments meet.

The road network is a directed graph, since many edges allow vehicles to pass only in one direction. There are also bidirectional edges in the road network, along which it is feasible to move in both directions.

Now let each edge of the road graph have an attribute, for example, the length of the edge. If one also knows the starting and ending vertices of the route, then one of the routing algorithms on the road graph (for example, Dijkstra's algorithm [2] is able to build a route of minimal length from the initial vertex to the final one. This route may turn out to be the fastest, but the routing algorithm does not take into account its security in any way. If one takes as an attribute of the edge the number of accidents that occurred on the corresponding segment of the road, calculated over several previous years, then such a route, although it will be the safest, since it minimizes the number of accidents along the route, is unlikely to be the shortest[1]. Obviously, an approach is needed that would make it possible to build a compromise route — safer, and at the same time not very long (compared to the minimum length route). Chapter 1 examines various approaches to achieving this goal, but here in the Introduction I will discuss our original approach to this problem.

The essence of this approach is to identify TA hotspots (TAHS) on the traffic route and then bypass them. I define TAHS as a compact area with an increased concentration of traffic accidents. Such an area can be a cluster detected by one of the numerous clustering algorithms [3] or a whole road segment with an increased number of TAs.

Approaches to clustering

In this Thesis, the well-known DBSCAN algorithm [4] is used for clustering, since it detects compact (with a given maximum distance between points) clusters of any shape. The resulting clusters will have a different size (the number of TAs in the cluster) from the specified minimum to the maximum determined by the data itself. Obviously, not all

[1] Estimates show that the length of the route calculated in this way will be 2-3 times longer than the minimum, which is hardly practical.

the clusters obtained need to be taken into account, since some of them will be just random noise, and if one had a second sample of accidents, clustering would show that only a part of the clusters is stably located in the same places, while the rest change their position.

In order to eliminate cluster noise, statistical tests and verification of statistical hypotheses are used in this Thesis. In other words, clusters obtained from real TA data are compared with clusters obtained using the same number of points evenly distributed over the road network. Having generated several hundred sets of uniformly distributed points, with the number of points in each set equal to the number of real accidents, and having carried out clustering for each such set, I will receive material for testing statistical hypotheses and will be able to identify "statistically significant" clusters [5]).

Using Statistically Significant TA Clusters

The resulting clusters can be easily interpreted due to the fact that when obtaining them, the maximum distance between accidents is set to 10 meters. Due to such a short distance, clusters very often look like "clots" of TAs located at crossroads, road intersections, where vehicles move from one line to another, etc. These locations require investigation and taking administrative measures (lowering the speed, increasing the transit time, etc.) designed to reduce the number of TAs there.

It is also possible to improve road safety by simply bypassing the selected clusters of accidents. If, for example, the road graph is modified in such a way that the edges (road segments) leading to the cluster receive an additional length (penalty)[2], the routing algorithm will take this into account and, if possible, bypass the cluster. For details on finding a route that bypasses the accident clusters, see Chapter 3.

[2] The penalty can also be practically infinite (a very large number). Then it will become prohibitive and the algorithm for finding the route will be forced to avoid such edges.

Road segments as TAHS

Cluster avoidance, as I have outlined, is about bypassing the designated road segments. Where does the question come from: is it possible to highlight «dangerous» road segments without any clusters? The answer is almost obvious: select those edges of the road graph (e.g. road segments) where the number of accidents exceeds a certain threshold defined by statistical tests, mark these edges as «dangerous», i.e. add an additional length (penalty) which will be considered by the routing algorithm when laying the route from the starting point to the end point. Note that this approach seems more natural, since the road graph itself is modified without involving additional objects (TA clusters). Clustering is also associated with a lot of manual work: it is necessary to consider each cluster and mark the edges of the road graph leading to it. Automation of this task is very cumbersome. See Chapters 4 and 5 for details of the route design that bypasses «dangerous» road segments.

Routing efficiency evaluation

Once the route is laid on the modified graph, the next problem arises: how to evaluate the effectiveness of the new routing, how much does it make traffic safer? To do this, firstly, I need to somehow create a set of routes covering the entire road graph, and, secondly, to use the average efficiency of the new routing.

In this Thesis, a computer simulation of all possible routes is constructed by overlaying a square grid on the entire road graph network. Then all points of the grid, except for identical ones, serve as target starting and ending points for routes. Actual starting and ending points of the route are the ones that are most proximal to the grid points and lie on

the road network (nodes). Thus, I get a set of routes of the type "the node closest to the initial point of the grid" - "the node closest to the final point of the grid".
Now for each pair of nodes let's calculate the relative risk ratio of accidents (RRR) as the ratio of the number of accidents along the modified route to the number of accidents along the original route (i.e. obtained on the original, unmodified graph). If the average of all such ratios is significantly less than 1, then I can say that the routes calculated on the modified graph are on average safer than the routes obtained on the original graph. It makes sense to average the RRR only for certain binned lengths of the unmodified route, which will make the picture more accurate and precise.

Theoretical and practical significance

The theoretical significance of the work consists in the creation of a new method of transport routing based on bypassing of the TAHS on the road, determined by statistical methods on road traffic accidents.

The practical significance of the work is apparent, and is that the routing methods described should lead to a reduction in the number of accidents and hence in the number of injuries and deaths. Bypassing of the TAHS can be particularly useful for those who have increased exposure to roads (taxi drivers) or drivers who are in other risk groups (the elderly, people with reduced reaction, etc.). The author has created a package of programs in Python that enables one to modify the road graph of any city in such a way that the route drawn on such a graph will be safer on average than the original route[3].

In addition, administrative measures (reduced speed, increased travel time, etc.) to reduce the number of accidents can be applied to the TAHS identified.

[3] The program package treats road edges as obstacles and is described in detail in Chapter 6. The program texts are given in Appendices 1-10.

Research methodology and methods

Various statistical methods are widely used in the Thesis: calculation of elementary statistics, getting confidence intervals by bootstrap method, clustering, random sampling method (Monte Carlo). The geographic information system (GIS) QGIS 3 is widely used to evaluate the results of the work and to visualize the road network.

Scientific specialty

The subject and content of this work fully corresponds to the passport of the scientific specialty 2.3.5. "Mathematical and software support of computer systems, complexes and computer networks", point 4 (Intelligent systems of machine learning, database and knowledge management, toolkits for development of digital products).

Author’s published articles on the topic of this thesis

1. A. M. Gershteyn and A. N. Terekhov “Hotspots of Traffic Accidents that cause injuries or death in Massachusetts from 2013 to 2018 “Computer tools in education”,no.1, pp.45–57, 2021(in Russian); DOI: http://dx.doi.org/10.32603/2071-2340-2021-1-46-58

2. A. M. Gershteyn and A. N. Terekhov, “Routing Algorithm for Vehicles that Avoids Severe Traffic Accident Hotspots on the Road Network (Using the City of Springfield, Massachusetts as a Case Study), ”Computer tools in education, no.2, pp.5–18, 2022(in Russian) http://cte.eltech.ru/ojs/index.php/kio/article/view/1729

3. Gershteyn A. M., Terekhov A. N. “A Simple Way to Improve Road Safety by Avoiding Dangerous Sections of the Route” *Programmnaya Ingeneria*, 2023, vol. 14, no. 3, pp. 103—109. DOI: https://dx.doi.org/10.17587/prin.14.103-109 (in Russian).

4. A. M. Gershteyn and A. N. Terekhov, “Bypassing Dangerous Sections of the Route as a Way to Improve Traffic Safety (Using St. Petersburg as an Example),” Computer tools in education, no. 1, pp. 30–39, 2023 (in Russian); http://cte.eltech.ru/ojs/index.php/kio/article/view/1755

Main scientific results

1. The relative risk ratio of accidents (RRR) is used as an indicator of the effectiveness of routing, equal to the average ratio of the number of accidents along the altered (taking into account obstacles on the roads) route to the number of accidents along the unaltered (not taking into account obstacles) route ([57], p. 8 par. 6).
2. Statistically significant clusters of accidents are used as obstacles in the construction of an alternative (safer) route ([48], p. 47 par. 6, [57], p. 8 par. 7).
3. Separate road segments (edges of the road graph) containing the number of accidents statistically exceeding the number of accidents obtained from the assumption of the uniformity of the distribution of accidents on the road network are used as obstacles ([58] p. 103 par. 2, p. 104 par. 6, [61] p. 30 par 1, p. 32 par. 2).
4. To design a safer route, each obstacle is subject to a penalty, i.e. a fixed number is added to the attribute of the length of the corresponding edge. By calculations, the optimal value of the penalty is determined, which ensures a minimum risk of a TA with a minimum increase in the length of the route and the number of traversable vertices of the road graph ((([58], p. 103 par. 2, [61], p. 32 par. 3).

All articles on the subject of the thesis were written in co-authorship with Dr. A.N. Terekhov. A.N. Terekhov carried out the general supervision of the work, the author of the dissertation proposed to use the average relative risk (RRR) of accidents and the corresponding confidence intervals as an indicator of the safety of the route, and the penalty length added to the appropriate attribute of the road graph edge (= road segment) as a means to reach a compromise between the route length and its safety profile. The author also performed all calculations using the Python language and graph libraries: OSmnx [6] and NetworkX [7].

Main points to be defended

1. Statistically significant clusters with severe traffic accidents have been shown to exist in recent datasets from villages (Newton, Springfield) to an entire State (e.g. Massachusetts) applying DBSCAN algorithm to real data and comparing to results of applying DBSCAN to simulated data based on uniform distribution of severe traffic accidents along pertinent network of roads.
2. Severe traffic accident clusters in Massachusetts have been shown to repeat from year to year at a rate of ~30%, thus suggesting that they are related to the specific location and road conditions in that vicinity. These stable and statistically significant clusters, with compact radius of ~20 meters, are identified as road areas that should be fixed first by municipal authorities.
3. In Moscow, by comparing the number of actual accidents with the number expected according to a uniform distribution over the entire road network (using Monte Carlo simulation), road segments with a statistically significant high number of accidents were found.
4. Optimal value of penalty at 2000 meters has been established for Moscow, that results in substantial risk reduction of 9-31%, while increasing route length by 6-11%.
5. The algorithm developed for Moscow (items 3-4) was tested on the road network of St. Petersburg. The penalty of 1000m should be considered optimum for St. Petersburg, leading to smaller increase in the length of a route in comparison with the penalty of 2000m, while the relative risk in this case practically does not change (the average length of a route increases by 8.0-10.0% for 1000m (in comparison with 8.8-14.8% for 2000m). The average relative risk appears to be in the range of 14.5 - 36% (1000m) compared to the

range of 13.9 - 36% for the 2000m penalty.

6. The routing algorithm developed for Moscow [58] is stable, that is, it can be applied to the road network of other cities with minimal changes (except for choosing the optimal penalty).

The approach applied to Moscow and St. Petersburg is planned to be combined in the future with real-time data (traffic jam data) to find alternative and safer routes.

Chapter 1. Review of Literature

Invented in the late 50s, the Dijkstra algorithm [2], the Bellman-Ford algorithm: [8], [9], as well as the algorithm A* [21], which appeared in 1968, allow to construct an optimal, in some sense, route on a weighted oriented graph G(V, E), where V is the set of vertices of the graph, and E is the set of its edges. Originally such simple properties as length or travel time were considered as weights of edges, the minimal sum of which is obtained in the process of construction of a route. This seemed to make these algorithms ideal at designing transport routes between two given graph vertices (two points chosen on a map). However, it soon became clear that in reality these simple properties need to be adjusted.

So, for example, in real conditions there are traffic jams that make building a route of minimum length meaningless. This means that it is necessary to somehow detect traffic jams in real time and adjust the weights of the corresponding edges of the road graph (for example, to make a traffic jam completely impassable by increasing the length of the corresponding edge of the road graph to infinity) - and then the routing algorithm will have to seek another route, in one way or another, free of traffic jams.

With the advent of smartphones and the widely available Internet access, this has become a realistic task. It is enough to install on a smartphone the appropriate application, which transmits the coordinates of the vehicle to the server so that the appropriate algorithms determine the location of the vehicle (the appropriate edge of the road graph) and vehicle speed, which makes it possible to identify the coordinates and the severity of traffic jams. Examples of such applications include Yandex, Waze and Google Maps.
Of course, the application installed on the smartphone is able to determine not only the coordinates of the vehicle, but also other features of its behavior (how often the driver slows down — accelerates, switches from one line to another, etc.) Therefore, there are systems that try to take into account not only traffic jams, but also the probability of

accidents within the road segment, other road features, and the behavior and preferences of the driver specified by himself in the mobile app. For example, the work [22] uses a weighted sum of the length of the road, the average time of its passage and the number of accidents registered on it during the year, and in [18] when determining the final route some segments of the road are excluded from the road network based on the analysis of driver behavior and preferences. For the remaining segments, the routing algorithm determines the final route, using a combination of the road segment quality indicator (road health), the average travel time and driver preferences as an edge attribute. This approach, in principle, can improve driving safety along with finding a route that is passed in the minimum time. A similar approach to [18] can be found in [23], but in the latter work the driver's behavior is determined by exchanging data between vehicles.

Another, perhaps somewhat unexpected, example of routing is related to the transportation of various dangerous goods. Safety becomes the priority of the route from the starting point to the destination, because an accident during the transportation of a poisonous or explosive substance can lead to an environmental disaster and immense human casualties. In contrast to the previous example, in the task of routing dangerous goods the priority becomes not the time of the route, but its safety. Often the problem can be solved using already accumulated data, and the solution itself will depend on two key factors - the probability of an accident on each of the road segments and the population density around each segment [10]. Consideration of the population density where each road segment is located will obviously depend on the nature of the cargo, its ability to affect larger or smaller areas in the event of an accident. Real-time weather data (e.g., wind speed) may also be required. A more detailed overview of the problem can be found in [11], [12], [15]

The third example of routing is related to the safety of the pedestrian route. A safety problem may arise when there are, for example, HRA associated with criminal activity [13]. The task of routing in this case is to bypass dangerous areas with increased criminal activity. As usual, the routing algorithm must know the "danger" value for each road section. To get this value from individual points - the coordinates of various crimes committed in the city - one can use the standard KDE (Kernel Density Estimation) technique [14], which allows us to find the risk value for any point in the city. In some works, such as [28], Bayes' theorem is used to determine the probability of a crime for a given place and time (see [30], [31]).

Since this kind of routing is done mainly for pedestrians, one encounters a new and important problem here: too high a price to pay for safety. The route that minimizes the risk may be too long, so one has to find a compromise, either by complicating the expression for the edge attribute for the routing algorithm (including the combination of the risk itself and the length of the road segment), or by obtaining several most appropriate routes and choosing from them a compromise: both safer and not very long.

In addition to a "safe" route with minimal criminal activity, a pedestrian may also seek to choose a route with maximum illumination, a sufficient number of landmarks, a minimum number of turns and maximum wide sidewalks [19].

Interestingly, for bicycle walking routes, unlike walking routes, the travel time from the starting point to the destination may not play a great role, so in [19] to find the optimal route, such properties of the road segment as slope, road type, its width, the presence of road signs and lighting are used. The work also applies an original algorithm for finding the optimal route based on dynamic programming (see, for example, [32]) for dynamic programming).

Let us move, finally, to the understanding of safety, adopted in this paper. It consists in the fact that it is necessary to construct such a route, along which the risk of getting into a severe traffic accident, that causes human damages (injuries or death), will be less than on the original route (with the same starting and ending points) that minimizes route length or travel time. For this purpose, the approach used in relevant literature is typically to alter attributes of edges, combining length of an edge or time of passage and (in one form or another) - risk of an TA[4].

For example, in [17] the weighted sum of the length of the edge and the estimate of the risk of accidents within this edge is used as an attribute of the edge. Note that the estimates of the risk of accidents in this work are purely subjective and are reported by

[4] This is due to the fact that building a route that minimizes only the risk of an accident will, according to our estimates, lead to a too long route, exceeding the original 2-3 times.

students, who indicate dangerous points in the campus road network in terms of accidents, along with an estimate of the risk of accidents. This approach to identifying dangerous places on the road network can be extended to a large city, for example, as part of the OpenStreetMap project [33].

Another idea of building a high-security route is related to obtaining several alternative routes and choosing among them a route of acceptable length. For example, in [16], a modification of the standard Dijkstra routing algorithm is used, which utilizes as an attribute of the edge the number of accidents that occurred in each segment of the road, and additional normalized indicators that take into account the weather, time of day, age of the driver, etc. Modification of the algorithm allows you to get several route options, and then select routes with a compromise combination of risk and length.

Additional to the number of previously recorded TAs indicators (current properties of the road segment, weather, drivers' behavior) are also used in [24], [25].

Several alternative routes are also considered in [20], which is interesting because it attempts to predict the probability of an accident for elderly drivers and cyclists using traffic properties (speed and density of flow, driver reaction time), as well as the length of the road segment and the coefficient of friction of tires and the road surface.

An attempt to predict the probability of accidents while driving along a given route is also made in [26]. To predict the probability of accidents, the volume of traffic, road characteristics and current weather conditions are used. The traffic volume (for those road segments where no traffic data are recorded) is obtained as a result of interpolation - determining the traffic for a given time by the nearest road segments for which this value is known. The probability of an accident for a given segment is determined using a trained binary classifier (see [34]), and the probability along the entire route is calculated as the result of a series of Bernoulli tests. If the obtained probability is used as an edge attribute, the routing algorithm will determine a route for which the probability of an accident is on average half as much (compared to the standard route, which takes the minimum time) and 1.7 times as long.

The paper then uses a combined edge attribute for routing that contains a weighted sum of travel time and TA probability to find an acceptable compromise between the length of the route and its safety.

The probability of an TA for each road segment is also proposed in [27], but with a completely different mathematical apparatus - Bayesian networks [35], using a combination of static (road type, map) and dynamic (weather, lighting, vehicle density, information about changing road lanes, vehicle speed, etc.) data, which are offered by the vehicle itself, and from the sensors located along the road.

Chapter 2. Identifying TA hotspots on Massachusetts roads in 2013-2018

Introduction

The purpose of this chapter is to find compact stable TA hotspots (TAHS) on roads, i.e., locations with a statistically significant increased density of severe traffic accidents (TA) that do not change their position over several consecutive years.
The danger of such hotspots is caused either by the road structure (places of intersection of roads, entry/exit to the highway), or by other factors (quality of road surfaces, poor visibility, etc.). All these factors can be adjusted to a certain extent by municipal or state authorities, moreover, these dangerous sections can be excluded from the routes of commercial and private transport.

Detection of dangerous areas on the earth's surface is an important step in the study of phenomena of various natures. For example, locations with higher crime densities identify the most dangerous areas as a prime target for police and other organizations whose activities are aimed at reducing crime [36], [37].

There are several methods of dangerous area detection: KDE (Kernel Density Estimates) [2, 4, 14], Moran I-statistics [39], Getis-Ord Gi* statistics [41], and various clustering algorithms. KDE uses various kernel functions to convert points on a surface (e.g., crime scenes) into some smooth function in an attempt to reconstruct the density distribution of these points. KDE identifies hazardous locations in the plane, but cannot estimate their statistical significance. Moran's I and Getis-Ord Gi* statistics identify statistically significant areas (clusters). According to their nature, it is not easy to use KDE, getis-ord GI* or Moran's statistics to detect dangerous areas of a given size. In addition, the Getis-Ord Gi* and Moran's I statistics work only in the 2d case [37], but the road network is essentially a one-dimensional object with different distance representations. Note that KDE can in principle be applied to the clustering of traffic accidents belonging to the road network, using special, rather complex kernels developed in [40]. However, this method requires much more time-consuming calculations, and the results are not so easy to interpret, and the shape of clusters in the case of application of KDE will be determined by the kernel used, while in the case of application of clustering methods, the shape of clusters can be arbitrary.

As for the clustering algorithms, such as DBSCAN [4], they can be easily adapted to search for clusters of traffic accidents belonging to a road network by specifying the corresponding distance matrix, which can be computed using, for example, the SANET package [47]. In addition, the DBSCAN algorithm, as shown in Section 2.2.1, can easily be configured to detect clusters with a given maximum distance between points.

There is another difficulty in detecting dangerous areas belonging to a road network: the lack of appropriate statistics. Getis-Ord Gi* and Moran's I statistics, as already mentioned, work only in the 2d case [37]; for clustering methods such statistics are very rare. So, computer simulations (Monte Carlo method) [5] are used to statistically justify the obtained clusters in this paper. The SANET package allows one to rapidly generate millions of points uniformly distributed over the road network. Thus, the plan of this chapter is to perform a cluster analysis on selected TA using distances over the road network, and then use the same network and the same number of evenly distributed points as in the real data to perform computational simulations at least several hundred times to obtain cluster size statistics. Then, I compare the size distributions for the real and simulated TA clusters. As a result, statistically valid clusters are identified.

2.1 Input Data

This Chapter uses road networks provided by the Massachusetts Department of Transportation [10] in Esri shapefile format, which is very easy to visualize using GIS applications such as QGIS [44] и OpenJUMP [45]. MassDOT portal data [46] on Massachusetts traffic accidents from 2013-2018 in ".csv" spreadsheet format, which can be easily converted to other formats, such as the same Esri shapefile, was also used.

As usual, the resulting data must be preprocessed. The isolated road network fragments (if any) must be connected to the main network. Disconnected Islands plug-in and QGIS editing tools were used to connect the fragments to the main network. In files containing information about TAs, one needs to leave only those records where the coordinates of the accident are present (approximately 96% of all records). Since I obtained the accident data and the files where the road network is stored from different sources, the individual accidents do not belong exactly to the elements of the network. Meanwhile, for some types of analysis, it is important that the accident and the road network be a single entity. Therefore, TA points were projected onto the elements of the network, for which I used the plugin QGIS NNJoin, which creates an additional layer in QGIS. This additional layer contains original coordinates of the accident location, and there is an additional attribute that stores the start and end coordinates of the endpoints of a line segment that is nearest to the accident location and belongs to the road network. These coordinates allows one to project the point of the accident on the corresponding line of the road network using a simple Python script:

```
def dropPoint2Line (point, line):
    #point (x,y)
    #line  (point1,point2)
    x0 = point[0]
    y0 = point[1]

    x2 = line[0][0]
    y2 = line[0][1]
    x3 = line[1][0]
    y3 = line[1][1]

    m = x3 - x2
    p = y3 - y2
    t = (m * x0 + p * y0 - m * x2 - p * y2)/(m**2 + p**2)
    x1 = m * t + x2
    y1 = p * t + y2
    return (x1,y1)
```

2.2 Research methods

2.2.1 DBSCAN

As a clustering algorithm, I chose DBSCAN (**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise) — a heuristic algorithm that finds clusters of the same minimum density (the letters DB in the name of the algorithm mean Density-Based) and has arbitrary shape. At the same time, the number of clusters is determined automatically. Each point after the algorithm operation turns out to either belong to some cluster or be classified as an outlier - a point that does not belong to any cluster. Another advantage of DBSCAN is also that this algorithm, implemented in the sklearn library, works very rapidly, which is important for processing large amounts of data and conducting statistical tests when the algorithm must be run hundreds of times.

DBSCAN uses a matrix of distances between points or the coordinates themselves (in the case of Euclidean distances), as well as two parameters: "eps" (distance) and "min_samples" (positive integers starting at 1). The algorithm first selects the main points — those in the eps neighborhood that contain at least "min_samples" of other points. The first main point allocated by the algorithm becomes the "seed" of the cluster. Let's denote it Q1-0. All points *reachable* from Q1 are assigned to the cluster. These can be points that are directly reachable, that is, located in the "eps" neighborhood of Q1-0, or points for which there is a path Q1-0 -> Q1-1 -> Q1-2 ..., and Q1-1 is in the "eps" neighborhood of Q1-0, and the point Q1-2 – in "eps"-neighborhood Q1-1. Note that all points of such a path, except the last one, must be the main ones. The cluster can include both main points and ordinary ones, for which there is a path from Q1-0, but which do not contain "min_samples" in their "eps" neighborhood.

When the cluster is formed, the algorithm will try to find the main point that does not belong to the already allocated cluster and form a new cluster around it. The algorithm will terminate when it cannot find the main point that does not belong to the already allocated clusters.

2.2.2 Matrix of intra-network distances (scale 20m)

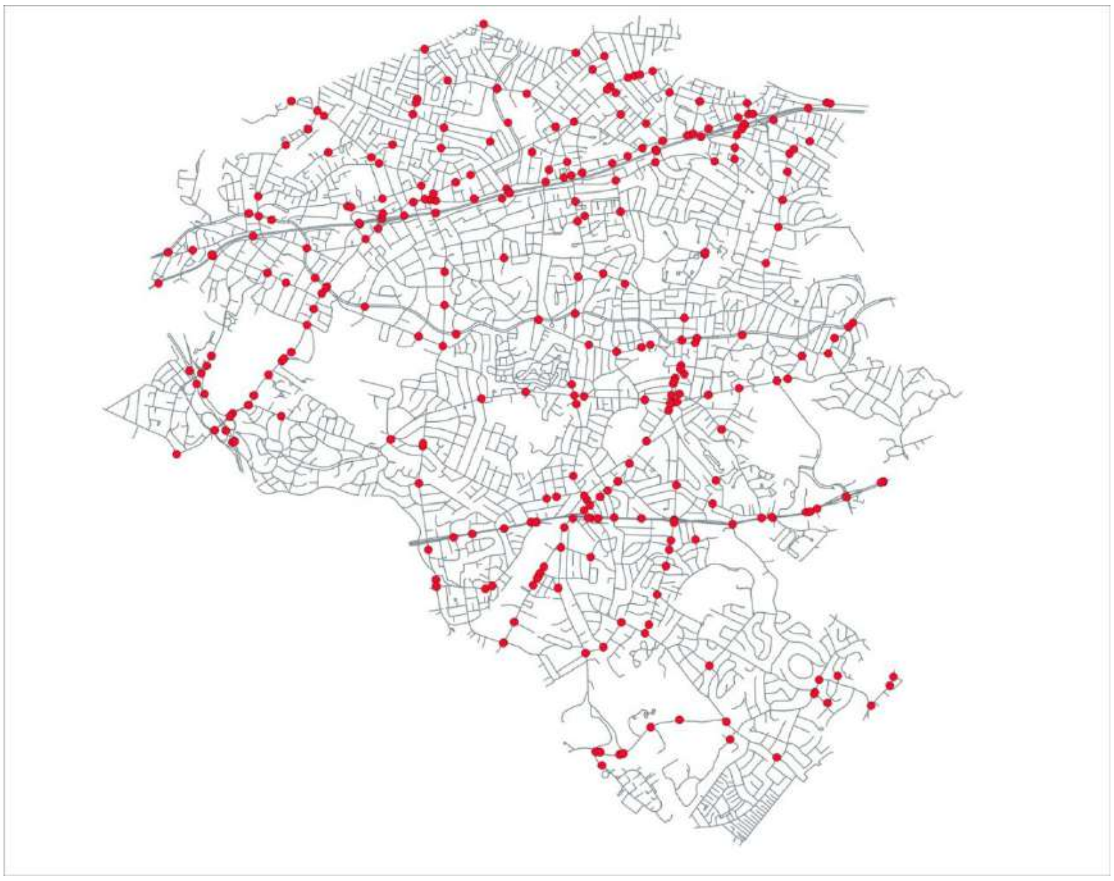

Figure 2-1. Severe Traffic Accidents (Newton MA, 2013)

Having prepared the data, I will try to detect High Risk Areas (HRA) in the city of Newton, Massachusetts, using data on severe accidents in 2013 (a total of 369 cases). In our first clustering attempt, I will use the network distance matrix computed using the SANET package and the parameters "eps"=20m and "min_samples" = 3 of the DBSCAN algorithm. The choice of the parameter "eps"=20m seems reasonable, since 20m is approximately the distance of the width of a road consisting of four to six lanes (3.6m per lane, 14.4 m – 21.6 m width) and the resulting clusters would apparently be compact enough to be easily avoided by using a suitable routing algorithm. Note that the DBSCAN algorithm is designed so that "eps"=20m specifies the "eps"-edge of the point, that is, the minimum density, not the cluster size. The clusters themselves can be arbitrarily shaped and can be stretched over a distance many times greater than 20m.

As a result, 63 clusters with sizes from 3 to 9 TAs were obtained. To select statistically significant clusters from these clusters, I formulate the null hypothesis: Accidents are distributed evenly across the road network. I will assume that a cluster of size n is statistically significant at level a if the probability of detecting at least one cluster of size greater than or equal to n (if the null hypothesis is correct) is less than a clusters [5]. Given that there are no statistics on cluster sizes in the case of the DBSCAN algorithm, I conducted a number of statistical tests using the Monte Carlo method. Using the SANET package, 1024 samples of points evenly distributed over the Newton road network were obtained (each sample contains 369 points, as in the original data), then these samples were used to calculate 1024 distance matrices along the road network (1024 is the number of Monte Carlo tests) and then the DBSCAN algorithm was applied to each matrix with the same parameters as in the case of real data ("eps"=20 and "min_samples"=3). The simulation results are shown in Table 2-1.

Table 2-1. Simulation results (1024 tests, Newton, Massachusetts road network)

Cluster size	The number of clusters of the same or larger size	P
3	59	0.058
4	0	0

Table 2-1 shows that clusters with n ≥ 4 are significant at the 0.05 level. Removing clusters with n=3 from the 63 initially detected yields 5 statistically significant clusters.

2.2.3 Hybrid approach to clustering

Clustering using the network distance matrix presented in the previous section requires very large computational resources and is possible (taking into account the need to conduct at least several hundred tests) only for small data sets, such as data on severe accidents in Newton, a suburb of Boston with a population of about 90 thousand people. Therefore, it is very tempting to replace the network distances between individual TAs with the corresponding Euclidean metric. In this case, the DBSCAN algorithm implemented in the sklearn package (Python) calculates the distance matrix extremely quickly, and for a scale of 20m it seems reasonable that the distances (Euclidean and over the network) are very similar. After repeating all the calculations — this time with Euclidean distances and obtaining a new set of statistically significant clusters, I can compare them with the clusters obtained in Section 2.2.2.

Comparing two sets of clusters, I get the following:

- 5 network clusters are located identically to their Euclidean counterparts;
- 2 Euclidean clusters are located differently;

identical clusters can be very different when zooming in.

Table 2-2. Newton 2013. Two clustering methods (Euclidean and Network, eps=20)

Method	N	Percentage of the total number of severe TAs	False clusters number/percentage	Corrupted clusters number/percentage
Network	5	7.3	0/(0%)	0/(0%)
Euclidean	7	10	2/(29%)	3/(43%)

So, considering network clustering as "true", I can list some disadvantages of Euclidean clustering:

1. False clusters — have no network counterpart

2. Corrupted clusters — have true (the same as in the network case) and false fragments (on different road lines or on different roads).

Now I can summarize the two approaches to clustering with "eps"=20 in Table 2-2.

2.2.4 Scale 10m

It follows from the previous section (Table 2-2) that the Network and Euclidean approach to clustering at eps = 20m differ significantly.

Meanwhile, it is obvious that at some sufficiently small scale, the network and Euclidean distance should be identical. Of course, this scale is less than 20 m. To estimate this scale, I investigate the dependence of the difference between Euclidean and Network distances on Euclidean distances and find the point of divergence where Euclidean distances become inadequate for our task (Fig. 2-2).

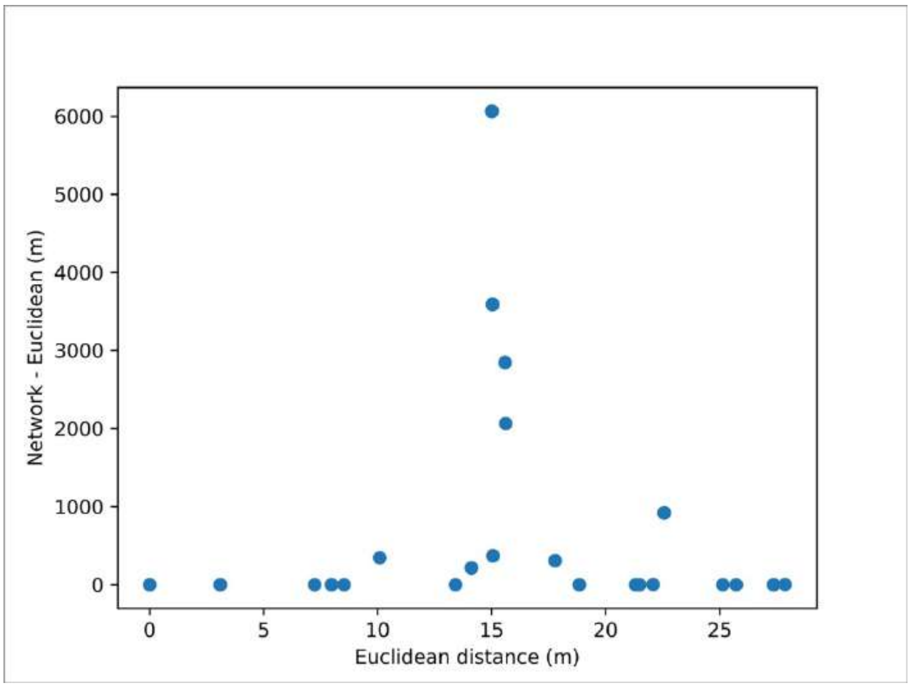

Figure 2-2. Newton, Massachusetts, 2013, severe TAs. Dependence of the difference between Euclidean and Network distances on Euclidean distances

As can be seen in Figure 2-2, the network distance is always at least as long as the Euclidean distance, because the straight line connecting the two points has the minimum length in the Euclidean space. One can also see that the discrepancy between the Euclidean distance and the Network distance starts at a scale of 10 m. Large spikes near 15 m are clearly visible in the graph, and this likely distorts the Euclidean clustering with eps = 20. Thus, I chose 10m as the scale, where the Euclidean distance, which requires much less computation, can replace the distance along the road Network.
The value of 10m seems very small, but in fact it is not. It is only slightly less than the width of a 3-lane highway (in the United States, each lane is about 12 feet or 3.6 meters wide. In other words, 10 meters is the natural scale for the movement of vehicles in one direction on the road network.

2.2.5 Hybrid clustering with 10m

Looking at Fig. 2-2 and taking into account the natural scale for the road network given at the end of the previous section, it is possible to justify a hybrid clustering method: use the road network only to generate sets of uniformly distributed points and perform all clustering (for detecting real clusters and for static tests) with “eps”=10 using Euclidean distances between TAs.

2.3 An example

2.3.1 Massachusetts 2013, clustering of severe TAs

According to available data, 30,696 severe accidents were registered in Massachusetts in 2013, of which 23,964 (78%) occurred on major roads.

The application of clustering with “eps”=10 and “min_samples”=3 to 23,964 severe TAs gives 1,340 clusters, from which statistically significant ones need to be identified. To do this, clustering was performed 1,502 times on sets of 23,964 points evenly distributed over the road network generated by the SANET software package. The results of the Monte Carlo tests are shown in Table 2-3.

Table 2-3. Monte Carlo simulation results for Massachusetts, 2013 (1,502 tests, 23,964 points in each)

Cluster size	The number of clusters of the same or larger size	P
3	1502	1
4	160	0.1
5	6	0.004

Thus, at the 0.05 level, clusters of size ≥5 should be considered statistically significant. Removing from the initial clusters those whose size is less than 5, I get 354 clusters, the total number of severe TAs in clusters is 2,301, that is, 9.6% of the total number of severe TAs.

Figure 2-3 shows statistically significant clusters of severe TAs that occurred in Massachusetts in 2013.

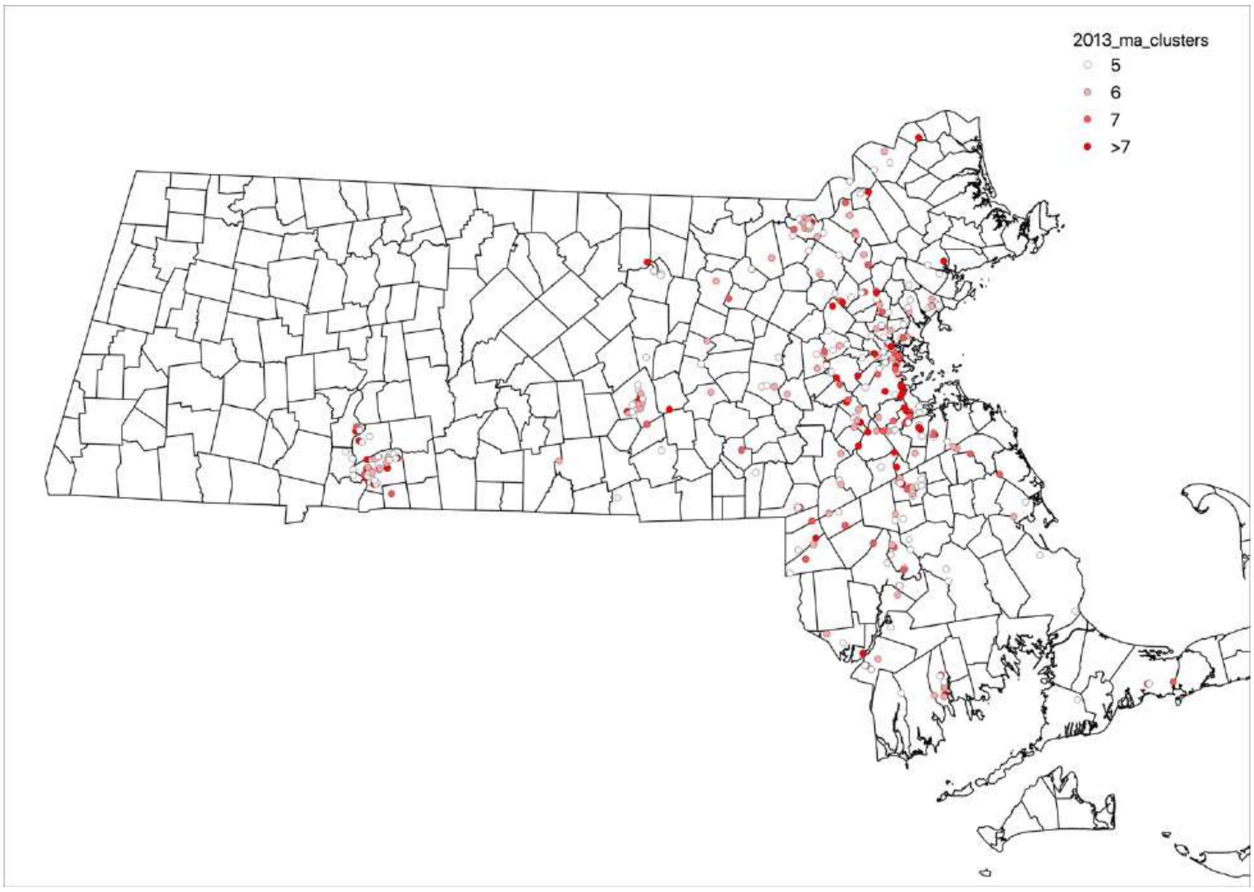

Figure 2-3. Massachusetts 2013, significant clusters of severe TAs (DBSCAN, eps=10)

As can be seen from Fig. 2-3, the spatial distribution of clusters throughout the state is far from uniform. 166 (47%) clusters were found in thirteen cities (Table 2-4). Newton with 3 clusters ranks 38th in the general list and is included in the table only because I studied it in detail in Section 2.

Table 2-4. Massachusetts cities with the maximum number of clusters

City	Number of Clusters	Number of Severe Traffic Accidents
SPRINGFIELD	38	234
BOSTON	24	209
WORCESTER	23	148
LOWELL	15	93
BROCKTON	14	87
BRAINTREE	8	68
NEW BEDFORD	9	55

Continuation of Table 2-4

City	Number of Clusters	Number of Severe Traffic Accidents
QUINCY	7	53
WOBURN	5	39
WEYMOUTH	5	36
HOLYOKE	5	35
RAYNHAM	7	35
WALTHAM	6	35
NEWTON	3	19

Figure 2-3 also shows that most clusters have sizes from 5 to 7.

2.3.2 Meta-clusters in Massachusetts: Hybrid Bottom-Up Approach

In the previous section, clusters of severe traffic accidents that occurred in Massachusetts in 2013 were analyzed. My next goal is to use data for 2013-2018 to identify time-stable clusters, that is, places that remain dangerous for at least several years. I will use two-stage clustering, that is, obtaining significant clusters for each year, and then clustering these clusters again with the parameter min_years ≥minimum number of years. For example, “min_years” = 4 indicates that the resulting cluster contains data for at least 4 different years.

As in Section 2.3.1, I will first get all (including statistically insignificant) clusters for 6 years: 2013, 2014, 2015, 2016, 2017, 2018. Then I will conduct Monte Carlo simulations to identify significant clusters. The results of the Monte Carlo simulations for all 6 years are collected in Table 2-5.

Table 2-5. Monte Carlo simulation results for six years, DBSCAN (eps=10, min_years=3) uniformly distributed random points on the road network

Year	Total severe TA	Number of trials	Cluster size	The number of clusters of a given or larger size	p
2013	23964	1502	3	1502	1.0
			4	160	0.11
			5	6	0.004
2014	24921	1444	3	1442	0.99
			4	171	0.12
			5	6	0.004
2015	25959	1386	3	1384	0.99
			4	208	0.15
			5	11	0.008
2016	26772	1344	3	1344	1.0
			4	214	0.16
			5	8	0.006
			6	1	0.0007
2017	26672	1349	3	1349	1.0
			4	206	0.15
			5	9	0.007
2018	24763	1453	3	1453	1.0
			4	201	0.14
			5	7	0.005

After removing statistically insignificant clusters, I will combine the clusters for all 6 years, adding a year label to each cluster, and then perform a second clustering with "eps"=10 and the "min_years"=3. Some statistical characteristics of meta-clusters are shown in Table 2-6. The number of severe TAs in all meta-clusters – 7,456 is 4.8% of 15,3051 – the total number of severe accidents used for two-stage clustering. Figure 2-4 shows the distribution of clusters on a map of Massachusetts. It is apparent that the spatial distribution of meta-clusters is similar to 2013.

Table 2-6. Meta-cluster statistics

Different years in the cluster	Clusters	Severe TA	Severe TA / Cluster	Severe TA median	Severe TA standard deviation
3	115	2269	19.7	19	3.29
4	74	1979	26.7	25	5.77
5	36	1412	39.2	36	10.68
6	29	1796	61.9	56	21.71
Total	254	7456	29.3	24	16.23

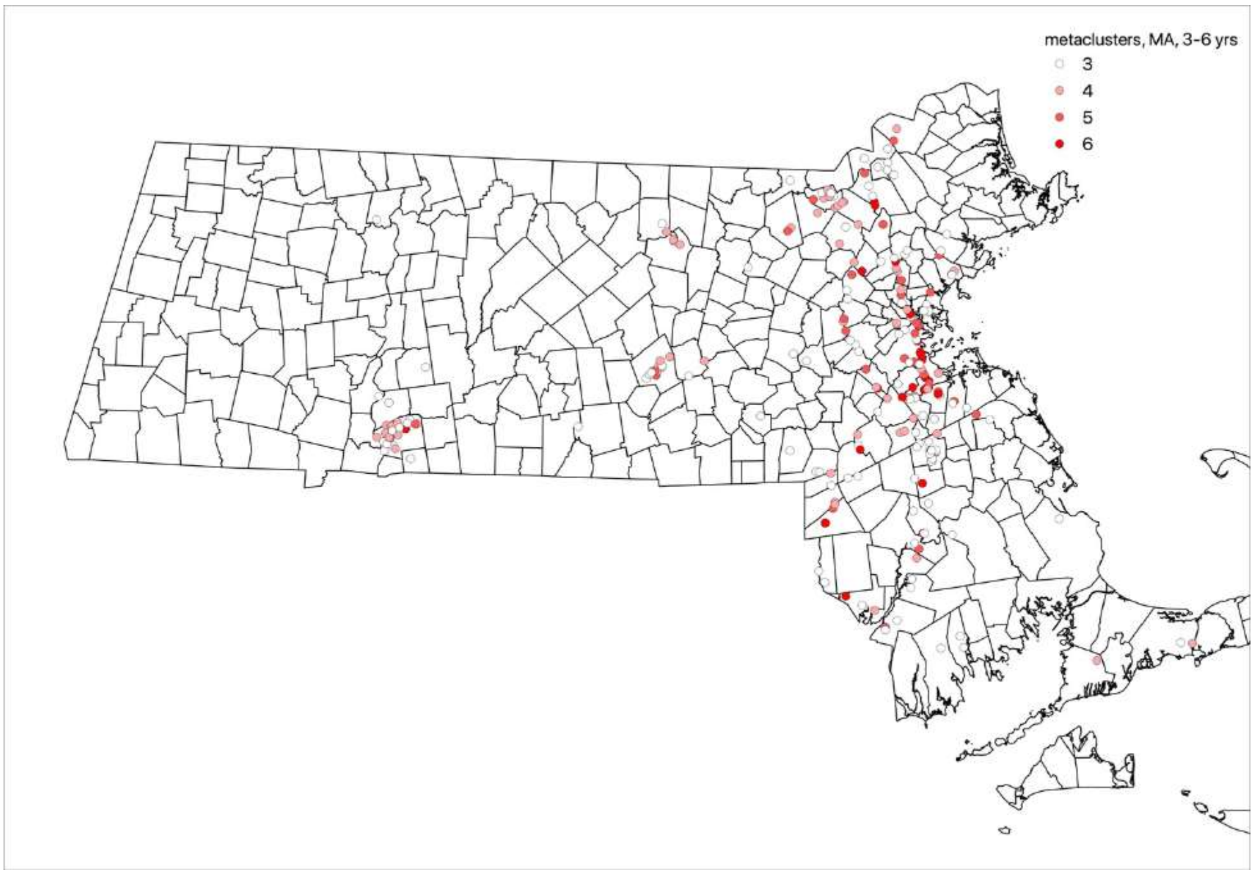

Figure 2-4. Meta-clusters in Massachusetts, 3-6 different years in each cluster

In addition to the statistical characteristics of meta-clusters containing a different number of years when severe TAs occurred, the characteristics of the repeatability of clusters from year to year are very interesting. For example, for each cluster of severe TAs that occurred in one year, one can find (if it exists) a spatially equivalent cluster of accidents that occurred in another year. The results are presented in Table 2-7, which resembles a distance matrix and contains the relative number of clusters observed during one year and remaining in subsequent years. For example, 0.36 at the intersection of row 2018 and column 2017 means that the ratio (number of clusters in 2017 found in 2018/total number of clusters in 2017) is 0.36. The total number of statistically significant clusters in the corresponding year is shown in the **Clusters** column.

Table 2-7. Relative repeatability of clusters from year to year

Годы	2017	2016	2015	2014	2013	Clusters
2018	0.36	0.35	0.35	0.33	0.36	378
2017		0.35	0.35	0.34	0.32	369
2016			0.39	0.33	0.3	374
2015				0.3	0.33	334
2014					0.36	349
2013						354

It is also interesting to find out how the repeatability of clusters depends on the annual interval. In Table 2-8, the data in Table 2-7 are presented somewhat differently. Column 1 year contains the "distances" of one year between sets of clusters (i.e. the repeatability of clusters for 2013-2014, 2014-2015 and so on), column 2 years represents data on two-year repeatability (i.e. 2013-2015, 2014-2016 and so on). The last column (3 years) represents data for a three-year repeatability. The last two rows of the table contain elementary repeatability statistics for different annual intervals. It can be seen that the "distance" between sets of clusters remains relatively constant, decreasing slightly with an increase in the time interval. Apparently, the road conditions for the studied accidents do not change significantly from year to year, i.e. they belong to the same distribution and can be combined if necessary, for further analysis.

Table 2-8. Repeatability of clusters for different time intervals

1 year: 13-14, 14-15, 15-16, 16-17,17-18 **2 years**: 13-15, 15-17, 14-16, 16-18 **3 years**: 13-16, 14-17, 15-18	**1 year**	**2 years**	**3 years**
	0.36	0.33	0.3
	0.3	0.33	0.34
	0.39	0.35	0.35
	0.35	0.35	
	0.36		
Average	0.35	0.34	0.33
STD	0.03	0.01	0.02

Conclusions

This Chapter shows that the hybrid DBSCAN clustering method using Euclidean distance applied to data on severe TAs in Massachusetts, combined with Monte Carlo simulations on the state road network, can identify statistically significant hotspots of severe traffic accidents. These hotspots (clusters) are compact, with about a third of the clusters repeated next year. It is advisable for drivers to avoid identified areas of increased risk of severe TAs, and city and state authorities need to use information about such areas to plan measures aimed at reducing injuries and deaths on theses roads.

Further research may be useful to identify spatial and other features of the TA hotspots (e.g. intersection, highway entry/exit, road junction, road width, speed limit, poor visibility, time of day, weather conditions, etc.).

Chapter 3. Vehicle routing in the presence of dangerous sections on the road (using the city of Springfield, MA as a case study)

Introduction

The task of constructing an optimal route in one sense or another for a vehicle is one of the problems that is well-studied and has been solved long ago. If a road network is represented by a bidirectional multigraph (i.e., a graph where several edges can pass through any two vertices, and each edge can be unidirectional or bidirectional), then routing is performed using the Dijkstra algorithm [2] or Bellman-Ford [8, 9] one. It is enough to assign a number to each edge (for Dijkstra's algorithm it is strictly non-negative) and the algorithms will construct an optimal (in the sense of minimizing the sum of such numbers) route.

For usual routing problems, an edge is assigned either its length or delay time - a much more complex parameter that depends not only on the geometry of the road network, but also on the current state of the route: presence of traffic jams, weather conditions, etc.. Unfortunately, conventional routing very rarely considers other important factors, in particular, road sections with compact sections of high risk (see [48]) or with a large number of severe (i.e., resulting in injuries or fatalities) TAs located within the same road segment (edge). There is a work [49] (see also the literature review in Chapter 1), where an attempt is made to create an integrated attribute for each edge of a road graph containing some weighted sum of the delay time and the crash risk indicator.

This approach is very labor-intensive and requires separate and complex calculations for each edge of the graph representing the road network. Meanwhile, there is a simple technique that allows bypassing dangerous sections by marking them as "impassable".

Imagine a road network graph where each edge is assigned its length as an attribute used by the routing algorithm. Then I can assign an infinite or, which is more practical, a very long length (several orders of magnitude greater than the length of the longest edge) to the edges leading to a HRA, and then the routing algorithm will automatically bypass the dangerous road section.

3.1 Tools and Data

Let's apply this simple technique to the road network of the city of Springfield (Springfield, MA), extracted from the OpenStreetMap (OSM) global map of the Earth's surface. OSM maps obtained as a result of many years of work of a non-commercial wiki project can be of different quality depending on the locality, but for the purposes of routing algorithms research (and as shown in [50] on the example of the Vienna road network, also for real routing) they are quite suitable.

The only difficulty is to extract the road network for a given surface area. When the OSM API was used for this, the task was far from trivial. Fortunately, the recently appeared OSMnx library for Python (see details in [51]) simplified this task to a couple of lines of code. So, to get a map of any city, it is enough to specify the city itself, the state (for the USA) and the country. For example, the Springfield road network is loaded using the graph_from_place() function of the OSMnx package:

```
import osmnx as ox
ox.config(use_cache=True, log_console=True, all_oneway=False)
G = ox.graph_from_place('SPRINGFIELD, MA, USA', network_type='drive')
```

The only task remaining is to save the G object containing the road network to disk for later use. Object G stores the road network in the form of a MultiDiGraph, that is, a graph with the possibility of the existence of many one- and two-way edges passing through the same pair of vertices. Such a graph format is not always convenient for geographic information systems (GIS) like QGIS or ArcGIS, so the OSMnx package has the ability to save in vector ".shp" files, standard for GIS. In ".shp" files information is stored separately for MultiDiGraph edges and vertices, which I will extensively use to determine edges, leading to the HRA.

Table 3.1 presents some statistical characteristics of the edges of the Springfield road graph.

Table 3.1 Elementary statistics of the lengths of the road graph edges of Springfield (in meters)

Average	130
Median	92
Standard deviation	116
Minimum	1.47
Maximum	2084.8

I will also need data on TAs in Massachusetts for 2013-2018, which can be obtained in text format ".csv" on the MassDOT portal website [5].

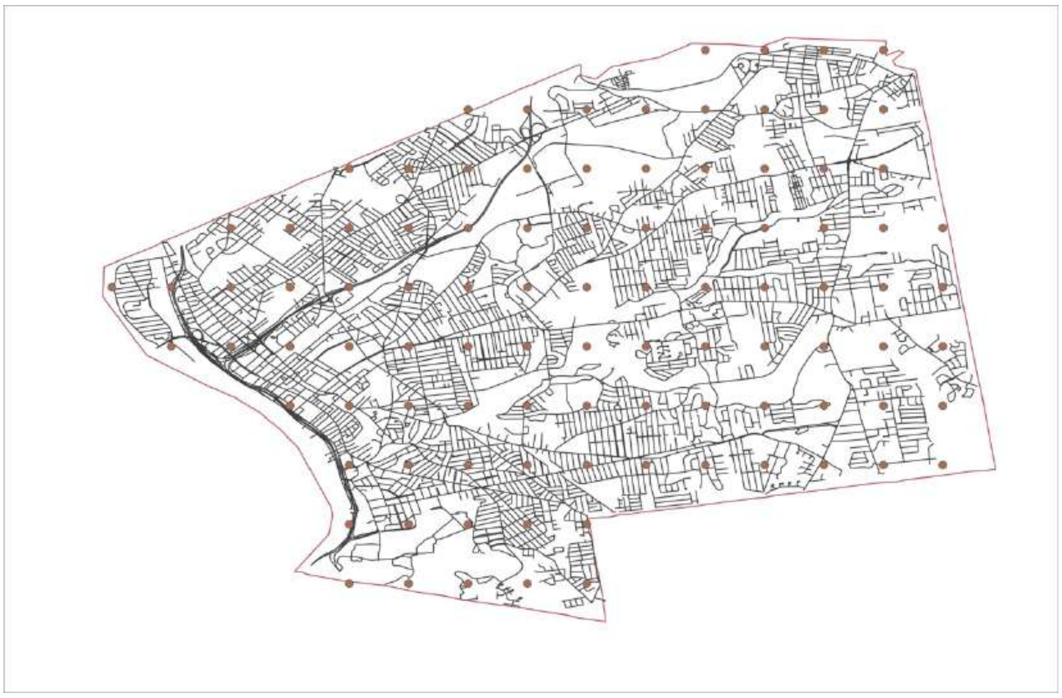

Figure 3-1. A grid of equidistant points placed on the map of Springfield

[5] https://massdot-impact-crashes-vhb.opendata.arcgis.com/search

3.2 Method

To study the effectiveness of routing in the presence of dangerous sections of the road network, I will set a grid of equidistant points and put it on the map of Springfield (Fig. 3-1). The map contains 95 grid nodes with a distance of about 830 meters between neighboring nodes, which gives 95 x 94 = 8930 different directional routes. My further actions can be put into the following scheme:

a. Combine all severe traffic accidents that occurred in Springfield between 2013 and 2018 with the traffic graph extracted from OSM. Note that the accident coordinates and the road graph are derived from independent sources, so the accident points may not coincide with the graph edges. Consequently, I need a procedure that projects each accident to the nearest edge. As a result, each edge of the graph will be matched with a non-negative number of accidents that occurred along it.

b. Let's prepare two versions of the road graph: G0 and G1. Version G0 will contain the original graph extracted from OSM. In version G1 the attribute storing the edge length will be replaced for some edges by a very large number, many orders of magnitude greater than the maximum edge length, which makes such edges practically impassable (in the presence of alternatives) for the routing algorithm.

c. Using the grid placed on the Springfield map, I will construct all possible routes for graphs G0 and G1. Then select from all the routes connecting the same points in the same direction only those that differ in G1 compared to G0, i.e. those affected by changes in graph G1.

d. For all selected routes, compare their length and number of vertices (according to the data stored in the graph G0), that is, see how the length and "divisibility" of the route change when passing obstacles.

e. For all selected routes, compare the number of accidents along the route, that is, enables one to see how the relative risk ratio of accidents changes when bypassing obstacles. By the Relative Risk Ratio (RRR) here I mean the ratio of accidents TAS1/TAS0, where TAS0 is the number of accidents counted along the original route on the graph G0, and TAS1 is the number of accidents along the route connecting the same starting and ending points, but on the graph G1.

3.3 Bypassing High Risk Areas (HRA)

Let's try to construct vehicle routes so as to bypass HRA (statistically significant severe TA meta-clusters), i.e. clusters resulting from secondary clustering of clusters of severe TAs that occurred in Springfield from 2013-2018. Of all such secondary clusters, only those containing clusters of crashes in at least three different years (31 in total) were selected. See [48] for details.

My task is quite complex, so it is reasonable to break it into several steps, saving (where necessary) intermediate data in ".csv" files:

1. Load two original road graphs from the hard disk drive and name them G0 and G1.
2. Find all edges leading to the HRA (meta-clusters). This task has to be performed manually, because a cluster may be located in different ways. If the cluster is located on a unidirectional edge, it is enough just to assign a very large number to the attribute storing the length of the edge, and the edge becomes almost impassable for routing algorithms, that is, the cluster will be excluded from the route laid on the graph G1.

3. If the cluster is on a bi-directional edge, the problem of graph modification becomes more complicated, because it will be necessary to attribute the cluster to one of the directions of the edge[6].
4. Finally, in the case of a cluster located at an intersection, I have to "close" all paths leading to the cluster, because there is no single dedicated edge where the accidents can be attributed to, instead accidents may have come from all directions. Thus, graph G1 can be easily changed if I prepare a list of "forbidden" edges in advance, considering each meta-cluster and all edges leading to it.
5. For all the edges of graph G1 defined in the previous paragraph, replace the value of the edge length attribute with a very large number 100,000,000,000, which is many orders of magnitude greater than the length of the longest edge in G1 [7].

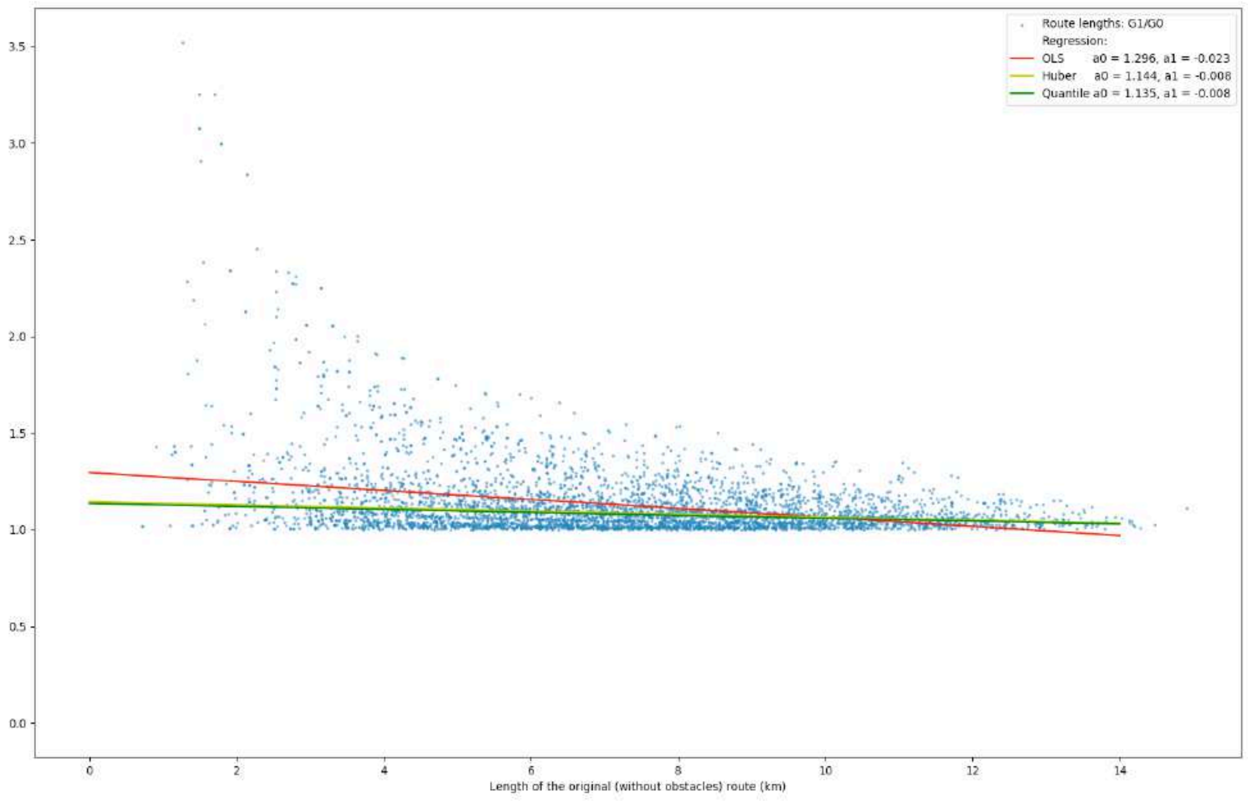

Figure 3-2. Ratios of route lengths calculated from graphs G1 (with obstacles) and G0 (without obstacles)

[6] For a small number of accidents that make up the cluster, it is possible if one uses additional information: address, direction of travel. If the cluster cannot be attributed to a particular direction, one must "prohibit" both.
[7] The maximum length of the edge according to Table 3-1 is 2085 meters, so the number 100,000,000,000, exceeding the maximum length by 48 million times, may from a practical point of view, be considered infinite.

6. For all start and end points given by the grid of equally spaced points (see Fig. 3-1) I calculate optimal routes and select those that are different for graphs G0 and G1 with the same start and end points of the routes. The result is 4,822 such routes, and the length of some routes (a total of 91, which is less than 2% of all obtained) exceeds 100,000,000,000, indicating that in these cases, the routing algorithm has not found ways around the obstacles. Excluding such routes from our data, I finally get 4,731 route pairs on the graphs G0 and G1, which have the same start and end points.

These remaining routes can be displayed as points on the graph, where the X-axis shows the original lengths of the routes without taking into account obstacles, and the Y-axis shows the ratio of the lengths of the routes: calculated from graphs G1 and G0 (Fig. 3-2) and the ratio of the number of vertices computed by the graph G1 to the number of vertices computed by the graph G0 (Fig. 3-3).

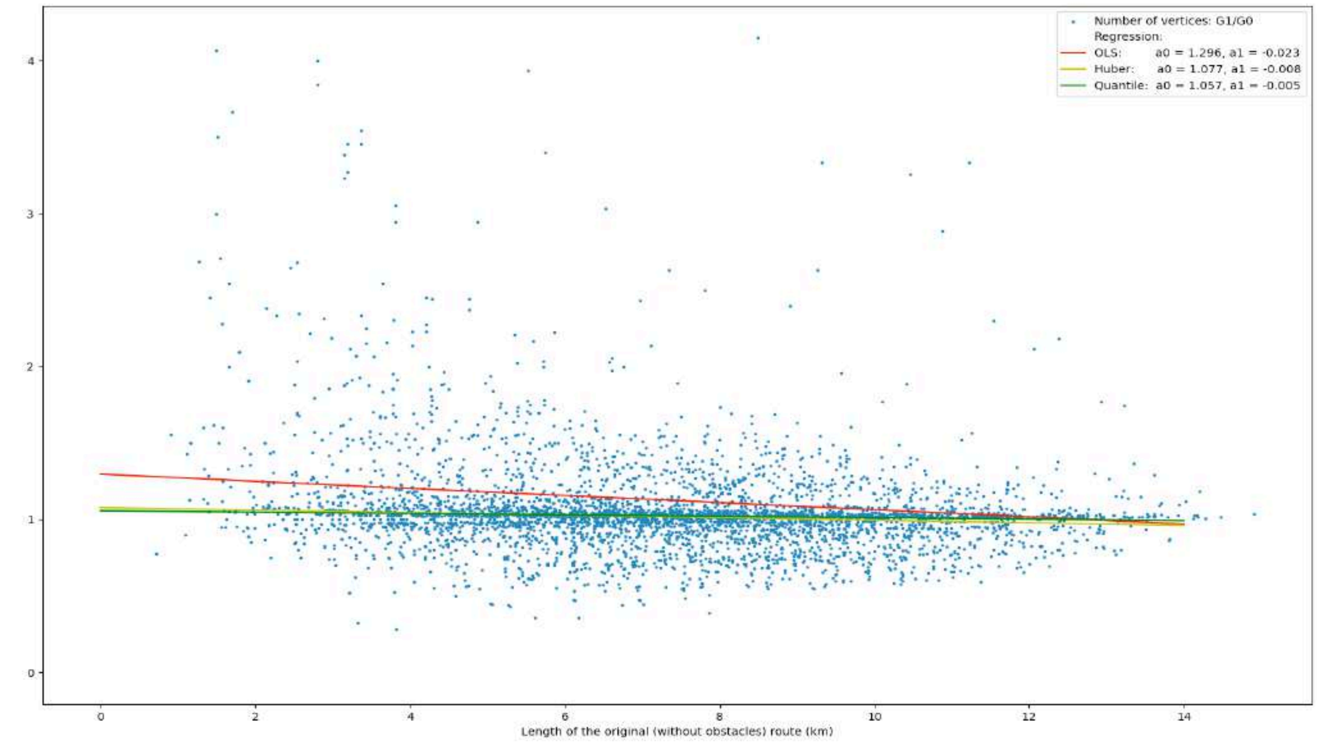

Figure 3-3. Ratio for number of vertices along route of graph G1 (avoiding obstacles) and G0 (original route, without avoiding obstacles)

To get a better idea of where the bulk of the data is concentrated, Figures 3-2 and 3-3 show different regression dependencies of the form y = a0 + a1*x of length ratios (Figure 3-2) or vertex number ratios (Figure 3-3). Since the data are highly scattered, the figures also show regression straight lines obtained using two robust (outlier-resistant) algorithms: Huber regression (see 52, 53]) (yellow line) and quantile regression (see [54]) (green line), which allows to predict arbitrary quantiles of the dependent variable. In this case, I used prediction for the median (50% quantile). The regression coefficients are shown in the legends to the figures.

From the diagrams shown in Figures 3-2 and 3-3, it is apparent that the path length in the case of obstacle avoidance is always greater than the "original" path built without obstacles. It is also evident that the ratio of the length of routes G1/G0 decreases as the length of the original path increases.

As for the ratio of the number of vertices G1/G0, for individual routes it can be either more or less than one, depending on which route the bypass route "lay" on. As the data in Table 3-3 (the last column) show, the average ratio of the number of vertices tends to one for long (more than 6 km. routes)[8], and increases (apparently due to the fact that bypass routes are significantly longer here) to 1.52 on the shortest routes.
Since the ratio of the number of vertices turns out to be uninformative and tends to 1.0 for sufficiently long routes, the main interest is the loss in distance that occurs when choosing an alternative (cluster avoiding) route.

First of all, I will provide general descriptive statistics on all routes that differ (at the same initial and final coordinates) (Table 3-2). Since the distribution of the ratios of various variables in this case is not Normal[9], the bootstrap method is used everywhere to calculate confidence intervals (see, for example, [55]). Specific calculations were performed using the bootstrapped Python package (https://github.com/facebookarchive/bootstrapped).

[8] The difference from the 1.0 of the average G1/G0 vertex ratio for routes with original (without HRA) lengths greater than 13 km. is probably due to the small sample size and the consequent loss of accuracy of the average calculation.
[9] For example, for length ratios whose descriptive statistics are shown in Table 3-2, the D'Agostino test (see [56]) scipy.stats.normaltest() gives pvalue = 0.0

Table 3-2. General descriptive statistics of route length ratios (length when bypassing the HRA/initial length)

Statistical measure	Value	95% confidence interval
Average	1.128	(1.122 - 1.133)
Median	1.07	(1.069 - 1.074)
Standard deviation	0.18	(0.161 - 0.203)
Min	1.00	
Max	3.52	

Table 3-3. The dependence of the average ratios of route lengths (Length when bypassing the HRA/Initial length) and the number of vertices along the routes (Number of vertices when bypassing the HRA/Number of vertices of the original route) for different lengths of the original (laid without avoiding HRA) route. 95% confidence intervals are indicated in parentheses for the ratio of lengths and the number of vertices

Distance intervals (km)	Routes	Average ratio of lengths	Average ratio of the number of vertices
1-2	63	1.58 (1.404 – 1.747)	1.52 (1.33 – 1.69)
2-3	156	1.36 (1.299 – 1.423)	1.25 (1.17 – 1.32)
3-4	354	1.22 (1.192 – 1.245)	1.18 (1.13 – 1.22)
4-5	471	1.16 (1.142 – 1.174)	1.09 (1.06 – 1.11)
5-6	562	1.13 (1.122 – 1.146)	1.07 (1.04 – 1.09)
6-7	659	1.12 (1.107 – 1.125)	1.02 (1.00 – 1.04)
7-8	626	1.10 (1.088 – 1.103)	1.02 (1.01 – 1.04)
8-9	593	1.09 (1.085 – 1.100)	1.02 (1.00 – 1.03)
9-10	502	1.08 (1.077 – 1.091)	1.00 (0.98 – 1.02)
10-11	349	1.07 (1.069 – 1.083)	1.00 (0.97 – 1.02)

Continuation of Table 3-3

Distance intervals (km)	Routes	Average ratio of lengths	Average ratio of the number of vertices
11-12	220	1.07 (1.060 – 1.076)	1.00 (0.96 – 1.03)
12-13	119	1.06 (1.054 – 1.072)	1.01 (0.97 – 1.04)
13-14	45	1.05 (1.040 – 1.061)	1.05 (1.00 – 1.09)
14-15	9	1.04 (1.021 – 1.058)	1.05 (1.01 – 1.08)

Table 3-4. Descriptive ratio statistics (Length at of HRA avoiding route/Original route Length) for selected intervals of the original route length. For the mean, median, and standard deviation, 95% confidence intervals are shown in parentheses

Length km	N	Average	Median	Standard deviation	Min	Max	Range (Max–Min)
1-5	1044	1.23 (1.216 – 1.255)	1.12 (1.106 -1.137)	0.317 (0.286 - 0.358)	1.001	3.521	2.52
5-9	2440	1.11 (1.105 - 1.114)	1.07 (1.061 - 1.072)	0.115 (0.102 - 0.121)	1.000	1.707	0.707
9-13	1190	1.08 (1.073 - 1.081)	1.06 (1.060 - 1.067)	0.069 (0.067 - 0.077)	1.000	1.439	0.439
13-15	54	1.05 (1.04 - 1.059)	1.04 (1.028 - 1.045)	0.034 (0.027 - 0.043)	1.005	1.151	0.146

The results of calculations showing the route that avoids HRAs compared to binned lengths of the original route are shown in Table 3-3 and in Fig. 3-4.

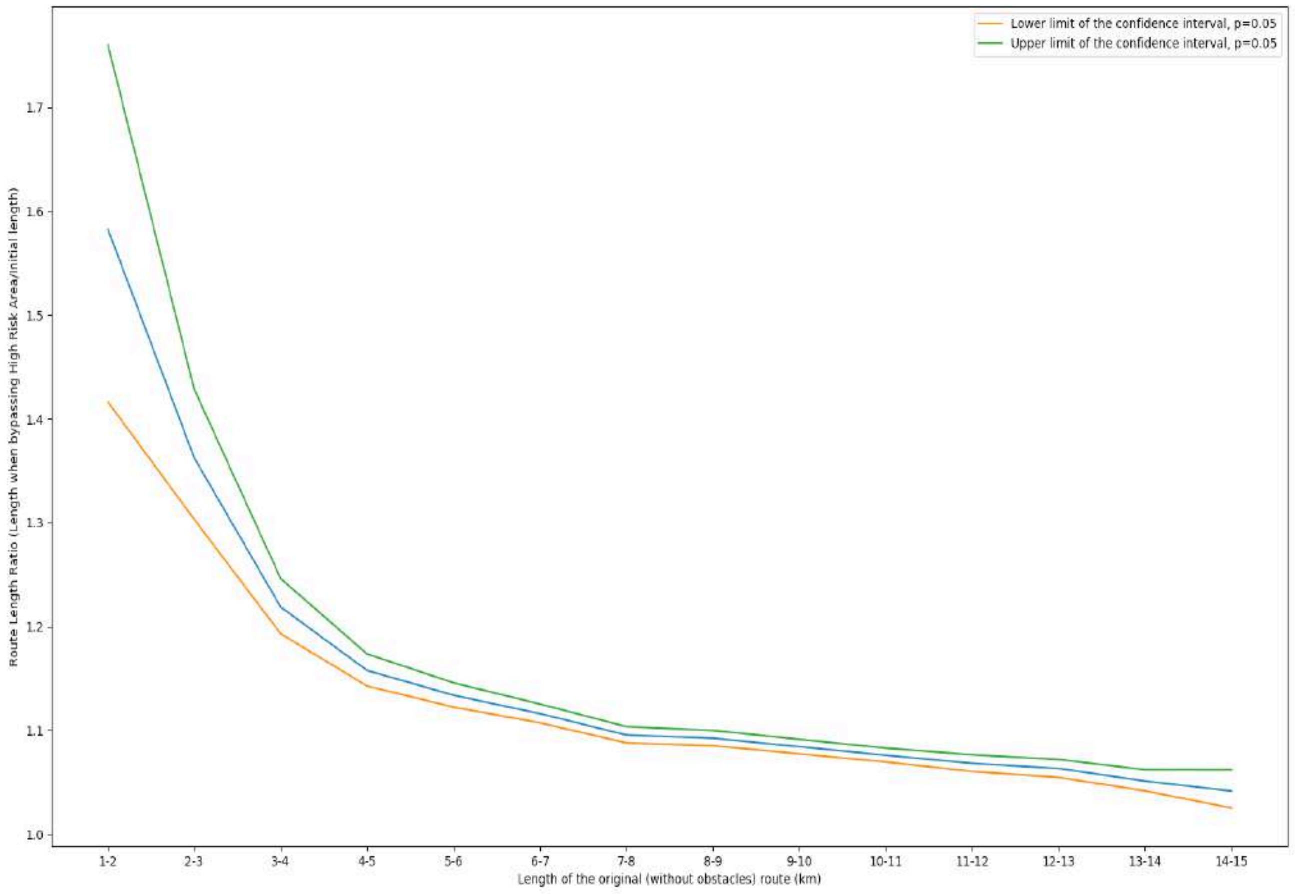

Figure 3-4. Dependence of the average ratios of route lengths (Length when bypassing the HRA/Original length) for different lengths of the original route constructed without considering the HRA areas

Finally, Table 3-4 provides more detailed descriptive statistics of the ratios (Length when bypassing the HRA /Initial length) for four-kilometer intervals of the original route length, from which it is apparent that all indicators (mean, median, standard deviation, the difference between the maximum and minimum values) gradually decrease as the length of the original route increases.

3.3 Relative risk of an accident when choosing a route bypassing the HRA

As evident from the previous section, bypassing the HRA guarantees an increase in the length of the route. Simultaneously, it is not at all obvious that the alternative route will be safer. It is therefore critical to assess the relative risk of an accident when choosing a route that bypasses the HRA.

Suppose it is necessary to plot a route from A to B. Let's denote A-----> B a route with a bypass of the HRA, and A-> B is a route without a bypass of the HRA. Let TA1 be the number of severe traffic accidents registered along route A-----> B, and TA2 be the number of severe traffic accidents along route A->B. Then the TA1/TA2 ratio will be a measure of the relative risk on the two routes (Relative Risk Ratio - RRR).

To calculate the relative risk, it is necessary to enumerate the coordinates of all severe traffic accidents that occurred in Springfield in 2013-2018 and determine for each severe traffic accident the closest edge to it in the graph G0. To do this, I use the get_nearest_edge() method of the OSMnx package.

Unfortunately, get_nearest_edge() gives unambiguous results only for unidirectional edges and is unable to determine which direction of the bidirectional edge a particular accident belongs to. To find tease this out, it would seem that you can compare the data on the direction of the accident with the data on the direction of the route (for example, S, W, N, E) to which this edge of the graph belongs. But the data on the direction of movement is not known for all TAs, besides, they are not particularly reliable and one observes that for a route going from north to south, the direction W or E may still be listed.

Thus, I can only assign half of all the accidents found to each direction of the bidirectional face and hope that averaging over many edges for a sufficiently long route compensates for the inaccuracy of determining the number of accidents for a specific direction of the bidirectional edge.

So, bearing in mind the comments about the number of accidents for a bidirectional edge, I will calculate for each pair of routes A-----> B and A-> B the number of accidents along the route, after which I can obtain general descriptive statistics of the TA1/TA2 ratios for all routes (Table 3-5).

Table 3-5. General descriptive statistics for ratio of severe Traffic Accidents (TAs along A-----> B/TAs along A-> B)

Metric	Value	95% confidence interval
Mean	0.91	(0.89 – 0.93)
Median	0.84	(0.83 – 0.842)
Standard deviation	0.73	(0.49 – 0.96)
Minimum	0.13	
Maximum	27.75	

Table 3-6. Dependence of average ratios (TAs along A-----> B/TAs along A-> B) for binned route lengths A-> B

Distance, km	N	Average ratio of TAs	95% confidence interval
1-2	63	1.32	(0.834 - 1.689)
2-3	156	1.16	(0.978 - 1.335)
3-4	354	1.02	(0.948 - 1.087)
4-5	471	0.94	(0.894 - 0.979)
5-6	562	0.88	(0.850 - 0.909)
6-7	659	0.85	(0.828 - 0.874)
7-8	626	0.85	(0.830 - 0.871)
8-9	593	0.84	(0.823 - 0.859)
9-10	502	0.82	(0.808 - 0.842)
10-11	349	0.82	(0.803 - 0.846)
11-12	220	0.81	(0.789 - 0.839)
12-13	119	0.84	(0.809 - 0.870)
13-14	45	0.84	(0.796 - 0.886)
14-15	9	0.83	(0.738 - 0.926)

Table 3-7. Descriptive statistics for ratio of severe TAs (TAs along A-----> B/TAs along A-> B) for selected intervals of the original route length. 95% confidence intervals are indicated in parentheses for the mean, median and standard deviation

Length km	**N**	**Average**	**Median**	**Standard deviation**	**Min**	**Max**	**Range Max-Min**
1-5	1044	1.02 (0.97 - 1.07)	0.85 (0.84 - 0.87)	0.80 (0.64 - 0.95)	0.17	10.0	9.83
5-9	2440	0.86 (0.84 - 0.87)	0.83 (0.82 - 0.84)	0.29 (0.27 - 0.31)	0.25	3.57	3.32
9-13	1190	0.82 (0.81 - 0.840)	0.84 (0.82 - 0.85)	0.20 (0.19 - 0.21)	0.27	1.56	1.29
13-15	54	0.83 (0.80 - 0.88)	0.85 (0.83 - 0.9)	0.15 (0.12 - 0.18)	0.43	1.22	0.79

Table 3-6 shows the relative risk of an accident as a function of the length of the original route A-> B. As before, the bootstrap method was used to calculate the confidence interval. The same results as in Table 3-6 are shown more clearly in Figure 3-5, where the gray horizontal line corresponds to a relative risk level of 1.0. The values themselves are shown in blue, and the 95% confidence intervals are shown in orange (lower bound) and green (upper bound).

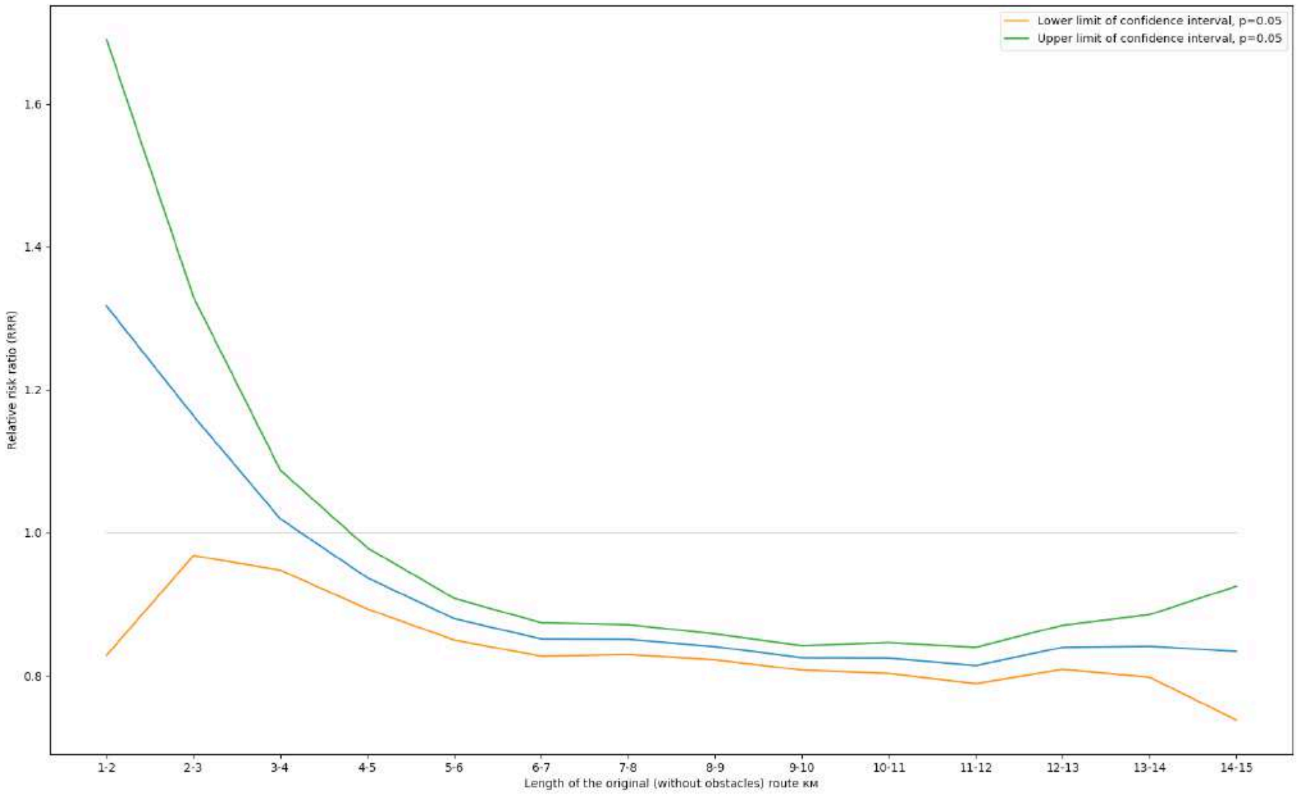

Figure 3-5. Dependence of the average ratios (TAs along A-----> B/TAs along A-> B) for binned route lengths A-> B

As is apparent, the reduction of relative risk occurs for routes that are at least of moderate length (in this case those that are longer than 4 kilometers). As for the shorter routes (less than 4 kilometers), there is a substantial increase in the length of the alternative route (avoiding HRAs), as per Table 3-3, and consequently an increased number of severe TAs along this new path.

In Table 3-7 a more detailed picture is presented of the descriptive statistics for the ratio of severe traffic accident (severe traffic accident along A-----> B/severe traffic accidents along A-> B) for 4-kilometer intervals of the original route. It is observed that, all the statistical metrics decrease (standard deviation, range) or reach a plateau (average, median) as the original route length increases.

Considering Table 3-4 and 3-7 together, one may juxtapose the increase in mean ratio of route length with the change in mean relative risk of severe Traffic Accidents for different bins of the original route length. The findings are summarized in Table 3-8.

For example, for the interval 5 to 9 kilometers, one observed an increase in the average path length of 11% and a reduction of relative risk of severe Traffic Accidents by 14%.

Table 3-8. Dependence of change in average relative risk (RRR) on average increase in route length bypassing the HRA, for different intervals of the original route.
The corresponding 95% confidence intervals are shown in parentheses

Length of the original route km	**Increase in the average ratio of route lengths, %**	**Change in the average relative risk of TA, %**
1-5	23% (21,6% - 25,5%)	+2% (-3%, + 7%)
5-9	11% (10,5% - 11,4%)	-14% (-13%, - 16%)
9-13	8% (7,3% - 8,1%)	-18% (-16%, -19%)
13-15	5% (4,1% - 5,9%)	-17% (-12%, -20%)

Note: “Collision risk” will henceforth be used as synonymous to “severe traffic accident risk”, for brevity.

For the original (not considering HRA) route lengths in the 1-5 km interval, the HRA bypass route was on average 23% (95% confidence interval 21.6% - 25.5%) longer than the route not including obstacles, and the collision risk increased by 2% on average (95% confidence interval: -3%, +7%).

For the original (not including HRA) route lengths in the 5-9 km interval, the HRA bypass route is on average 11% (95% confidence interval 10.5% - 11.4%) longer than the route that does not consider obstacles, and the collision risk decreases by an average of 14% (95% confidence interval: 13%-16%).

For the original (not including HRA) route lengths in the 9-13 km interval, the HRA bypass route is on average 8% (95% confidence interval 7.3% - 8.1%) longer than the route that does not consider obstacles, and the collision risk of crash decreases by an average of 18% (95% confidence interval: 16%-19%).

For the original (not including HRA) route lengths in the 13-15 km interval, the HRA bypass route is on average 5% (95% confidence interval 4.1% - 5.9%) longer than the route that does not take into account obstacles, and the collision risk decreases by an average of 17% (95% confidence interval: 12%-20%).

Conclusions

The meta-clusters of severe traffic accidents identified in Chapter 2 (see also [48]) may be used by the appropriate services to identify the causes leading to the occurrence of accidents and reduce their incidence.

For car drivers, taxi driver, and buses operators in the city of Springfield, when using a route with a length of more than 4 km, I can recommend an algorithm that allows one to bypass the meta-clusters of severe TAs described in this chapter, and thereby reduce the risk of a severe traffic accident by an average of 16% with an increase in the length of the route by an average of 8%.

Chapter 4. A simple way to improve road safety by bypassing dangerous sections of the route (using the city of Moscow as a case study)

Introduction

It is shown in [57], by the case-study of the city of Springfield, MA that by bypassing dangerous sections of the road network (clusters of TAs), it is possible to reduce the relative risk of an accident with an associated limited increase in the length of the route and the number of traversable vertices of the graph representing the road network. Building a safer route required some manual work in [57]: it was necessary to consider all the roads (edges of the road graph) leading to a HRA (cluster) and mark them in such a way that they became impassable for the routing algorithm. For a small city, this did not require much effort, but for a megalopolis like Moscow, this task becomes too time-consuming. In addition, there are always doubt whether it is worth banning all paths leading to the cluster, or perhaps only some of the most "dangerous" ones?

This chapter proposes a slightly different method based on the method of determining the danger of individual road sections (edges of the road graph) without clustering. The null hypothesis was that severe traffic accidents are distributed uniformly along the road network. Accordingly, 1,000 Monte Carlo computer simulations were executed and the number of simulated severe traffic accidents on each road segment was tabulated along with the 95th and 99th percentiles. Subsequently, those road segments along which the actual observed number of severe traffic accidents was greater than the 95th percentile, or 99th percentile of the simulated data, were considered statistically significant at the 5% and 1% level, respectively. These statistically significant edges were marked in the modified road graph by adding a fixed number to their length attribute, which made a conventional routing algorithm, such as the Dijkstra algorithm [63] or Bellman-Ford algorithm [69, 70], change the route and include less dangerous (unmarked) edges of the road graph. Note that in [57] an essentially infinite number was added to the edge length attribute, which led to the fact that such an edge was practically excluded from the route. As shown in this chapter, this approach was not optimal and adding to the length attribute a finite number, comparable with the average route length, leads to better results.

4.1 Data

I will use the global OpenStreetMap (OSM) map of the world to obtain the road network of Moscow. OSM maps are usually of excellent quality and are widely used in works devoted to routing (see, for example, [13] which touches a similar problem of pedestrian avoidance of criminally dangerous areas). The easiest way to get the road network of Moscow is to use the OSMnx library for the Python language (see [6]). The resulting road network is a multidigraph, that is, a directed graph, the edges of which can be unidirectional and bidirectional, and through any two vertices of the graph can pass any number of edges. The OSMnx package allows you to save the original multidigraph as two separate objects - edges and vertices - in vector ".shp" format. This allows you to display the road network in a geographic information system (for example, in QGIS [59]).

The Moscow road network obtained from OSM contains 25,397 edges (15,696 unidirectional and 9,701 bidirectional) and 16,797 vertices. Descriptive statistics of some edge attributes are given in Table 4-1.

As for the accident data, they are obtained from an official GIBDD source [60]. For this study, I selected data on 27,798 road accidents recorded in Moscow in 2019-2021. Since the Moscow road network and accident data are obtained from independent sources, it is necessary to delete accidents that clearly do not belong (for example, due to a registration error) to the road network. For this purpose, I included for further analysis only those accidents whose distance to the nearest edge of the road network is less than 35m[10]. There were 21,956 such accidents (79% of the total number).

Table 4-1. Descriptive statistics of some attributes of Moscow road graph edges

Statistical measure	Length m	Speed km/h	Travel time min.
Average	217.4	51.7	15.9
St. Deviation	275.5	12.13	20.9
Min	1.06	5	0.1
25% percentile	47.2	41.9	3.3
Median	127.9	50.5	9.2
75% percentile	291.6	53.1	21.2
Max	8043	100	673

4.2 Tools and method

The list of filtered TAs that occurred in Moscow in 2019-2021, which I discussed in the previous paragraph, contains the coordinates of the TA, the distance from the accident to the nearest edge and the edge itself in the form of <vertex1> <vertex2> <key>, where <vertex1> and <vertex2> are the identifiers of the vertices of the road graph through which an edge passes, and <key> is the number of the edge passing through specific vertices. In the case when there is only one edge <key> = 0. In the case when there are two edges, there are two keys numbered 0,1, etc. Thus, the keys allow you to identify edges passing through the same vertices. Now you can sort through all the edges of the road graph and search for each one in this list, thereby determining which accidents belong to this edge. Naturally, for bidirectional edges, you need to look for both the edge <vertex 1> <vertex 2> <key> and the edge <vertex 2> <vertex 1> <key>. The resulting list of edges can be sorted by the number of accidents belonging to them, and then significant edges can be identified by conducting Monte Carlo simulations.

[10] If I consider the width of the lane to be 3.5 m, and the maximum number of lanes to be 10, then 35 m will serve as an estimate of the maximum width of the road in Moscow.

For Monte Carlo simulations, I used the software package SANET [47], which allows one to obtain millions of points evenly distributed along the road network. The tests consist in forming groups of 21956 points (as many as there were severe traffic accidents in 2019-2021) and for each group all edges are enumerated again and the number of points belonging to each edge is found. By performing, say, 1000 runs, I get the statistics of accidents when the accidents are evenly distributed over the road network. If the number of real accidents of an edge exceeds the 95th or 99th percentile obtained from the Monte Carlo simulations, I will consider the number of accidents of that edge to be significant at 5% and 1% level, respectively. The "dangerous" edges obtained in this way can be used in the same way as in [57].

To do this, I will create two identical road graphs G0 and G1. I will leave graph G0 unchanged, and in graph G1 I will change the <Length> attribute of the edges selected as statistically significant, adding to it a sufficiently large number, the same fixed value for all such edges, to show the routing algorithm that this edge is undesirable in the route being laid and it should be avoided whenever possible.

I just need to compare all the routes laid on graphs G0 and G1 and make sure that the routes on graph G1 are in some sense safer. To do this, I will set a square grid with a fixed step equal to 3 km in our case. (Fig. 4-1), and I will construct all routes starting and ending at the nodes of this grid[11].

Let's denote any two grid nodes as A and B and find the route A->B on graph G0 and the route A—>B on graph G1. To compare these routes, I calculate the relative risk ratio (RRR) equal to the ratio of accidents that occurred along route A—>B to the number of accidents that occurred along route A->B. If the upper bound of the 95% confidence interval for the average RRR is < 1, then the routes laid on graph G1 are safer than the routes defined on the original graph G0.

[11] More precisely, beginning and ending in the vertices of the graph closest to the nodes of the grid

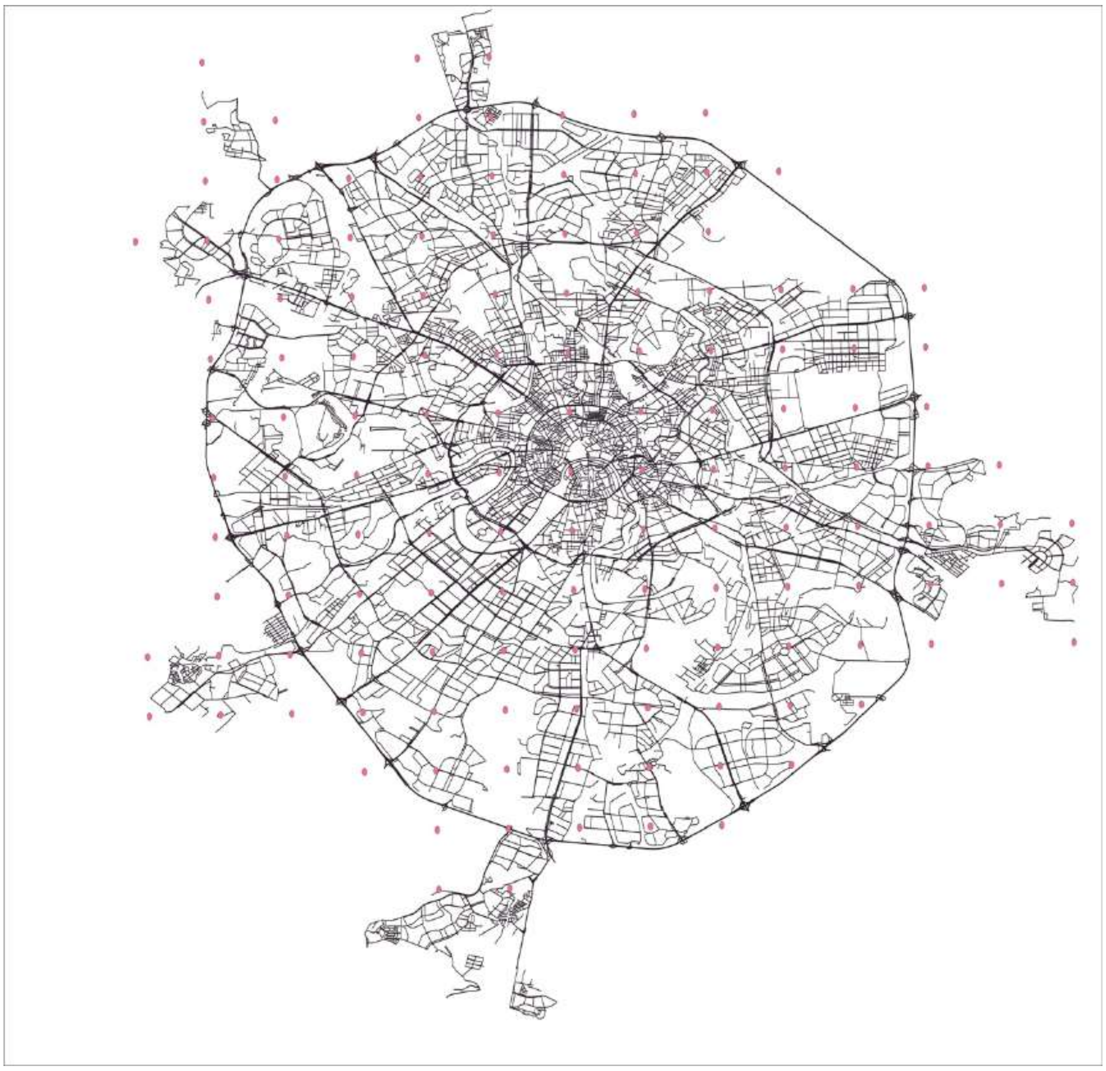

Figure 4-1. Nodes of the square grid placed on the road map of Moscow

Note that in [57], only routes that differ for graphs G0 and G1 were used to collect statistics. In this chapter, all routes are used, because depending on the value added to the <Length> attribute, the number of routes that differ for G0 and G1 will be different.

4.3 Results

For statistical tests I chose the first 2000 edges of the road graph, which contain the largest number of accidents, because the calculations for all 25397 edges are extremely time consuming and provide minimal added value. One thousand (1,000) tests were performed for every edge out of 2000, 815 edges were significantly "dangerous" at 0.05

level and 469 edges of the road graph - at 0.01 level. The elementary statistics for the "dangerous" edges are shown in Table 4-1. Table 4-1 demonstrates that at the 0.01 significance level, obstacles are characterized by a greater average number and a greater dispersion (as per Interquartile Range, Q3-Q1) in the number of severe TAs.

Table 4-2. The elementary statistics for the "dangerous" edges

P	p = 0.05	p = 0.01
Total edges	815	469
Average	6.6	7.8
St. deviation	5.4	6.3
Min	3	3
25% percentile (Q1)	4	4
Median	5	6
75% percentile (Q3)	7.5	9
Max	58	58

Fig. 4-2 shows 815 obstacle edges on the map of Moscow. Edges with different numbers of accidents are binned and shown in different colors (3-7 accidents – green, 7-20 accidents – red, more than 20 accidents - yellow). It should be noted that the edges with the largest number of accidents are located on the Central Ring Road, as well as on the section of Kutuzovsky Avenue from Minskaya Street to Alexey Sviridov Street and on the section of the Third Ring from the Enthusiasts Highway to Novaya Perevedenovskaya.

After obtaining obstacles, the order of actions becomes the same as in [57]. It is necessary, using a square grid placed on the map of Moscow (see Fig. 4-1), to sort through all the starting and ending points and for each route A->B calculated without obstacles, calculate the route A—>B having the same starting and ending points, but considering obstacles. Next, one should compare all such routes and understand how the relative risk of an accident (RRR), the length of the route and the number of traversable vertices of the road graph changes on average. In contrast to [57], I conducted this analysis for different values of the penalty imposed on the obstacle in order to understand how it affects the effectiveness of alternative routing.

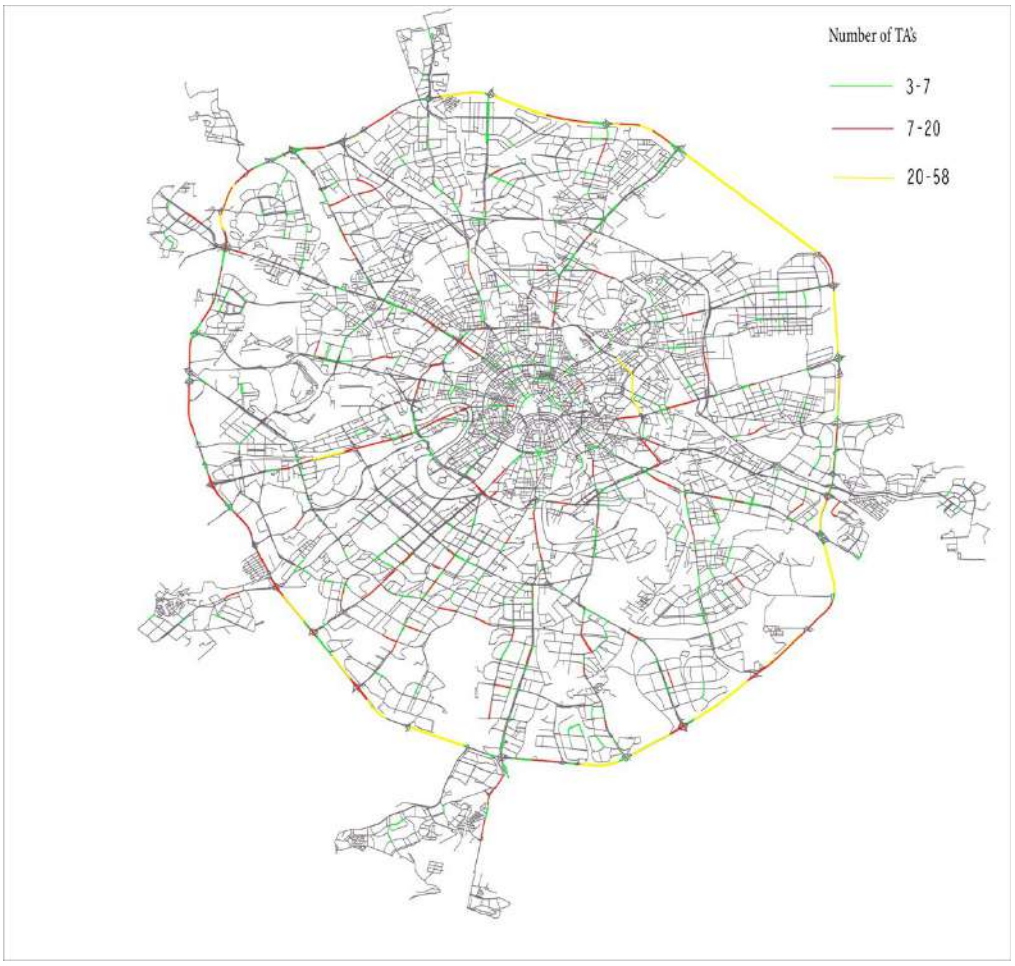

Figure 4-2. Obstacles displayed on the Moscow road map

The results of the calculations are shown in Figures 4-3...4-5 and in Tables 4-3...4-5. In addition to the values themselves, Tables 4-3...4-5 show in a smaller font and the figures show the less saturated color of the 95% confidence intervals of the values obtained by the bootstrap method. The first 800 edges significant at the 0.05 level were used as obstacles, and the values of 200, 500, 1000, 2000, 5000, and 10000000000000 meters were used as penalties. The latter value, which far exceeds all conceivable distances in Moscow, can be considered infinite, as shown in Figures 4-3...4-5 and Tables 4-3...4-5.

As can be seen from the figures and tables, the optimal value of the penalty is equal to 2000m. At this value the curve showing the dependence of the average relative risk of accident on the length of the original route (Fig. 4-3) lies slightly below all others and the minimum value of the relative risk reaches, according to Table 4-3, the value of 0.686.

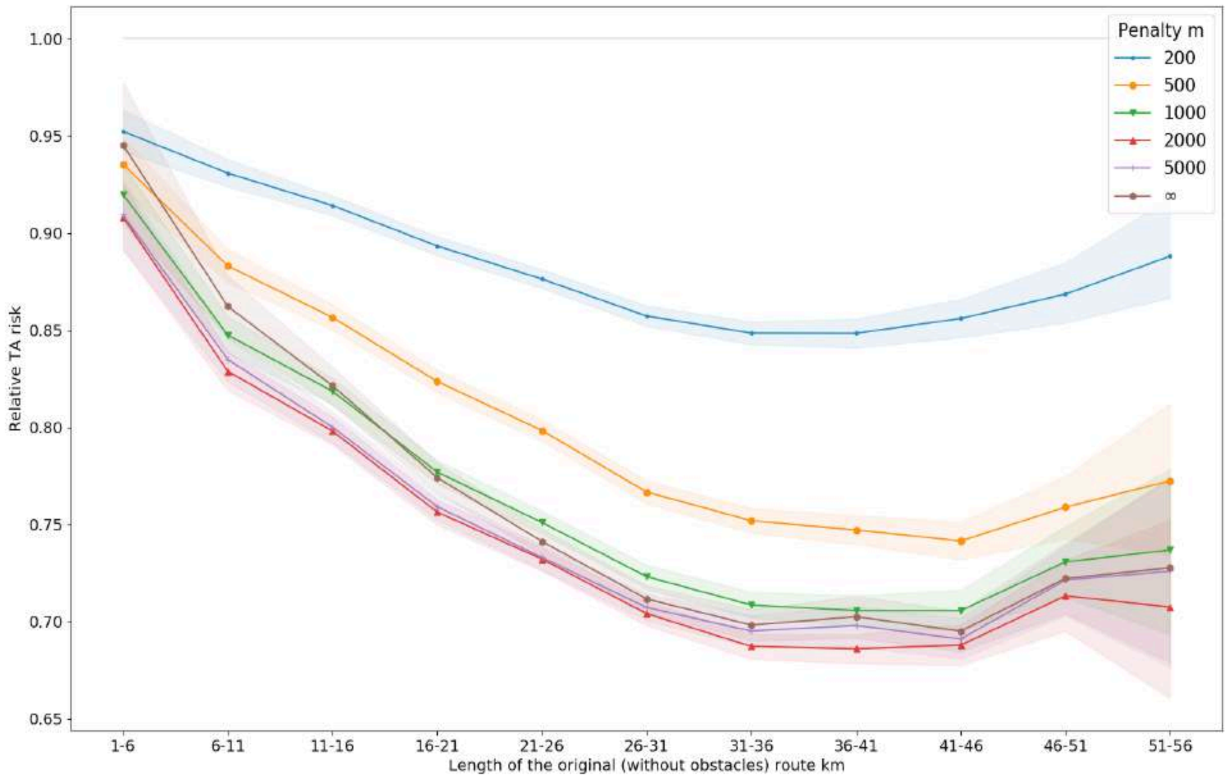

Figure 4-3. Relative TA risk as a function of the length of the original route and the penalty for an obstacle edge (with 95% confidence intervals)

Table 4-3. Relative TA risk as a function of the length of the original route and the penalty for an obstacle edge (with 95% confidence intervals)

Interval km	**Number of routes**	**200 m**	**500 m**	**1000 m**	**2000 m**	**5000 m**	∞
1-6	535	0.952 0.942-0.963	0.935 0.922-0.949	0.920 0.905-0.935	0.908 0.892-0.925	0.909 0.892-0.927	0.945 0.907-0.977
6-11	1524	0.931 0.924-0.938	0.883 0.874-0.891	0.847 0.838-0.857	0.829 0.819-0.839	0.835 0.824-0.846	0.862 0.846-0.877
11-16	2679	0.914 0.909-0.919	0.857 0.850-0.863	0.819 0.812-0.825	0.798 0.791-0.805	0.800 0.792-0.808	0.821 0.812-0.831
16-21	3065	0.893 0.888-0.898	0.824 0.818-0.829	0.777 0.771-0.783	0.756 0.750-0.763	0.759 0.752-0.766	0.774 0.766-0.782
21-26	3255	0.876 0.872-0.881	0.798 0.793-0.804	0.751 0.745-0.757	0.732 0.726-0.738	0.733 0.727-0.739	0.741 0.734-0.748
26-31	2854	0.857 0.852-0.863	0.767 0.761-0.773	0.723 0.717-0.729	0.704 0.698-0.710	0.707 0.701-0.714	0.712 0.705-0.719
31-36	2196	0.849 0.843-0.855	0.752 0.745-0.758	0.709 0.702-0.715	0.687 0.681-0.694	0.695 0.687-0.703	0.698 0.690-0.706
36-41	1315	0.848 0.841-0.856	0.747 0.740-0.755	0.706 0.698-0.713	0.686 0.678-0.693	0.698 0.687-0.708	0.703 0.692-0.713

Continuation of Table 4-3.

Interval km	Number of routes	200 m	500 m	1000 m	2000 m	5000 m	∞
41-46	591	0.856 0.846-0.866	0.742 0.732-0.751	0.706 0.695-0.716	0.688 0.678-0.698	0.691 0.680-0.701	0.695 0.685-0.706
46-51	179	0.869 0.854-0.885	0.759 0.743-0.775	0.731 0.713-0.749	0.713 0.695-0.731	0.721 0.703-0.740	0.722 0.704-0.741
51-56	32	0.888 0.867-0.917	0.772 0.735-0.813	0.737 0.693-0.780	0.708 0.660-0.752	0.726 0.676-0.774	0.728 0.679-0.774
Share of changed routes		0.7	0.84	0.9	0.92	0.94	0.94

Moreover, at this penalty the dependence of route length ratios (length including obstacles/length without obstacles) on the length of the original route (Fig. 4-4) behaves much more smoothly than at a penalty equal to infinity and 5000m. According to Table 4-4, the average route length ratio at a 2000m penalty ranges from 1.064 to 1.105, that is (see Tables 4-3, 4-4) an average relative risk reduction of 9-31% is achieved at a 2000m penalty by increasing the average route length by 6-11%.

As can be seen from Figure 4-4, with an infinite value of the penalty and small lengths of the original path, the ratio of route lengths increases greatly, apparently because the routing algorithm has difficulty in finding an alternative route. Meanwhile, the 2000m penalty gives a very smooth dependence and a small change in the average path length ratio after/before (6%-10.5%)

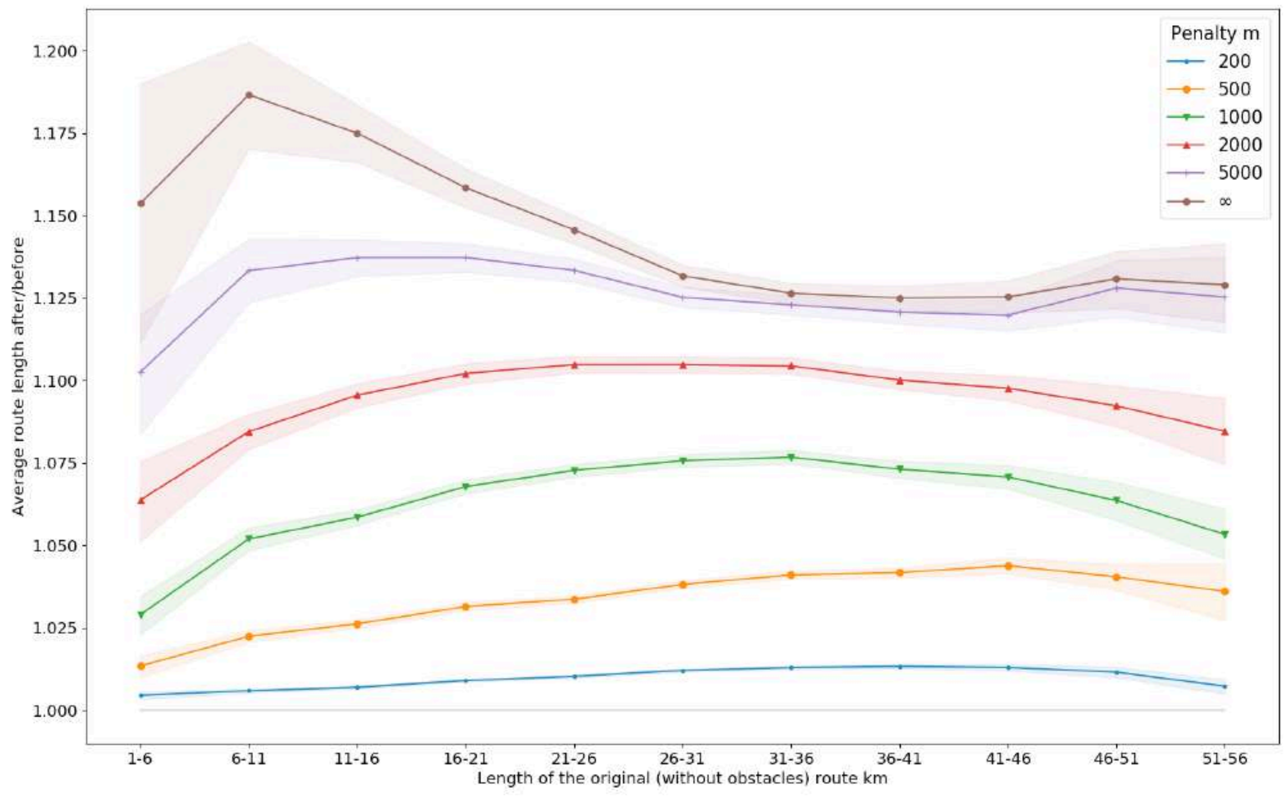

Figure 4-4. Dependence of the average ratios of route lengths (Length when bypassing the HRA/ Initial length) for different lengths of the original route and different values of the penalty for the obstacle edge

Table 4-4. Average ratios of route lengths (after/before) for different lengths of the original route and different penalty values for the obstacle edge (with 95% confidence intervals)

Interval km	**Number of routes**	**200** m	**500** m	**1000** m	**2000** m	**5000** m	∞
1-6	535	1.005 1.003-1.006	1.013 1.010-1.016	1.029 1.023-1.035	1.064 1.051-1.075	1.103 1.084-1.120	1.154 1.111-1.190
6-11	1524	1.006 1.005-1.007	1.022 1.021-1.024	1.052 1.048-1.055	1.084 1.079-1.090	1.133 1.124-1.143	1.187 1.170-1.203
11-16	2679	1.007 1.007-1.007	1.026 1.025-1.027	1.059 1.056-1.061	1.096 1.092-1.099	1.137 1.132-1.143	1.175 1.166-1.184
16-21	3065	1.009 1.009-1.009	1.031 1.030-1.033	1.068 1.066-1.070	1.102 1.099-1.105	1.137 1.133-1.142	1.158 1.152-1.164
21-26	3255	1.010 1.010-1.011	1.034 1.033-1.035	1.073 1.071-1.075	1.105 1.102-1.107	1.133 1.130-1.137	1.146 1.141-1.150
26-31	2854	1.012 1.012-1.013	1.038 1.037-1.039	1.076 1.074-1.078	1.105 1.102-1.107	1.125 1.122-1.128	1.132 1.128-1.135
31-36	2196	1.013 1.012-1.014	1.041 1.040-1.042	1.077 1.075-1.079	1.104 1.102-1.107	1.123 1.120-1.126	1.126 1.123-1.130
36-41	1315	1.013 1.013-1.014	1.042 1.040-1.043	1.073 1.071-1.076	1.100 1.097-1.103	1.121 1.117-1.124	1.125 1.121-1.129

Continuation of Table 4-4

Interval km	**Number of routes**	**200** m	**500** m	**1000** m	**2000** m	**5000** m	∞
41-46	591	1.013 1.012-1.014	1.044 1.042-1.046	1.071 1.067-1.074	1.098 1.094-1.102	1.120 1.115-1.125	1.125 1.120-1.130
46-51	179	1.012 1.010-1.013	1.040 1.037-1.044	1.064 1.058-1.069	1.092 1.086-1.098	1.128 1.119-1.136	1.131 1.122-1.139
51-56	32	1.007 1.005-1.009	1.036 1.027-1.044	1.053 1.046-1.061	1.085 1.074-1.095	1.125 1.114-1.137	1.129 1.118-1.141

The decrease in relative risk is accompanied, as it follows from Fig. 4-5 and Table 4-5, by another loss: an increase in the number of traversable vertices of the road graph by about 6-27.6%, depending on the length of the original route.

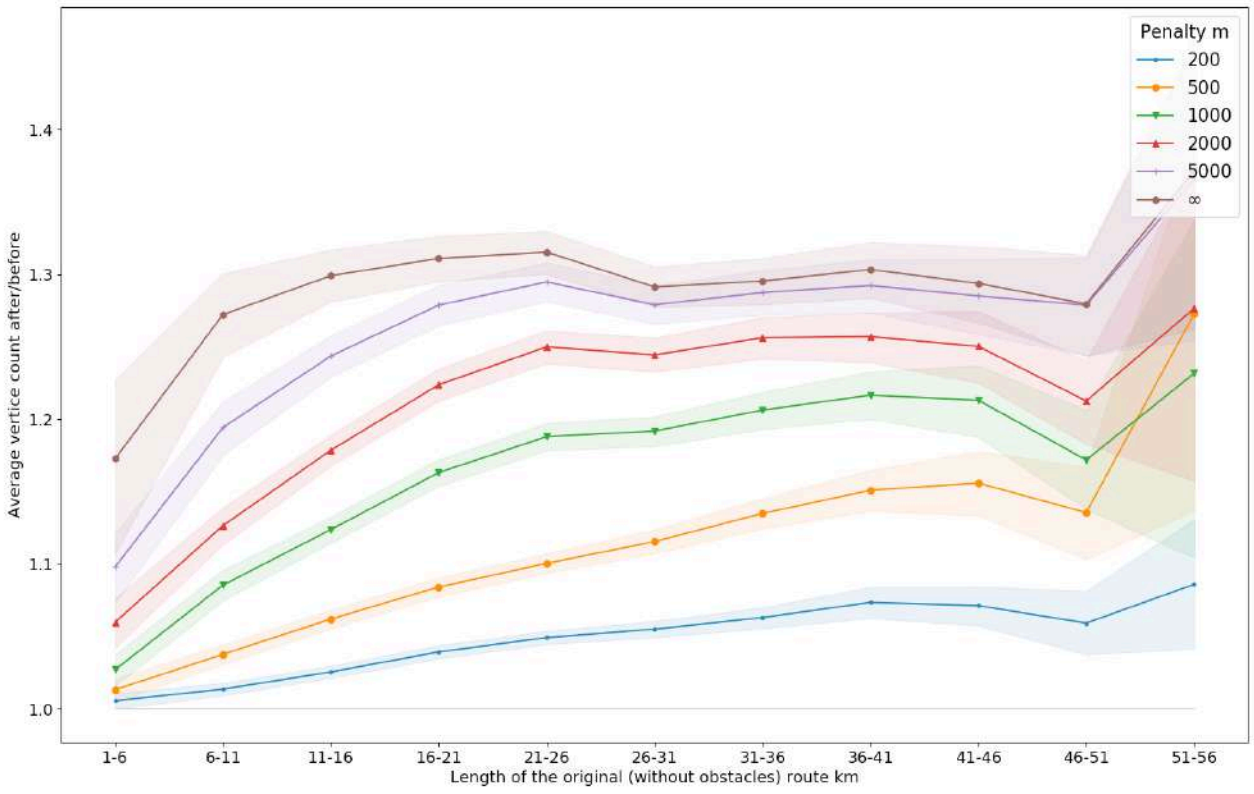

Figure 4-5. Average ratios of the number of vertices of routes (after/before) for different lengths of the original route and different penalty values for the obstacle edge (with 95% confidence intervals)

Table 4-5. Average ratios of the number of vertices of routes (after/before) for different lengths of the original route and different penalty values for the obstacle edge (with 95% confidence intervals)

Interval km	**Number of routes**	**200** m	**500** m	**1000** m	**2000** m	**5000 m**	∞
1-6	535	1.006 1.000-1.011	1.013 1.005-1.021	1.027 1.015-1.038	1.060 1.042-1.076	1.098 1.073-1.121	1.173 1.108-1.227
6-11	1524	1.014 1.009-1.018	1.038 1.031-1.045	1.086 1.075-1.096	1.127 1.114-1.139	1.194 1.175-1.213	1.272 1.243-1.300
11-16	2679	1.026 1.021-1.030	1.062 1.056-1.068	1.124 1.115-1.132	1.179 1.168-1.190	1.244 1.229-1.258	1.299 1.281-1.317
16-21	3065	1.039 1.035-1.044	1.084 1.078-1.091	1.163 1.154-1.172	1.224 1.213-1.235	1.279 1.265-1.292	1.311 1.296-1.326
21-26	3255	1.049 1.045-1.054	1.101 1.094-1.107	1.188 1.179-1.197	1.250 1.238-1.261	1.295 1.281-1.308	1.315 1.300-1.329
26-31	2854	1.055 1.050-1.061	1.116 1.107-1.124	1.192 1.182-1.202	1.244 1.233-1.256	1.279 1.266-1.292	1.291 1.277-1.305
31-36	2196	1.063 1.056-1.070	1.135 1.124-1.145	1.206 1.193-1.219	1.256 1.241-1.271	1.287 1.271-1.303	1.295 1.279-1.311
36-41	1315	1.074 1.063-1.084	1.151 1.137-1.165	1.217 1.200-1.232	1.257 1.240-1.274	1.292 1.274-1.310	1.303 1.284-1.322
41-46	591	1.071 1.058-1.084	1.156 1.133-1.177	1.213 1.188-1.237	1.250 1.225-1.275	1.285 1.259-1.311	1.294 1.268-1.319
46-51	179	1.059 1.038-1.082	1.136 1.103-1.167	1.172 1.137-1.206	1.213 1.183-1.240	1.279 1.243-1.311	1.280 1.243-1.312
51-56	32	1.086 1.041-1.131	1.273 1.139-1.395	1.232 1.103-1.339	1.276 1.151-1.375	1.366 1.252-1.458	1.372 1.261-1.463

Finally, I will briefly focus on the optimal choice of obstacles (their number and level of significance), which would provide the best value of the relative risk of an accident. Unfortunately, this problem still has to be solved empirically – with the help of rather laborious calculations. First of all, let's compare the first 400 obstacles with a significance level of 0.05 and 0.01, designated as 95_400 and 99_400 in Fig. 4-6. It is perceptible that the reliability of 0.01 turns out to be preferrable, which is not surprising, because according to Table 4-2 (and common sense), edges that are reliable at the level of 0.01, contain more accidents (on average) and are more suitable for the role of obstacles. Now let's compare the different number of obstacles, reliable at the level of 0.05, and indicated in Fig. 4-6 as 95_400, 95_500 and 95_800. It is apparent that in our particular case, the largest number of obstacles 800 turns out to be optimal, although it is impossible to deduce from this the rule "the more, the better", since too many obstacles may leave few alternatives for changing the route, which will lead to an increase in the relative risk of an accident instead of reducing it.

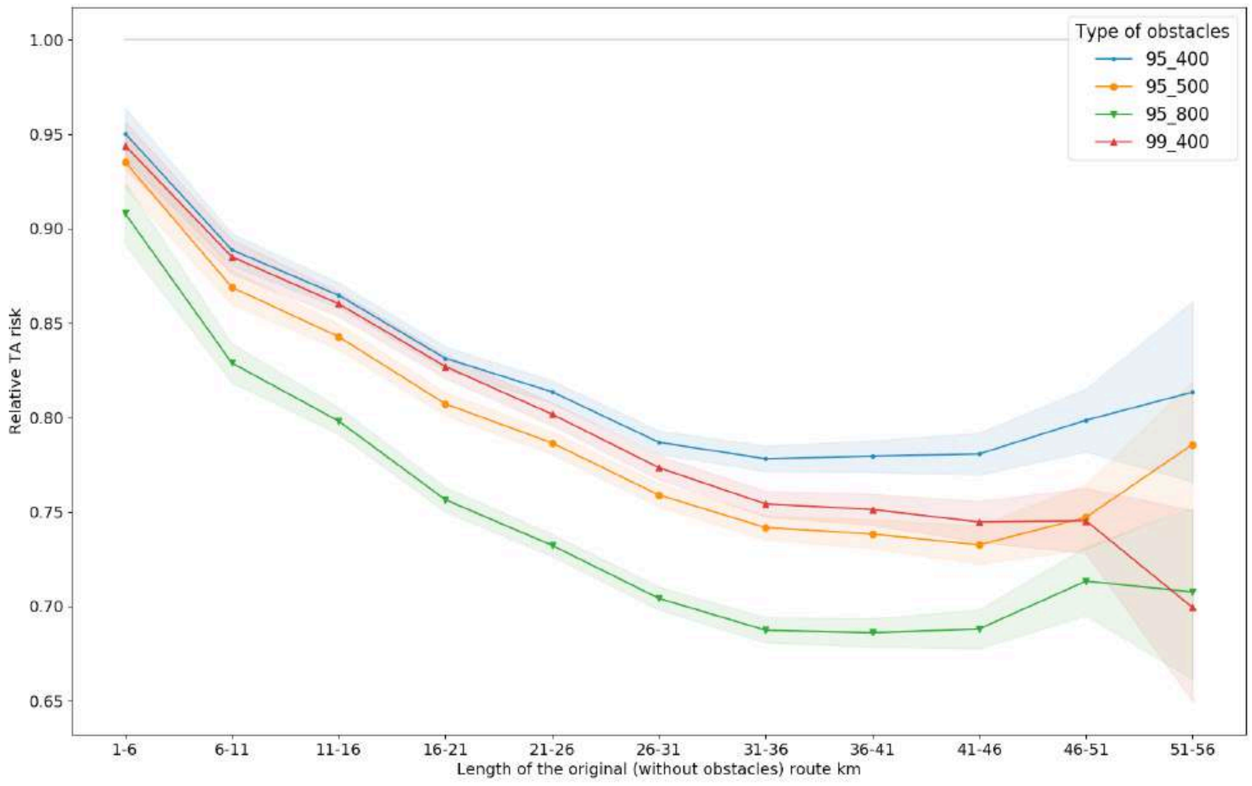

Figure 4-6. Dependence of the average relative risk of accidents for different obstacles and different lengths of the original route (penalty = 2000m)

Conclusions

Using the example of Moscow, a way to improve road safety is considered, which consists in building a route that bypasses obstacles identified on the road map (graph). The obstacles are the edges of the road graph containing a statistically significantly higher number of severe traffic accidents (TAs).

To check the effectiveness of routing, the indicator proposed in [57] is used – the relative risk of an accident is equal to the ratio of the number of accidents along the route that takes into account obstacles to the number of accidents counted along the original route built without taking into account obstacles.

It is shown that obstacle avoidance reduces the relative risk of an accident by 9-31% (depending on the length of the original route) by increasing the average length of the route by 6-11% and increasing the average number of traversable vertices of the road graph by 6-27.6%.

Chapter 5. Bypassing dangerous sections of the route to improve traffic safety (case study of St. Petersburg)

Introduction

In Chapter 4, using the example of Moscow, a way to improve road safety was developed, consisting in building a route that bypasses obstacles identified on the road map (graph). The obstacles are the edges of the road graph (road segments) containing a statistically significantly higher number of severe traffic accidents (TAs). Each identified obstacle is subject to a penalty – adding some value to the length attribute of the corresponding edge of the road graph, after which the usual routing algorithm is launched, determining the route of the minimum (taking into account penalties) length.

In this chapter, the same approach is applied to the city of St. Petersburg (see [61]). The results of calculations show that in the case of St. Petersburg, the selection and subsequent bypass of obstacles can significantly reduce the average relative risk of an accident due to a limited increase in the length of the route and the number of vertices of the road graph traversed along the route. Hence, the algorithm developed in Chapter 4 is validated by the new data provided by Saint-Petersburg traffic history.

5.1 Data

As in the previous chapter, I will use the OpenStreetMap (OSM) global map of the world to obtain the road network, this time for the city of St. Petersburg. The easiest way to load the road network is to use the OSMnx library for the Python language (see [6]).

The resulting road network is a multidigraph, that is, a directed graph, the edges of which can be unidirectional and bi-directional, and through any two vertices of the graph can pass any number of edges. The package OSMnx allows you to save the original graph as two separate objects - edges and vertices in vector format .shp, which allows you to display the road network in a geographic information system, such as QGIS [59].

Table 5-1. Parameters of the Moscow and St. Petersburg road networks

Metric	**Moscow**	**St. Petersburg**
Total vertices	16797	14624
Total edges	25415	22787
One-directional edges	15696	15222
Two-directional edges	9719	7565

Table 5-2. Descriptive statistics for lengths (in meters) of edges of road networks for Moscow and St. Petersburg

Metric	**Moscow**	**St. Petersburg**
Average length	217.4	155.1
St. deviation	275.5	219.2
Min	1.06	0.66
25% percentile	47.2	24.1
Median	127.9	81.9
75% percentile	291.6	197.9
Max	8043	4542

Despite the much smaller population of St. Petersburg compared to Moscow, it's road network obtained from OSMnx is not much simpler, as shown in Table 5-1, compared to that of Moscow and has 30352 edges (35134 edges for Moscow). The statistics of the lengths of the edges of the road graph for both cities, given in Table 5-2, shows that the road segments (edges of the road graph) for St. Petersburg are on average shorter than for Moscow (by 36%, as measured by the median metric).

As for the data on traffic accidents, they were obtained from the official source of the traffic police (GIBDD) [60]. For this study, I selected records of 16,865 serious (that is, resulting in injury or death to at least one of the participants) traffic accidents recorded in

St. Petersburg in 2019-2021. Since the St. Petersburg road network and accident data were obtained from independent sources, it was necessary to remove accidents that clearly did not belong (e.g., due to registration error) to the road network. For this purpose, only those accidents whose distance to the nearest edge of the road network is less than 35m[12] were left for further analysis. The total number of such accidents was 13503 (80% of the total number).

5.2 Tools and method

As in [58], for each edge of the road graph, one can count the number of severe TAs belonging to it and then find edges in which the number of accidents exceeds, say, the 95th percentile of the values obtained as a result of Monte Carlo numerical simulations described in sufficient detail in [58]. Considering the number of accidents of such edges statistically significant (at the level of 0.95), I will continue to use such "dangerous" edges as obstacles.

To do this, I will create a graph G0 representing the road network of St. Petersburg, and it’s copy - graph G1. I will leave graph G0 unchanged, and in graph G1 I will add a sufficiently large number (penalty) to the length attribute of the edges selected by hypothesis testing using Monte Carlo simulations. The penalty will be the same for all these dangerous edges and will imply to the routing algorithm that these edges are undesirable and should be avoided if possible.

I just need to compare all the routes laid on graphs G0 and G1 and make sure that the routes on graph G1 are in some sense safer. To do this, I will set a square grid with a fixed step equal to 3 km in this case (see Fig. 5-1), and I will plot all routes starting and ending at the vertices of the graph closest to the grid nodes.

[12] If I consider the width of the lane to be 3.5 m, and the maximum number of lanes to be 10, then 35 m will serve as an estimate of the maximum width of the road in Moscow.

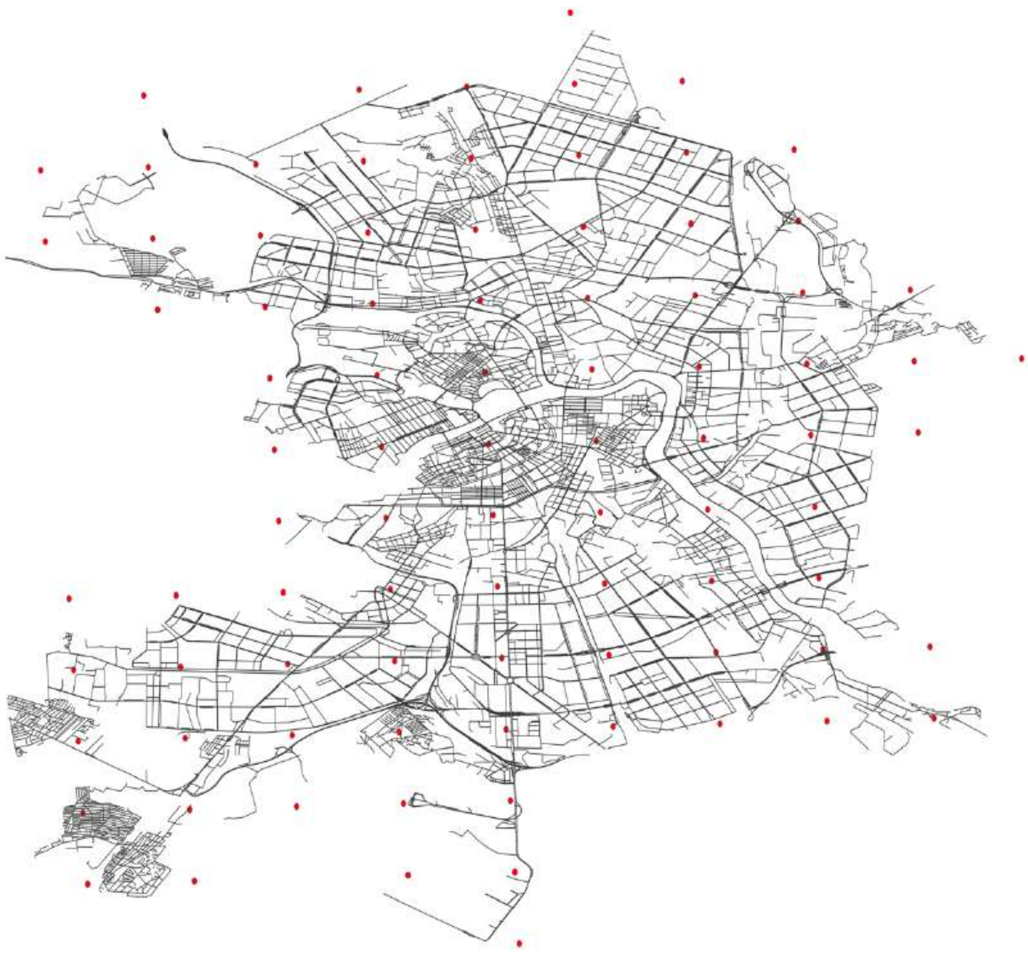

Figure 5-1. Nodes of the square grid placed on the road map of St. Petersburg

Let's denote any two grid nodes as A and B and find the route A->B on graph G0 and the route A—>B on graph G1. To compare these routes, I calculate the relative risk ratio of accidents (RRR) equal to the ratio of accidents that occurred along route A—>B to the number of accidents that occurred along route A->B. If the upper bound of the confidence interval for the RRR < 1, then the routes along the graph G1 are safer than the routes defined on the original graph G0.

5.3 Results

The first 2,000 edges of the road graph containing the largest number of severe traffic accidents were chosen for hypothesis testing, because calculations for all 30352 edges are extremely time-consuming[13] and provide little added value.

[13] All calculations were performed on a workstation with a quad-core processor and 32GB of RAM

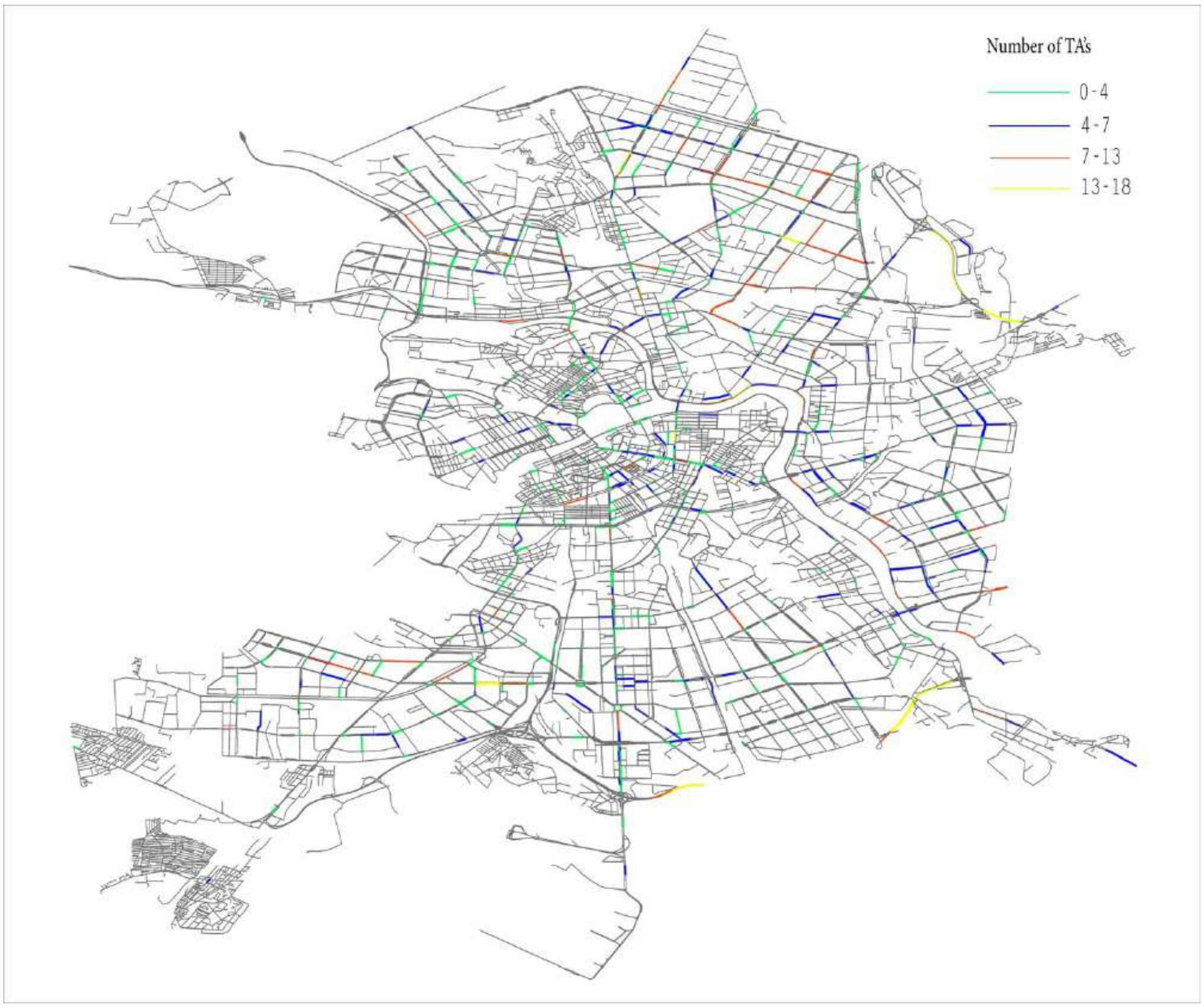

Figure 5-2. Obstacles displayed on the road map of St. Petersburg

In total, 1000 tests were carried out for each edge out of 2000, significantly "dangerous" were at the level of 0.95 – 983 edges, of which, as in [58], the first 800 edges containing the largest number of severe traffic accidents were used. Fig. 5-2 shows 983 obstacle edges on the map of St. Petersburg. Edges with different numbers of accidents are binned and shown in different colors (0-4 accidents – green, 4-7 accidents – blue, 7-13- red, 13-18 accidents - yellow). Table 5-3 shows a list of 10 road segments (with the coordinates of the beginning and end of the segment) with the largest number of accidents.

After receiving obstacles, the procedure steps is the same as in [58]. It is necessary, using a square grid placed on the map of St. Petersburg, to sort through all the starting and ending points of the grid and for each route A->B laid without obstacles, calculate the route A->B having the same starting and ending points, but laid with obstacles. Next, one should compare all such routes and understand how the relative risk of an accident, the length of the route and the number of traversable vertices change on average. This analysis was carried out for various values of the penalty imposed on the obstacle to understand how it affects the effectiveness of alternative routing.

Table 5-3. Ten Road segments in St. Petersburg with the highest number of severe traffic accidents

Road segment	Start (latitude. longitude)	End (latitude. longitude)	TAs
KAD[14] 58-60 км. внутр.	59.8523, 30.4745	59.8352, 30.4484	18
KAD 81-83км. внешн.	59.8344, 30.4477	59.8524, 30.4753	18
KAD 74км	59.8114, 30.3423	59.8151, 30.3615	17
KAD 35-37км	60.0155, 30.4664	59.9931, 30.4798	16
Liteyniy Ave from . Pestelja St. to Nekrasova St.	59.9429, 30.3484	59.9390. 30.3482	15
Engel’sa Ave from Pr. Lunacharskogo. to Vyborg highway	60.0438, 30.3279	60.0361, 30.322	14
Leninskiy Ave from ul. Ziny Portnovoy to Bulvar Novatorov	59.8518, 30.2564	59.8518, 30.2678	14
Sverdlovskaya Naberezhnaya from Ul. Vatutina to Ul. Arsenal’naya	59.9595, 30.3839	59.9549, 30.3746	14
Prospect Nauki from Grazhdanskiy Prospect to Ul. Butlerova	60.0130, 30.3978	60.011, 30.4058	14
KAD 102km	59.9824, 30.5070	59.9866, 30.4904	14

Calculation results are shown in Figures 5-3...5-5 and in Tables 5-4...5-6. In addition to the values themselves, Tables 5-4...5-6 show in smaller font, and the figures show in less saturated color the 95% confidence intervals of the values obtained by the bootstrap

[14] St. Petersburg Ring Road (КАД in Russian)

method. The first 800 edges significant at the 0.95 level were used as obstacles, and the values of 200, 500, 1000, 2000, and 5000 meters were used as penalties.

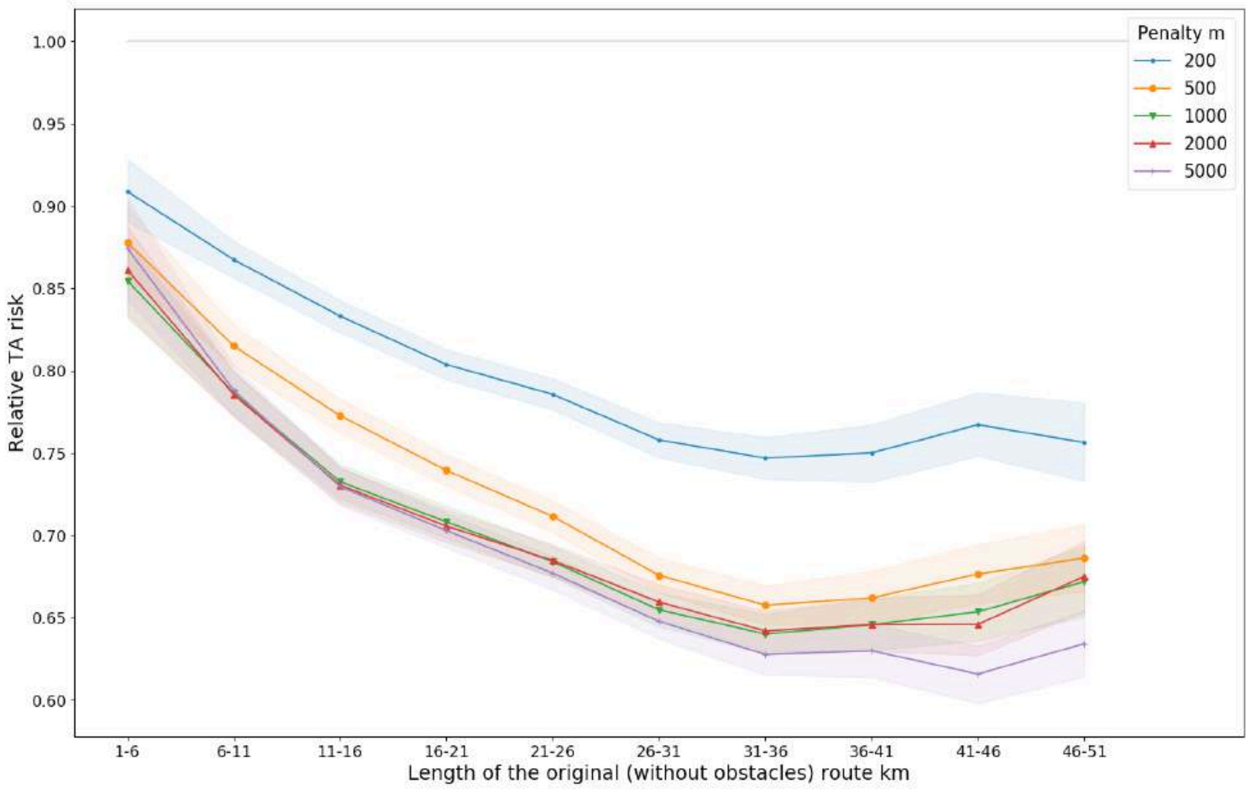

Figure 5-3. Relative risk of an TA depending on the length of the initial route and the penalty for the obstacle edge with 95% confidence intervals

If only the relative risk of an accident is taken into account, then a penalty equal to 5000m will be optimal (see Figure 5-3). Table 5-4 shows that the minimum risk value for this penalty is 0.616. But at the same time, with a penalty of 5000m (see Figures 5-4 and 5-5), the cost in the length of the route and the number of traversable vertices of the road graph will greatly increase.

As Figure 5-3 and Table 5-4 show, penalties equal to 1000 m and 2000 m give almost the same dependence of the average relative risk on the length of the initial route. But at the same time, as it follows from Figures 5-4 and 5-5, for a penalty of 1000 m, the loss in the length of the route and the number of traversable vertices of the road graph is significantly reduced. Therefore, a penalty of 1000 m should be considered optimal for Saint-Petersburg.

Table 5-4. Relative risk of an TA depending on the length of the initial route and the penalty for the obstacle edge with 95% confidence intervals

Interval km	Number of routes	200 m	500 m	1000 m	2000 m	5000 m
1-6	297	0.909 0.890-0.928	0.878 0.857-0.899	0.855 0.831-0.879	0.861 0.834-0.888	0.874 0.844-0.904
6-11	853	0.867 0.857-0.878	0.815 0.803-0.828	0.786 0.773-0.800	0.785 0.771-0.799	0.788 0.773-0.803
11-16	1231	0.833 0.824-0.843	0.773 0.763-0.783	0.733 0.722-0.743	0.730 0.720-0.742	0.730 0.718-0.741
16-21	1272	0.804 0.795-0.813	0.739 0.730-0.749	0.708 0.699-0.718	0.706 0.696-0.715	0.703 0.692-0.713
21-26	1088	0.786 0.776-0.795	0.712 0.702-0.721	0.684 0.674-0.694	0.685 0.675-0.694	0.677 0.667-0.687
26-31	768	0.758 0.747-0.769	0.676 0.665-0.686	0.655 0.645-0.665	0.660 0.649-0.670	0.648 0.637-0.658
31-36	493	0.747 0.734-0.760	0.658 0.646-0.670	0.640 0.628-0.652	0.642 0.630-0.654	0.628 0.615-0.640
36-41	233	0.750 0.733-0.768	0.662 0.646-0.678	0.646 0.630-0.661	0.646 0.630-0.662	0.630 0.613-0.646
41-46	97	0.767 0.748-0.787	0.677 0.659-0.695	0.654 0.636-0.672	0.646 0.628-0.664	0.616 0.598-0.633
46-51	53	0.757 0.733-0.780	0.686 0.666-0.707	0.672 0.651-0.694	0.675 0.654-0.696	0.634 0.614-0.655
Proportion of changed routes		0.82	0.88	0.92	0.94	0.95

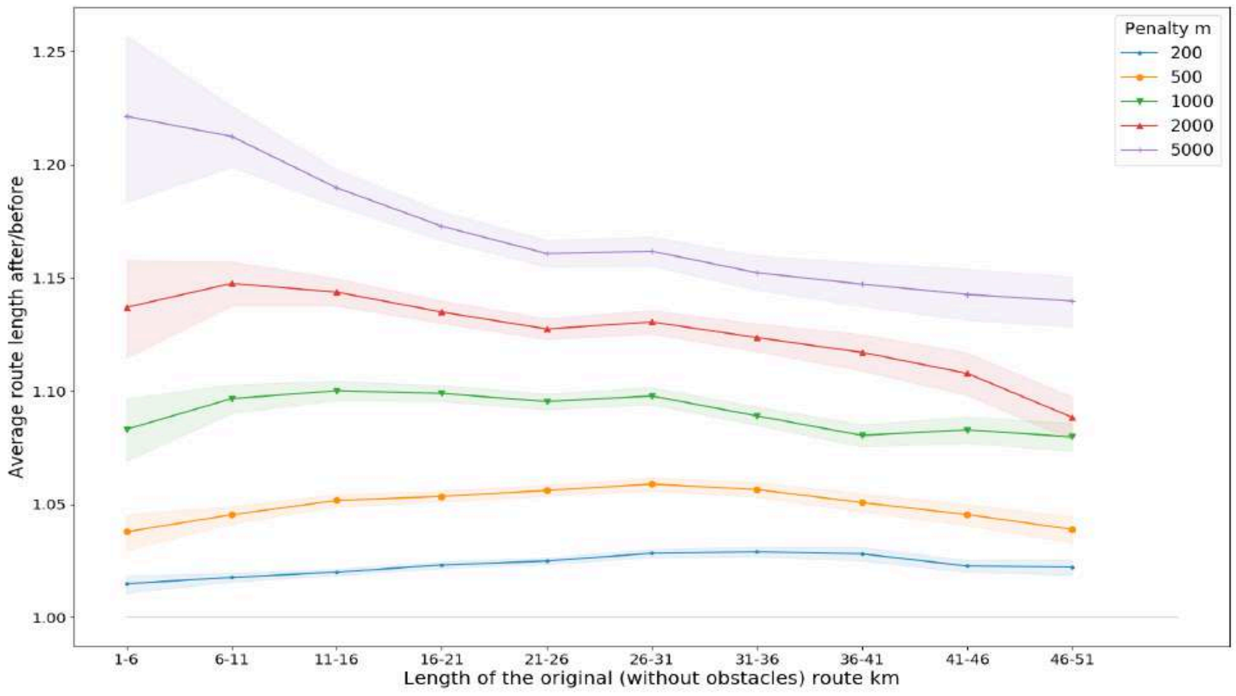

Figure 5-4. Dependence of the average ratios of route lengths (Length when bypassing the HRA/ Initial length) for different lengths of the original route and different values of the penalty for the obstacle edge

Table 5-5. Average ratios of route lengths (after/before) for different lengths of the original route and different penalty values for the obstacle edge (with 95% confidence intervals)

Interval km	Number of routes	200 m	500 m	1000 m	2000 m	5000 m
1-6	297	1.015 1.011-1.018	1.038 1.030-1.045	1.083 1.070-1.096	1.137 1.115-1.158	1.221 1.183-1.257
6-11	853	1.018 1.016-1.019	1.045 1.041-1.049	1.097 1.090-1.103	1.148 1.138-1.157	1.213 1.199-1.226
11-16	1231	1.020 1.018-1.021	1.052 1.049-1.054	1.100 1.096-1.104	1.144 1.138-1.150	1.190 1.182-1.198
16-21	1272	1.023 1.022-1.025	1.053 1.051-1.056	1.099 1.095-1.103	1.135 1.130-1.140	1.173 1.167-1.179
21-26	1088	1.025 1.023-1.026	1.056 1.054-1.059	1.095 1.092-1.099	1.127 1.123-1.132	1.161 1.155-1.166
26-31	768	1.028 1.026-1.030	1.059 1.056-1.062	1.098 1.094-1.101	1.130 1.125-1.136	1.162 1.155-1.168
31-36	493	1.029 1.027-1.031	1.056 1.053-1.060	1.089 1.085-1.093	1.124 1.117-1.130	1.152 1.145-1.160
36-41	233	1.028 1.025-1.031	1.051 1.047-1.055	1.080 1.075-1.085	1.117 1.109-1.125	1.147 1.138-1.157
41-46	97	1.023 1.020-1.025	1.045 1.041-1.050	1.083 1.077-1.089	1.108 1.098-1.117	1.143 1.131-1.154
46-51	53	1.022 1.019-1.026	1.039 1.033-1.045	1.080 1.074-1.086	1.088 1.078-1.097	1.140 1.128-1.150
Proportion of changed routes		0.82	0.88	0.92	0.94	0.95

For this penalty value, according to Table 5-5, the average ratio of route lengths ranges from 1.08 – 1.1, that is (see Table 5-5,) the average reduction in relative risk by 14.5-36% is achieved with a penalty of 1000m by increasing the average route length by 8-10%. The decrease in relative risk is accompanied, as it follows from Fig. 5-6 and Table 5-6, by another cost: an increase in the number of traversable vertices of the road graph by about 3-12% (for a fine of 1000 m), depending on the length of the original route.

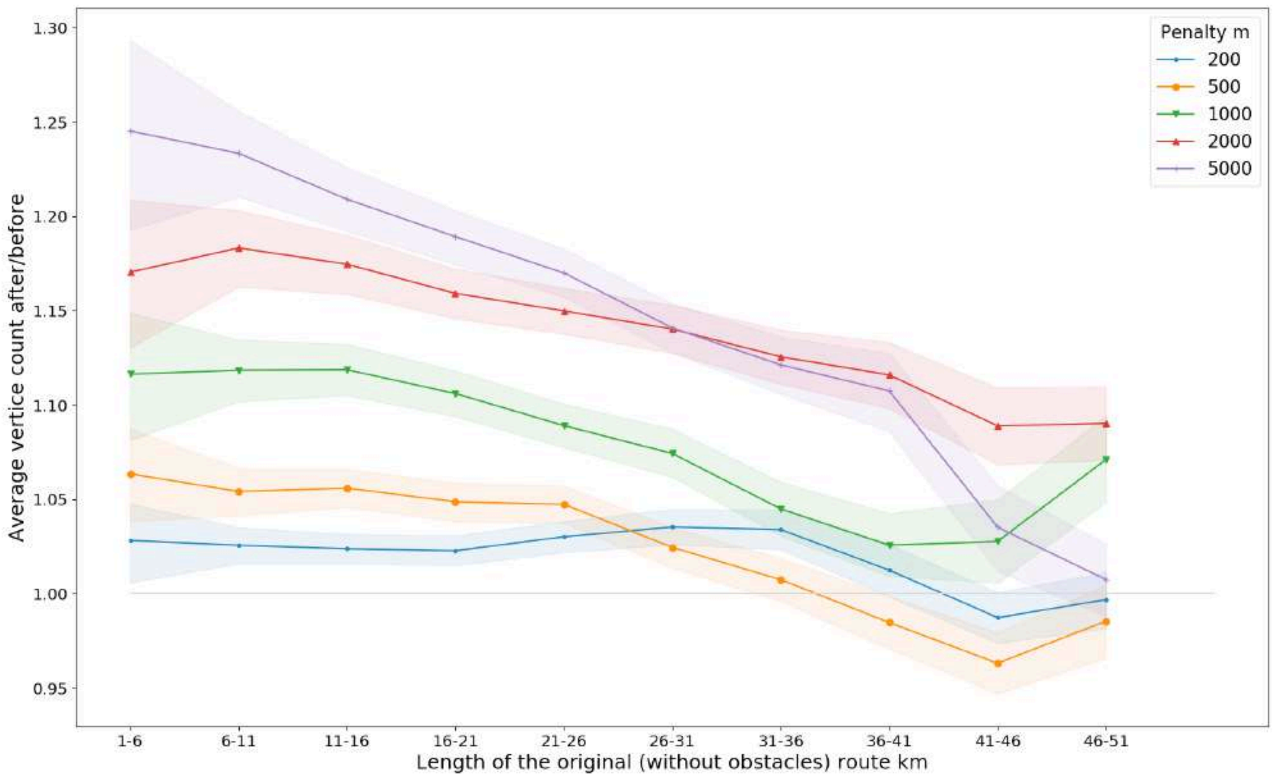

Figure 5-5. Average ratios of the number of vertices of routes (after/before) for different lengths of the original route and different penalty values for the obstacle edge (with 95% confidence intervals)

Table 5-6. Average ratios of the number of vertices of routes (after/before) for different lengths of the original route and different penalty values for the obstacle edge (with 95% confidence intervals)

Interval km	Number of routes	200 m	500 m	1000 m	2000 m	5000 m
1-6	297	1.028 1.007-1.047	1.064 1.038-1.088	1.116 1.081-1.149	1.171 1.129-1.208	1.245 1.193-1.296
6-11	853	1.026 1.016-1.035	1.054 1.041-1.067	1.118 1.102-1.134	1.183 1.163-1.203	1.233 1.210-1.256
11-16	1231	1.024 1.016-1.032	1.056 1.046-1.066	1.119 1.105-1.132	1.175 1.159-1.190	1.209 1.192-1.226
16-21	1272	1.023 1.015-1.031	1.049 1.039-1.059	1.106 1.094-1.118	1.159 1.146-1.173	1.189 1.175-1.203
21-26	1088	1.030 1.022-1.038	1.047 1.037-1.057	1.089 1.077-1.101	1.150 1.138-1.162	1.170 1.158-1.183
26-31	768	1.035 1.026-1.045	1.025 1.014-1.036	1.074 1.061-1.087	1.140 1.127-1.153	1.141 1.128-1.154
31-36	493	1.034 1.023-1.045	1.007 0.996-1.019	1.045 1.031-1.059	1.126 1.111-1.140	1.121 1.106-1.136

Continuation of Table 5-6

Interval km	Number of routes	200 m	500 m	1000 m	2000 m	5000 m
36-41	233	1.012 0.999-1.026	0.985 0.971-0.999	1.026 1.009-1.042	1.116 1.098-1.133	1.107 1.086-1.128
41-46	97	0.987 0.973-1.001	0.963 0.947-0.980	1.028 1.006-1.050	1.089 1.069-1.109	1.035 1.012-1.058
46-51	53	0.997 0.982-1.011	0.985 0.966-1.004	1.071 1.049-1.094	1.090 1.071-1.110	1.007 0.988-1.026
Share of changed routes		0.82	0.88	0.92	0.94	0.95

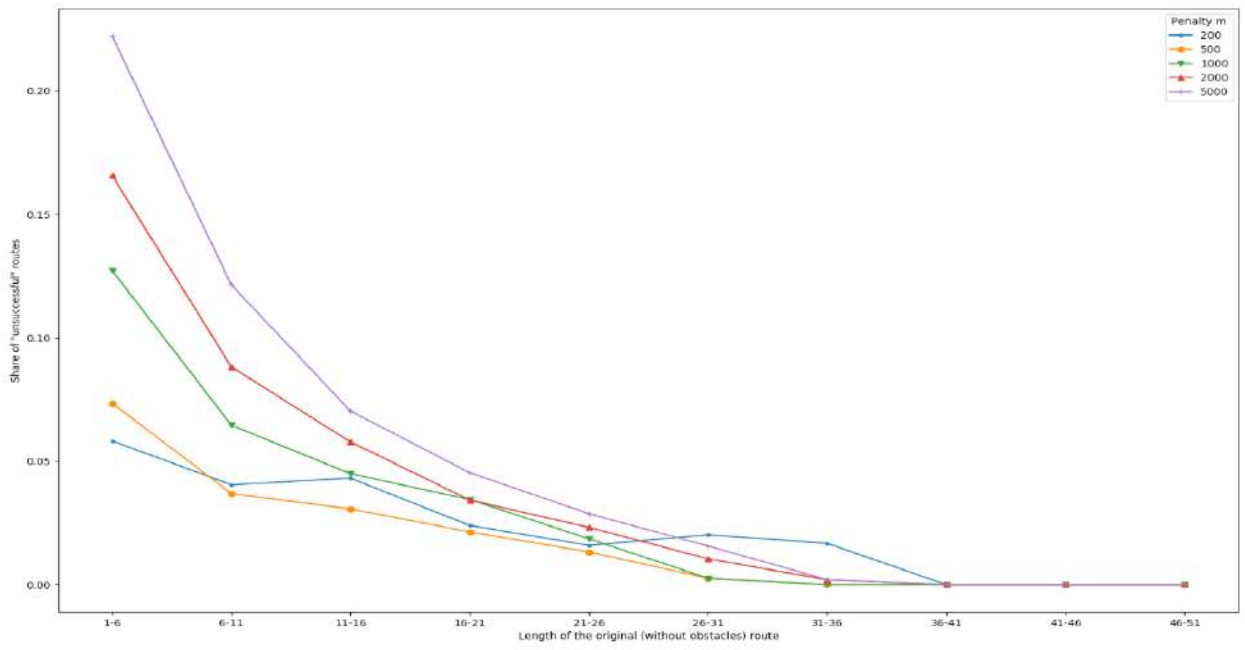

Figure 5-6. The proportion of routes for which a safety gain is not achieved, depending on the length of the original route and the penalty

Figures 5-3...5-5 and Tables 5-4...5-6 give values of average values (average relative risk, average route lengthening etc.). However, a particular route calculated with obstacles in mind may not coincide with these average values. In particular, a route may not provide a safety benefit compared to a route based on the original (unchanged) road network. Figure 5-6 shows the proportion of these routes as a function of the length of the original route.

It is clearly seen From Fig. 5-6 that the share of unsuccessful routes is maximal for short (1-6 km) original route lengths and reaches 0.23 (e.g. 23%) for the highest penalty value (5000m). For the optimal value of the penalty (1000 m) the share of unsuccessful routes ranges from 12.7% (for the shortest routes with length less than 6 km) to 0% (for routes longer than 31 km). Hence, for practical calculations it is necessary to check each calculated route and if it is not safer use a different value of the penalty or refuse alternative routing at all.

Findings

Bypassing obstacles, which are the edges of the road graph of St. Petersburg[15], containing a statistically significantly large number of severe traffic accidents (TAs), reduces the average relative risk of accidents by 14.5-36% (depending on the length of the original route) by increasing the average length of the route by 8-10% and increasing the average number of passable vertexes of the road graph by 3-12%.

Obtaining similar results for St. Petersburg by repeating the algorithm previously used for Moscow [58], allows us to talk about the stability of the proposed method and the possibility of its application for other cities.

[15] It means that a fine of 1000m is imposed on each obstacle.

Chapter 6. A package of programs modifying a road graph according to the number of accidents belonging to it

6.1 Introduction

This chapter will describe in detail a package of Python programs that allows one to modify a road graph in such a way that applying conventional routing algorithms to it leads to the construction of safer routes on average, as shown in Chapters 4 and 5. Constructing a modified road graph is a rather complex task (both algorithmically and in terms of computational size). It seems to me that it is more convenient to solve this problem using a set of sequentially applied compact programs than using one immense program. In addition, I use third-party programs (SANET) to solve the problem, which allow us to generate points uniformly distributed over the road network, necessary for Monte Carlo simulations and statistical hypothesis testing (see 4.2).

6.2 Getting a road graph in OSMNX format and (separately) edges and vertices of the road network in ShapeFile format

Appendix 1 shows a program that allows one to download a vector map (road graph) of the selected city from the OpenStreetMap repository. In Appendix 1, this is St. Petersburg. The loading process is controlled by the ox.config( ) function, where the most important option - all_oneway = False means that the bidirectional edge of the road graph is stored as one edge with the oneway = 1 attribute (and the unidirectional edge with the oneway = 0 attribute).
The graph loading itself is performed by the method ox.graph_from_place(), the following two methods ox.add_edge_speeds(G) and ox.add_edge_travel_times(G) load an estimate of the maximum speed and minimum travel time through the edge of the graph (the length of the edge is simply divided by the maximum speed) accordingly.

The remainder of the program saves the original graph and its representations as sets of edges and vertices (as a file in ShapeFile format) onto the disk.

6.3. Combining TAs and a road graph

By definition, each TA must belong to a road graph, that is, be located on one and only one of its edges. But since the road graph and TAs are obtained from different sources, they do not coincide with each other. This means that I will have to determine for each accident the edge of the road graph closest to it and then assume that it was on this edge of the road graph that the accident occurred. Simultaneously, in the case of a bidirectional edge, a difficulty arises due to the inability to determine which direction of the edge belongs to the TA. In principle, this can be found out by examining the metadata of the TA, which indicates the street and the house near which the accident occurred, but it is very difficult to algorithmize this process, and it is impossible to manually sort out several thousand accidents. Therefore, where necessary, I have to "split" the accident, assigning half of it to one direction, and half to another. The Python program that defines the edge of the road graph closest to an accident is shown in Appendix 2. The «heart» of the program is a string

```
u, v, key, dist = ox.get_nearest_edge(Gp, (y, x), return_dist=True)
```

In which, for a TA with coordinates (y, x), four values are calculated – (u, v, key) – three values defining the edge closest to the accident, where u is the identifier of the beginning of the edge (= the identifier of the vertex of the graph located at the beginning of the edge), v is the identifier of the end of the edge, and key is an integer number by which one of the edges passing through two vertices (u, v) is determined. If only one edge passes through the given vertices, the key has a single value of 0. If two edges pass through these vertices, the key has two values – 0,1, etc.

Note that the ox.get_nearest_edge() method has as parameters the projected (projected) graph Gp and coordinates (x,y), not the expected latitude and longitude (lat, lon). This is done in order to compute the distances between an accident and an edge of the road graph, since this cannot be done when the accident is defined by latitude and longitude.

In this thesis, the UTM (Universal Transverse Mercator) coordinate system (see e.g. [62]) is used to determine the distances. The advantage of the UTM system is that the distance between two points can be calculated with high accuracy

using the usual Euclidean formula d = sqrt((x1-x2)**2 + (y1-y2)**2)

Appendix 3 provides a program that evaluates the accuracy of calculating distances between two points in the UTM system. The coordinates (latitude and longitude) of the northernmost and southernmost point of St. Petersburg are taken as two points.

The distance calculated taking into account the curvature of the earth surface in the line dist = geopy.distance.geodesic(coords_1, coords_2).m is 31,879 meters.

Calculations show that the difference between the "true" distance and the distance calculated by Euclid's formula using UTM coordinates is only 4m. Since I am interested in distances < 35m, the accuracy of the distance calculation can be considered quite satisfactory

Transition to UTM coordinates for the road graph (see Appendix 2) is performed by the project_graph() method:

```
Gp = ox.project_graph(G)
```

And the projection of the accident point utilizes the ox.projection.project_geometry method

```
point_geom_proj, crs = ox.projection.project_geometry(Point(reversed(point)),{
to_crs=Gp.graph['crs']) ,
```

where point is the latitude and longitude, and crs is the UTM zone, equal to 36 for St. Petersburg.

I only need to describe the input and output of the program. The input is a list of traffic accidents in .csv format:

```
YEAR,DATE,TIME,LAT,LON,X,Y
19,31.01.2019,14:50,59.896364,29.842107,658989.2947785741,6643282.0244597355
19,31.01.2019,11:20,59.752223,30.288019,347630.30501833995,6626933.342702196
19,31.01.2019,22:35,59.884464,30.387443,353794.9774347632,6641428.438857373
19,31.01.2019,12:10,60.004061,30.436099,357032.93212231743,6654634.490192406
```

Each line, as can be seen from the file header, contains a description of one accident: year (last two digits), date, time, latitude and longitude, and x,y coordinates, the origin of which was not clear from the data provided by the traffic police GIBDD, so it was decided to switch to the UTM system.

The output of the program is a .csv file:

```
YEAR,DATE,TIME,ID,X,Y,LAT,LON,NODE1,NODE2,KEY, DIST
9, 31.01.2019, 11:20, 126439, 347630.30501808145, 6626933.341987754, 59.752223,
30.288019, 800275641, 1038407432, 0, 2.23623
19, 31.01.2019, 11:20, 126439, 347630.30501808145, 6626933.341987754, 59.752223,
30.288019, 1038407432, 800275641, 0, 2.23623
19, 31.01.2019, 22:35, 126440, 353794.97743462917, 6641428.438138614, 59.884464,
30.387443, 1710211567, 300429771, 0, 0.72741
19, 31.01.2019, 12:10, 126441, 357032.93212221784, 6654634.489469756, 60.004061,
30.436099, 25896964, 276609, 0, 1.30547
```

This is also a list of TAs, but now 4 parameters have been added to each accident: (NODE1, NODE2, KEY) - the ID of the nearest edge and DIST - the distance of the accident to the nearest edge.

6.4. Counting the number of TAs for each edge of the road graph

In the previous paragraph, I went through all the registered TAs and for each one I determined the edge of the graph to which the accident belongs. The result is a list where, for each accident, the identifiers of the edge to which the accident belongs are included.

Nothing prevents one from performing the reverse operation: iterate through all the edges of the graph and look for each edge in the file where all TAs are listed. How many times an edge of the graph will be found in this file, corresponds to how many accidents belong to this edge. The program counting the number of accidents for each edge is given in Appendix 4.

At the input of the program, the previously obtained road graph G and the list of TAs with the ids of the edges to which the TA's belong are used:

(tas_to_edges_2019_2021_twoway_proj_spb_gibdd_flt_35.csv).

It is futile to search for some lines directly in the .csv file. Therefore, in the program the .csv file is converted to a Pandas object using the pd.read_csv() method

ta_edges =

pd.read_csv('data/tas_to_edges_2019_2021_twoway_proj_spb_gibdd_flt_35.csv')

After the information about the TAs goes into the Pandas object, the calculation of accidents for each edge is reduced to a search implemented in the main function of the tas_along_edge() program:

```
def tas_along_edge(u, v,  key):
    edges_outg = ta_edges.loc[ta_edges['NODE1'] == u]
    edge_tas = edges_outg.loc[(edges_outg['NODE2'] == v) &\
                             (edges_outg['KEY'] == key)]
    tas = edge_tas.shape[0]
    oneway = G.edges[u, v, key]['oneway']
    if not oneway:
       tas = tas/2.0
    return tas
```

The search is implemented in the first three lines of the function. The remaining lines are needed to check the edge for bidirectionality. If the edge is bidirectional, I need to split the found TA's between two directions. I have already mentioned in paragraph 6.3 that it is extremely difficult, if not impossible, to assign an accident to a direction using real accident data. Therefore, in the program I simply assign half of the found accidents to each edge. This should not affect the routing results much, since bidirectional edges, as it follows from Table 5-1, make up about one third of all edges, and for long routes (the most important from the safety point of view), which include dozens of edges, there should be averaging of errors associated with attributing an accident to the wrong direction of a bidirectional edge.

It remains for us to mention only some features of the program. The collect_edges() function prepares a list of all edges of the road graph. It would seem that this list can be obtained from the edges attribute of the G: G.edges (keys = True, data = True), but it turns out that some edges in this initial list are repeated, and the collect_edges() function simply filters it.

The main function of the program tas_to_edges(G) iterates through all edges from the edge_list[] list and for each edge calculates the number of accidents, the density of accidents equal to the number of accidents divided by the length of the edge, as well as the coordinates of the beginning and end of the edge. Next, the list of edges is sorted, so that the edges with the largest number of accidents are the first in the list (descending order). The calculation result is recorded in the edges_most_populated_spb_sorted_by_ta_flt_35_debug.css file, the first lines of which are similiar to listing 6.4.1.

Listing 6.4.1. A file containing a sorted list of edges of the road graph, the number and density of accidents for each edge

```
cn,node_start,node_end,key,lat0,lon0,lat1,lon1,tas,ta_density
0,10593917,10593888,0,59.8523445,30.4745321,59.835247,30.4484255,18,0.007
1,339921466,248196221,0,59.834425,30.4477147,59.8523655,30.4752937,18,0.007
2,251397970,251398032,0,59.9159712,30.4137958,59.8928928,30.4469675,17,0.005
3,2490216682,187587141,0,59.8113625,30.3422923,59.8151242,30.361476,17,0.014
4,587052882,245828,0,60.0154892,30.4663785,59.9930789,30.4798397,16,0.006
5,9708925354,9708927324,0,59.9428586,30.3484161,59.9390119,30.3481549,15,0.035
```

The file header shows that there are 10 parameters in the line: the edge number, its identifiers (node_start,node_end,key), the coordinates of the beginning and end, the number of accidents (tas) and the density of accidents (ta_density).

6.5 Statistical tests to identify edges with a statistically significant number of TAs

I will consider as statistically significant such a value of the number of TAs for a given edge that exceeds the 95th or 98th percentile of the distribution of accidents for this edge - provided that the accidents are distributed uniformly over the road network.

To obtain points uniformly distributed over the road network, the SANET software package [47] is used, which allows us to obtain (in ShapeFile format) millions of points uniformly distributed over the road network. To obtain these points, it is enough to have the edges of the road graph in ShapeFile format (their obtaining is briefly described in Section 6.2).

The statistical tests themselves are performed by the program shown in Appendix 5. In the while True loop: first, the list of edges sorted by the number of TAs belonging to them is read line by line, from which the edge identifier (edge1, edge2, key) and the number of TA's are extracted:

```
if edge_cnt > tot_edges: break;
      line = ta_edges.readline()
      vals = line.split(',')
      if len(vals) < 10: break
      vals[-1] = vals[-1].strip() #delete last symbol '\n'
      edge1 = int(vals[hp.index('node_start')])
      edge2 = int(vals[hp.index('node_end')])
      key   = int(vals[hp.index('key')])
      tas = vals[hp.index('tas')]
```

Then, from the general list of lists of uniformly distributed points sanet_trials, the points for a particular trial are extracted and an edge (edge1, edge2, key) is searched among these points. The number of times an edge is found is the number of points that belong to it. A total of 1000 trials are conducted for a given edge and their results are recorded in the sanet_trials_file file as follows:

```
node_start,node_end,key,tas,trial_values
10593917,10593888,0,18,15#14#9#7#12#8#10#6#8#6#7#13#9#15#10#8#8#14#8#8#12#7#9#6…
```

First comes the identifier of the edge for which the trials are conducted (10593917,10593888,0), then the number of TA's for this edge, and then the results of the trials themselves (1000 values separated by the '#' sign). I only need to mention the auxiliary function prepare_sanet_points(trials, size), which divides the total list of generated uniformly distributed points into portions of size equal to the real number of accidents. In total, 1000 such portions are required for 1000 trials.

6.6 Selecting edges with a statistically significant number of TAs

I will consider the number of TAs belonging to an edge of the road graph significant if it exceeds the 95th percentile of the values obtained as a result of Monte Carlo simulation from a uniform distribution along the road network. This means that having the file obtained in Section 6.5 as input, it is necessary to read it line by line, extracting the number of accidents from each line and calculating the 95th percentile of Monte Carlo simulations. Then it is necessary to compare the number of TAs and the percentile, and if the number of TAs is higher, write the corresponding line to the output file, which will contain edges with statistically significant high numbers of TAs. The program that performs the actions described above is provided in Appendix 7.
In the main loop of the program the while True: the actions described above are performed. The percentile of the values of Monte Carlo simulations is found using the function percentile(vals, p), which has as a parameter the values of Monte Carlo simulation vals and percentile, for example, 95.

6.7 Constructing routes on the modified graph and collecting statistics of relative TA risk

Finally, I am ready to modify the road graph in such a way that edges with statistically significant high number of TAs will receive a "penalty" - the same increase in the length of each edge. But this increase may not always lead to the construction of a safer route. To make sure that the route from point A to point B is safer, let us calculate the ratio of TAs on the modified route/TAs on the original (on the unmodified graph) route (let us call this metric relative risk of TA). If this ratio is less than one, the route based on the modified graph is safer.
Naturally, in order to determine the optimal value of the penalty, it is necessary to collect statistics on some set of routes and make sure that the average value of the relative risk of an accident is statistically significantly less than one.
To specify the set of routes I will use a square grid placed on a road graph, and to obtain the relative risk statistics I will construct the routes between all the different pairs of nodes on the grid.
To create the grid, I will use the program shown in Appendix 8. This program, although quite lengthy, is quite simple in essence. It starts with the coordinates of the upper left corner of the lattice ilat, ilon and width and height of the lattice: width, height. Then the latitude and longitude of the grid start are translated into UTM system, which I briefly described in section 6.3:

```
ini_coords = utm.from_latlon(ilat, ilon)
```

Then the origin coordinates are slightly corrected (moved 3 km to the left and 4.5 km up). Finally, the grid nodes are computed in an intuitive way and saved in a file. This file is then opened to filter the nodes of the grid - leaving only those nodes that are located no further than half a grid step. This is done so that the nodes of the lattice are located on the road graph itself. Nodes that are far away from the nearest road are useless for us. The grid created for St. Petersburg is shown in Figure 5.1.

Now that the grid has been created and written to a file, I can start the main task - collecting statistics on the relative risk of road accidents. The program shown in Appendix 9 does this. Unlike other programs, this one is run from the command line. The point is that the program performs a large amount of calculations and runs for quite a long time. To speed up the calculations, several copies of the program can be executed with different parameters from the command line. If, say, the processor has 4 cores, one can easily run 3 copies of the program with different parameters.

The program itself first loads the necessary files (two copies of the road graph - G and GO, a square grid and a list of edges with a statistically significant high number of TA's), and then modifies the GO graph so that edges with a statistically significant high number of accidents receive a penalty - an increase in length by the value entered on the command line. This is done in the set_obstacles(fname, penalty, maxp) function, where fname is the name of the file with the list of edges containing a statistically significant number of TAs (see Section 6.6), penalty is the value in meters, and maxp is the number of edges that participate in the GO graph modification, i.e. the program allows using either all edges with a significant number of accidents or a subset of them.

In the main loop of the program

```
for rin,rend in product(range(grd_pnt), range(grd_pnt)):
```

all non-identical pairs of nodes in the grid are enumerated. For each pair of nodes, the nearest nodes of the graph are calculated:

```
start_node = ox.get_nearest_node(G, start)
end_node = ox.get_nearest_node(G, end) ,
```

where start, end are the coordinates of the corresponding nodes of the grid.

For GO and G, routes with common start (start_node) and common end (end_node) are constructed, and for each pair of routes the relative risk of accidents is calculated, which is equal to the ratio of the number of accidents along the modified route along the GO graph to the number of accidents along the original route along the G graph. The calculation of accidents along the route is performed by the function tas_along_route(G, route), where route is the list of graph vertices traversed along the route.

In addition, for each route, its length and the number of road graph vertices traversed along the route are calculated.

The calculation results are written to a .csv file with the following structure:

Listing 6.7.1. The structure of the output file

```
BEGIN,END,ROUTE_LENGTH,ROUTE_O_LENGTH,ROUTE_NODES,ROUTE_O_NODES,TA_DIRECT,TA_AVOID
,AFFECTED
60.035373133218066#30.08738874095845, 60.0084702205044#30.089756684912672,
4734.367, 4734.367, 10, 10, 0.0, 0.0, no
60.035373133218066#30.08738874095845, 59.873953156252696#30.10152938432629,
40672.12399999999, 40781.512999999984, 186, 186, 163.0, 155.5, yes
60.035373133218066#30.08738874095845, 59.847049243899896#30.10387060306932,
40015.45499999998, 40124.843999999975, 180, 180, 162.5, 155.0, yes
60.035373133218066#30.08738874095845, 59.82014516525566#30.10620740887939,
44565.799000000006, 44629.78200000001, 228, 223, 165.5, 160.0, yes
60.035373133218066#30.08738874095845, 60.03654801588501#30.141163834961077,
5827.031, 5827.031, 13, 13, 2.0, 2.0, no
```

Here BEGIN, END - coordinates of the beginning and end of the route, ROUTE_LENGTH - length of the unmodified route, ROUTE_O_LENGTH - length of the modified route, ROUTE_NODES and ROUTE_O_NODES - number of nodes of the graph along the original and modified route respectively, TA_DIRECT - number of accidents along the original route, TA_AVOID - number of accidents along the modified route, AFFECTED is yes if the route has changed, no if not.

6.8. Final data processing and visualization

Having several files for different penalty values (their structure is shown in Listing 6.7.1), it is possible to plot graphs of the main routing indicators (relative risk of an TA, the ratio of the lengths of the modified and the original route, as well as the ratio of the number of vertices of the road graph for the modified and the original route) depending on the length of the original, built without modification graph G, route. On such a graph, it is necessary to show not only average values, but also confidence intervals for averages. All these tasks are performed by the program shown in Appendix 10.

Main function plot_dependency(ylabel, column, label, legend): receives as the input ylabel – an inscription on the ordinate axis, column – displayed values, for example, an array of relations Length of the modified route / Length of the original route, label – a list of penalties (in the form of strings), and legend – the graph title.

To display the data, it is necessary to calculate their averages and confidence intervals of the averages. This is done by the interval_stat() function, which in turn uses the bootstrap package bootstrapped. For more information about using bootstrap ratings, see section 3.3.

The rest of the program is not too complicated and is not of much interest, since it prepares data for the plot_dependency(ylabel) function.

Examples of graphs constructed by this program are shown in Fig. 5.3, 5.4, 5.5.

Examining the constructed graphs, studying the corresponding tables, as described in Chapters 4-5, it is possible to find the optimal value of the penalty, which, paired with a list of edges with a statistically significant high number of accidents (see section 6.6), gives complete information necessary to modify the road graph. The set_obstacles(fname, penalty, max) function, briefly described in section 6.7, is suitable for modification, which should be used with the optimal penalty value for this graph.

Conclusion (main scientific results)

1. The relative risk ratio of accidents (RRR) is used as an indicator of the effectiveness of routing, equal to the average ratio of the number of accidents along the altered (taking into account obstacles on the roads) route to the number of accidents along the unaltered (not taking into account obstacles) route ([57], p. 8 par. 6).
2. Statistically significant clusters of accidents are used as obstacles in the construction of an alternative (safer) route ([48], p. 47 par. 6, [57], p. 8 par. 7).
3. Separate road segments (edges of the road graph) containing the number of accidents statistically exceeding the number of accidents obtained from the assumption of the uniformity of the distribution of accidents on the road network are used as obstacles ([58] p. 103 par. 2, p. 104 par. 6, [61] p. 30 par 1, p. 32 par. 2).
4. To design a safer route, each obstacle is subject to a penalty, i.e. a fixed number is added to the attribute of the length of the corresponding edge. By calculations, the optimal value of the penalty is determined, which ensures a minimum risk of a TA with a minimum increase in the length of the route and the number of traversable vertices of the road graph (([58], p. 103 par. 2, [61], p. 32 par. 3).

Acknowledgements

With acknowledgement, the author is grateful to Dr. A.N. Terekhov for his scientific guidance, support, and assistance in working on the Thesis. The author is grateful to his parents Mikhail and Masha, his wife Galina and children: Arielle, Oscar and Alik for their constant encouragement and support.

Glossary of terms

Bootstrap — a statistical method for estimating quantitative statistical measures by averaging estimates over several small samples of data. Samples are created by extracting observations from a large sample of data one at a time and returning them to the data sample after they have been selected.

Confidence interval — the interval of values of a random variable within which the mean of that variable falls with a given probability, for example, the 95% confidence interval.

DBSCAN — clustering algorithm that selects clusters of points of arbitrary shape and a given local density.

GIS – Geographic information system - a system for collecting, storing, analyzing and graphically visualizing spatial (geographic) data

HRA — **H**igh **R**isk **A**reas – in this Thesis, either a cluster of accidents or a road segment with an increased number of accidents

KDE – kernel density estimation

Monte-Carlo method — a broad class of computational algorithms that rely on repetitive random sampling to obtain numerical results. In this Thesis, sets of points evenly distributed over the road network are generated to assess the statistical significance of the values.

Relative Risk Ratio — the ratio of the number of accidents along the modified (presumably safer) route to the number of accidents of the original route having the same starting and ending points.

RRR — **R**elative **R**isk **R**atio of accidents

TA – **T**raffic **A**ccident

References

1. F. A. Novikov, discrete mathematics for programmers // Piter, 2002 (in Russian).
2. E. W. Dijkstra, A note on two problems in connexion with graphs // Numerische Mathematik volume 1, pages 269–271 (1959), doi:10.1007/BF01386390
3. Rodriguez MZ, Comin CH, Casanova D, Bruno OM, Amancio DR, Costa LdF, et al. (2019) Clustering algorithms: A comparative approach. PLoS ONE 14(1): doi: 10.1371/journal.pone.0210236
4. Ester M., Kriegel H-P., J. S, Sander J., Xu X., A density-based algorithm for discovering clusters in large spatial databases with noise, AAAIPress, 1996, pp. 226–231.
5. Yiqun Xie, Shashi Shekhar, 2019. Significant DBSCAN towards Statistically Robust Clustering. In '19: Proceedings of the 16th International Symposium on Spatial and Temporal Databases, August 2019, Pages 31–40. DOI: 10.1145/3340964.3340968
6. Boeing, G. 2017. "OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks." Computers, Environment and Urban Systems 65, 126-139. doi:10.1016/j.compenvurbsys.2017.05.004
7. Edward L. Platt, Network Science with Python and NetworkX Quick Start Guide: Explore and visualize network data effectively // Packt Publishing, 2019, 190pp.
8. R. Bellman: On a Routing Problem // Quarterly of Applied Mathematics. 1958. Vol 16, No. 1. C. 87-90, 1958.
9. L. R. Ford, Jr., D. R. Fulkerson. Flows in Networks, Princeton University Press, 1962.
10. Highway routing of hazardous materials, NHI Course 38064, National Highway Institute.
11. Gustavo Alfredo Bula. Vehicle Routing for Hazardous Material Transportation. Operations Research [cs.RO]. Université de Technologie de Troyes; Universidad nacional de Colombia, 2018. English. NNT : 2018TROY0014 . tel-03146101

12. Safe route-finding: A review of literature and future directions, Soheil Sohrabi, Yanmo Weng Subasish Das, Stephanie German Paal, Accident Analysis & Prevention, ISSN: 0001-4575, Vol: 177, Page: 106816

13. Urban navigation beyond shortest route: The case of safe paths, Esther Galbrun, Konstantinos Pelechrinis, Evimaria Terzia, Information Systems, Volume 57, April 2016, Pages 160-171

14. https://en.wikipedia.org/wiki/Kernel_density_estimation

15. Safe and secure vehicle routing: a survey on minimization of risk exposure, Georg E. A. Fröhlicha, Margaretha Gansterera, and Karl F. Doernera, Intl. Trans. in Op. Res. 0 (2022) 1–35, DOI: 10.1111/itor.13130

16. Driver Route Planning Method Based on Accident Risk Cost Prediction, Xiaoleng Liao, Tong Zhou, Xu Wang, Rongjian Dai, Xuehui Chen and Xiangmin Zhu, Journal of Advanced Transportation Volume 2022, Article ID 5023052, DOI:https://doi.org/10.1155/2022/5023052

17. A Route Navigation System for Reducing Risk of Traffic Accidents, Ryo Takeno, Yosuke Seki, Masahiko Sano, Kenji Matsuura, Kenji Ohira, Tetsushi Ueta, 2016 IEEE 5th Global Conference on Consumer Electronics.

18. Abdelhamid, S., Elsayed, S. A., AbuAli, N. & Hassanein, H. S. Driver-centric route guidance. 2016 2016. IEEE, 1-6.

19. Bao, S., Nitta, T., Yanagisawa, M., Togawa, N., 2017. A Safe and Comprehensive Route Finding Algorithm for Pedestrians Based on Lighting and Landmark Conditions. In: IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, pp. 2439–2450.

20. Chandra, S., 2014. Safety-based path finding in urban areas for older drivers and bicyclists. Transp. Res. Part C: Emerg. Technol. 48, 143–157.

21. Hart, P., Nilsson, N. and Raphael, B. (1968) A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions Systems Science and Cybernetics, 4, 100-107. http://dx.doi.org/10.1109/TSSC.1968.300136

22. Hayes, S., Wang, S. & Djahel, S. Personalized Road Networks Routing with Road Safety Consideration: A Case Study in Manchester. 2020 2020. IEEE, 1-6.

23. Hoseinzadeh, N., Arvin, R., Khattak, A.J., Han, L.D., 2020. Integrating safety and mobility for pathfinding using big data generated by connected vehicles. J. Intell. Transp. Syst. 24, 404–420.

24. Jiang, S., Jafari, M., Kharbeche, M., Jalayer, M., Al-Khalifa, K.N., 2020. Safe route mapping of roadways using multiple sourced data. IEEE Trans. Intell. Transp. Syst.

25. Jiang, S., Zhang, Y., Liu, R., Jafari, M., Kharbeche, M., 2022. Data-Driven Optimization for Dynamic Shortest Path Problem Considering Traffic Safety. IEEE Trans. Intell. Transp. Syst.

26. John Krumm, Eric Horvitz, Risk-aware planning: methods and case study on safe driving routes, Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence February 2017 Pages 4708–4714.

27. Liu, Q., Kumar, S. & Mago, V. SafeRNet: Safe transportation routing in the era of Internet of vehicles and mobile crowd sensing. 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), 8-11 Jan. 2017 2017. 299-304.

28. Mata, F., Torres-Ruiz, M., Guzm´an, G., Quintero, R., Zagal-Flores, R., Moreno-Ibarra, M.,Loza, E., 2016. A Mobile Information System Based on Crowd-Sensed and Official Crime Data for Finding Safe Routes: A Case Study of Mexico City. Mobile Inf. Syst. 2016, 1–11.

29. Ouyang, W., Yu, C.-W., Yu, K.-M., Lin, K.-J., Chang, H.-T., 2014. Safe path planning strategy for bike net. Wireless Pers. Commun. 78, 1995–2007

30. Scott Hartshorn, Tell Me The Odds: A 15 Page Introduction To Bayes Theorem, 2017 (Kindle Edition)

31. https://ru.wikipedia.org/wiki/Теорема_Байеса (in Russian)

32. https://ru.wikipedia.org/wiki/Динамическое_программирование (in Russian)

33. Jamal Jokar Arsanjani, Alexander Zipf, Peter Mooney, Marco Helbich (eds.),OpenStreetMap in GIScience: Experiences, Research, and Applications // Springer International Publishing, 2015, 324pp.

34. https://en.wikipedia.org/wiki/Binary_classification

35. Heckerman, D. Geiger, and D. M. Chickering, Learning bayesian networks: The combination of knowledge and statistical data, Machine learning, vol. 20, no. 3, pp. 197–243, 1995.

36. Chainey S, Tompson L, Uhlig S., 2008. "The utility of hotspot mapping for predicting spatial patterns of crime". Secur J 21(1):4–28, DOI: 10.1057/palgrave.sj.8350066

37. Chainey, S., Ratcliffe, J., 2005. GIS and Crime Mapping. John Wiley and Sons, UK. DOI: 10.1002/9781118685181

38. Gramacki Artur, Nonparametric Kernel Density Estimation and Its Computational Aspects, Springer International Publishing, 2018. DOI: 10.1007/978-3-319-71688-6

39.Moran, P. A. P., 1950. Notes on Continuous Stochastic Phenomena. Biometrika, Vol. 37, No. 1/2 (Jun., 1950), pp. 17-23. DOI: 10.2307/2332142.

40.Okabe, A., Sugihara, K., 2012. Spatial Analysis along Networks: Statistical and Computational Methods; John Wiley & Sons: Hoboken, NJ, USA, DOI:10.1002/9781119967101

41.Songchitruksa, P., Zeng X., 2010. Getis–Ord Spatial Statistics to Identify Hot Spots by Using Incident Management Data. Transportation Research Record: Journal of the Transportation Research Board. 2165. pp 42-51. DOI: 10.3141/2165-05.

42.Yingjie, L., Liwei,Zh., Junping,Y., Pengtao,W., Ningke,H., Wei,Ch., and Bojie,F.2017. Mapping the hotspots and coldspots of ecosystem services in conservation priority setting. Journal of Geographical Sciences,27(6):681-696. DOI: 10.1007/s11442-017-1400-x

43.Massgis data-massachusetts department transportation massdot roads // URL: docs.digital.mass.gov.

44.QGIS // www.qgis.org, URL:https://www.qgis.org/en/site/

45.Open jump // www.openjump.org. URL:http://www.openjump.org/

46.MassDOT Crash Open Data Portal // Mass.gov. URL:https://massdot-impact-crashes-vhb.opendata.arcgis.com/search

47.SANET //sanet.csis.u-tokyo.ac.jp. URL:http://sanet.csis.u-tokyo.ac.jp/

48. A. M. Gershteyn and A. N. Terekhov “Hotspots of Traffic Accidents that cause injuries or death in Massachusetts from 2013 to 2018 “Computer tools in education”,no.1, pp.45–57, 2021(in Russian); DOI: http://dx.doi.org/10.32603/2071-2340-2021-1-46-58
49. Sahnoon, Iyad & Ahmed, Mohamed & Alghafli, Abdulla. (2018). Integrating Traffic Safety in Vehicle Routing Solution. 251-263. DOI:10.1007/978-3-319-60441-1_25.
50. Is OSM Good Enough for Vehicle Routing? A Study Comparing Street Networks in Vienna Anita Graser, Markus Straub and Melitta Dragaschnig, 11th International Symposium on Location-Based ServicesAt: Vienna, AustriaVolume: Progress in Location-Based Services 2014
51. Boeing, G. 2017. "OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks." Computers, Environment and Urban Systems 65, 126-139. doi:10.1016/j.compenvurbsys.2017.05.004
52. Peter J. Huber, Elvezio M. Ronchetti, Robust Statistics Concomitant scale estimates, pg 172
53. Art B. Owen (2006), A robust hybrid of lasso and ridge regression. https://statweb.stanford.edu/~owen/reports/hhu.pdf
54. Koenker, Roger and Kevin F. Hallock. “Quantile Regression”. Journal of Economic Perspectives, Volume 15, Number 4, Fall 2001, Pages 143–156
55. Alan Agresti, Christine Franklin, Bernhard Klingenberg Statistics: The Art and Science of Learning from Data, Fourth Edition, Pearson, 2018
56. D’Agostino, R. and Pearson, E. S. (1973), “Tests for departure from normality”, Biometrika, 60, 613-622
57. A. M. Gershteyn and A. N. Terekhov, “Routing Algorithm for Vehicles that Avoids Severe Traffic Accident Hotspots on the Road Network (Using the City of Springfield, Massachusetts as a Case Study), ”Computer tools in education, no.2, pp.5–18, 2022(in Russian) URL: http://cte.eltech.ru/ojs/index.php/kio/article/view/1729
58. Gershteyn A. M., Terekhov A. N. “A Simple Way to Improve Road Safety by Avoiding Dangerous Sections of the Route” *Programmnaya Ingeneria*, 2023, vol.

14, no. 3, pp. 103—109. DOI: https://dx.doi.org/10.17587/prin.14.103-109 (in Russian).

59. https://www.qgis.org/ru/site/

60. http://stat.gibdd.ru/

61. A. M. Gershteyn and A. N. Terekhov, “Bypassing Dangerous Sections of the Route as a Way to Improve Traffic Safety (Using St. Petersburg as an Example),” Computer tools in education, no. 1, pp. 30–39, 2023 (in Russian); URL: http://cte.eltech.ru/ojs/index.php/kio/article/view/1755

62. https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system

Appendix 1. Loading a road graph in native OSMNX format and in ShapeFile format (for GIS application)

```
import osmnx as ox
import pickle

city = 'Petersburg'
loc = city + ',RU'
ox.config(use_cache=True, log_console=True, all_oneway=False)

G = ox.graph_from_place(loc, network_type='drive')
G = ox.add_edge_speeds(G)
G = ox.add_edge_travel_times(G)

ox.io.save_graph_shapefile(G, 'data/' + 'spb_one_way_false', encoding='utf-8')

with open(city + '.p', 'wb') as f:
    pickle.dump(G, f)
```

Appendix 2. Finding the edges of the road graph closest to an TA

```
#edges_for_all_tas_spb_projected_gibdd.py
import pandas as pd
import networkx as nx
import osmnx as ox
from shapely.geometry import LineString,Point
import geopandas as gpd
from ipyleaflet import *
import pickle
import matplotlib.pyplot as plt
import datetime


max_dist = 35
save_oneway = False

with open("data/Petersburg.p", 'rb') as f:  # notice the r instead of w
    G = pickle.load(f)
Gp = ox.project_graph(G)


if save_oneway:
   f = open('zzz_tas_to_edges_2019_2021_oneway_proj_spb_gibdd.csv','w')
else:
   f = open('zzz_tas_to_edges_2019_2021_twoway_proj_spb_gibdd.csv','w')
f.write('YEAR,DATE,TIME,ID,X,Y,LAT,LON,NODE1,NODE2,KEY,DIST\n')

# input TAs
ta_data = 'data/tas_raw_spb.csv'
fta = open(ta_data,'r')
ta_header = 'YEAR,DATE,TIME,LAT,LON,X,Y'
ta_items = ta_header.split(',')
fta.readline() # Skip header

def prepareRow(year, date, time, id, x, y, lat, lon, node1, node2, key, d):
    row = str(year) + ', '
    row += date + ', '
    row += time + ', '
    row += str(id) + ', '
    row += str(x) + ', '
    row += str(y) + ', '
    row += str(lat) + ', '
    row += str(lon) + ', '
    row += str(node1) + ', '
    row += str(node2) + ', '
    row += str(key) + ', '
    #row += str(d) + '\n'
    row += "{0:.5f}".format(d) + '\n'
    return row


#def write_csv_row(year, edge, ta_id, x, y, lat, lon, oneway):
def write_csv_row(year, date, time, edge, ta_id, x, y, lat, lon, oneway):
    node1 = edge[0]
    node2 = edge[1]
    key = edge[2]
    d = edge[3]
    row = prepareRow(year, date, time, ta_id, x, y, lat, lon, node1, node2, key,
d)
```

Continuation of Appendix 2

```
    f.write(row)
    if not oneway:
       #if the edge is two-way, add adge with opposite direction
       row = prepareRow(year, date, time, ta_id, x, y, lat, lon, node2, node1,
key, d)
       print('two way:', node1,node2,key)
       f.write(row)

def extract_year(date):
    return date.split('-')[0]


count = 0
ta_id = 126438
print(datetime.datetime.now().time())
while True:
    ta = fta.readline().split(',')
    if len(ta) < 7: break
    year = ta[ta_items.index('YEAR')]
    date = ta[ta_items.index('DATE')]
    time = ta[ta_items.index('TIME')]
    lat = ta[ta_items.index('LAT')]
    lon = ta[ta_items.index('LON')]
    point = (float(lat), float(lon))
    point_geom_proj, crs = ox.projection.project_geometry(Point(reversed(point)),\
        to_crs=Gp.graph['crs'])
    x, y = point_geom_proj.x, point_geom_proj.y
    u, v, key, dist = ox.get_nearest_edge(Gp, (y, x), return_dist=True)
    edge = (u, v, key, dist)
    if dist < max_dist:
      if save_oneway:
         write_csv_row(year, date, time, edge, ta_id, x, y, lat, lon, True)
      else:
         oneway = G.edges[u,v,key]['oneway']
         write_csv_row(year, date, time, edge, ta_id, x, y, lat, lon, oneway)
      ta_id += 1
      count += 1
      if count % 100 == 0:
         print(count)
f.close()
```

Appendix 3. Estimation of the accuracy of calculating distances in UTM coordinates using the Euclid formula

```
#_test_utm_eucl.py
import osmnx as ox
import pickle
import math
from shapely.geometry import LineString,Point
from pyproj import Proj

coords_1 = (float(60.098310), float(30.305087))
coords_2 = (float(59.814286), float(30.374339))

with open("data/Petersburg.p", 'rb') as f:  # notice the r instead of w
    G = pickle.load(f)
Gp = ox.project_graph(G)


dist = geopy.distance.geodesic(coords_1, coords_2).m

point1 = (coords_1[0], coords_1[1])
point2 = (coords_2[0], coords_2[1])

point_geom_proj, crs = ox.projection.project_geometry(Point(reversed(point1)),\
    to_crs=Gp.graph['crs'])
x1, y1 = point_geom_proj.x, point_geom_proj.y

point_geom_proj, crs = ox.projection.project_geometry(Point(reversed(point2)),\
    to_crs=Gp.graph['crs'])
x2, y2 = point_geom_proj.x, point_geom_proj.y

eucl_dist = math.sqrt((abs(x1-x2)**2) + (abs(y1-y2))**2)
print(eucl_dist - dist)
```

Appendix 4. Counting the number of accidents for each edge of the road graph

```
#edges_most_populated.py
import pandas as pd
import networkx as nx
import osmnx as ox
from shapely.geometry import LineString
import geopandas as gpd
from ipyleaflet import *
import pickle
import matplotlib.pyplot as plt
import numpy as np
import datetime
from operator import itemgetter


with open("data/Petersburg.p", 'rb') as f:  # notice the r instead of w
    G = pickle.load(f)
ta_edges =
pd.read_csv('data/tas_to_edges_2019_2021_twoway_proj_spb_gibdd_flt_35.csv')
avoidf = open('edges_most_populated_spb_sorted_by_ta_flt_2_debug.csv', 'w')
avoidf.writelines('cn,node_start,node_end,key,lat0,lon0,lat1,lon1,tas,\
       ta_density\n')

elfn = "edge_list.csv"
elf = open(elfn, 'w')
elf.writelines('NODE1,NODE2,KEY\n')

def tas_along_edge(u, v,  key):
    edges_outg = ta_edges.loc[ta_edges['NODE1'] == u]
    edge_tas = edges_outg.loc[(edges_outg['NODE2'] == v) &\
                             (edges_outg['KEY'] == key)]

    tas = edge_tas.shape[0]
    oneway = G.edges[u, v, key]['oneway']
    if not oneway:
       tas = tas/2.0
    return tas

def equal_nodes(edge1, edge2):
    ret = False
    if (edge1[0] == edge2[0]) and (edge1[1] == edge2[1]) and\
       (edge1[2] == edge2[2]):
       ret = True
    return ret

edge_list = []
def collect_edges(G):
    edges = G.edges(keys = True, data = True)
    for edge in edges:
        current_edge = [edge[0],edge[1],edge[2]]
        llen = len(edge_list)
        if llen != 0:
           pe = edge_list[llen-1]
           prev_edge = [pe[0],pe[1],pe[2]]
        if llen == 0:
           edge_list.append(edge)
        elif not equal_nodes(prev_edge, current_edge):
             edge_list.append(edge)
```

Continuation of Appendix 4

```
def save_paths_to_avoid(edges,n):
    for i in range(n):
        density = f"{edges[i][8]:.3f}"
        line = str(i) + ',' + str(edges[i][0]) + ',' + str(edges[i][1]) +\
               ','+ str(edges[i][2]) + ','+ str(edges[i][3]) +\
               ','+ str(edges[i][4]) +',' + str(edges[i][5]) +\
               ',' + str(edges[i][6])+ ',' + str(edges[i][7])+\
               ',' + str(density) + '\n'
        avoidf.writelines(line)
    avoidf.close()

ds = []
ts_edges = []
def tas_to_edges(G):
    for edge in edge_list:
        node0 = edge[0]
        node1 = edge[1]
        n_edges =  G.number_of_edges(node0, node1)
        for key in range(n_edges):
            tas = tas_along_edge(node0, node1, key)
            l = G.edges[node0, node1, key]['length']
            d = tas/l
            lon0 = G.nodes[node0]['x']
            lat0 = G.nodes[node0]['y']
            lon1 = G.nodes[node1]['x']
            lat1 = G.nodes[node1]['y']
            ts_edge = [node0, node1, key, lat0, lon0, lat1, lon1, tas, d]
            ts_edges.append(ts_edge)
collect_edges(G)
print('Total edges:', len(edge_list) )
for edge in edge_list:
    row = str(edge[0]) + ',' + str(edge[1]) + ',' + str(edge[2]) + '\n'
    elf.writelines(row)
elf.close()
tas_to_edges(G)
ts_sorted = sorted(ts_edges, key=itemgetter(7), reverse=True)
save_paths_to_avoid(ts_sorted, len(ts_edges))
```

Appendix 5. Statistical tests on a road graph with uniformly distributed points

```
"""
Script:
collect_trials.py - performs Monte-Carlo trials on most TA-populated edges
Input:
Road network (.p file for example "Peterburg.p")
.csv list of Sanet-generated points and respective edges points belong to.
.csv list of the TA-populated edges sorted by TA.
Output:
.csv - TA-populated edges along with results of Monte-Carlo trials.
"""

import pandas as pd
import networkx as nx
import osmnx as ox
from shapely.geometry import LineString
import geopandas as gpd
#import geopy.distance
#import json
from ipyleaflet import *
import pickle
import matplotlib.pyplot as plt
import numpy as np
#import sys
import datetime



tsize = 13503

tot_trials = 1000
tot_edges = 2000

with open("data/Petersburg.p", 'rb') as f:  # notice the r instead of w
    G = pickle.load(f)

#points uniformly distributed over edges
points_sanet_edges = pd.read_csv('data/sanet_points_to_edges_flt_35_spb.csv')
print(points_sanet_edges)

"""
def currentTime():
    e = datetime.datetime.now()
    time = "%s:%s:%s" % (e.hour, e.minute, e.second)
    return time
"""

sanet_trials =[]
def prepare_sanet_points(trials, size):
    #df.iloc[2:5]
    for i in range(trials):
        init = i * size
        end = (i+1) * size - 1
        trial = points_sanet_edges.iloc[init:end]
        sanet_trials.append(trial)
```

Continuation of Appendix 5

```
prepare_sanet_points(tot_trials, tsize)
print(len(sanet_trials))

ta_edges = open('data/edges_most_populated_spb_sorted_by_ta_flt_35_debug.csv' ,
'r')
mhdr = ta_edges.readline()
hp = mhdr.split(',')
hp[-1] = hp[-1].strip()
print(len(hp))
sanet_trials_file = open('sanet_trials_flt_35_half_two_way.csv', 'w')
hdr = 'node_start,node_end,key,tas,trial_values\n'
sanet_trials_file.writelines(hdr)

edge_cnt = 0
while True:
      if edge_cnt > tot_edges: break;
      line = ta_edges.readline()
      vals = line.split(',')
      if len(vals) < 10: break
      vals[-1] = vals[-1].strip() #delete last symbol ‘\n’
      edge1 = int(vals[hp.index('node_start')])
      edge2 = int(vals[hp.index('node_end')])
      key   = int(vals[hp.index('key')])
      tas = vals[hp.index('tas')]
      #oneway = G.edges[edge1,edge2,key]['oneway']

      row = str(edge1)+','+str(edge2)+','+str(key)+','+tas+','
      for trial in sanet_trials:
          pi = trial.loc[trial['EDGEI'] == edge1]
          pe = pi.loc[pi['EDGEE'] == edge2]
          pk = pe.loc[pe['EDGEK'] == key]
          pnts = pk.shape[0]
          #if not oneway: pnts = pnts/2.0
          row = row + str(pnts) + '#'
      row  += '\n'
      sanet_trials_file.writelines(row)
      print('edge number:', edge_cnt)
      edge_cnt += 1
sanet_trials_file.close()
```

Appendix 6. Representation of uniformly distributed points obtained using the SANET program in the form of .csv file

```
from configparser import InterpolationError
import shapefile
import networkx as nx
import osmnx as ox
from datetime import datetime
import numpy as np
import pandas as pd
import pickle


with open("data/Petersburg.p", 'rb') as f:  # notice the r instead of w
    G = pickle.load(f)

#G_proj = ox.projection.project_graph(G)

def currentTime():
    now = datetime.now()
    current_time = now.strftime("%H:%M:%S")
    return now

fname = 'data/sanet/SANETRandomPoint20m.shp'
#fname = '../prepare_data/data/moscow_random30m.shp'

out_name = 'sanet_points_to_edges_flt_35_spb.csv'
outf = open(out_name,'w')
outf.writelines('N,LAT,LON,EDGEI,EDGEE,EDGEK\n')



print(currentTime(), ' Start reading sanet points')
sf = shapefile.Reader( fname )
mkrecords = sf.records()
print(currentTime(), ' End reading sanet points')


#trials = 1000
trials = 1000
tsize = 13503
#tsize = 10
tr = 0
cnt = 0
X = []
Y = []
```

Continuation of Appendix 6

```
for record in mkrecords:
    lat = record[2]
    lon = record[1]
    X.append(record[1])
    Y.append(record[2])
    cnt += 1
    if cnt >= tsize:
       edges =  ox.distance.get_nearest_edges(G,X,Y,method='balltree')
       for i, edge in enumerate(edges):
           out_str = str(tr) + ',' + str(Y[i]) + ',' + str(X[i]) + ',' +
str(edge[0]) + ',' + str(edge[1]) + ',' + str(edge[2]) + '\n'
           outf.writelines(out_str)
       X = []
       Y = []
       print(currentTime(), ' Trial: ', tr)
       tr += 1
       cnt = 0


       if tr >= trials: break
outf.close()
print(currentTime(), 'End')
```

Appendix 7. Finding statistically significant values of road traffic accidents

```
"""
Script:
select_significant_edges.py - select statistically significant edges
Input:
sanet_trials_flt_35_half_two_way.csv - sanet points with results of trials
Output:
sanet_trials_flt_35_signifcant_<p>.csv - statistically significant edges used as
input for routing script
routing_finite_penalty.py.

"""
import numpy as np


p = 95

def percentile(vals,p):
    vals[-1] = vals[-1].strip()
    tot_vals = vals[hdrps.index('trial_values')][:-1]
    str_vals = tot_vals.split('#')
    str_vals[-1] = str_vals[-1].strip()
    ivals = [int(str_val) for str_val in str_vals]
    #vls = np.array(ivals)
    percentile = np.percentile(ivals,p)
    return percentile


trialsf = open('data/sanet_trials_flt_35_half_two_way.csv','r')
outpf = open('sanet_trials_flt_35_signifcant_' + str(p)+ '.csv','w')
hdr = trialsf.readline()
outpf.writelines(hdr)
hdrps = hdr.split(',')
hdrps[-1] = hdrps[-1].strip()
while True:
    row = trialsf.readline()
    rowps = row.split(',')
    if len(rowps) < 5: break
    tas = rowps[hdrps.index('tas')]
    perc = percentile(rowps,p)
    if float(tas) > perc:
       outpf.writelines(row)
trialsf.close()
outpf.close()
```

Appendix 8. Creating a square grid superimposed on a road graph

```
import pickle
import osmnx as ox
import utm
import sys

with open("data/Petersburg.p", 'rb') as f:  # notice the r instead of w
     G = pickle.load(f)


# upper left 55,915650900000003, 37,366904099999999
ilat = 60.05
ilon = 30.14
width = 50000
height = 50000
step = 3000

csv_header = 'num,X,Y,LAT,LON\n'
gname = 'spb_grid_' + str(step)+ '.csv'
gf = open(gname,'w')
gf.write(csv_header)

# X,Y
ini_coords = utm.from_latlon(ilat, ilon)
print (ini_coords)
x0 = ini_coords[0] - 3000
y0 = ini_coords[1] + 4500

irange = int(width/step)
jrange = int(height/step)

k = 0
for i in range(irange):
    x = x0 + i*step
    for j in range(jrange):
        y = y0 - step*j
        #latlon = utm.to_latlon(x,y, 37, 'U')
        latlon = utm.to_latlon(x,y, 36, 'U' )
        row = str(k) + ',' + str(x) + ',' + str(y) + ',' + str(latlon[0]) + ','\
              + str(latlon[1]) + '\n'
        gf.write(row)
        k += 1
gf.close()

#Filter grid poionts
fname = 'grid_filtered_' + str(step) + '.csv'
ff = open(fname,'w')
fg = open(gname,'r')
header = fg.readline()
ff.write(header)
max_dist  = step/2
```

Continuation of Appendix 8

```
while True:
    row = fg.readline()
    rvals = row.split(',')
    if len(rvals) < 5: break
    lat = float(rvals[3])
    lon = float(rvals[4])
    nodeDist = ox.get_nearest_node(G, (lat,lon), method='haversine',\
               return_dist=True)
    print(nodeDist)

    if nodeDist[1] < max_dist:
       ff.write(row)

ff.close()
```

Appendix 9. Program for collecting relative risk statistics

```
import numpy as np
import pandas as pd
import networkx as nx
import osmnx as ox
from shapely.geometry import LineString
import geopandas as gpd
import geopy.distance
from ipyleaflet import *
import pickle
import matplotlib.pyplot as plt
import datetime
from itertools import product
import sys


# To run with command line

if len(sys.argv) < 2:
   print ('Usage: routing_finite_penalty.py <p>, <max_path_to_avoid> <obstacles>
<penalty>')
#example: routing_finite_penalty.py 95 800
data/sanet_trials_flt_35_signifcant_95.csv 200
   sys.exit()

p = int(sys.argv[1])
max_paths_to_avoid = int(sys.argv[2])
fobst = sys.argv[3]
penalty = int(sys.argv[4])
print('penalty=', penalty)



"""
p = 95
max_paths_to_avoid = 800
fobst = 'data/sanet_trials_flt_35_signifcant_95.csv'
penalty = 1000
"""


debug = True
#fobst = 'data/paths_to_avoid_moscow_sorted_by_ta.csv'
#node_start,node_end,key,tas,trial_values
#fobst = 'data/paths_to_avoid_moscow_sorted_by_ta_flt_35_half_two_way.csv'
grid_step = 3000
#max_paths_to_avoid = 500
#penalty = 500
obst_edges = []
```

Continuation of Appendix 9

```
def read_obst_edges(fname, maxp):
    cnt = 0
    fl = open(fname, 'r')
    hdr = fl.readline().split(',')
    hdr[-1] = hdr[-1].strip()

    while(True):
         ln = fl.readline()
         if ln[0] == '#': continue
         print(ln)
         if(ln == ''): break
         lnp = ln.split(',')
         if len(lnp) < 5: break
         start_node = int(lnp[hdr.index('node_start')])
         end_node = int(lnp[hdr.index('node_end')])
         key = int(lnp[hdr.index('key')])
         edge = [start_node, end_node, key]
         obst_edges.append(edge)
         cnt += 1
         if cnt == maxp: break
    fl.close()

def split_row(header):
    hdrps = header.split(',')
    hdrps[-1] = hdrps[-1].strip()
    return hdrps,len(hdrps)


def set_obstacles(fname, penalty, maxp):
    global GO # with obstacles
    global G  # original graph
    fl = open(fname, 'r')
    hd = fl.readline()
    hdr,hl = split_row(hd)
    cnt = 0
    while(True):
         ln = fl.readline()
         if ln[0] == '#': continue
         #print(ln)
         lnp, lcs = split_row(ln)
         if len(lnp) < hl: break
         start_node = int(lnp[hdr.index('node_start')])
         end_node = int(lnp[hdr.index('node_end')])
         key = int(lnp[hdr.index('key')])
         origEdgeLength = G.edges[ start_node, end_node, key]['length']
         GO.edges[ start_node, end_node, key]['length'] = origEdgeLength + penalty
         cnt += 1
         if cnt == maxp: break
    fl.close()

with open("data/Petersburg.p", 'rb') as f:  # notice the r instead of w
    G = pickle.load(f)

with open("data/Petersburg.p", 'rb') as f:  # notice the r instead of w
    GO = pickle.load(f)

# Read grid points
gridfn = 'data/grid_filtered_' + str(grid_step) + '.csv'
df = pd.read_csv(gridfn)
print(df.shape)
```

Continuation of Appendix 9

```
# Read TAs with the edges TAs belong to
ta_edges =
pd.read_csv('data/tas_to_edges_2019_2021_twoway_proj_spb_gibdd_flt_35.csv')
print(ta_edges)


def tas_along_edge(u, v,  key, oneway, year=0):
    if year == 0:
       edges_outg = ta_edges.loc[ta_edges['NODE1'] == u]
    else:
       edges_outg = ta_edges.loc[ta_edges['NODE1'] == u & ta_edges['YEAR'] ==\
                    year]
    edge_tas = edges_outg.loc[(edges_outg['NODE2'] == v) & (edges_outg['KEY'] ==\
               key)]
    tot_tas = edge_tas.shape[0]
    if not oneway:
       tot_tas = 0.5 * edge_tas.shape[0]
    return tot_tas


def tas_along_route(G, route, year=0):
    total_tas = 0
    rl = len(route)
    node0 = route[0]
    for i in range(1,rl):
        node1 = route[i]
        n_edges =  G.number_of_edges(node0, node1)
        if n_edges == 1:
           edge = 0
        else:
           #Find edge with min length in case there are several edges between two
           # nodes
           edge = 0
           l_dest = G.edges[node0, node1, 0]['length']
           for k in range(1, n_edges):
               l_attr = G.edges[node0, node1, k]['length']
               if l_attr  < l_dest:
                  l_dest = l_attr
                  edge = k
        oneway =  G.edges[node0, node1, edge]['oneway']
        if year == 0:
           total_tas += tas_along_edge(node0, node1, edge, oneway)
        else:
           total_tas += tas_along_edge(node0, node1, edge, oneway, year)
        node0 = node1    # next edge begins where previous one ends
    return total_tas

# Compare two routes
def routesAreEqual(route1,route2):
    result = True
    if len(route1) != len(route2): return False
    for i in range(len(route1)):
        if route1[i] != route2[i]:
           result = False
           break
    return result
```

Continuation of Appendix 9

```
# Check if an edge is in list of "prohibited" edges
def edgeWithPenalty(node1, node2, key):
    ne = obst_edges_n.shape[0]
    for i in range(ne):
        v = obst_edges_n[i]
        trio = np.array([node1,node2,key])
        if (v==trio).all():
           return True
    return False


def inspect_route(G,route):
   extra_length = 0
   penalties = 0
   for i in range(len(route)-1):
        u = int(route[i])
        v = int(route[i + 1])
        n_edges =  G.number_of_edges(u, v)
        ek = 0
        if n_edges > 1:
           # Find edge with min length
           minl =  G.edges[u,v,0]['length']
           for i in range(1, n_edges):
              li = G.edges[u,v,i]['length']
              if li < minl:
                 minl = li
                 ek = i
        if edgeWithPenalty(u,v,ek):
              extra_length += penalty
              penalties += 1
   return extra_length, penalties

read_obst_edges(fobst,max_paths_to_avoid)
obst_edges_n = np.array(obst_edges)

set_obstacles(fobst, penalty, max_paths_to_avoid)
grd_pnt = df.shape[0]

f_tar_name = 'routes_' + str(grid_step) + '_pen_' + str(penalty) +\
             '_edges_sorted_by_ta_sign' + str(p) + '_' + str(max_paths_to_avoid)+\

             '.csv'
f_tar = open(f_tar_name, 'w')
f_tar.write("BEGIN,END,ROUTE_LENGTH,ROUTE_O_LENGTH,ROUTE_NODES,ROUTE_O_NODES,\
             TA_DIRECT,TA_AVOID,AFFECTED\n")

print(datetime.datetime.now().time())

cin = 0
cend = 900000000
penalties_G = []
penalties_GO = []
penalties_G = []
```

Continuation of Appendix 9

```
for rin,rend in product(range(grd_pnt), range(grd_pnt)):
    ind = rin * grd_pnt + rend
    print(ind, ' of ', grd_pnt * grd_pnt )
    if ind < cin: continue
    if ind >= cend: break
    if rend == rin: continue
    row_from = df.loc[rin,:]
    row_to = df.loc[rend,:]
    start = (row_from[3],row_from[4])
    end = (row_to[3],row_to[4])
    dist = geopy.distance.distance(start, end).km
    start_node = ox.get_nearest_node(G, start)
    end_node = ox.get_nearest_node(G, end)
    try:
        route = nx.shortest_path(G, start_node, end_node, weight='length')
        penalties_G.append(inspect_route(G, route)[1])
        route_length = nx.shortest_path_length(G, start_node, end_node,
weight='length')
        route_nodes = len(route)
    except:
        continue
    try:
        route_o = nx.shortest_path(GO, start_node, end_node, weight='length')
        total_penalty = inspect_route(GO, route_o)[0]
        penalties_GO.append(inspect_route(GO, route_o)[1])
        if total_penalty != 0:
           print('penalty=', total_penalty)
        route_o_length = nx.shortest_path_length(GO, start_node, end_node,
weight='length') - total_penalty
        route_o_nodes = len(route_o)
    except:
        continue

    tas_direct = tas_along_route(G, route)
    equal_routes =  routesAreEqual(route, route_o)
    if not equal_routes:
        affected = ', yes'
        tas_avoid = tas_along_route(G, route_o)
    else:
        affected = ', no'
        tas_avoid = tas_direct

    row = str(start[0])+'#'+str(start[1]) + ', ' + str(end[0])+\
          '#'+str(end[1]) + ', ' + str(route_length) + ', ' +\
           str(route_o_length) +\
           ', '
    row += str(route_nodes) + ', ' + str(route_o_nodes) + ', ' +\
           str(tas_direct) + ', ' + str(tas_avoid) + affected + '\n'
    f_tar.write(row)
s = pd.Series(penalties_GO)
print('Modified route penalties')
print(s.describe())


s = pd.Series(penalties_G)
print('Original route penalties')
print(s.describe())

f_tar.close()
print(datetime.datetime.now().time())
```

Appendix 10. Data visualization

```
from os import stat_result
import geopy.distance
from math import sqrt
import numpy as np
import pandas as pd
from scipy import stats
import bootstrapped.bootstrap as bs
import bootstrapped.stats_functions as bs_stats
import matplotlib.pyplot as plt
import matplotlib.markers as markers



params = {'xtick.labelsize':'x-large',
          'ytick.labelsize':'x-large'}
plt.rcParams.update(params)

colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b',
'#e377c2', '#7f7f7f', '#bcbd22', '#17becf']


edges = ['200','500','1000','2000','5000']
fnames = ['routes_3000_pen_200_edges_sorted_by_ta_sign95_800.csv',
          'routes_3000_pen_500_edges_sorted_by_ta_sign95_800.csv',
          'routes_3000_pen_1000_edges_sorted_by_ta_sign95_800.csv',
          'routes_3000_pen_2000_edges_sorted_by_ta_sign95_800.csv',
          'routes_3000_pen_5000_edges_sorted_by_ta_sign95_800.csv',
          ]
pnlts = ['200','500','1000','2000','5000']
paths = open('routes_3000_pen_500_edges_sorted_by_ta_sign95_800.csv', 'r')

header = paths.readline()
hv = header.split(',')

markers = [".","o","v","^","+","h","s","*",]
```

Continuation of Appendix 10

```
def plot_dependency(ylabel, column,label,legend):

    rheader = 'eucl_dist','length_before','route_length_after/before',\
              'tas_after_before','nodes_after_before','penalty','affected']
    plots = []
    for i, pnlt in enumerate(pnlts):
        mean_ta_rt = []
        mean_ta_ci_l = []
        mean_ta_ci_u = []
        x = []
        xax = []
        start = 1
        total_routes = 0
        affected = 0
        print(pnlt)
        while start < 50:
            xax.append(str(start) + '-' + str(start + step))
            x.append(stat_result)
            ta_ratios = []
            for r in routes:

                val = r[1]/1000
                if r[5] == pnlt:
                    total_routes += 1
                    aff = r[rheader.index('affected')].strip()
                    if aff == 'yes':
                       affected += 1
                    if val > start and val < start + step:
                       ta_ratios.append(r[rheader.index(column)])
            stats = interval_stat(ta_ratios)
            mean = stats[1]
            mean_ta_rt.append(mean)
            mean_ta_ci_l.append(stats[2])
            mean_ta_ci_u.append(stats[3])
            print(start,start+step, ' ', len(ta_ratios), "{:.3f}".format(mean),\
                  "{:.3f}".format(stats[2]), '-', "{:.3f}".format(stats[3]))
            start = start + step
        print('Affected routes:', affected/total_routes)
        print('\n')

        if pnlt == '10000000000000': pnlt = '∞'
        y = np.array(mean_ta_rt)
        yl = np.array(mean_ta_ci_l)
        yu = np.array(mean_ta_ci_u)
        plot = [y,yl,yu]
        plots.append(plot)
    fig, ax = plt.subplots()
    ax.labelsize:'medium'  # fontsize of the x any y labels
    for j in range(len(pnlts)):
        ax.plot(xax,plots[j][0],label = label[j], color = colors[j],\
                marker=markers[j])
        ax.fill_between(xax, plots[j][1], plots[j][2], color=colors[j], alpha=.1)
    ht = np.ones(20)
    l = np.linspace(0,10,20)
    ax.plot(l,ht, color = 'lightgrey')
    lg = ax.legend(loc='upper right', fontsize=16)
    lg.set_title(legend,prop={'size':16})
    plt.xlabel('Длина первоначального (без препятствий) маршрута, км',fontsize=18)
    plt.ylabel(ylabel, fontsize=18)
```

Continuation of Appendix 10

```
def interval_stat(values):
    ntest = stats.kstest(values, 'norm')
    i_len = len(values)
    lr = np.array(values)
    samples = lr
    mean_ci = bs.bootstrap(samples, stat_func=bs_stats.mean, alpha=0.05)
    med = np.median(values)
    med_ci = bs.bootstrap(samples, stat_func=bs_stats.median, alpha=0.05)
    istat = [i_len, mean_ci.value , mean_ci.lower_bound, mean_ci.upper_bound,\
             med, med_ci.lower_bound, med_ci.upper_bound]
    return istat

def geoDist(col1,col2, prts, header):
    latlon1 = parts[hv.index(col1)].split('#')
    latlon2 = parts[hv.index(col2)].split('#')
    lat1 = float(latlon1[0])
    lon1 = float(latlon1[1])
    lat2 = float(latlon2[0])
    lon2 = float(latlon2[1])
    return geopy.distance.distance((lat1,lon1), (lat2,lon2)).km


routes = []
cnt = 0
for i in range(len(pnlts)):
    pnlt = pnlts[i]
    fname = fnames[i]
    paths = open(fname, 'r')
    header = paths.readline()
    hv = header.split(',')
    hv[-1] = hv[-1].strip()
    while(True):
        line = paths.readline()
        parts = line.split(',')
        if len(parts) < 9: break
        dist = geoDist('BEGIN', 'END', parts, hv)
        d = round(dist,3)
        tas_after = float(parts[hv.index('TA_AVOID')])
        tas_before = float(parts[hv.index('TA_DIRECT')])
        if tas_before != 0:
            tas_after_before = tas_after/tas_before
        else:
            continue
        length_after = float(parts[hv.index('ROUTE_O_LENGTH')])
        length_before = float(parts[hv.index('ROUTE_LENGTH')])
        length_after_before = length_after/length_before

        nodes_after = float(parts[hv.index('ROUTE_O_NODES')])
        nodes_before = float(parts[hv.index('ROUTE_NODES')])
        nodes_after_before = nodes_after/nodes_before
        affected = parts[hv.index('AFFECTED')].strip()
        r = [dist, length_before, length_after_before,\
             tas_after_before, nodes_after_before, pnlt, affected]
        routes.append(r)
    paths.close()
```

Continuation of Appendix 10

```
step = 5

plot_dependency(‘Relative TA risk','tas_after_before', edges, 'Penalty m')
plt.show()

plot_dependency(Average route length after/before’,'route_length_after/before',
edges, 'Penalty m')
plt.show()

plot_dependency('Average vertice count after/before','nodes_after_before', edges,
'Penalty m')
plt.show()
```