INTRODUCTION

This work describes the process and the results of the work on the development of a recommendation system for building tourist routes based on solving the multi-criteria Orienteering Problem.

The problem of planning a tourist itinerary is very relevant.

Modern world is developing rapidly. Process of globalization is increasing the possibilities for travel. Now people get more and more opportunities for development, learning new things, researching a different culture, learning new languages, meeting new people. Modern types of transport allow us to quickly cover great distances in short periods of time. A large number of travelers allows transport companies to organize transport regularly and frequently. Many countries are adapted to tourists, for example, direction signs to attractions, cheap hostels, low-cost tariffs on flight, information points for tourists and etc. All these things make travel affordable, safe and comfortable.

That's why a lot of people travel around the world. And most of them face the problem of travel planning. Everyone wants to see as many attractions as possible during their time in the city. For this it is necessary to think over the optimal route across sightseeings with maximum profitability.

Depending on desire and financial capabilities, the tourist can choose the way of travel. There are different types of transport: public transport, taxi or, possible, by foot. Also separation of tourists may be dependent on physical training. Some may prefer to be more outdoors and minimizing being in transport (within a reasonable distance). Some may prefer, on the contrary, to look at the world from the bus window. These are the two extreme sides of the ability to move, it can be taken combined with particular coefficients.

Also the problem of movement to solve the problem of moving between non-trivial points, for example, from "island" to "continent", from northwest to northeast, from north to south. This problem is non-trivial, because usually all directions of the city go to the center.

Also in modern cities it is necessary to take into account traffic on the road. For planning time of itinerary with adaptation to real conditions. Also algorithms take into account bus schedules.

Main goal of route calculation is to minimize the number of transfers and minimize travel time.

Recommendation system of tourist itinerary can take into account some wishes of tourists. Main goal of this system is to maximize "income" deceived from visiting places in a limited time. Each place has its own rewards for visiting.

Task for work:

1. Make an analytical review of methods, algorithms about Orienteering Problem;
2. Make an analytical review of methods about route planning on transportation networks;
3. Make an analytical review of methods about geographic data storing methods
4. Analysis and development of algorithms for routes by public transport;
5. Development algorithm for routes by foot and by taxi;
6. Comparing of received results when building routes by public transport;
7. Implement route building by public transport modification;
8. Development of methods for multi-criteria Orienteering Problem construction;
9. Integration with Framework for Orienteering Problem solving;
10. Analyzed received results of the Orienteering Problem.

4 IMPLEMENTATION OF BUILDING ROUTE BY PUBLIC TRANSPORT

4.1 Description of task

For building tourist routes, it is necessary to take into account all possible ways of moving between attractions. This is on foot, by public transport and by taxi. In this chapter, we will describe algorithms for constructing itinerary on public transport. The results of the algorithm should be as close as possible to the real ones. Therefore, the route will take into account the bus schedule and traffic jams on the road in real time.

The construction of this model will be implemented in several steps:

1. Parsing data about public transport routes;
2. Parsing data about traffic-jams in real time;
3. Processing the received data to obtain general information about itinerary on public transport;
4. Building direct route by public transport;
5. Building routes with one transfer by public transport;
6. Building routes with two transfers by public transport.

All these steps will help you build routes in accordance with the actual situation on the roads. In order to adapt the routes to comfortable conditions for tourists, the following chapters will consider modifications of this algorithm.

The algorithm optimization priority goes in the following order:

1. Minimize the number of transfers;
2. Minimize the time spent on the trip.

The option with three or more transfers is not considered because in a large city this is quite a rare phenomenon, and it is also as uncomfortable as possible for tourists. Also, all the popular attractions of the city are usually within the reach of a maximum of two transfers. Otherwise, perhaps this attraction probably does not deserve a visit.

4.2 Parsing data about public transport

Data processing for further work takes place in two stages. The first step is to work with static data. Static data tke from GTFS (General Transit Feed Specification) Feed [36]. This data contains information about all routes and detailed information about each of them (stops in right order, distance between stops and other information). For convenience it is necessary to convert data to the following format as in [37]. Its required date for work can be seen in Table 4.1. The data about city sightseeing takes from [38]. Data from other columns (route short name, full name of route, type of transport, name of stops) contains a description of the route that can be used for full and detailed information for the user.

Table 4.1 – Example of converted data format

index	route_id	direction	stop_id	next_stop_id	stop distance
1	1125	direct	16396	16367	0.85
2	1125	direct	16367	16371	0.4
…	…	…	…	…	…
14	1125	reverse	33093	16348	0.1
…	…	…	…	…	…

This data format makes it easier to work with the data. Since all the stops for each route are in order. This allows work not with data itself, but with their indexes. Each row has its own index.

Next, it is necessary to parse data about the real location of public transport GTFS- realtime Vehicle Position from [39]. Parse data every 3 minutes during two weeks. An example of finding a transport in real time is reflected in Figure 4.1. The download data received in format JSON.

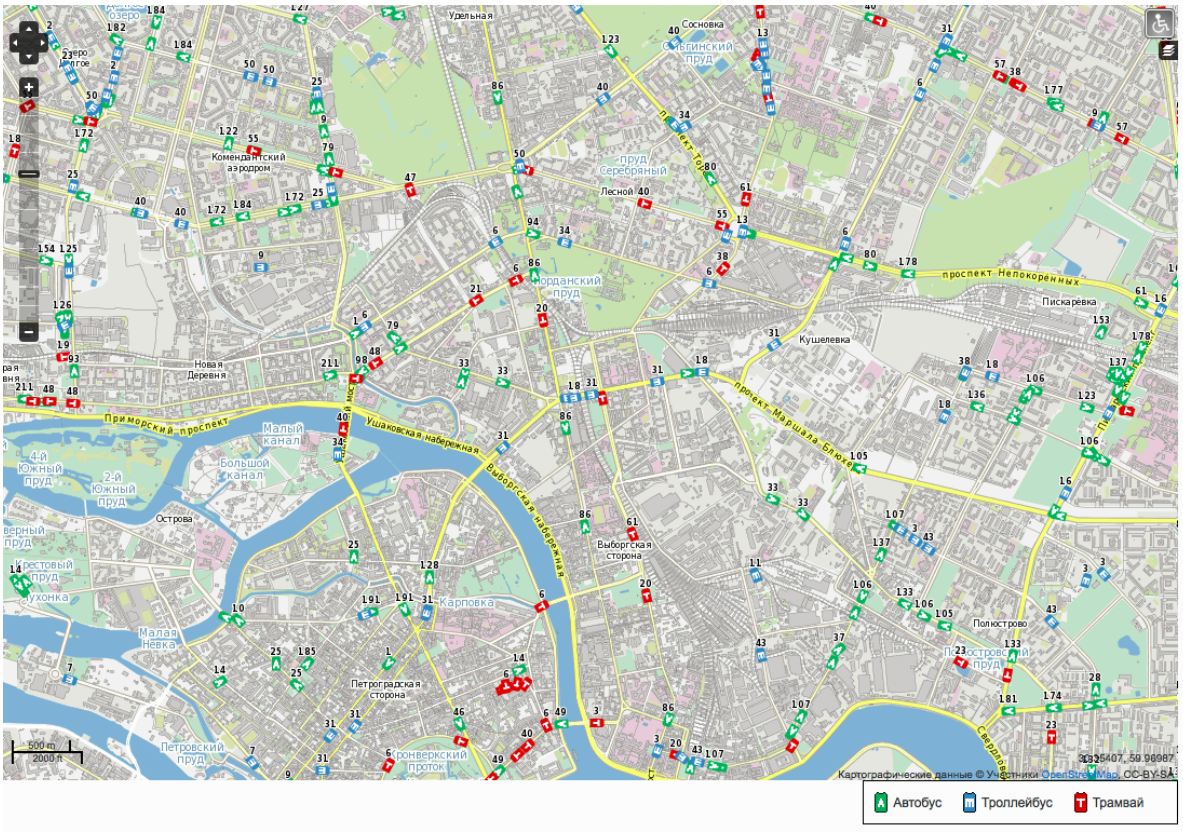

Figure 4.1 – Position of public transport in real time [39]

4.3 Processing the received data about public transport

The main idea of data processing is union information from both resources. That is, get data with stops in the right order and with data about the time between stops.

Data will be merged every 10 minutes. That is data in the model will be changed every 10 minutes. Main data from realtime data that is used in the algorithm is current coordinates of transport, timestamp in the moment of sending data and route's id.

For every 10-minutes interval there are approximately 3-4 files. Take information from data about both weeks.

Let's work with files with real-time information.

It consists of trips. Each trip(t) matches to some route (r) . Accordingly, route for particular time interval can be written as $\vec{t}(r) = (t^0_r, t^1_r, \dots)$ and $t \in r$.

So, consider each trip from a set of trips. From static file data take information about the current route (which corresponds to the trip). The information that is needed is set of stops for this route $\vec{p}(r) = p^0_r, p^1_r, \dots$ And information from real data is set of point about location current transport $\vec{point}(t) = point^0_t, point^0_t, \dots$

Now it is needed to find for each point the appropriate nearest stop if possible. To do this, calculate the distance between current point and each stop. Now is taken from all calculations the smallest result is $\min(\Delta\tau_{fp}(p^{i}_{r}, point^{j}_{t}))$. And if this minimum result is less than 5 minutes in foot path (that is approximately 400 meters), then assign to it stops timestamp equal to timestamp of this current point $\tau(p^{i}_{r}) = \tau(point^{j}_{t})$. Do the same algorithm for each point.

At this step, finding indexes for stops with known timestamps. These indexes can contain errors that appeared when calculating the nearest stop. This is seen as a separate index for comparison with others. It can be solved with deleting outliers. If the difference between index from indexes and expected values of all indexes is less than standard deviation of all indexes, then the index is satisfied by the group of these indexes.

Now define minimal and maximal indexes with non - zero stops timestamp. In this way it is possible to calculate how many kilometers $\Delta dist_{bpt}(p^{min}_{r,}\ p^{max}_{r})$ the transport travels during this interval of time $\Delta\tau_{bpt}(p^{min}_{r,}\ p^{max}_{r})$. Using this data can be calculated to average speed of transport. And so, time between nearby stops is calculated $\Delta\tau_{bpt}(p^{j}_{r,}\ p^{j+1}_{r})$.

After processing all trips from a particular route is getting a set of stops with time. This data has duplicates. To fix this will merge all trips from route by column stop_id, next_stop and direction. If there is at least one value for some stops than take mean value from all trip $\Delta\tau_{bpt}(p^{j}_{r,}\ p^{j+1}_{r}) = \text{mean}(\Delta\tau_{bpt}(p^{j}_{r,}\ p^{j+1}_{r}))$, else $\Delta\tau_{bpt}(p^{j}_{r,}\ p^{j+1}_{r}) = NA$. Also having this data it is possible to note whether there was a stop of this bus on this route in this period of time. And it should be noted by the new characteristics of data for route: available. If $\Delta\tau_{bpt}(p^{j}_{r,}\ p^{j+1}_{r})$ is not NA available$(p^{j}_{r}) = 1$; else available$(p^{j}_{r}) = 0$.

Also for every 10-minutes interval file is to create 6 columns with time between buses from the next 5 files. When to calculate a common time of trip, if the accumulated sum of time when divided by 10 gives an integer part equal to 1, then move to column timeBeweenStops1 and proceed to calculate the accumulated sum of time. Until the route ends or until the sum is divided by 10 with an integer part equal to 2, then the calculation moves to column timeBeweenStops2 and etc. (it is assumed that the journey on one transport continues no more than an hour). This instruction for the function $\Delta\tau_{bpt}(p_{b}^{ri},\ p_{e}^{ri})$.

To an approximate calculation the transport travel time in the current time interval, is also needed to know the time between other stops. It is possible to predict this with a neural network. The work of this prediction is presented in another chapter.

The pseudocode of data processing is presented below:

Input: GTFS-realtime (2 weeks, every 3 minutes), GTFS Feed

allTransport = convert GTFS Feed to format as [37]

Output: $path^{best}$

MainData = Processing(GTFS-realtime week#1 & week#2, day, 10-minutes interval)

function **Processing**(GTFS-realtime, day, 10-minutes interval){

for each t in unique set t:

$\vec{p}(r) = p^0_r, p^1_r, \ldots$ for static stops in r

$\vec{point}(t) = point^0_t, point^0_t, \ldots$ for real points in t ($t \in r$)

for each $point^j_t$, in $\vec{point}(t)$:

for each p^i_r, in $\vec{p}(r)$:

if $\min(\Delta\tau_{fp}(p^i_r, point^j_t)) < 5$: $\tau(p^i_r) = \tau(point^j_t)$

for each p^i_r from $\vec{p}(r)$:

if $\tau(p^i_r)$ is not Null:

$\Delta\tau_{bpt}(p^{min}_r, p^{max}_r) = \tau(p^{max}_r) - \tau(p^{min}_r)$

$\Delta dist_{bpt}(p^{min}_r, p^{max}_r) = dist(p^{max}_r) - dist(p^{min}_r)$

$speed = \Delta dist_{bpt}(p^{min}_r, p^{max}_r) / \Delta\tau_{bpt}(p^{min}_r, p^{max}_r)$

For each p^j_r between p^{max}_r and p^{min}_r :

$\tau(p^j_r) = dist(p^j_r) / speed$

$\Delta\tau_{bpt}(p^j_r, p^{j+1}_r)$ time between nearby stops

now graph has duplicates r, fix it:

for each r^i in set r:

merge set t from r^i by p^j_r and direction

if $\Delta\tau_{bpt}(p^j_r, p^{j+1}_r)$ at least 1 from all t is not NA:

$\Delta\tau_{bpt}(p^j_r, p^{j+1}_r) = mean(\Delta\tau_{bpt}(p^j_r, p^{j+1}_r))$ from all t

$available(p^j_r) = 1$ ($\Delta\tau^{wait}(p^{ri}) < 10$)

else: $\Delta\tau_{bpt}(p^j_r, p^{j+1}_r) = NA$; $available(p^j_r) = 0$

fill NA values $\Delta\tau_{bpt}(p^j_r, p^{j+1}_r)$ with neural networks

4.4 Building direct route by public transport

Direct route is the most preferred, comfortable type of trip. In addition - it is the frequent type of trip, when the purpose of travel is tourism. Because usually the most famous attractions are located in the city center or in the vicinity of it.

Let's build a direct route from source point g_{src} to the target point g_{tgt}. First is needed to find all potential stops set p^b, set p^e in radius 1.5 km, from which it is possible to begin and end the trip by transport. Radius can be changed depending on the desire of tourists. 1.5 km is approximately equal to 15 minutes walk. These stops are searched for using the k-d Tree method.

Then searching all transport set r^b that pass through set of stops set p^b. It also should meet the requirements of the transport schedule: available(p^i_r) == 1. It means that this transport passes through this stop in this 10-minute time interval. The same search does for set p^e.

Now are looking for the intersection of those two sets r^b and r^e. And as the result is a new set r^{suit}. Now at this step the shortest time is chosen. The time from source point to begin stop, the travel time and the time from end stop to target point are taken into account.

So the best direct path is found.

The pseudocode of finding the direct path is presented below.

function **pathWithDirectBus** (g_{src}, g_{tgt}){

set p^b, set p^e = k-d TreeSearch(g_{src}, g_{tgt}, radius = 1.5 km)
set r^b, set r^e = { for i in $\overrightarrow{r}$:
if p^b in $\overrightarrow{p}(r^i)$ & available(p^i_r) =1:
then $r^b = r^i$ }
r^{suit} = {for each r^i on r^b:
for each r^j on r^e :
if r^b == r^j:
then $r^{suit} = r^i$}
$time^{best}$, r^{best}= {
for each r^i in r^{suit} :
$time^{best}$ =min($\Delta\tau_{bpt}(p_b^{ri}, p_e^{ri}) + \Delta\tau_{fp}(p^{ri}{}_e, g_{tgt}) + \Delta\tau_{fp}(g_{src,} p^{ri}{}_b)$)}

The example of building routes in Figure 4.2.

Figure 4.2 – Example of direct route: From Church of the Savior on Spilled Blood to Temple Theodore Icon of the Mother of God. From the bus station Ст. метро "Невский проспект" to bus station Пр. Бакунина with bus number 10.Time in way = 17 minutes. Fare=50 rubles.

4.5 Building route with one transfer by public transport

If the path is not found when building a direct route. Then it is needed to build a route with one transfer.

The main idea is to find the bus stops intersection of buses passing through the start and end points.

Set of stops and buses in the start point and set of stops and buses in the end point is known after the previous step (searching direct path). Now it is a task to find the intersection between the set of stops $p^{b\ ri}{}_{stop\ k}$ for each route $r^b{}_i$ from the general set r^b for the begin point and the set of stops $p^{e\ rj}{}_{stop\ v}$ for each route $r^e{}_j$ from the general set r^e for end point. Also it reduces the sets of buses in begin and end points. Found intersection stops set Tr means that here cross a bus from begin point and end point and here tourists can change the transport. This algorithm also checks whether the bus passes through desired stops in the current 10 - minutes interval or not.

In this step is having a set of bus in the near of start points set r^b, a set of bus in the near of end points set r^e and the set of their common stops set Tr. The current task is to find minimum time from summarizing of foot path from source point to begin stop $\Delta\tau_{fp}(g_{src,} p^i{}_b)$, footpath from end stop to target point $\Delta\tau_{fp}(p^j{}_e, g_{tgt})$ and route by transport $\Delta\tau_{bpt}(p^i{}_b, p^i{}_{tr}) + \Delta\tau_{bpt}(p^j{}_{tr}, p^j{}_e)$.

So the best path with one transfer is found. The example of building a route with one transfer is in Figure 4.3.

The pseudocode of finding the path with one transfer is presented below:

function **pathWithOneTransfer** (g_{src}, g_{tgt}){

set p^b, set p^e , set r^b, set r^e from **pathWithDirectBus** (g_{src}, g_{tgt},)

set$\vec{p}$(set r^b),set$\vec{p}$(set r^e)

set Tr = {

for each r^b_i from set r^b{

for each $p^{b\ ri}_{stop\ k}$ from r^b_i{

for each r^e_j from set r^e{

for each $p^{e\ rj}_{stop\ v}$ from r^e_j{

if $p^{b\ ri}_{stop\ k}$ == $p^{e\ rj}_{stop\ v}$ & available(p^p) ==1 & available($p^e_{stop\ v}$)==1:

set Tr = $p^{b\ ri}_{stop\ k}$; set r^b = r^b_i ;set r^e = r^e_i }

for each i from set r^b:

for each j from set r^e:

$time^{best}$, r^{best}_b, r^{best}_e, tr^{best} = min($\Delta\tau_{bpt}(p^i_b, p^i_{tr}) + (\Delta\tau_{bpt}((p^j_{tr}, p^j_e) + \Delta\tau_{fp}(p^j_e, g_{tgt}) + \Delta\tau_{fp}(g_{src}, p^i_b)$)

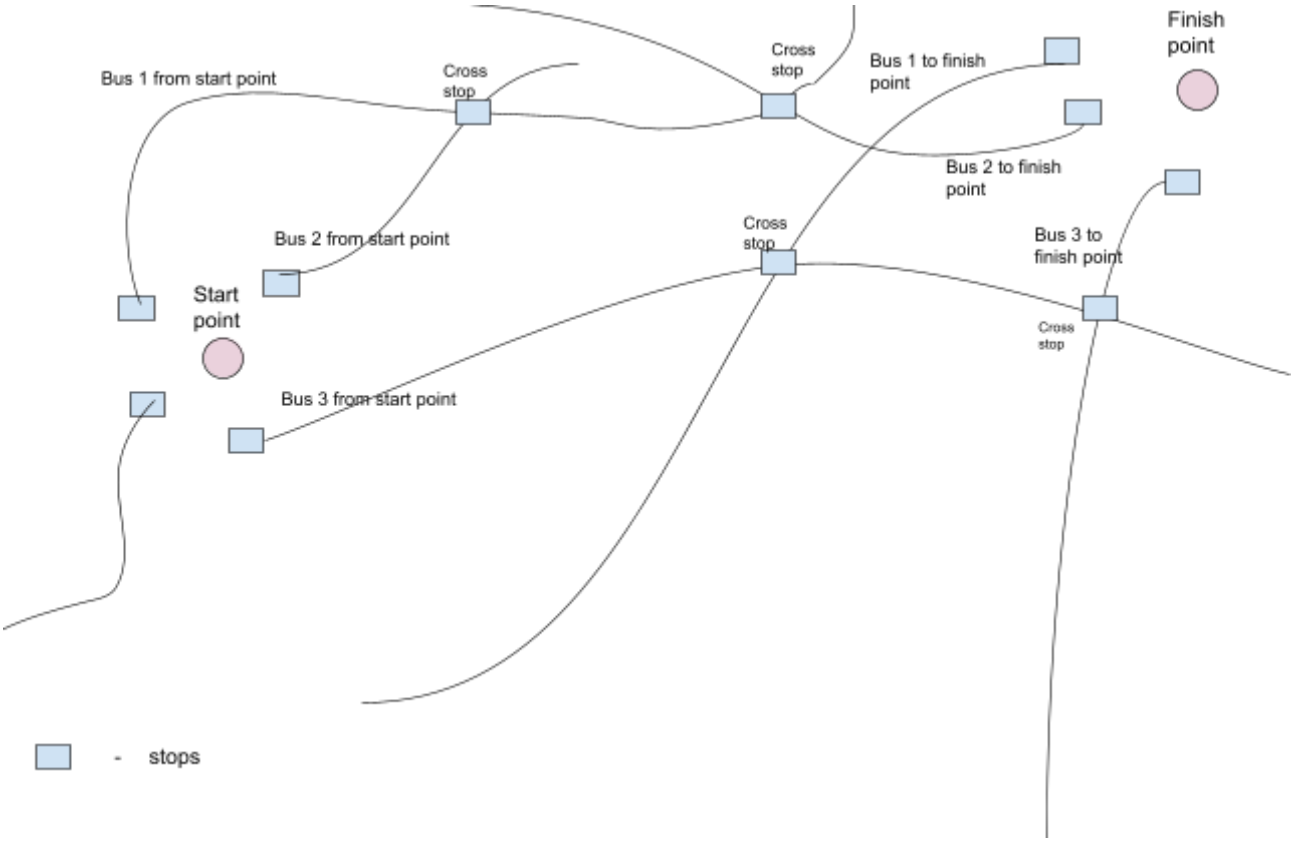

Figure 4.3 – Scheme of route with one transfer

4.6 Building route with two transfers by public transport

If the path is not found when building a route with one transfer. Then it is needed to build a route with two transfers.

The main idea to build a route with two transfers can be presented in a few step:

1. Divide a path between source and target points into 10 parts
2. Retreat from the starting point 1/10 (“part1”) of the distance;
3. Find all stops set p^{b}_{part1} within a radius of 5 kilometers from the “found” point and have direct transport set set r^{b}_{part1} a from the starting point to this “found” point (each stop from this set of stops is potential stop for first transfer);
4. Save only those stops where the largest number of buses stop. Let's leave the top-30 most popular stops (pseudocode is available in function findPotentialTransfers);
5. Retreat from the finish point 1/10 of the distance and repeat step 3 and 4 for this point. So, it is getting a set of stops set p^{e}_{part2} and set of buses set set r^{e}_{part2} ((each stop from this set of stops is potential stop for second transfer);
6. Now there are 2 sets of buses that pass from starting point and first found point(p1) and that passes from end point and second found point (p2). Let's look for buses that pass through a set of stops p1 - set p^{b}_{part1} and p2 - set p^{e}_{part2}. Now there is a set r^{join};
7. After this retreat from point p2 - 1/10 of the distance. And rerun all steps. Retreat 1/10 and rerun all step 1-6 until part2 < 1;
8. After this retreat from point p1 - 1/10 of the distance and repeat steps 1 - 7. Until part1 < 1. (In order to avoid repeating a points there is a condition after each step of part1: full = full - 0.1);
9. After all these steps there are a full set of possible routes r^{join};
10. The further task is find route to stop of first transfer from begin and route to stop from second transfer to end with pre-calculated time (pseudocode is available in function preprocessingDataForFindingBusToTransfers);
11. Here all parts of the route are connected in one path (route to first transfer point, route between transfers and route from second transfer point). there will be a set of possible paths. Also here it is checked that the stops for the first transfer happened at the same stop; also it is checked for the stop of the second transfer. And at this step, the availability of the bus at this stop in this time interval is checked. Choosing the route with minimum time.

function **pathWithTwoTransfers**($g_{src,}$ g_{tgt}){

set p^b, set p^e , set r^b, set r^e from **pathWithDirectBus** (g_{src}, g_{tgt},)

set$\vec{}$p(set r^b),set$\vec{}$p(set r^e)

full = 1.0

for part1 = 0.1 ; part1 < 1; part1 =+ 0.1:

for part2 = 0.1; part2 < full; part2 =+ 0.1:

set p^b_{part1} ,set set r^b_{part1} = **findPotentialTransfers**(set r^b, maxRadius, lat^b, lon^b, lat^e, lon^e, part1)

set p^e_{part2} , set set r^e_{part2} = **findPotentialTransfers**(set r^e, maxRadius, lat^b, lon^b, lat^e, lon^e, part2)

for each set r^e_{part1} from set set r^e_{part1} :

for each set r^b_{part2} from set set r^b_{part2} :

for each r^e_{part1} from set r^e_{part1} :

for each r^b_{part2} from set r^b_{part2} :

if r^b_{part2} == r^e_{part1}:

set r^{join}

set $p^b_{r,\ transfer1}$

set $p^e_{r,\ thansfer2}$

full = full - 0.1

$\Delta\tau_{bpt}$($p^b_{r1,}$ $p^e_{r1,\ transfer1}$), set(p^b_{r1} , $p^e_{r1,\ transfer1}$) =

preprocessingDataForFindingBusToTransfers(

$p^b_{r,\ transfer1}$, set p^b, set r^b) # route to point of first transfer from begin

$\Delta\tau_{bpt}$($p^e_{r2,}$ $p^b_{r2,\ transfer2}$), set(p^e_{r2} , $p^b_{r2,\ transfer2}$)= **preprocessingDataForFindingBusToTransfers**(

$p^e_{r,\ thansfer2}$, set p^e, set r^e) # route to point from second transfer to end

for each r^{join}, $p^b_{r\ join,\ transfer1}$, $p^e_{r\ join,\ thansfer2}$ from set (r^{join}, $p^b_{r,\ transfer1}$, $p^e_{r,\ thansfer2}$)

for each p^b_{r1} , $p^e_{r1,\ transfer1}$ from set(p^b_{r} , $p^e_{r,\ transfer1}$) :

for each p^e_{r2} , $p^b_{r2,\ transfer2}$ from set(p^e_{r} , $p^b_{r,\ transfer2}$) :

if $p^e_{r1,\ transfer1}$ == $p^b_{r\ join,\ transfer1}$ & $p^e_{r\ join,\ thansfer2}$ == $p^b_{r2,\ transfer2}$ & available(p^b_{r}) ==1 1 & available($p^b_{r,\ transfer1}$) ==1 & available($p^e_{r,\ transfer2}$) == 1:

$$time = \min(\Delta\tau_{bpt}(p^{b}_{r,}\ p^{b}_{r,\ transfer1}) + \Delta\tau_{bpt}(p^{e}_{r,}\ p^{e}_{r,\ transfer2}) + \Delta\tau_{bpt}(p^{b}_{r,\ transfer1,}\ p^{e}_{r,\ transfer2}))$$

The function findPotentialTransfers searches the most frequently visited stops in the near of some point. This point is calculated with section formulas in coordinate geometry. Then it is searching the set of nearest stops set p^{cross} in the near of this point and is checking whether at least one bus from the beginning set r^{b} passes through it stops. And after this it is counting the all buses that pass through each of these stops. Sorted stops by number of passing buses. And take the top-100 of this sorting set.

function **findPotentialTransfers** (set r^{b}, maxRadius, lat^{b}, lon^{b}, lat^{e}, lon^{e}, part){

lat^{cross} = part * lat^{b} + (1 - part) * lat^{e}

lon^{cross} = part * lon^{b} + (1 - part) * lon^{e}

set p^{cross} = k-d TreeSearch(lat^{cross}, lon^{cross} , maxRadius)

set p^{cross}, set $countBus^{p\ cross}$ =countBus(set p^{cross}) # count numbers of buses visited each stop

take top-100 the most visited stops by bus

set $countBus^{p\ cross}{}_{top}$ = sort(set $countBus^{p\ cross}$)[:100]

if len(set $countBus^{p\ cross}{}_{top}$) < 50 : **tryWithSmallBegin** (set r^{b}, minRadius = 0, maxRadius + 1, lat^{b}, lon^{b}, lat^{e}, lon^{e}, part)

}

The function preprocessingDataForFindingBusToTransfers matches stops for the trip from source point to the first transfers $r^{pb} == r^{pb\ transfer1}$ and pre-calculated time of trip $\Delta\tau_{bpt}(p^{b}_{r,}\ p^{b}_{r,\ transfer1})$. Also for a trip from the second transfer to the target point.

function **preprocessingDataForFindingBusToTransfers** ($p^{b}_{r,\ transfer1}$, set p^{b}, set r^{b}) {

set $p^{b}_{r,\ transfer1}$, set p^{b}

for each $p^{b}_{r,\ transfer1}$ from set $p^{b}_{r,\ transfer1}$:

for p^{b} from set set p^{b} :

if $r^{pb} == r^{pb\ transfer1}$ & same direction:

$\Delta\tau_{bpt}(p^{b}_{r,}\ p^{b}_{r,\ transfer1})$

set(p^{b}_{r}, $p^{b}_{r,\ transfer1}$)

return for set $p^{e}_{r,\ transfer2}$, set p^{e}

}

In Figure 4.4 is presented a route with two transfers.

Figure 4.4 – Example of route with two transfers. Route 1. From Church of the Savior on Spilled Blood to transfer1. From bus station СУВОРОВСКАЯ ПЛОЩАДЬ [49] to bus station СЕРДОБОЛЬСКАЯ УЛ. with bus number 46.Time in way 26 minutes. Route 2. From transfer1 to transfer2. From bus station ЛАБОРАТОРНЫЙ ПР. [137]< to bus station КТ with bus number К-271.Time in way 29 minutes. Route 3. From transfer2 to Genium Park. From bus station СЕРДОБОЛЬСКАЯ УЛ. to bus station СМОЛЬНЫЙ УНИВЕРСИТЕТ with bus number 137.Time in way 24 minutes. Fare = 150 rubles.

5 RESULTS OF WORKING OF THE ALGORITHM AND ITS MODIFICATION

5.1 Description of modification and its implementation

Algorithms can be modified for meeting the needs of tourists. Further will be presented modifications,its description, differences of their constructions and differences of built routes:

1. Main algorithm.

Conditions:

- transfer can only be at the same stop;
- minimum pedestrian route.

2. Algorithm for tourists who like walking in reasonable limits (limit is 1.5 km for each possible walking part of route)

Conditions:

- transfer can be in different stops within in distance 1.5 km;
- maximum pedestrian route.

3. Algorithm for tourists who like walking in reasonable limits without possibility of change stop in transfer point (limit is 1.5 km for each possible walking part of route)

Conditions:

- transfer can only be at the same stop;
- maximum pedestrian route.

Results of comparisons of algorithms 1- 3 presented in Table 5.1. Illustration of these algorithms is presented in Figure 5.1 - 5.2.

4. Algorithms with possibility choose preferred walking distance coefficient. For example, a tourist who likes to walk can choose the options for walking to a stop with a minimum coefficient (this will mean that the tourist will willingly pass the maximum possible distance provided in the algorithm = 1.5 km in each possible foot path). or on the other hand, if the tourist is tired, he can choose the maximum coefficient to spend as much time as possible in transport. Here, the time in transport is taken into account completely, and the walking time is taken into account with a coefficient. Thus, at 0, only the time on the bus is minimized, and at 20, the total time is minimized. (but since the bus is moving faster, the time spent on the bus is preferable). That is, the coefficient determines how willingly a tourist walks. Or for global minimization of a walking path, it is possible to set the coefficient with a walking route equal to 100, for example. Received results can be called “Comfort level”.

Results of working of algorithm 4 presented in Table 5.1 - 5.3.

5. Complex cases of algorithms. Building routes between atypical points:

- from north to south;
- from north east to north west;
- from island to continent.

Results of working algorithms in the construction of complex algorithms are presented in Table 5.4.

Table 5.1 – Results of comparison results of working of algorithms 1 - 3

Type of route	Algorithm 1	Algorithm 2	Algorithm 3	By taxi	By foot
Direct route	52 m	52 m	75 m	46 m	80 m
Route with one transfer	81 m	82 m	80 m	69 m	126 m
Route with two transfers	103 m	-	106 m	69 m	126 m

The conclusion from the comparison of these algorithms can be made as follows: the best time is expected to be shown by the first algorithm, with the minimum walking distance. Since the walking path takes much longer than the path in transport. But the differences are minor. The tourist can choose the preferred option for him.

Algorithm 2 has '-' in the cell for a route with two transfers because it is difficult to find a path which will not be found with a single transfer in the conditions of the possibility of changing the situation at the transfer stop.

The implementation of modification 4 is presented in Table 5.2 and Figure 5.1 and Figure 5.2.

Table 5.2 – Results of working of algorithm 4 for direct route

Walking preference coefficient	Comfort level	Time in transport	Time by foot	Common time
0	12	12	34	46
0.5	28	13	31	44
...	...	...	...	...
2	75	13	31	44
2.5	88	42	18	61
...	...	...	...	...
10	226	42	18	61

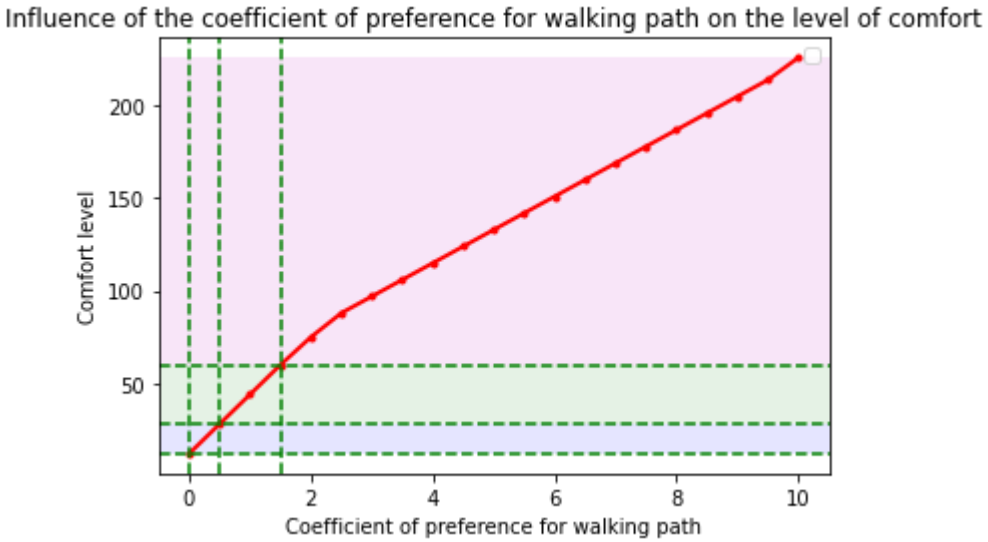

Figure 5.1 – Influence of the coefficient of preference for walking path on the level of comfort for direct route

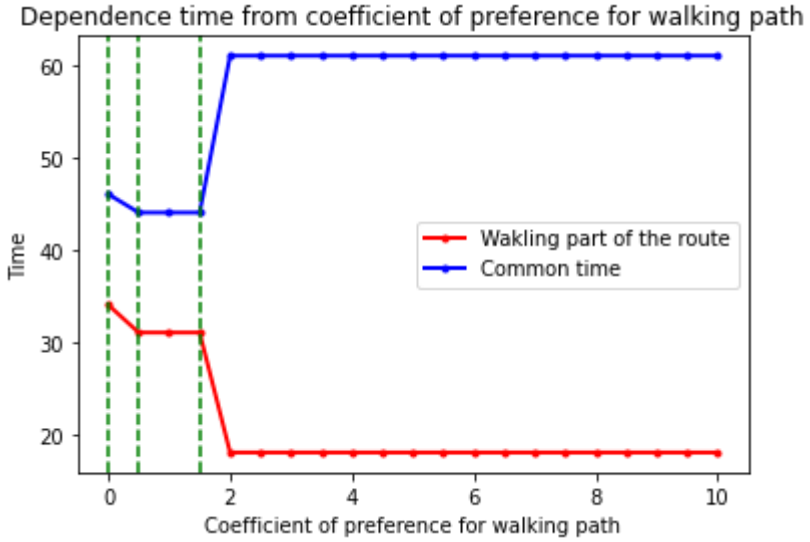

Figure 5.2 – Dependence time from coefficient of preference for walking path for direct route

When considering the Table 5.2 and Figure 5.1 - 5.2 of changes in the level of comfort when changing the coefficient of preference of the walking path, it can be concluded that with an increase in the coefficient, the length of the walking path decreases, and the path on transport increases. At the same time, it can be noted that with a coefficient in the range from 0 to 0.05, the pedestrian path is the same and, consequently, this path more satisfies the needs of a tourist with a coefficient of 0. Further, the coefficient from 0.05 to 1.5 also does not change. The walking part reaches its minimum at a coefficient of 2 and does not change in the future. This means that this route is less comfortable for tourists with high coefficients than with a coefficient of 2.

The next example for a route with one transfer is presented in Figure 5.4 - 5.5 and table 5.3.

Table 5.3 – Results of working of algorithm 4 for route with one transfer

Walking preference coefficient	Comfort level	Time in transport	Time by foot	Common time
0	0	0	96	96
0.5	14	0	96	96
1	27	14	52	66
1.5	34	14	52	66
2	41	19	48	67
...	...	...	...	...
5.5	80	19	48	67
6	85	37	41	78
...	...	...	...	...
10	105	37	31	78

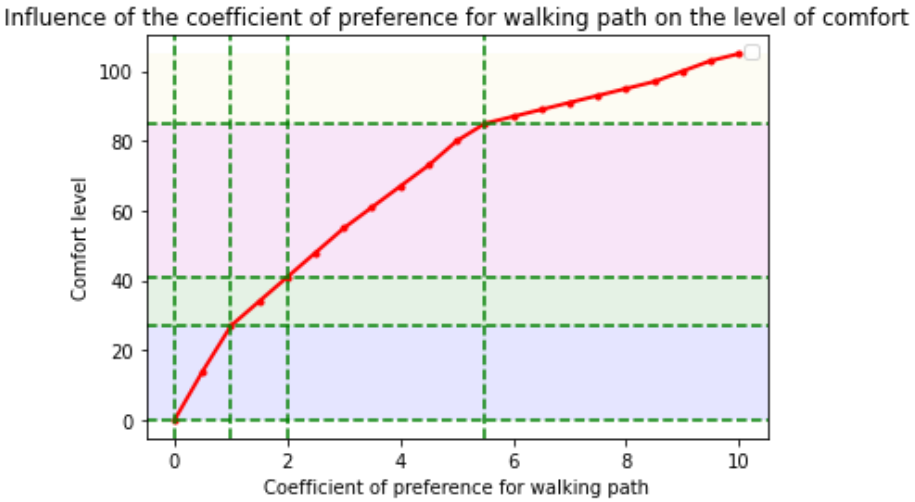

Figure 5.4 – Influence of the coefficient of preference for walking path on the level of comfort for route with one transfer

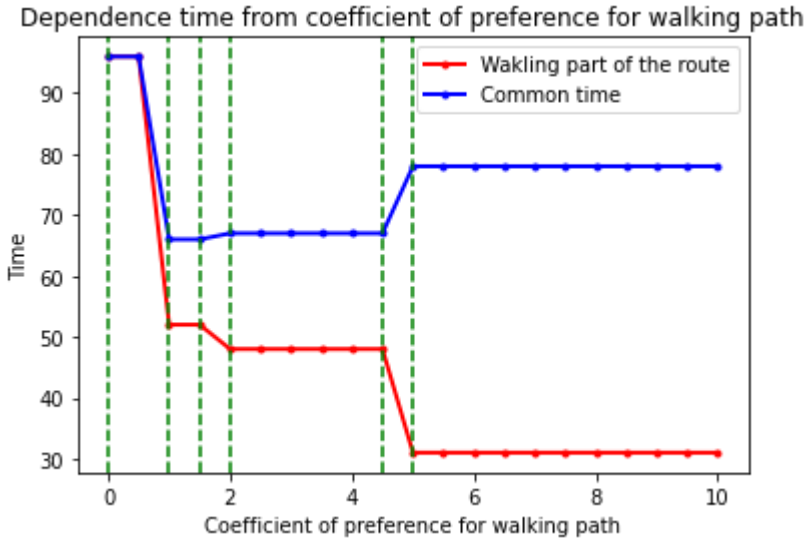

Figure 5.5 – Dependence time from coefficient of preference for walking path for route with one transfer

When considering Table 5.3 and Figure 5.4 - 5.5 it can be noted that with a coefficient in the range from 0 to 1 - for the tourist with such preferences it is most reasonable to go all path by foot.

In case of coefficient from 1.5 to 2 the pedestrian path is the same and, consequently, this path more satisfies the needs of a tourist with a coefficient of 1.5. Further, the coefficient from 4.5 to 5 also does not change. The walking part reaches its minimum at a coefficient of 5 and does not change in the future. This means that this route is less comfortable for tourists with high coefficients than with a coefficient of 5.

The results for complex cases are presented in Table 5.4.

Table 5.4 – Complex cases

Case	Possibility of change stop of transfer	Time of trip	Number of transfers	Time by foot
From north to south	1	103 m	1	325 m
From north to south	0	178 m	2	325 m
From north east to north west	0	160 m	2	180 m
From north east to north west	1	88 m	1	180 m
From island to continent	0	44 m	0	40 m

The algorithm works for complex cases, it is possible to conclude that the algorithm works for complex cases and that the possibility of changing the stop of transfer significantly reduces the travel time and reduces the number of transfers.

The example of a route from North to South is presented in Figure 5.6.

Figure 5.6 – The example of a route from North to South. Route 1. From North to transfer. From bus station ПР. КУЛЬТУРЫ to bus station ИРИНОВСКИЙ ПР. with bus number 102.Time in way 82 minutes. Route 2. From transfer to South. From bus station ИРИНОВСКИЙ ПР. to bus station КТ with bus number К-388.Time in way 51 minutes. Fare = 100 rubles.

5.2 Comparison of the developed algorithm with other similar algorithms

Comparison of the algorithm can be based on several criteria:

1. Quality attribute (when calculating the path, if the algorithm builds a path between points with the less time in the trip and the less the numbers of transfer, that means that this algorithm builds a better route). Comparison based on this attribute is shown on Table 5.5;

2. Time to build a route (a faster algorithm is preferred). Comparison based on this attribute is shown on Table 5.6;
3. Enabling various build conditions (for example, accounting for transport schedules, accounting the traffic jam etc.). Comparison based on this attribute is shown on Table 5.6.

Here the algorithms will be compared on a qualitative basis (Table 5.5). A description of all presented algorithms can be found on the theoretical part of the work.

Table 5.5 – Comparative analysis of the quality of algorithms

Type of building route (the path that my algorithm shows)	Name of the algorithm	Time in trip	Number of transfers	Taxi ride time	Percentage of quality *
Direct route	ATFutures	28 m	3	21 m	133%
Direct route	Trip-Based Public Transit Routing	42 m	1	21 m	200%
Direct Route	Presented algorithm	33 m	0	21 m	157%
Route with one transfer	ATFutures	59 m	5	51 m	116%
Route with one transfer	Trip-Based Public Transit Routing	87 m	1	51 m	170%
Route with one transfer	Presented algorithm	87 m	1	51 m	170%
Route with two transfers	ATFutures	92 m	7	82 m	112%
Route with two transfers	Trip-Based Public Transit Routing	130 m	4	82 m	158%
Route with two transfers	Presented algorithm	161 m	2	82 m	196%

*Percentage of quality define as the result of dividing time in trip by taxi ride time * 100%.

From the comparative table it is possible to conclude that algorithm ATFutures shows results with the best time for trip, but this trip includes too many transfers. It is very difficult for realization, because in real conditions transport doesn't run exactly on schedule (especially for the case of St.Petersburg). Most likely in real conditions it will take much more time. It will also cost a lot.

When considering the second algorithm it can be seen that it shows good results. But it works very slowly. Thus, the presented algorithm is competitive and has the right to exist.

Now let's compare algorithms by another criteria [40] (by enabling different conditions and time of work).

Table 5.6 – Comparison of algorithms by operating time and other criteria taken into account when building routes

Algorithm	Time	Schedule	Traffic	Time of trip	Transfers	Number of day	Walking path	Fare	Multiple route
Presented algorithm	fast	+	+	+	+	7	+	-	-
ATFuture (CSA)	fast	+	-	+	-	1	+	-	-
Trip-Based Public Transit Routing	slow	+	-	+	+	7	+	-	+
Raptor	slow	+	-	+	+	1	+	-	-
rRaptor	slow	+	-	+	+	1	+	-	-
TP	fast	+	-	+	+	7	-	+	+
PTL	fast	+	-	+	+	1	-	-	-
CSA	fast	+	-	+	-	1	+	-	-
CH	fast	+	-	+	-	periodic	-	-	-
ACSA	fast	+	-	+	+	2	-	-	-
MLS	medium	+	-	+	+	1	-	+	-
McRAPTOR	medium	+	-	+	+	1	-	+	-

Describing the names of the columns in the Table 6:

Algorithm - Name of the algorithm;

Time - Running time;

Schedule - Schedule accounting;

Traffic - Traffic accounting;

Time of trip - Optimization time of trip;

Transfers - Optimization number of transfers;

Number of day - Number of day that it cover;
Walking path - Ability to choose the duration of walking path;
Fare - Fare calculating;
Multiple route - Calculating multiple route options.

Looking at Table 5.6 it is possible to conclude that the developed algorithm is competitive, it takes into account a lot of adding options, that makes the results of work close to real conditions. This algorithm will be useful for building routes by public transport.

6 DESCRIPTION AND IMPLEMENTATION OF MULTI-CRITERIA ORIENTEERING PROBLEM

6.1 Formalization of multi-criteria Orienteering Problem

Consider a set of nodes N = {1 , . . . , | N|}, where each node i ∈ N has non-negative evaluation S_i. The source and the target nodes are attached to nodes 1 and |N| ,respectively.

The main goal of OP - is to determine a route, that is limited by a given time budget T_{max}, that visits some nodes from N and maximizes common gained score [8]. Non-negative time for the passing path between nodes i and j is marked as t_{ij}. The variables are presented as follows: X_{ij} = 1, this means that node i is visited after visiting node j, and X_{ij} = 0 otherwise. The variable u_i will be used to exclude subtours and for the determination of the position of visited nodes on the route [41].

Following formula system (1) – (7) describes it [8].

Objective function, that is to maximize the total collected score is shown in (1).

$$\sum_{i=2}^{|N|-1} \sum_{j=2}^{|N|} S_i X_{ij} \rightarrow max \tag{1}$$

Constraint that guarantee that the route starts from node 1 and ends in node |N| is shown in (2).

$$\sum_{i=1}^{|N|-1} x_{iN} = \sum_{j=2}^{|N|} x_{1j} = 1;\ \forall k = 2, \ldots, (|N| - 1) \tag{2}$$

Constraint ensures the connectivity of the path and guarantees that each node is visited at most once or not visited at all. It is shown in (3).

$$\sum_{i=1}^{|N|-1} x_{ik} = \sum_{j=2}^{|N|} x_{kj} \leq 1;\ \forall k = 2, \ldots, (|N| - 1) \tag{3}$$

Constraint limits the total travel time within the time budget Tmax. It is shown in (4).

$$\sum_{i=2}^{|N|-1} \sum_{j=2}^{|N|} t_{ij} X_{ij} \leq |Tmax|;\ \forall i = 2, \ldots, |N|$$

(4)

Two constraints prevent subtours. It is shown in 5.

$$2 \leq u_i \leq |N|;\ \forall i = 2, \ldots, |N|$$

$$u_i - u_j + 1 \leq (|N| - 1)\left(1 - x_{ij}\right);\ \forall i = 2, \ldots, |N|$$

(5)

Next constraints (6) guarantee that the departure time of a current chosen node is equal to the sum of the departure time of the previous point, travel time between two points.

$$\sum_{i=1}^{|N|-1} \sum_{t=1}^{|T_{ih}|} [W_{iht} + (\theta_{iht} W_{iht} + \eta_{iht} X_{iht})] = \sum_{j=2}^{|N|-1} \sum_{t=1}^{|T_{ih}|} W_{hjt};\ \forall h = 2, \ldots, |N| - 1$$

(6)

Constraint (7) ensures that departure time is in the right time slot.

$$X_{ijt}\ \tau_{ijt} \leq W_{ijt} \leq X_{ijt}\ \tau_{ij(t+1)};\ i = 1, \ldots, |N|-1,\ j = 2, \ldots |N|\ \forall t$$

(7)

Wijt - the departure time in time slot t when travelling from node i to j;

Θijt - slope coefficient of the linear time-dependent travel time;

ηijt - intercept coefficient of the linear time-dependent travel time;

τijt - lower limit of time t for arc (i, j);

Tij - number of time slots for arc (i, j);

The details about calculating parameters θ_{ijt}, η_{ijt} and T_{ij} can be referred to the work of [8].

In constraint (4) variable t_{ij} is calculated as travel time between city's sightseeing according to the scheme, presented in Figure 6.1.

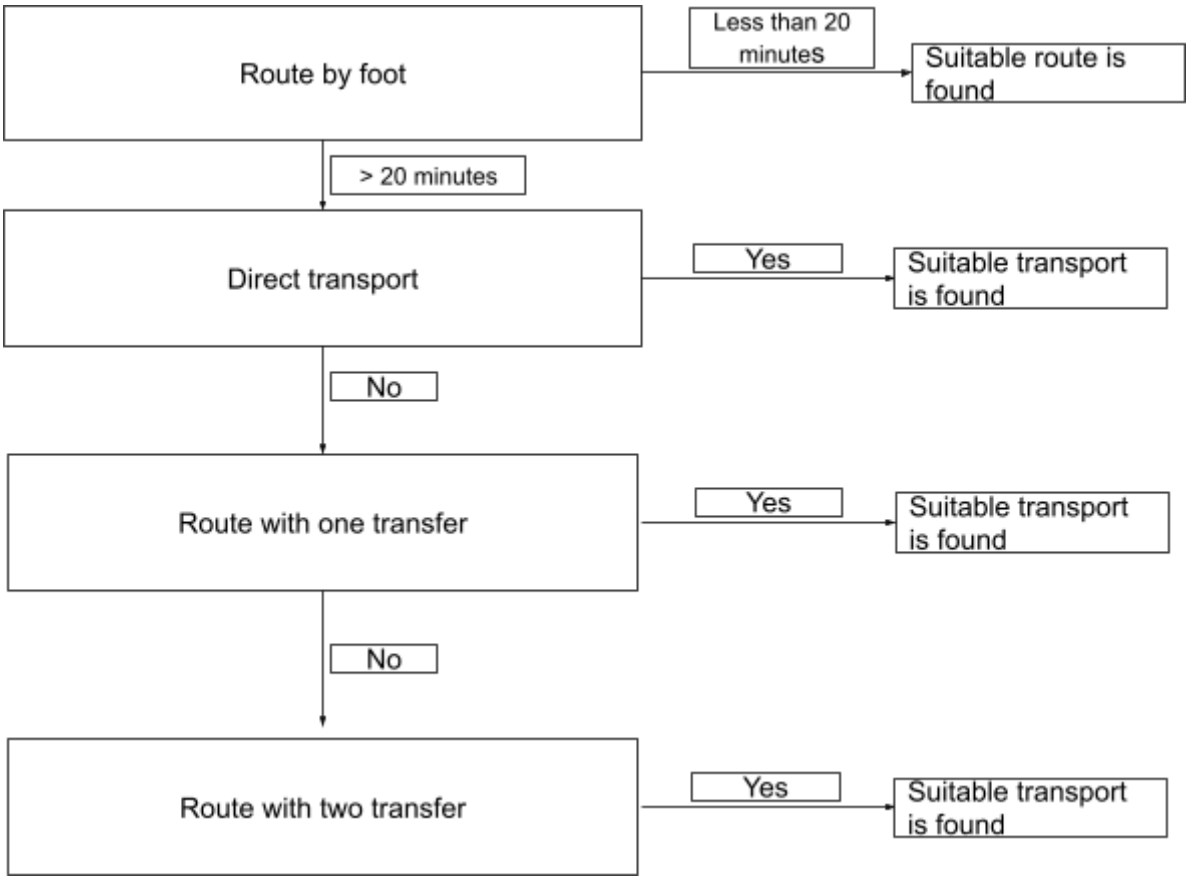

Figure 6.1 – Scheme of calculating time for trip

Let's calculate the average running time of the algorithm for a set of attractions in Saint-Petersburg. Results of calculating are presented in Table 6.1.

Table 6.1 – Average running time of the algorithm for a set of attractions in Saint-Petersburg.

Way of movement	Time (average)
By foot	485 ms
Direct transport	369 ms
With one transfer	1223 ms
With two transfers	3506 ms
Total	709 ms

To build most routes, it is enough to use the algorithm to build a direct path. Less often it is needed to make one transfer on the route. And even more rarely - two transfers.

Empirically, it was revealed that about 3/4 of all movements between the objects of this are direct routes. 7/36 with one transfer and 1/18 with two transfers. (Observations are approximate, obtained empirically from 36 experiments).

6.2 Integration with Framework for Orienteering Problem solving

Integration of a developed algorithm with framework for Orienteering Problem [6] solving a new Orienteering Problem, that constructs routes with considering current situation on road and looking for the best way to overcome the distance between attractions - is Multi-Criteria Time-Dependent OP (MC-TDOP). The framework is implemented in [6] and available in [42].

Results of implementation of our models with time-limit is equal to 600 (10 hours). Routes are built for a maximum sample of 5000 points with a start point in Technologicheskii Institut metro station and finish point in the Mineralogy Museum.

The following routes is building with the model MC-TDOP :

Tekhnologichesky Institut - Vasilyevsky Island - Channel Griboedova Embankment - Nevsky Prospekt - Church of the Savior on Spilled Blood - Peter and Paul Fortress - Trinity Bridge - Mineralogy Museum.

Total score = 692090; Running time = 434 ms.

The schema is presented in Figure 6.2.

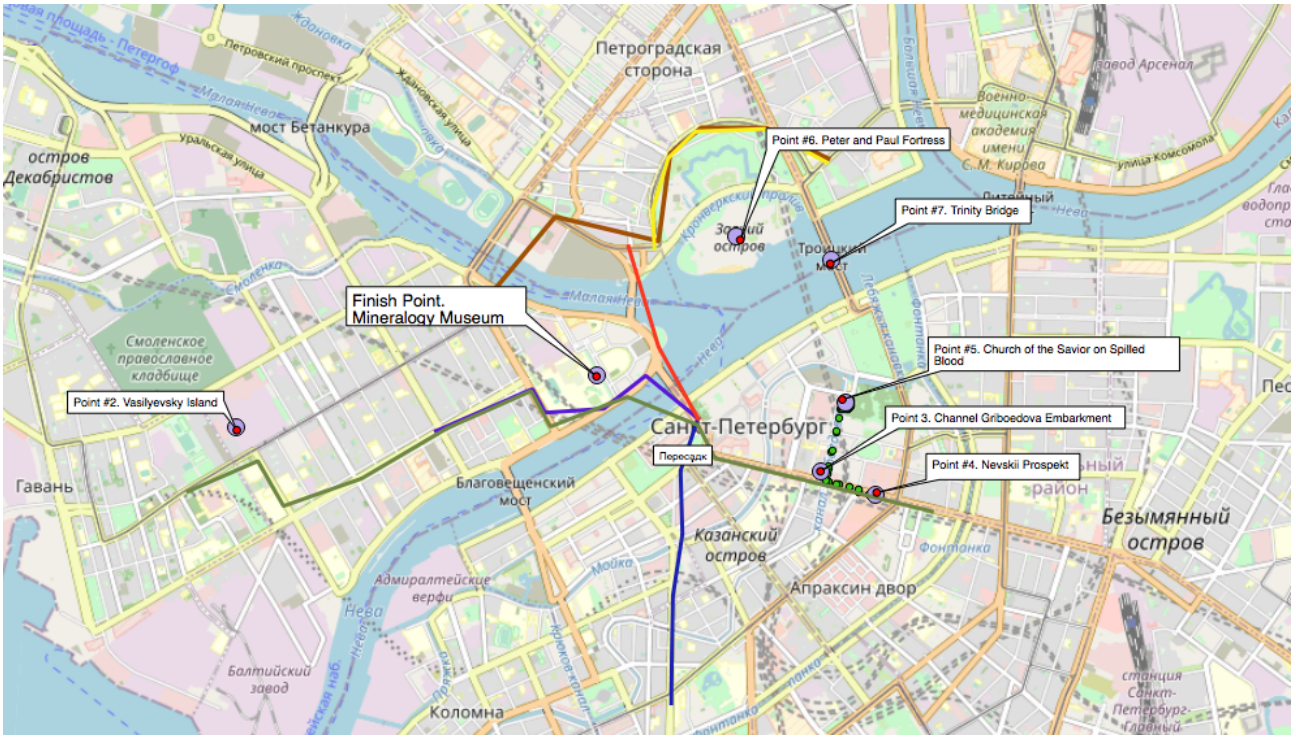

Figure 6.2 – Building a route building with MC-TDOP. Common time = 565 minutes. Common fare = 300 rubles.

Result calculating some variants of Orienteering Problem in Table 6.2.

Table 6.2 – Result calculating some variants of Orienteering Problem

Algorithm	Original running time	Original total score	Actualized running time	Actualized total score
TDOP	867141	720 ms	824444	1.5 s

A comparative analysis of the construction of routes using the original metric of the distance between the sights and developed in this paper is carried out. We can conclude that the developed method counts longer, which is predictable, because it performs much more calculations. And the total score is a little less, since the route is built according to all the bends of the roads and other obstacles.

7 DATA ANALYSIS AND PREDICTION OF UNKNOWN TIME VALUES BETWEEN STOPS

7.1 Description of analyse data

To predict the path between stops, is taken the following data:

1. Geographic coordinates of current stop (latitude and longitude) (type float);
2. Geographic coordinates of next stop (latitude and longitude) (type float);
3. Distance between stops (type float);
4. Distance between stop and city center (Kazan Cathedral) (type float);
5. Direction of movement (to or from city center) (type binary).

Description of analyzed data is presented in Table 7.1.

Table 7.1 – Description of analyzed data

	lon	lng	timeBetweenStop	distanceToCenter	lonNextStop	lngNextStop	inCenter
count	10696	10696	3560	10696	10652	10652	10696
mean	59.9217	30.28589	135.7	14.42	59.92	30.28	0.5
std	0.117841	0.236769	143.9	12.07	0.11	0.235	0.5
min	59.55057	28.971	20.1	0.07	59.55	28.97	0
25%	59.85151	30.2352	79.4	7.19	59.85	30.23	0
50%	59.9231	30.3289	109.4	10.26	59.92	30.32	1
75%	59.9987	30.4108	155.5	19.45	59.99	30.41	1
max	60.36348	31.33243	4191	88.15	60.36	31.33	1

According to this analysis, it is possible to conclude that the point near the Zvenigorodskaya metro station is the median of all stops in the city. Mean time between stops is 135 seconds (equal about 2.25 minutes). And other conclusions about data properties.

Target variable is timeBetweenStop. It has 7136 unknown values.

Let's test the hypothesis of equality of averages for the following data: a time between stops for transport moving from the center and a time between stops for transport moving to the center. Histograms for these data sets with the average values marked on them are presented in Figure 1.

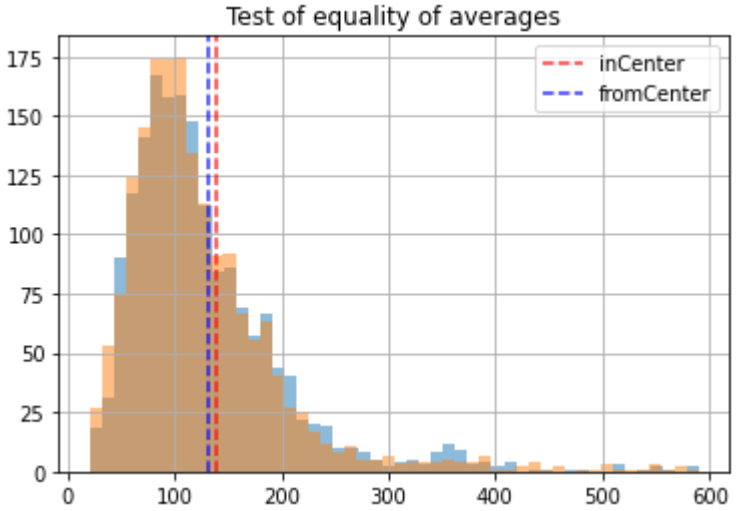

Figure 7.1 – Test the hypothesis of equality of averages

There are a lot of outliers in the data and deleting them completely is not the right action. Therefore, it is possible to use nonparametric criteria to compare the average values. For example, the Mann-Whitney test.

Let’s define the hypothesis:

H0: the averages of both samples are equal;

H1: the averages of both samples are different.

p-value = 0.0324 - that means that there is reason to reject the hypothesis. Conclusion: the average time between stops is different, the difference between these values is not random.

Let's test the hypothesis of equality of averages for the following data: a time between stops for transport moving in the near city center (less than 5 km) and a time between stops for transport is far from city center (more than 5 km). Histograms for these data sets with the average values marked on them is presented in Figure 7.2.

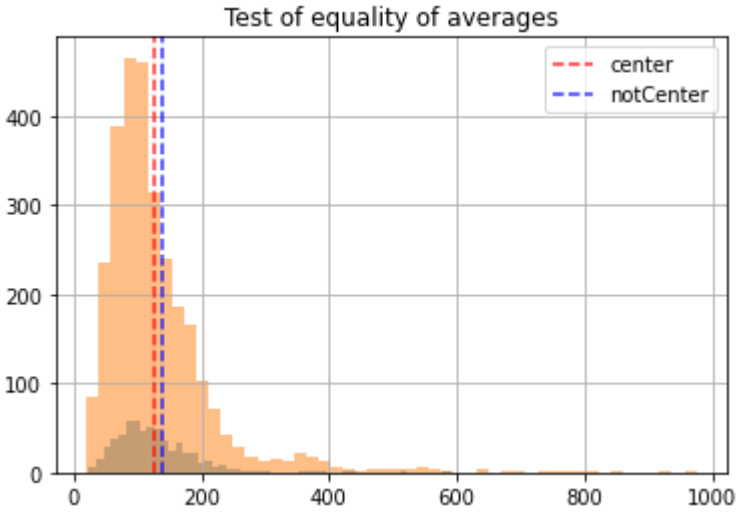

Figure 7.2 – Test the hypothesis of non-equality of averages

There are a lot of outliers in the data and deleting them completely is not the right action. Therefore, it is good to use nonparametric criteria to compare the average values. For example, the Mann-Whitney test.

p-value = 0.0019 - that means that there is reason to reject the hypothesis. Conclusion: the average time between stops in the city center and in not the city center is different, the difference between these values is not random.

Let's test the hypothesis of not equality averages for the following data: a time between stops for transport moving in the north of the city and in the south of the city. Histograms for these data sets with the average values marked on them is presented in Figure 3.

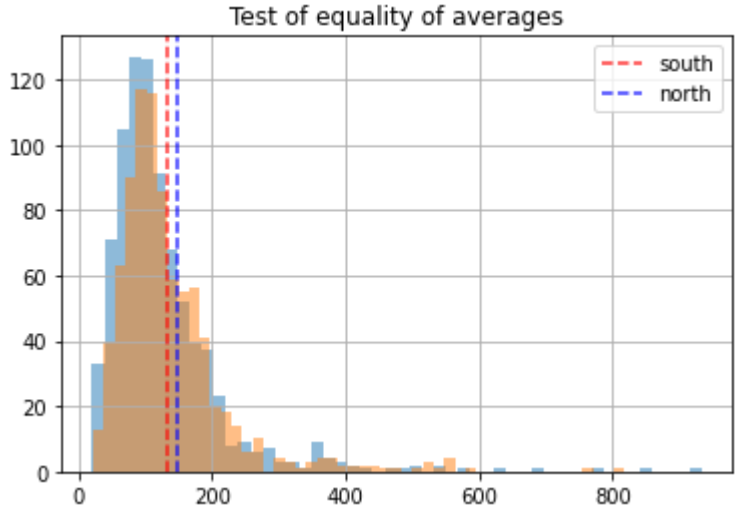

Figure 7.3 – Test the hypothesis of non-equality of averages

Here is also using nonparametric criteria to compare the average values - the Mann-Whitney test.

Let’s define the hypothesis:
H0: the averages of samples with south data is less than averages of samples with north data;
H1: the averages of samples with south data is greater than averages of samples with north data.

p-value = 0.998 - that means that there is no reason to reject the hypothesis. Conclusion: the averages of samples with south data is less than samples with north data. This difference is statistically significant.

7.2 Clusters analysis

Let’s perform a cluster analysis of the data.
The resulting analysis will look at the city's roads, their congestion, depending on the position and direction of movement.
The type of cluster analysis to be performed is K-means.
There is a set of data. The next step is to remove the outliers using a boxplot. Some of these are presented in Figure 7.4 - 7.5.

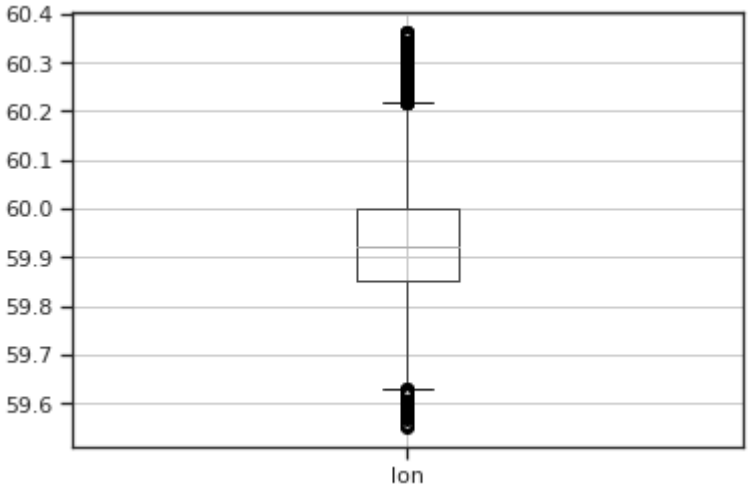

Figure 7.4 – Boxplot for latitude

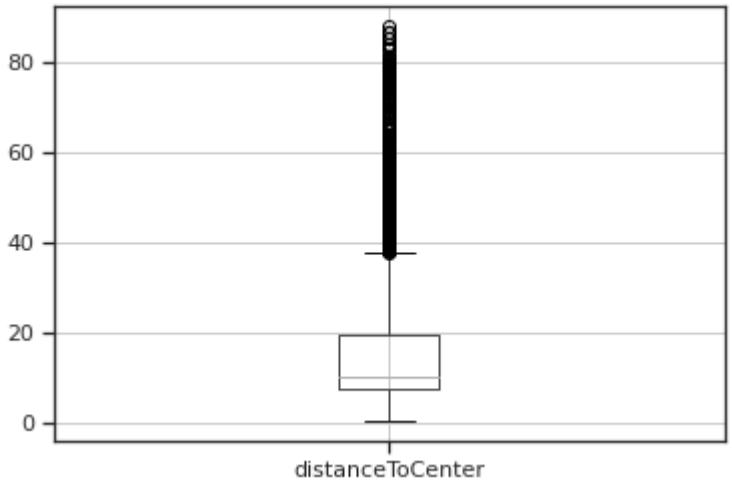

Figure 7.5 – Boxplot for distance to center

After the outliers are removed, the dataset consists of 2997 elements.

In order to determine the optimal number of clusters, it is a good way to use the Elbow method. It is presented in Figure 7.6.

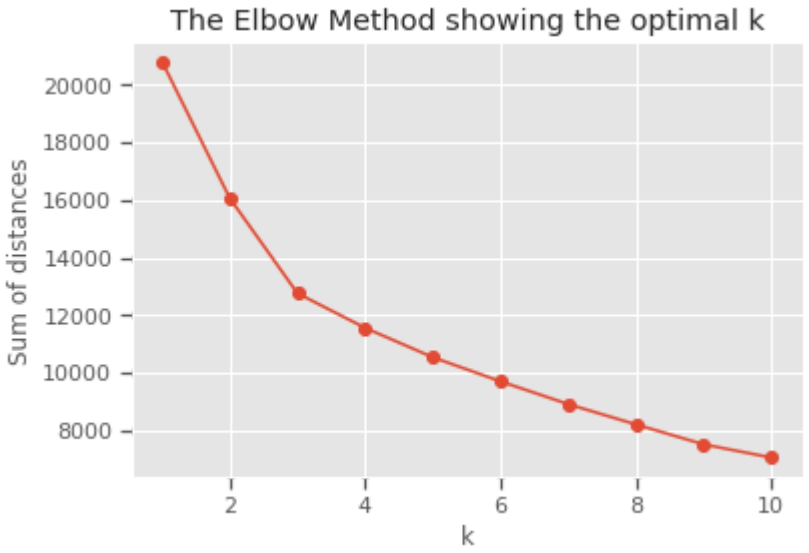

Figure 7.6 – Elbow Method

It is possible to assume that 3 clusters will be the optimal choice.

Let's do an analysis and see the results. It is presented in Table 7.6.

Table 7.6 – Results in division into clusters

cluster	lon	lng	speed	distanceToCenter	inCenter
0	59.9371	30.334109	16.3	8.155217	0.000000
1	59.9129	30.357086	14.2	9.893334	0.554795
2	59.93859	30.332484	15.4	8.409536	1.000000

The numbers of elements in each cluster presented in Table 7.

Table 7.7 – The numbers of elements in each cluster

0	438
1	1318
2	1217

Interpretation of received results:

Mean distance to center for all clusters is approximately equal. Speed in the second cluster is less than in other clusters. That means this means that cluster 2 represents the busiest roads. The first and the third clusters have approximately equal speed, but speed for direction in center a bit less than in direction from center.

The visualization of this analysis is presented in Figure 7.10.

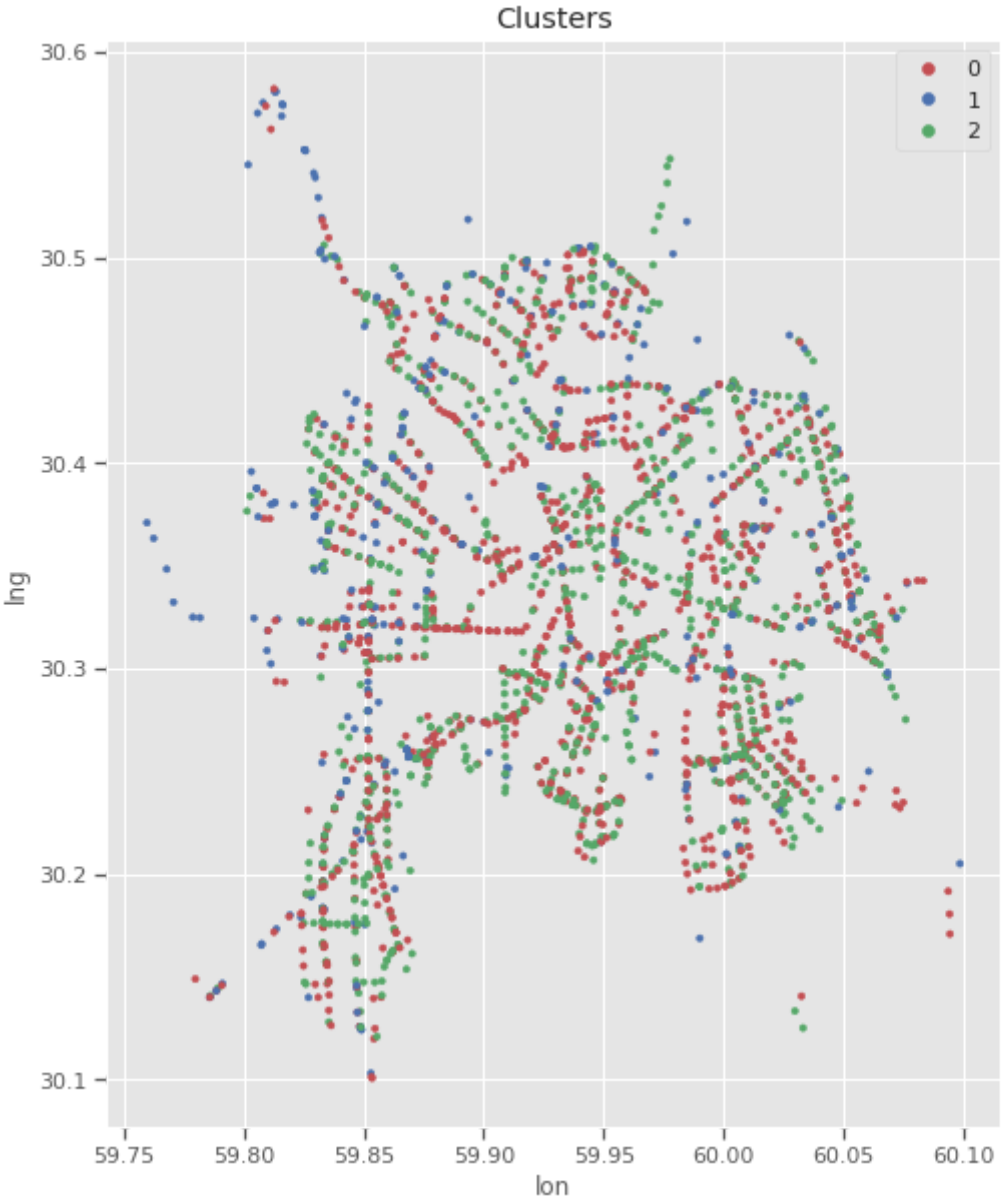

Figure 7.10 – Visualization of clusters analysis

7.3 Prediction of unknown time values in datasets

The task is to find the best method for predicting the time between stops for unknown values.

The Table 7.8 - 7.9 has the results of the implemented method.

Table 7.8 – Results of prediction on the test sample

Model	R^2	MSE
Linear Regression	0.23	3004
Polynomial Regression (degree = 2)	0.24	2987
Polynomial Regression (degree = 3)	0.23	3027
Decision Tree	0.238	2997
Random Forest	0.241	2986
Gradient Boosting	0.25	2955
XGBoost	0.25	2953
CatBoost	0.27	2865

Table 7.9 – Results of prediction on the test sample with neural network

Output activation function	Hidden activation function	Number of hidden layers	Number of neurons	Optimizer	MSE
Linear	RELU	6	57	Adam	2947
Linear	RELU	5	77	Adam	2981
Linear	RELU	4	107	Adam	3029

Let’s consider the results of sume implemented models.

Random Forest

With the number of trees is equal to 6 trees and maximum depth of tree is 5.

Feature importance is presented in Table 7.10.

Table 7.10 – Feature importance for random forest

	feature	importance
0	stop_distance	0.616225
2	lng	0.252665
3	distanceToCenter	0.071989
1	lat	0.045977
4	inCenter	0.013143

The most important feature is stop distance and longitude.

Decision Tree

The maximum depth in the Decision Tree is equal 4.

Feature importance is presented in Table 7.11.

Table 7.11 – Feature importance for Decision Tree

	feature	importance
0	stop_distance	0.478116
1	lat	0.227372
2	lng	0.204549
3	distanceToCenter	0.087620
4	inCenter	0.002344

The most important feature is stop distance, longitude and latitude.

Linear Regression

Feature importance is presented in Table 7.12.

Table 7.13 – Feature importance getting from CatBoost model

features	importance
stop_distance	53.471282
lng	15.785745
latitude	14.886548
distanceToCenter	11.292725
inCenter	4.563699

The most important feature is stop_distance, longitude and latitude

Most models showed the greatest influence of the following features: stop_distance, latitude and longitude.

The best result has a model CatBoost.

8 IMPLEMENTATION GEOGRAPHIC DATA STORING TREE

In this work, it's constantly required to find a location in a radius of several meters. Dataset, that describes the operation of public transport has 23126 ones.

In this work was implemented two methods to search stops in particular radius.

Brute-force method consists in calculating distance between current point and all points from the dataset and keeping only those that fit the parameters. For example, those that calculated distances are less than particular values.

This method has complexity O(n) in all cases and space O(n).

For best effectiveness of searching for this operation some storage systems are required.

After researching this area was made in favor of the k-d tree method. (In our case it is a 2-d tree). The k-d tree sorts them into two halves (around the midpoint) - either left and right, or top and bottom, alternating x and y separated to every level. More detailed description of this algorithm was presented earlier in the theoretical part of this work. Complexity of search is O(n) in the worst case and O(log(n)) in average.

In work was implemented two algorithms (brute-force and k-d tree).

Comparing those is in Table 8.1.

Comparing is held on the current dataset of describing public transport (size = 23126 ones).

Table 8.1 – Comparable time of work of two algorithms depends from distance

Distance	Brute-force	K-d tree
0.8 km	83 ms	71 ms
1.8 km	86 ms	75 ms
0.3 km	84 ms	72 ms
3 km	87 ms	73 ms

Results of experiment: stable less running time on algorithms for k-d tree methods then brute-force. Time of work is approximately equal for all distances.

Comparing is held on the current dataset of describing public transport in Table 8.2 (distance = 23126 ones).

Table 8.2 – Comparable time of work of two algorithms depends from distance

Size of graph	Brute-force	K-d tree
23126	81	72
13000	54	45
5000	34	27

Results of experiment: Stable less time of work on algorithms for k-d tree methods then brute-force. Time of working depends on the size of the dataset.

9 BUILDING A WALKING ROUTE AND ROUTE BY TAXI

9.1 Parsing data from Overpass API

The Overpass API is a read-only API that gives the user the necessary data from OSM maps. In its operation, it is similar to the principles of operation of a database over the Internet: custom sends a query to the API and gets back a datasets with information according to the sent query. The required data can be information about location of city sightseeing, about roads, about cafes and other types of places to eat, about cycle networks, about mountains and other geographic objects. Overpass API has a query language with a lot of features.

For beautiful data visualization it is convenient to use the programm QGIS Desktop.

QGIS is a free and open-source platform that allows visualizing and analyzing geospatial data. QGIS can load geospatial data also in GeoJSON format, that is very comfortable when main analyzing this data is happening in other programs.

Example of query for receiving roads for Vasileodtrovskii district:

```
[out:json];
area[name="Санкт-Петербург"]->.b;
rel(area.b)[name="Василеостровский район"];
map_to_area -> .a;
way[highway~"^(motorway|trunk|primary|secondary|tertiary|uncl
assified|living_street|pedestrian|bus_guideway|residential)$"](are
a.a);
out geom;
relation[highway~"^(motorway|trunk|primary|secondary|tertiary
|unclassified|living_street|pedestrian|bus_guideway|residential)$"
](area.a);
out geom;
```

Some results of work with Overpass API and QGIS Desktop image in Figure 9.1. Here is shown all museums in St. Petersburg. Data was received from server Overpass API and visualized with QGIS.

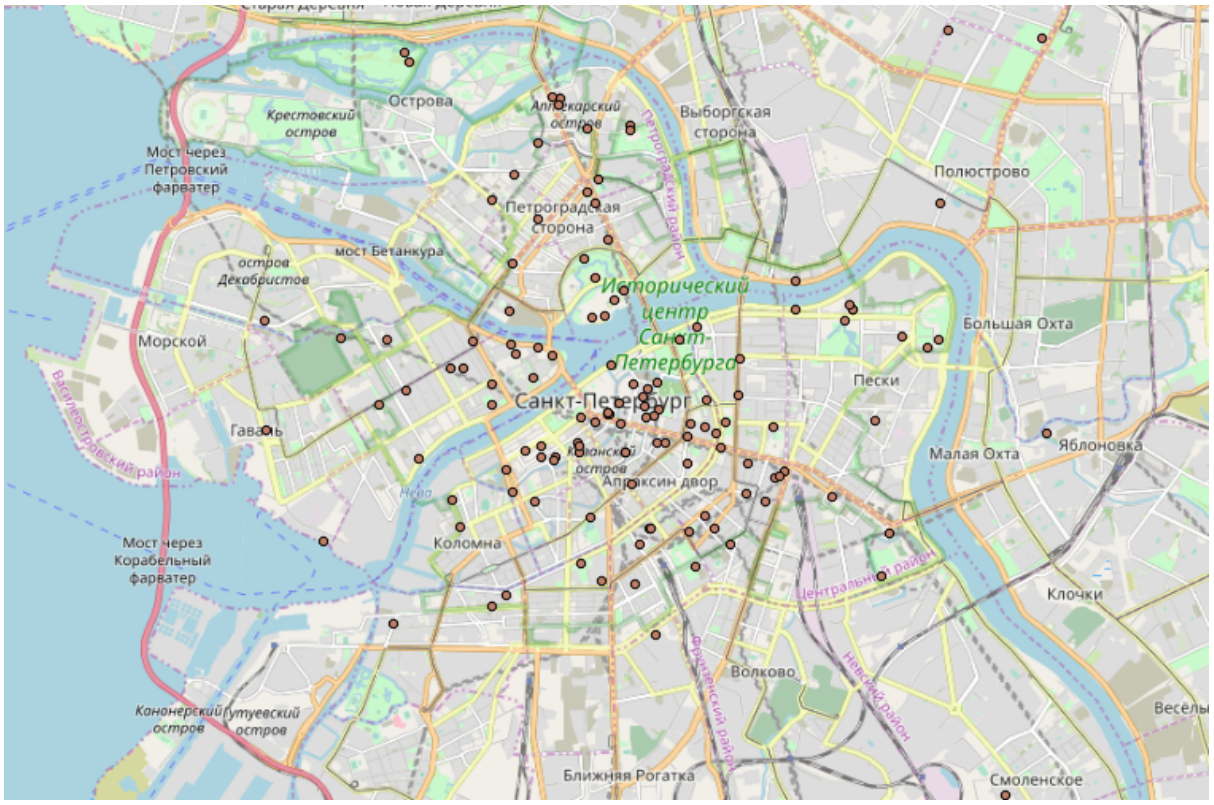

Figure 9.1 – Museums in Saint Petersburg, Russia

The following Figure 9.2 also shows the roads of a part of St. Petersburg obtained using the Overpass API. And using this data is built route by foot.

Figure 9.2 – The roads of a part of St. Petersburg received from Overpass API

9.2 Building a route by foot

For building route by foot it is necessary to receive data about all roads from server Overpass API. The most comfortable way is to download data to files for each city district.

Data has coordinates in field ‘geometry’. All these received coordinates of point can be presented as a non-oriented graph with weights as length of path between nodes.

This produced graph allows to build the shortest routes between points by foot using the Dijkstra algorithm.

In the work it is necessary to build routes between attractions. But coordinates of attraction do nor refer to the coordinates of roads. Therefore, it is required to find the nearest points from the graph to starting and to finishing places and build the edges from attractions to these points also. So, in this way it is possible to build a route by foot between attractions. The example of the path is presented in Figure 21.

9.3 Building a route by taxi

Building a route with a taxi also requires building a non-oriented graph. Graph is constructed from the coordinates of stops of public transport. Also when building a route it is also necessary to take into account traffic-jams.

Since it is having information about the time between stops (taking into account traffic jams). It is possible to calculate the coefficients of how much longer the transport goes compared to normal driving on the road (at a speed of about 50 km/h). And multiply the length of the path between the stops by this coefficient. Thus, the path between stops is lengthened according to the actual length of the path. On the resulting graph is built the shortest route using Dijkstra's algorithm.

So, in this way it is built the shortest path between attractions by taxi considering traffic-jams.

CONCLUSION

As a result of the work the goals were executed. Several variants of routes were built: by foot, by public transport, by taxi. Public transport includes buses, trams and metro. The preferable variants of public transport are buses and trams because its achieve the target point with more precision. Also before building algorithms were analyzed existing articles about building routes by public transport. In the work with public transport were developed algorithms for direct routes, routes with one transfer, routes with two transfers. The algorithms with more numbers of transfer are not considered because their realization in practice is difficult and expensive. And the largest number of city attractions are in easy transport accessibility, because they are in great demand. The algorithms use a list with all transport, where the sequence of stops is written for each transport number. This type of data representation is sufficiently comfortable. There are some variations in building tourist routes. Depending on the preferability of a particular tourist the durability of the pedestrian way can change. Tourists can choose this coefficient and depending on it will be building the route with a particular coefficient of comfort for this tourist. In the work also is presented variations of route with opportunity to change stop of transfer (at an acceptable distance between these stops). The algorithms are tested on complex cases: route from South to North, from “island” to “continent” and from northwest to northeast. These cases are difficult because usually the direction from public transport is coming to the center and back. All routes are built with traffic and timetable for a particular 10-minutes interval. Results of work of the developed algorithm were compared with existing algorithms and it is concluded that the developed algorithm gives good results and has the right to exist.

Routes were built by taxi and on foot using the server Overpass API, that allows downloading roads.

In the work were analyzed the existing modifications of the Orienteering Problem. An example route was constructed using the Orientation Problem, taking into account different modes of transport in real time. And the results of building were obtained.

In the course of the work, all the tasks were completed.

REFERENCES

1. Pietz J, Royset JO. Generalized orienteering problem with resource dependent rewards. Naval Research Logistics. 2013; 60(4):294–312.
2. Yi Mei, Flora Dilys Salim, Xiaodong Li. Efficient Meta-heuristics for the Multi-Objective Time-Dependent Orienteering Problem. European Journal of Operational Research. 2016; 254(2): 443 - 457.
3. Vansteenwegen P, Souffriau W, Oudheusden DV. The orienteering problem: A survey. European Journal of Operational Research. 2011;209(1):1–10.
4. Güneş Erdoğan, Jean-François Cordeau, Gilbert Laporte. The Attractive Traveling Salesman Problem. European Journal of Operational Research. 2010;203(1):59-69.
5. Gunawan A, Lau HC, Vansteenwegen P. Orienteering Problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research. 2016;255(2):315–332.
6. Ksenia D. Mukhina, Alexander A. Visheratin, Denis Nasonov Orienteering Problem with Functional Profits for multi-source dynamic path construction. Plos One. 2019; 14(4): e0213777.
7. Rosenwein M, Kantor G, Moshe B. The Orienteering Problem with Time Windows. Journal of the Operational Research Society. 1992;43(6):629–635.
8. Fomin FV, Lingas A. Approximation algorithms for time-dependent orienteering. Information Processing Letters. 2002;83(2):57–62.
9. Gendreau M, Laporte G, Semet F. A branch-and-cut algorithm for the undirected selective traveling salesman problem. Networks. 1998;32(4):263–273.
10. Schilde M, Doerner KF, Hartl RF, Kiechle G. Metaheuristics for the bi-objective orienteering problem. Swarm Intelligence. 2009;3(3):179–201.
11. Güneş Erdoğan, Gilbert Laporte. The orienteering problem with variable profits. Networks.2013;61(2):104-116.
12. Masoud Shahmanzari, Deniz Aksen. Multi-Period Travelling Politician Problem: A Hybrid Metaheuristic Solution Method. Operation Research Proceedings. 2020; 21:1 - 10.
13. Archetti, Claudia & Carrabs, Francesco & Cerulli, Raffaele. The Set Orienteering Problem,European Journal of Operational Research. Elsevier. 2018;267(1):264-272.
14. Camelia-M. Pintea, Petrică C. Pop & Camelia Chira. The generalized traveling salesman problem solved with ant algorithm. Complex Adaptive Systems Modeling. 2017; 8.
15. Fang SH, Lu EHC, Tseng VS. Trip recommendation with multiple user constraints by integrating point-of-interests and travel packages. In: Proceedings—IEEE International Conference on Mobile Data Management. vol. 1. IEEE; 2014. p. 33–42.

16. Pietz J, Royset JO. Generalized orienteering problem with resource dependent rewards. Naval Research Logistics. 2013;60(4):294–312.

17. Yu VF, Jewpanya P, Lin SW, Redi AANANP. Team orienteering problem with time windows and time-dependent scores. Computers and Industrial Engineering. 2019;127:213–224.

18. Freeman NK, Keskin BB, Capar I. Attractive orienteering problem with proximity and timing interactions. European Journal of Operational Research. 2018;266(1):354–370.

19. Yao B, Yan Q, Zhang M, Yang Y. Improved artificial bee colony algorithm for vehicle routing problem with time windows. PLOS ONE. 2017;12(9):1–18.

20. Fomin FV, Lingas A. Approximation algorithms for time-dependent orienteering. Information Processing Letters. 2002;83(2):57–62.

21. Gavalas D, Konstantopoulos C, Mastakas K, Pantziou G. A survey on algorithmic approaches for solving tourist trip design problems. Journal of Heuristics. 2014; 20(3):291–328.

22. Yu VF, Jewpanya P, Lin SW, Redi AANANP. Team orienteering problem with time windows and time dependent scores. Computers and Industrial Engineering. 2019; 127:213–224.

23. ImanRoozbeh, John W.Hearne, DelaramPahlevani. A solution approach to the orienteering problem with time windows and synchronisation constraints. Heliyon.2020;6(6): e04202.

24. Faigl J, Pěnička R, Best G. Self-organizing map-based solution for the orienteering problem with neighborhoods. In: Proceedings of the IEEE international conference on systems, man, and cybernetics. 2016 : 1315–1321.

25. Sascha Witt. Trip-Based Public Transit Routing. Lecture Notes in Computer Science 9294; 1025-1036, 2015.

26. Sascha Witt.Trip-Based Public Transit Routing Using Condensed Search Trees. KIT, 2016.

27. Julian Dibbelt, Thomas Pajor, Ben Strasser, Dorothea Wagner. Connection Scan Algorithm. ACM Journal of Experimental Algorithmics; Article No.: 1.7, 2018

28. Ben Strasser and Dorothea Wagner. Connection scan accelerated. In Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14), pages 125–137. SIAM, 2014.

29. Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-Based Public TransitRouting. In Algorithm Engineering and Experiments (ALENEX), pages 130–140, 2012.

30. Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13), volume 7933 of Lecture Notes in Computer Science, pages 43–54. Springer, 2013.

31. Hannah Bast, Sabine Storandt. Frequency-based search for public transit. In ACMSIGSPATIAL International Conference on Advances in Geographic Information Systems,pages 13–22. ACM Press, November 2014.

32. Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato F. Werneck. Public TransitLabeling. In Experimental Algorithms, volume 9125 of Lecture Notes in Computer Science (LNCS), pages 273–285. Springer, 2015.

33. Robert Geisberger. Contraction of Timetable Networks with Realistic Transfers. InExperimental Algorithms, volume 6049 of Lecture Notes in Computer Science (LNCS),pages 71–82. Springer, Heidelberg, 2010.

34. B. Rama. An Efficient Smart Search Using R Tree on Spatial Data. Journal of Advanced Research in Dynamical and Control Systems 2017; 9(11):226.

35. Dr. Mohammed Otair. Approximate K-Nearest Neighbour Based Spatial Clustering Using K-D Tree. International Journal of Database Management Systems 2013; 5(1)

36. GTFS data for Saint - Petersburg: http://transport.orgp.spb.ru/Portal/transport/internalapi/gtfs/feed.zip

37. Data about public transport https://data.gov.spb.ru/opendata/7830001067-routes_transport/

38. Mukhina, Ksenia; Visheratin, Alexander. Replication Data for: Orienteering Problem with Functional Profits for multisource dynamic path construction https://doi.org/10.7910/DVN/KCAIXS, Harvard Dataverse, V1

39. Data about position of public transport in real time: http://transport.orgp.spb.ru/Portal/transport/internalapi/gtfs/realtime/vehicle

40. Ben Strasser. Dynamic Time-Dependent Routing in Road Networks Through Sampling, 2017, KIT.

41. Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Muller-Hannemann4, Thomas Pajor,Peter Sanders, Dorothea Wagner and Renato F. Werneck Route Planning in Transportation Networks, 2015.

42. Framework for solving Orienteering Problem https://github.com/mukhinaks/fops