Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій

Кафедра оптимального керування і економічної кібернетики

Кваліфікаційна робота

на здобуття ступеня вищої освіти «магістр»

«**Оптимізація маршрутів з урахуванням значущості проміжних пунктів**»

«**Optimization of routes, considering the significance of intermediate points**»

Виконав здобувач денної форми навчання
спеціальності 113 Прикладна математика
Освітня програма Прикладна математика
Веремйов Кирил Валерійович

Керівник
Канд.техн. наук, доц. Мазурок І.Є.

Рецензент
Канд. техн. наук, доц. Мороз В.В.

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від _______________ р.

Завідувач кафедри

_________ _____________________
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК №

протокол № ____ від _____________ р.

Оцінка______________/___ ___/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

___________ _________________________
(підпис) (прізвище, ініціали)

Одеса 2023

TABLE OF CONTENTS

INTRODUCTION 2
ВСТУП 4
TASKING 6
SECTION 1. Approaches to solving the routing problem 7
1.1. Inspirock service 7
1.2. Sight Safari 8
1.3. Mapy.cz 10
1.4. Conclusion 11
SECTION 2. Mathematical model and methods for solving the problem 12
2.1. The case of a directed route with points to avoid 12
2.2. The case of a directed route with points to be visited 16
2.3. The case of a circular route 18
SECTION 3. Software tools and implementation details 20
SECTION 4. The testing of the developed software 23
4.1. The case of a directed route with points to be avoided. 23
4.2. The case of a directed route with points to be visited 24
4.2.1. Test C++ version examples 24
4.2.2. City examples 25
4.3. The case of circular route with points to be avoided 29
4.4. The case of circular route with points to be visited 30
CONCLUSIONS 33
ВИСНОВКИ 36
LITERATURE 40
ADDITION A 43
ADDITION B 47

INTRODUCTION

Today there are many cases when people need to construct a route taking into account not only the starting and destination points, but also the intermediate ones. This topic is quite relevant nowadays and there are many examples, given how often people need to go somewhere, visit or deliver something. For example, if a ship is sailing in a reef zone, it is important not only to lay out a route to the port, but also that it goes in the safe zone. That is, priority is given not to the shortest, but to the safest route, to avoid reef areas. Because the priority is safety, not distance. Or, if a tourist is in an unfamiliar city, he may be interested in visiting some attractions or seeing something. That is, he may prefer not the fastest route, but a more interesting one, which will allow him to get acquainted with local attractions and other interesting places. Thus, he can make a choice in favor of attractiveness rather than travel time, and it is important for him not just to arrive at the destination, but what the intermediate places are.

Therefore, the topic of this paper is the development of a solution that can allow to take into account significance of intermediate points (in the examples above, reefs or attractions) when constructing a route. We can consider points with different natures of significance in relation to other route criteria, not just the same ones. In previous examples — safety and distance, attractiveness and time or, again, distance.

Thus, in this article we will consider several cases.

Firstly, as already mentioned, if you need to come somewhere, but you have free time and are interested in the surrounding area. For example, you are a tourist in a new city or you left home a little earlier and would like to take a walk. So, this is a case of a directed route, when you are interested in arriving at some interesting points.

Secondly, the case of the ship. Here, too, you need to balance two criteria: safety — so as not to go out on the reef and the distance of the route — to finally arrive at the port. That is, this is also a directed route, but the difference is that in this case some points need to be avoided rather than trying to visit them.

Thirdly, case of a route around the surrounding area. E.g. you have some free time while waiting for a train at the station or a plane at the airport, and you are wondering

what you can see in the surrounding area. So this is a case of a circular route with points to be visited.

Fourthly, if the ship needs to go to several places, again in some dangerous (reef) zone. We can also think of this as a case of a circular route, but some points should be avoided here.

Thus, the object of study is the route and its intermediate points. The subject of the study is the method of constructing routes taking into account the significance of these points. Including taking into account more than one of their parameters (possible route criteria), such as interestingness, attractiveness, travel time, distance, safety and so on. As research methods were used some mathematical ideas and algorithms described below and implemented in computer programs.

For this problem we didn’t find a lot of well-known solutions (during research were found just few and with different approach). Thus, the scientific value of this article is that it may suggest a different approach to the problem, which in some cases may be a little more effective, simple and versatile.

As a practical value, we can note the versatility of the developed model, which can allow its use in various areas where we are talking about the significance of intermediate points and, possibly, more than one route criterion. Such as tourism, logistics, etc. and applications for them.

ВСТУП

Сьогодні чимало випадків, коли людям необхідно побудувати маршрут з урахуванням не лише пунктів відправлення та призначення, а й проміжних. Ця тема є досить актуальною в наш час і прикладів тому багато, враховуючи, як часто людям потрібно кудись їхати, відвідати чи щось доставити. Наприклад, якщо судно йде у рифовій зоні, важливо не лише прокласти маршрут до порту, а й щоб воно йшло у безпечній зоні.

Тобто пріоритет віддається не найкоротшому, а найбезпечнішому маршруту, щоб уникнути рифових зон. Тому що пріоритетом є безпека, а не відстань. Або якщо турист знаходиться в незнайомому місті, йому може бути цікаво відвідати якісь пам'ятки або щось подивитися. Тобто він може віддати перевагу не найшвидшому маршруту, а більш цікавому, який дозволить йому познайомитися з місцевими пам'ятками та іншими цікавими місцями. Таким чином, він може зробити вибір на користь привабливості, а не часу шляху, і йому важливо не просто прибути до пункту призначення, а також важливі й проміжні місця.

Тому темою цієї статті є розробка рішення, яке дозволить враховувати значущість проміжних точок (у прикладах вище – рифів або визначних пам'яток) під час побудови маршруту. Ми можемо розглядати точки з різною природою значущості по відношенню до інших критеріїв маршруту, а не лише тої самої. У попередніх прикладах – безпека та відстань, привабливість у часі або знову ж таки відстань.

Отже, у цій статті розглядається кілька випадків.

По-перше, як вже говорилося, якщо потрібно кудись приїхати, але є вільний час і ви цікавитеся околицями. Наприклад, ви турист у новому місті або вийшли з дому трохи раніше та хотіли б прогулятися. Отже, це випадок спрямованого маршруту, коли ви зацікавлені в тому, щоб дістатися якихось цікавих точок.

По-друге, випадок із кораблем. Тут теж потрібно збалансувати два критерії: безпека – щоб не вийти на риф та дальність маршруту – щоб нарешті прийти до

порту. Тобто це також спрямований маршрут, але різниця в тому, що в цьому випадку деяких точок потрібно уникати, а не намагатися їх відвідати.

По-третє, випадок маршруту околицями. Наприклад. у вас є трохи вільного часу в очікуванні поїзда на вокзалі або літака в аеропорту, і вам цікаво, що можна подивитися в околицях. Отже, це нагода для кругового маршруту з точками, які потрібно відвідати.

По-четверте, якщо кораблеві треба зайти в кілька місць, знову ж таки через якусь небезпечну (рифову) зону. Ми також можемо думати про це як про круговий маршрут, але тут слід уникати, а не відвідувати деякі точки.

Таким чином, об'єктом дослідження є маршрут та його проміжні точки. Предметом дослідження є спосіб побудови маршрутів з урахуванням значимості цих точок. У тому числі з урахуванням більш ніж одного з параметрів (можливих критеріїв маршруту), таких як цікавість, привабливість, час у дорозі, відстань, безпека і так далі. Як методи дослідження використовувалися деякі математичні ідеї та алгоритми, описані нижче і реалізовані в комп'ютерних програмах.

Для цього завдання ми знайшли не так вже й багато відомих рішень (в ході дослідження їх було знайдено всього кілька і з різним підходом). Таким чином, наукова цінність цієї статті полягає в тому, що вона може запропонувати інший підхід до цього питання, який у ряді випадків може виявитися більш ефективним, простим і універсальним.

Як практичну цінність можна відзначити універсальність розробленої моделі, що може дозволити використовувати її в різних галузях, де мова йдеться про значущість проміжних точок і, можливо, більш ніж одного критерію маршруту. Такі як туризм, логістика тощо та застосунки для них.

TASKING

The main goal of this work is to propose a solution for constructing routes that consider intermediate points and their cost, impact on neighboring points and routes, or other properties.

The algorithm should

- Take into account the properties of these points, the value of the attributes.
- Work both with values whose contribution needs to be maximized and with those that need to be minimized.
- Have polynomial indicators of algorithmic complexity and work quickly enough.
- Be versatile enough to be able to work together with other algorithms. This work considered a combination with algorithms for constructing two types of routes that should take into account intermediate points:
  - Directed. From the starting location to the ending point.
  - Circular. Or unlooped, in some neighborhood

The results must be clearly displayed.

SECTION 1. Approaches to solving the routing problem

In this section, we provide an overview of existing theoretical and practical solutions on the topic of the work, criteria for selecting methods for use in the work, and a comparative analysis of the solutions considered according to the relevant criteria.

1.1. Inspirock service

One of the services that we found closest to our topic was the Inspirock [1] service. It provided several options for places to visit over time.

Figure 1.1.1. Presentation of Inspirock results as a map and plan

It offers a detailed plan for visiting places in the form of a timeline, map (Figure 1.2.1.) or a calendar (Figure 1.2.2.). This app also allows you to plan trips for several days or even weeks. But the map, as you can see, gives an approximate route, because the algorithm selects points and plans a timeline, rather than drawing a route point by point. Which may not be very convenient at first, especially for short trips.

Hence there is a criterion: a convenient and detailed visual representation.

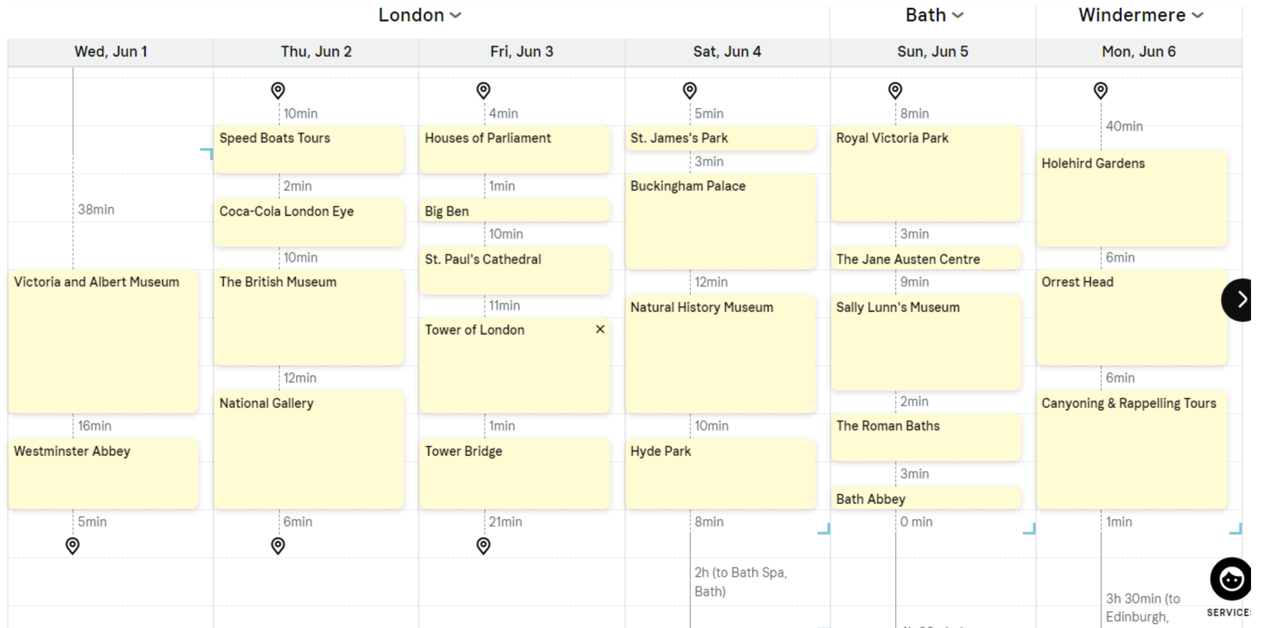

Figure 1.1.2. Presentation of Inspirock results as a calendar

Also, it may be too less informative for other cases where a detailed route is important (for example, the case of a ship). Also, the question may again be about the universality of the algorithm.

1.2. Sight Safari

Another analogue with a different approach was “Site Safari” by Egor Smirnov[2].

Attention is also paid to intermediate points of the tourist route. It also allows you to select categories of points of interest and gives you the ability to build both circular and directional routes. The idea of a directed route based on visibility polygons and label analysis (example in Figure 1.2.1.). The idea of a circular route is based on a genetic algorithm [3] (example in Figure 1.2.2.).

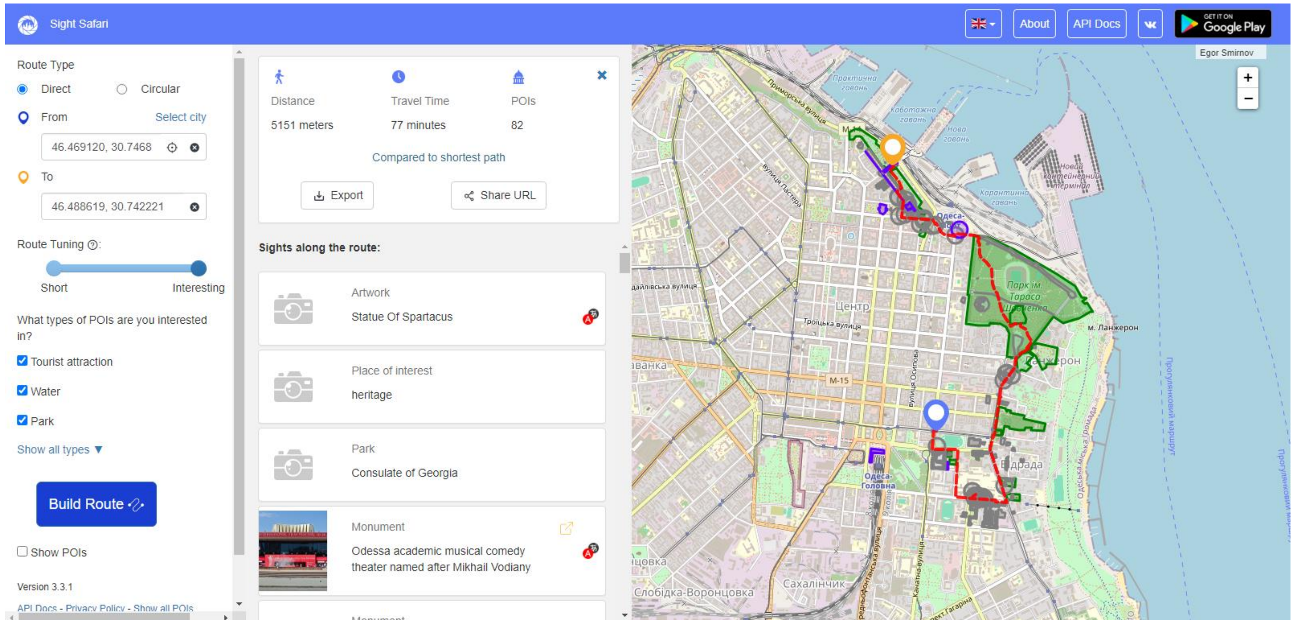

Figure 1.2.1. Site Safari directed route results

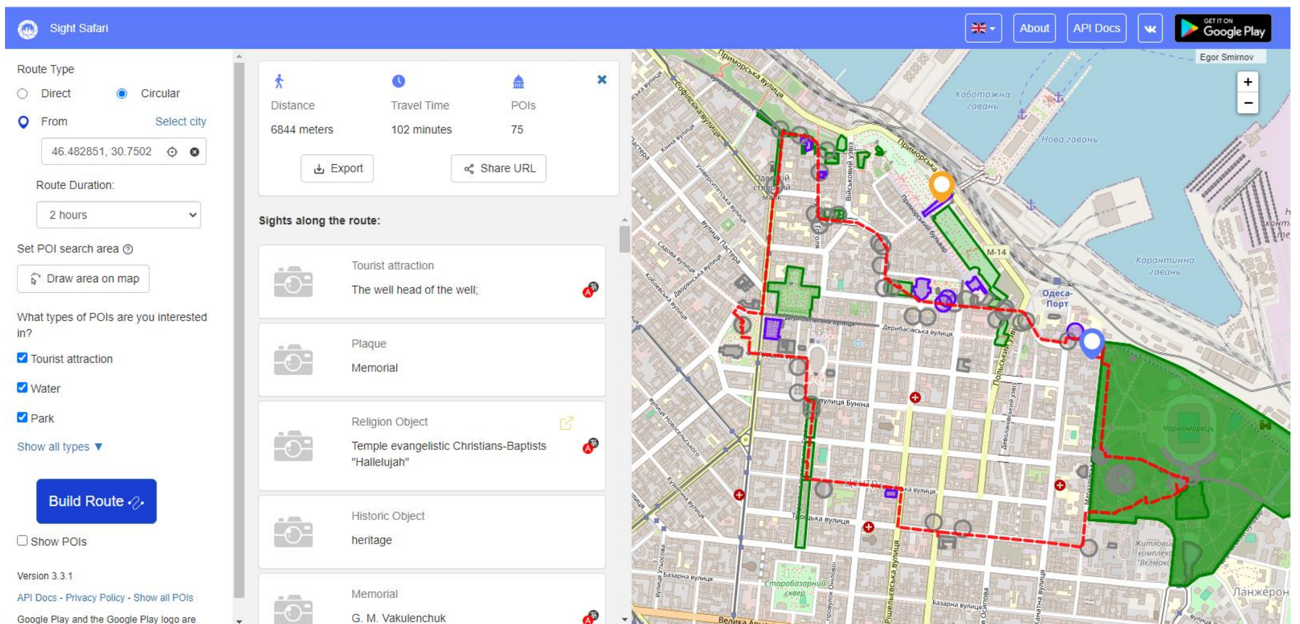

Figure 1.2.2. Site Safari circular route results

But the idea of the application was just for tourist routes. It has not been tested for other areas of mathematical problems (or we did not find information).

Hence another criterion: the universality of the algorithm.

Also, the route construction time is quite long: up to 10 seconds. This may be due to the peculiarities of the algorithm used in the application.

Hence, the following criteria may be used: the computational complexity of the algorithm and the computation time.

1.3. Mapy.cz

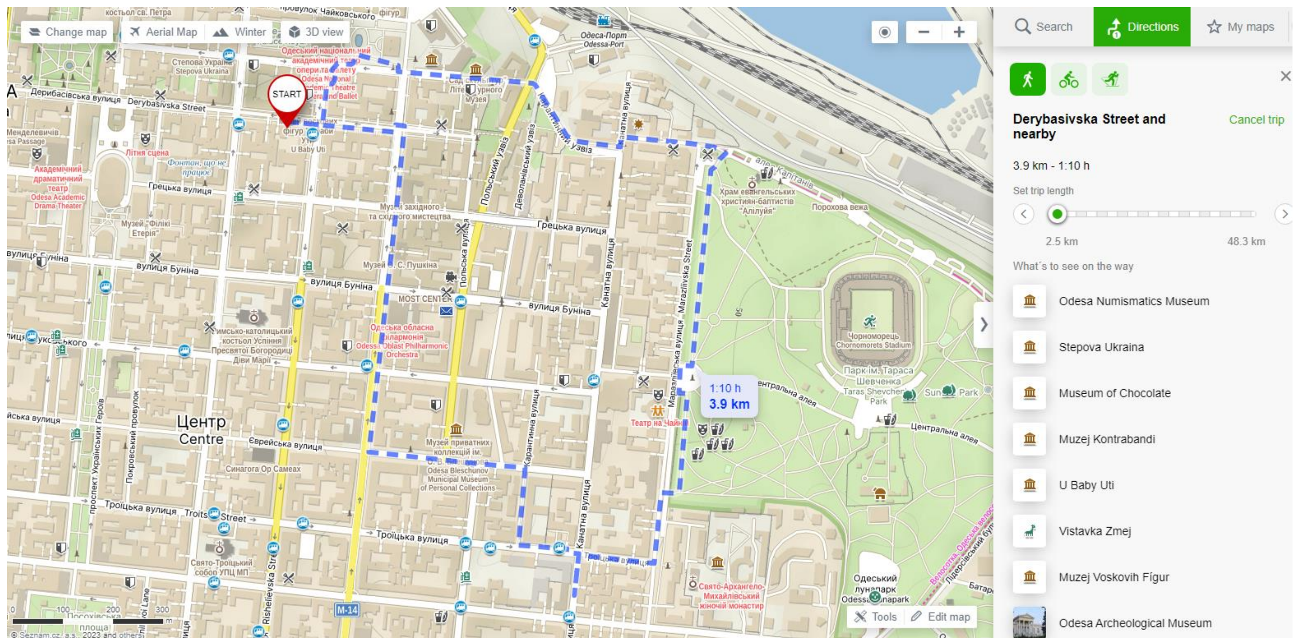

Figure 1.3.1. Mapy.cz circular route results

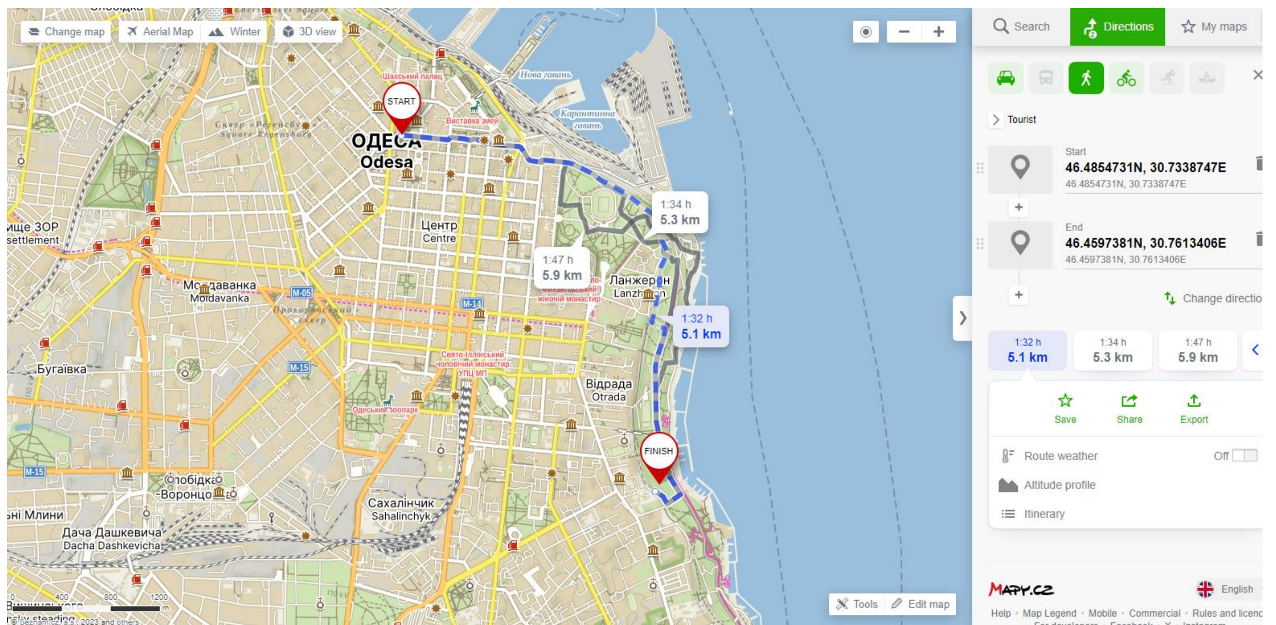

Figure 1.3.2. Mapy.cz directed route results

The travel service Mapy.cz [4] can provide both circular (Figure 1.3.1.) and directional (Figure 1.3.2.) routes with attention to intermediate points of interest (tourist mode). But the question again is the universality of its algorithm (we did not find information about what methods they use for construction). In this functionality it is very similar to Sight Safari, but much faster.

1.4. Conclusion

The comparison results are presented in the form of table 1.4.1.:

	Time of working	**Convenient and detailed presentation**	**Versatility**
Site Safari	Poor	Good	Poor
Inspirock	Good	Poor	Unknown
Mapy.cz	Good	Good	Unknown

Table 1.4.1. Comparison of analogues according to selected criteria

We can see that most of examined services and algorithms have their pros and cons, but some of them or are not too much versatility, or there is no information about it. So, the versatility is one of the main criteria of the developed solution, but the others are quite important too.

Additionally, we found a very similar idea about a potential field in the book Artificial Intelligence: A Modern Approach [5]. But there was a modification precisely in the context of A*, so the question again is its universality. This article presents a very similar idea, but it is not considered only for one algorithm. It is also considered as a possible pre-processing.

SECTION 2. Mathematical model and methods for solving the problem

In the article we will look at several use cases and what algorithm idea can be used and how to adapt it for them.

2.1. The case of a directed route with points to avoid

This is the case when you need to go from one point to another, but some points are undesirable. An example would be a ship and reefs. Where you need to balance between travel distance to reach your destination and safety to avoid dangerous points. In the case of a ship, of course, safety criteria may be more important, but in other examples they may be equal or in a different ratio.

We can model the area where the route should be built as a graph. Nodes can be places where a moving object can be, and edges can be paths between them. The desired path can be understood as a sequence of edges that can lead from the starting point (node) to the ending point.

So, the task can be formulated as multi-objective optimization problem: find a sequence of nodes $\{v_i\}$, such as

$$\begin{cases} \sum_{i=1}^{N-1} distance(v_i, v_{i+1}) \to min \\ \sum_{i=1}^{N} D(v_i) \to min \end{cases}$$

Where N is the number of route nodes, $D(v)$ is the vertex danger function, which can return the vertex danger value. Or it could be any other property that should be minimized. We can also assign a property not for nodes, but as another attribute for edges and reformulate in terms of edges:

$$\begin{cases} \sum_{i=1}^{E} distance(e_i) \to min \\ \sum_{i=1}^{E} D(e_i) \to min \end{cases}$$

Where e_i are edges and E is number of edges of a path.

To solve this problem, we can use some well-known algorithms with slight modifications. For the purposes of previous use case, we can take A*[6]. A * at each step selects the node with the minimum value

$$f(n) = g(n) + h(n)$$

where n is the next node on the path, $g(n)$ is the cost of the path from the starting node to n, and $h(n)$ is a heuristic that estimates the cost of the cheapest path from n to the destination. Thus, the classical algorithm tries to minimize the path length.

We also want to minimize another value (danger in the example) so that we can simply add that value to the function. We can also provide coefficients k_g, k_d, k_h to make the model more flexible. This way we can get

$$f(n) = k_g g(n) + k_d D(n) + k_h h(n)$$

But in this case, the path may be very close to a dangerous node, which may be unsafe. For example, in case of stormy weather at sea, it is better to avoid reefs and other dangerous areas at some distance. We can try to fix this by giving some information that there is some danger point near the node, or by modeling a danger zone around the point. The closer a node is to the dangerous point, the higher its dangerous value. And at the point itself the value should be much higher — in order to prevent visiting a dangerous point (namely a reef), even if a moving object (ship) is located in its vicinity.

To do this, we can use a modification of the two-dimensional Gaussian function[7].

$$G(n, p, \sigma) = D(x, y)e^{-\frac{(p_1 - x)^2 + (p_2 - y)^2}{2\sigma^2}}$$

It can definitely give the highest result when node $n = (x, y)$ is exactly the dangerous point (node) $p = (p_1, p_2)$ and the lower, the further (x, y) from (p_1, p_2). By adding this function with a large value of σ to $f(n)$, we can describe how far a node is from the danger point. That is, to simulate a possible dangerous zone near the point. For example, for stormy weather this zone may be larger than for calm weather. And

by adding another Gaussian with small σ, we can describe the dangerous node and the real dangerous zone near it. These too can have weights to manipulate how much weight each term should have. For a one-dimensional function, Figures 2.1.1, 2.1.2, 2.1.3 can serve as illustrations.

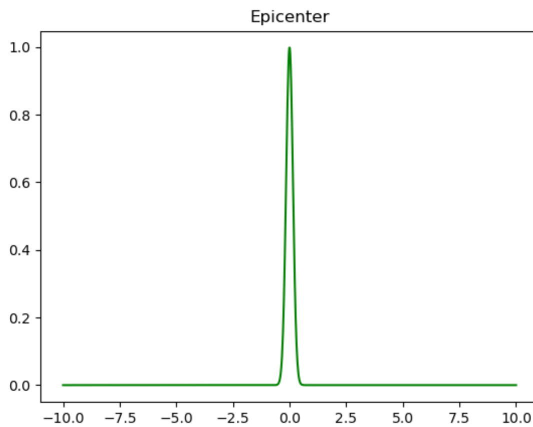

Figure 2.1.1. Epicenter modeling

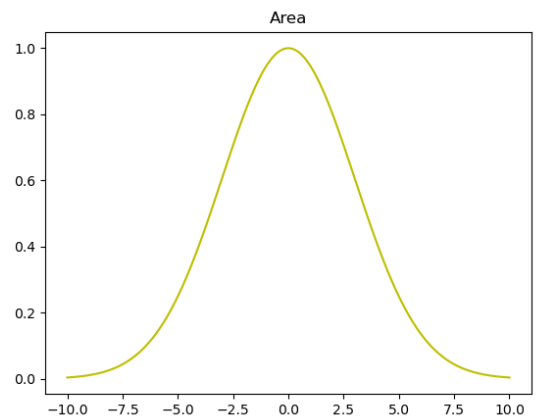

Figure 2.1.2. Neighborhood modeling

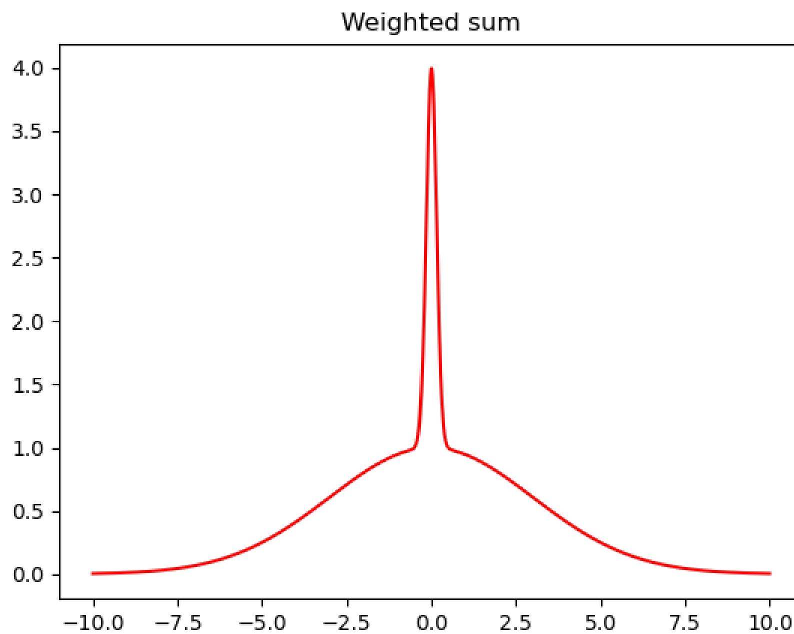

Figure 2.1.3. Weighted sum result

But this is only for one node. Moreover, a node can be influenced not by one point nearby, but by many of them. Thus, we can model the total danger point function for a node as the sum of the weighted sum of two two-dimensional Gaussian functions over all danger points near it.

$$TD(n) = \sum_{p \in P} k_l G(n, p, sigma_{local}) + k_w G(n, p, sigma_{wide})$$

Where P is a set of dangerous points near a node (which can affect it), and k_l, k_w are coefficients for local and wide Gaussian functions.

This way the final version of the utility function can be:

$$f(n) = k_g g(n, origin) + k_h h(n, destination) + TD(n)$$

At each step of the algorithm, the node with the minimum function $f(n)$ value will be considered as the probable next one. Thus, the algorithm tries to obtain the minimum value of $f(n)$ for the end node that is equal to the value of the entire path.

We can simplify the notation and use the A* standard if we add $TD(n)$ using weights to the other node values before performing the calculations. For edges, for example, we can combine $TD(n)$ with the edge length before the computation gets the new resulting edge weight and runs the algorithm with it. This can be done as a pre-processing of the graph, which can simplify and speed up the calculations. The advantage is also that we can run the algorithm (or even run other algorithms for different problems) on the modified graph many times without re-computing.

The worst case asymptotic estimate of the number of operations for such a graph modification in terms of Big O notation is $O(V * P)$, where V is the number of vertices (nodes) in the graph, and P is the number of dangerous points. But in practice, as already mentioned, we can not calculate the influence of each point on each node, but select only those points whose influence can be noticeable. Since the Gaussian function decreases from the epicenter, we can simply take several nodes in a certain radius of the epicenter (dangerous point), for which the value of the Gaussian function will differ from zero. Thus, in practice we can obtain lower complexity of the algorithm.

The worst case asymptotic of the number of operations A* is $O(E \log V)$, where E is the number of edges of the graph. Thus, the overall worst-case complexity of the algorithm for the first use is $O(E \log V + VP)$. But the advantage of the described approach is that the graph can be modified only once, and then we can run algorithms

on it repeatedly. We can calculate the danger function for nodes only once and then build many routes. And if we need to change some points or nodes, we can simply change them locally, in that area, and only those that will be affected. For example, if there is a storm at sea, we should not recalculate the values of all nodes. We can simply count those near reefs or other danger zones.

Moreover, the advantage of this approach is that we can run different algorithms for different problems on the modified graph without re-computing it. For example, on a modified graph we can run not just A* for a directed route search, but also some other algorithm for finding a circular route (described in part 2.3), and it will take into account the features of the modified graph.

Thus, we can build routes taking into account intermediate points and their properties, minimizing both criteria: distance and danger.

Modifications are also possible. For example, we can calculate the weight not of nodes, but of graph edges, if the implemented version of A* works with them. Then, accordingly, the asymptotic estimate of the graph modification in the worst case will be equal to $O(EP)$. But here, again, we can take into account the distance from dangerous points to edges and take into account only the essential ones, which in practice can reduce the running time of the algorithm. And of course, instead of danger, there may be another attribute that should be minimized. There can also be a weighted sum of more than two Gaussian functions and even different functions. For example, we can take a third, much broader and smaller value that can inform the algorithm when choosing the direction of movement and prevent approaching dangerous areas.

2.2. The case of a directed route with points to be visited

This case may be similar to the previous one, but in this case you should not avoid, but visit some points. E.g. this is the case when you need to walk from one point to another, but you have free time and would like to visit some attractive places and attractions along the way.

In the previous examples we tried to minimize the dangerous function, but in this one we should try to maximize the total interestingness function. We can denote it as

$I(n)$. But maximizing the sum $I(n)$ is equivalent to minimizing the sum $-I(n)$. So we can use the previous version of the function by simply adding a minus sign to the attribute.

$$G(n,p,\sigma) = I(x,y)e^{-\frac{(p_1-x)^2+(p_2-y)^2}{2\sigma^2}}$$

$$TI(n) = \sum_{p\in P} k_l G(n,p,sigma_{local}) + k_w G(n,p,sigma_{wide})$$

$$f(n) = k_g g(n, origin) + k_h h(n, destination) - TI(n)$$

A wide Gaussian function can “guide” the algorithm to a node of interest, and their concentration can describe some interesting area (for example, a historical center). While narrow, local may encourage the algorithm to visit the point directly.

But A* and some other graph algorithms cannot work with negative values. We can try to fix this by simply adding to all the interesting values of the function the absolute value of the smallest one. Thus, we can invert the function and create a total “uninterestingness” function whose values will be non less than zero.

$$TUI(n) = TI(n) + |\min_n TI(n)|$$

Minimizing “uninterestingness” is equivalent to maximizing interestingness. So, the final version of the utility function for this case is:

$$f(n) = k_g g(n, origin) + k_h h(n, destination) + TUI(n)$$

The estimates are the same as in the previous case. Also modifications and other comments are also possible for this case. In the case of three Gaussian functions, we can interpret the narrowest as for directly visiting the point of interest, the middle as for the area where you can see it, and the widest as for the area that is close to it and moving in its direction may soon lead you to it.

In cases where a combination of edge and node values (or just different parameter values) is used (for example, using edges to account for improving values and having them combined with node values), there may be several options: for example, considering the values separately (e.g. first flipping the nodes values, create “uninterestingness” from interestingness, and calculate them with edge values, e.g.

length); or consider them together: calculate everything together and make it non-negative in the end. The first case can be used if the edge parameter is very important and its value should not be underestimated. But in the second case, the parameter can be made important by increasing the value of the coefficient. So the second case: making all values non-positive (if necessary) might be more universal and efficient.

2.3. The case of a circular route

This is the case when, for example, you have free time at the station or a free evening at the hotel. Thus, it is a case of getting around the attractions closest to you so that you are interested in the route around the area where you are.

Another example would be the delivery of goods to several cities by ship and return to the port of departure. Also known as the traveling salesman problem (TSP). It can be formulated with some other conditions as [8]

$$x_{ij} = \begin{cases} 1 \text{ if the path is from node } i \text{ to node } j, \text{which are nodes to visit} \\ 0 \text{ otherwise} \end{cases}$$

$$\sum_{i=1}^{N} \sum_{i \neq j, j=1}^{N} c_{ij} x_{ij} \to \min$$

Where c_{ij} is some value (distance, cost, etc.) that needs to be minimized.

As mentioned, the advantage of the graph modification approach described above is that we can run various algorithms on the modified graph without recalculating the resulting values. And another value (for example, “danger” or “uninterestingness”) will be automatically minimized, as well as the previous one (distance, cost, etc.). We can adjust the weight of their contribution by weighting sum coefficients.

So, we can simply run one of the approximate solutions of the TSP algorithms on the modified graph. For example, the Asadpour Algorithm, whose worst-case complexity [9] is $O(\log V / \log \log V)$. Or some other, more optimal for a specific situation.

If there is no need to return to the starting point (in other words, if it doesn't matter where you end the route) the “circle” of the route can be broken. If there is a source point condition (if a ship, tourist, or other moving object is already at a location), we

can “break the circle,” remove the edge that returns to the origin node. If there is no such condition, we can simply remove the edge with the largest value that we are trying to minimize. This may be the case when a tourist plans a route through some area to walk and get acquainted with it, and he does not care where to start and end the route.

In the example of a tourist route, some points may not be mandatory to visit. So, there may be some algorithm that can evaluate their value and, in the case of a TSP decision, select some of them to visit.

The algorithm has a peculiarity: it prefers longer edges (with a large attribute value) than several small ones. If we take two paths: the first is simply the path from point A to point B, and the second is the path from A to point C, and then to B, where B is the point of interest, the distances of the paths are the same, but in the second case, the value of C (which may be non-zero) is also added to the values (“uninterestingness” or danger) of A and B. Therefore, the algorithm prefers longer edges (also in the case of points to be avoided). And sometimes, as in the case in Figure 4.4.2., this may not have such a positive effect. Perhaps this can be fixed by adding the edge length as a weight to the value. Or split the edges into smaller ones.

SECTION 3. Software tools and implementation details

The link to the project repository is in the literature list [13]. Additionally, the Jupyter notebooks code is in addition A for cases where points should be avoided, and in addition B for cases where points should be visited.

For directed route cases, the metric Euclidean distance function was chosen as a heuristic. Also, if we take a heuristic that returns zero, we can get a modification of Dijkstra's algorithm, which was also considered a little during testing. For city cases, the distance was calculated using coordinates in degrees, but it might be better to use projections to calculate in kilometers or something like that. At the very least, this may allow the results of the resulting route to be presented in a more convenient and understandable form.

The data was obtained from the Open Street Map service. The implementation was built in Python using the OSMnx [10], NetworkX [11] and GeoPandas [12] libraries. We used version 3.2.1 of the NetworkX libraries and version 1.7.0 of the OSMnx libraries. Some standard libraries such as Matplotlib and NumPy were also used. And Shapley in detail. We chose them because they provide a convenient way to display results and have the functionality we need: obtaining, filtering and displaying geodata, working with and displaying graphs, and also implementing graph algorithms.

The NetworkX implementation was used as an A* implementation. To solve the TSP problem, the Asadpour algorithm was used, which was also implemented in the NetworkX library.

Visualization was carried out using NetworkX tools in the case of a moving ship and OSMnx tools in the case of a tourist route. In both cases, with image correction using matplotlib.

Due to the implementation of the NetworkX library, the additional value (danger, interestingness) was on the edges. It was obtained by assigning end node value for directed graphs and maximum nodes values for undirected ones.

The first proof-of-concept implementation was built in C++ for a scenario using a directed route and points to be visited using test cases. The points and their values,

as well as the barrier locations, were manually selected to test the algorithm for various conditions and use cases. The visualization was in the console.

The second version of the implementation was implemented in Jupyter notebooks in Spyder. They provide a convenient way to visualize the results in the same place where the code was written, and the results can be exported.

In the case of the ship, the region (sea) was represented by a grid. Danger points were manually selected to test the algorithm under different conditions. Point concentration areas can model reef zones or areas with danger weather conditions or some other specific properties.

For the city, crossroads were represented as nodes, and streets as edges of the graph. The plot was created from the loaded (from file or network) OSM data, as were the points of interest. The city of Odessa was chosen for testing. It has quite interesting areas (for example, the historical center with such attractions as the Odessa Opera House, Deribasovskaya Street, Odessa City Garden, etc.), as well as a rather diverse urban structure (squares, square blocks, long and short streets (and, accordingly, the edges), etc.). Open Street Data also provides a lot of additional information about buildings, streets, places, etc. that can used to assess the value of the interestingness of a site or area. For example, in this work, only points of interest tagged with tourism corresponding to the scenario were taken. For points of interest that could be visited, the nodes closest to them were assigned as graph nodes. But the Gaussian function worked calculated exactly for the points.

For this implementation, the pedestrian network type was chosen for the city graph, but the OSMnx library also provides other types such as bike, drive, drive service that can be used for various use cases like this. For example, not a case of a walk, but a car trip between cities or, again, the option of using a city circular route. The algorithm can use not only distance as edge weight, but also something else, such as travel time, etc., which can be useful for these scenarios.

As said, the running time can be much faster than $O(VP)$ if we take for modification only those points that can be affected by a particular point (the addition

of which can be non-zero), but in this work the influence of points was calculated for all nodes of the graphs.

SECTION 4. The testing of the developed software

The solution was tested for the city of Odessa and some test graphs. For cities, results depend on the quality and completeness of the Open Street Map data for that area. You can see some of the results in the screenshots in this section.

4.1. The case of a directed route with points to be avoided.

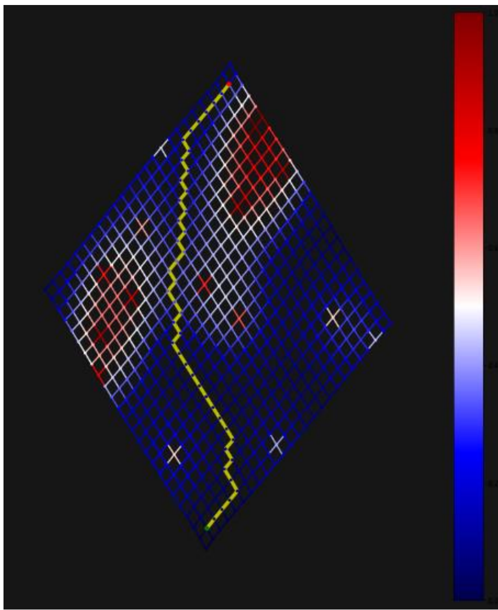

Figure 4.1.1. Route example 1

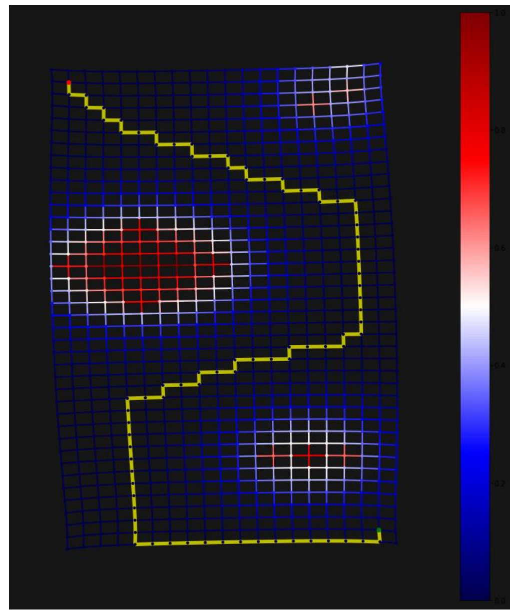

Figure 4.1.2. Route example 2

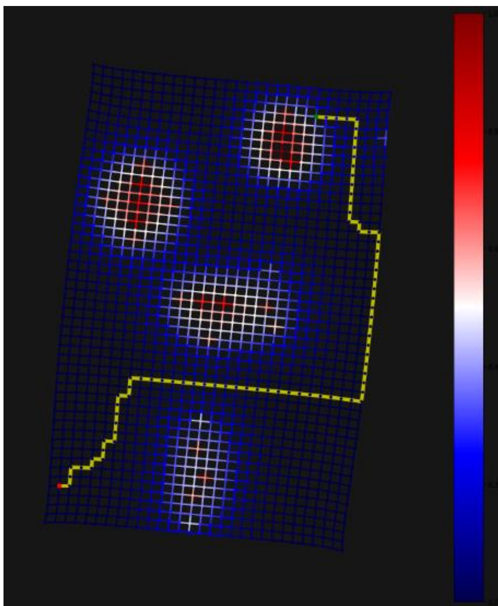

Figure 4.1.3. Route example 3

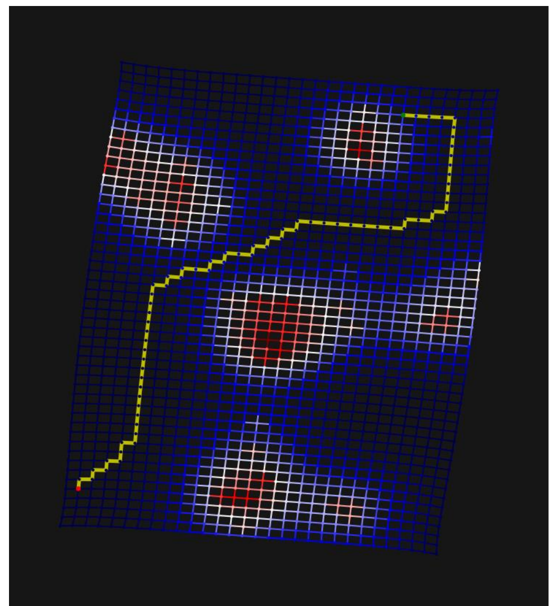

Figure 4.1.4. Route example 4

Figures 4.1.1.-4.1.4. show examples of simulation results for the case of a vessel's route in a reef zone. It can be seen that the algorithm avoids dangerous areas well and at the same time tries to get to the destination node using a shorter route. In other words, the algorithm tries to maintain a balance between safety and route distance, which was its goal.

This result is very similar to the result obtained in the book “Artificial Intelligence: A Modern Approach” [5], but here danger zones are considered not only for the A* algorithm, but, as seen in section 4.3, as a broader tool. Because if we use this approach before processing, we can not only solve directed pathfinding problems like in these examples, but also different problems and run different algorithms even without recalculating the previous values of the modified graph, which can save time and computational costs, while allowing the required attributes of intermediate points to be taken into account. Which was the task.

4.2. The case of a directed route with points to be visited

4.2.1. Test C++ version examples

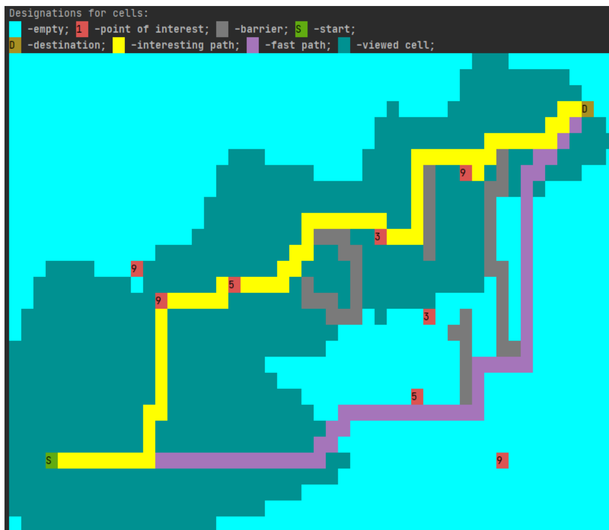

Figure 4.2.1.1. Route example 1

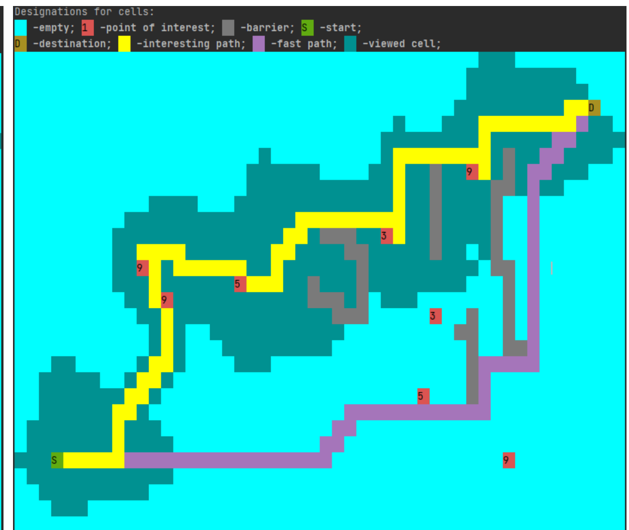

Figure 4.2.1.2. Route example 2

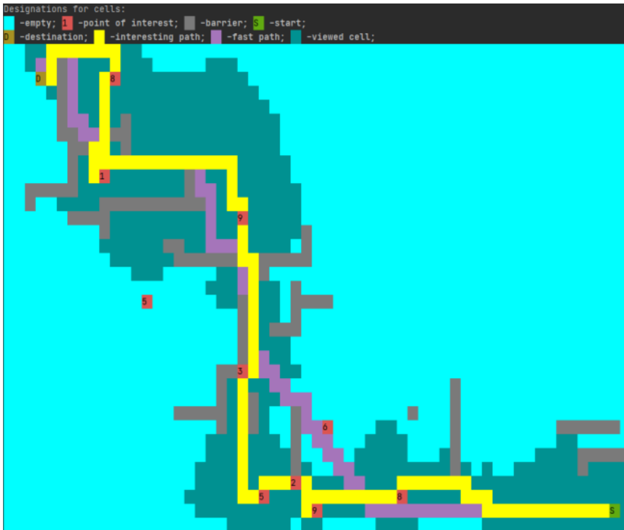

Figure 4.2.1.3. Route example 3

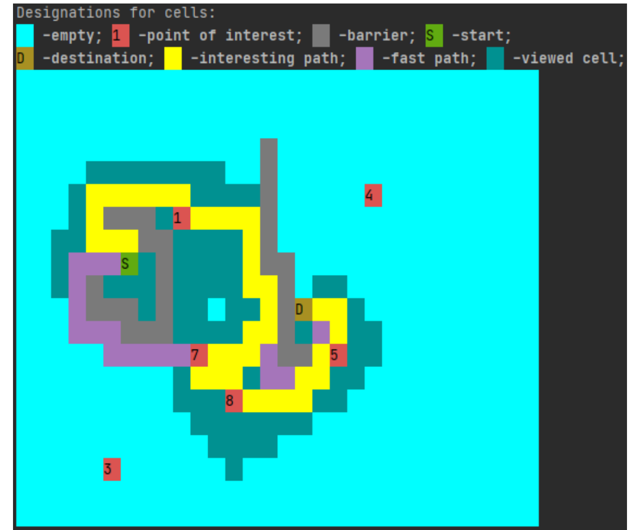

Figure 4.2.1.4. Route example 4

In Figures 4.2.1.1. to 4.2.1.4. you can see the results of an early trial implementation created in C++ using test cases. In these figures we see that the algorithm also builds a path in the opposite direction from the destination point if the value of the point of interest is large enough. Which may be quite logical in the case of a tourist route. This is probably due to the fact that in this case the calculations were also carried out with negative values, which could not only increase, but also reduce the intermediate result.

As a result, we see that the proposed route is slightly longer than the fastest route but allows you to visit some interesting places. What was the task in this case.

4.2.2. City examples

The center of Odessa and the surrounding area were chosen for testing. You can see a graphical representation of the city in Figure 4.2.2.1.

Figure 4.2.2.1. City Odessa graph illustration

To match the graph model with the terrain, screenshots of a map with points of interest are provided. Purple is the fastest route, yellow is an interesting one.

The results are presented in Figures 4.2.2.2.-4.2.2.6.

Figure 4.2.2.2. Map image of area of route of example 1

Figure 4.2.2.3. Example 1 directional route image

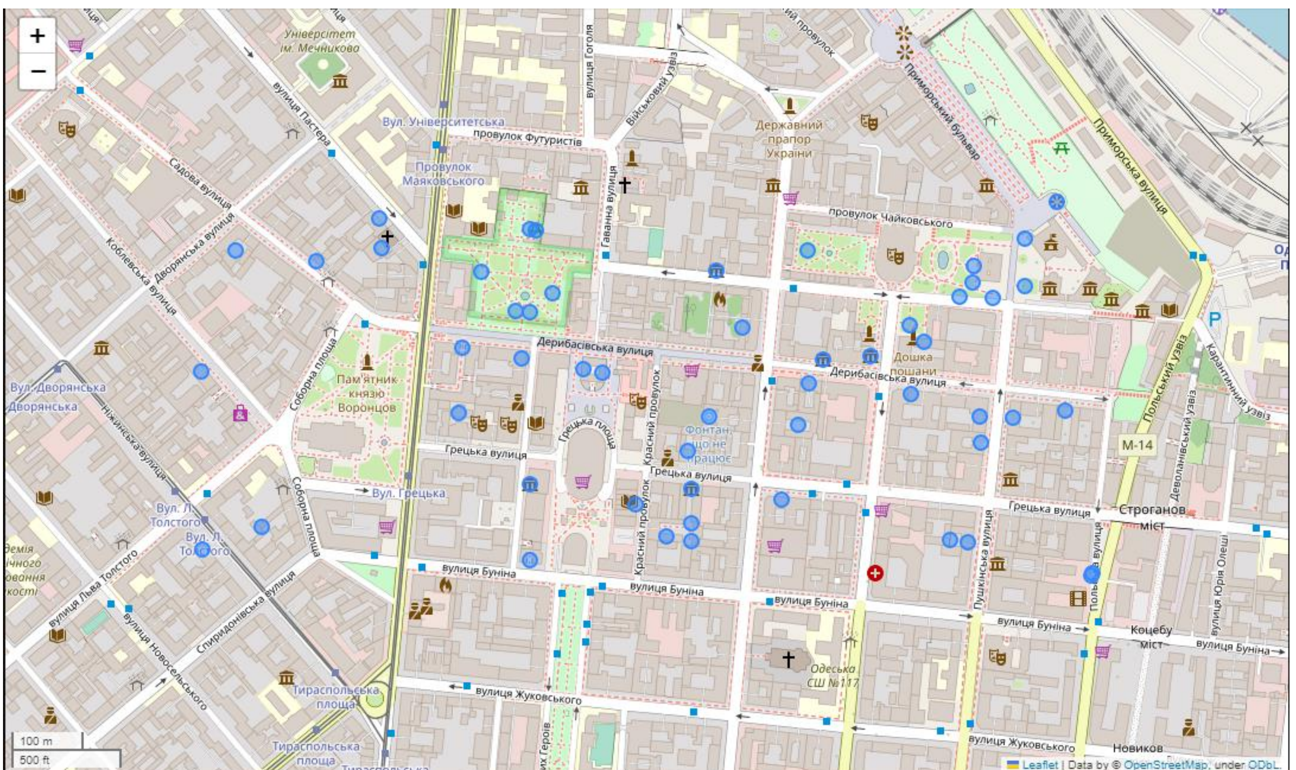

Figure 4.2.2.4. Map image of area of route of example 2

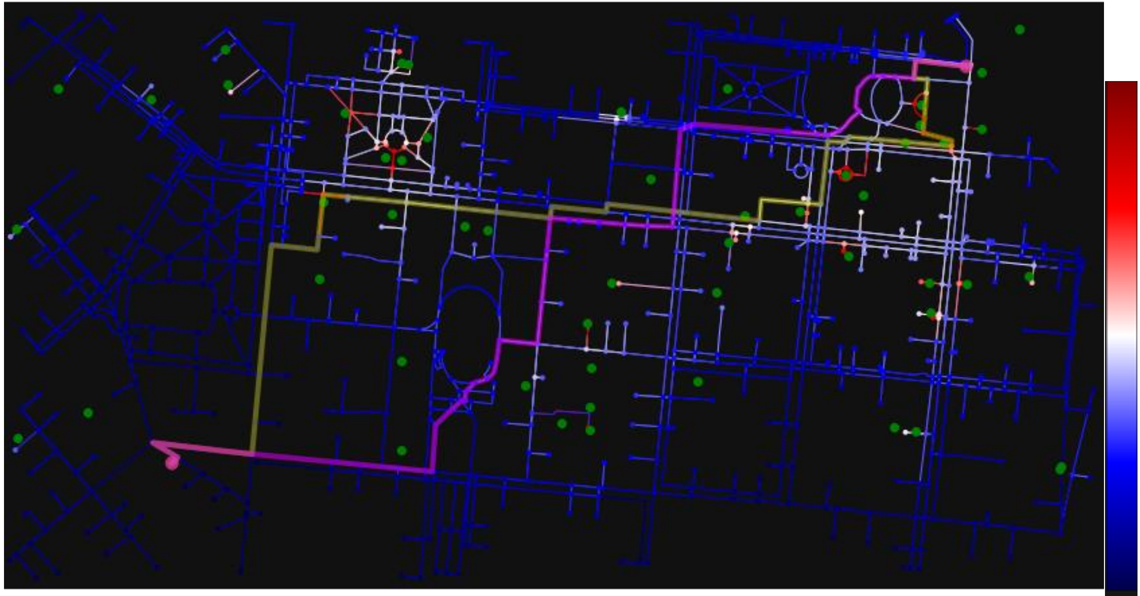

Figure 4.2.2.5. Example 2 directional route image

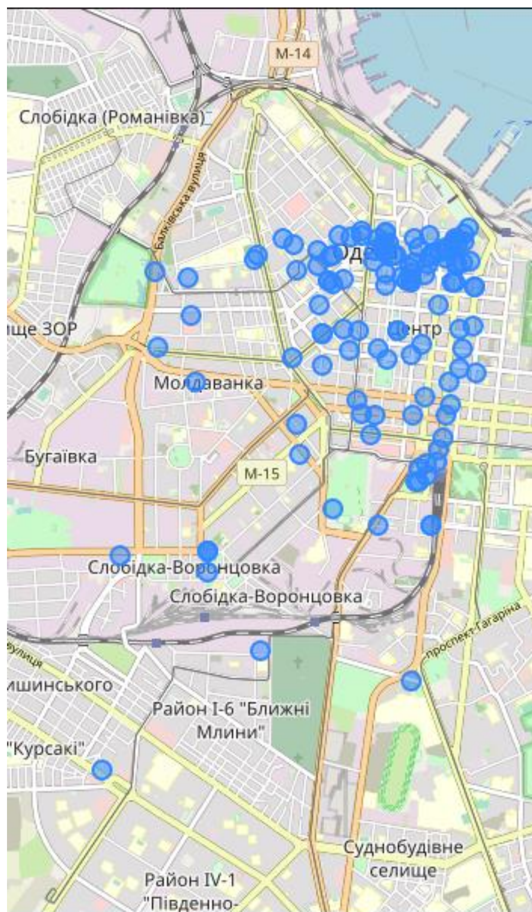

Figure 4.2.2.6. Example 3 map image

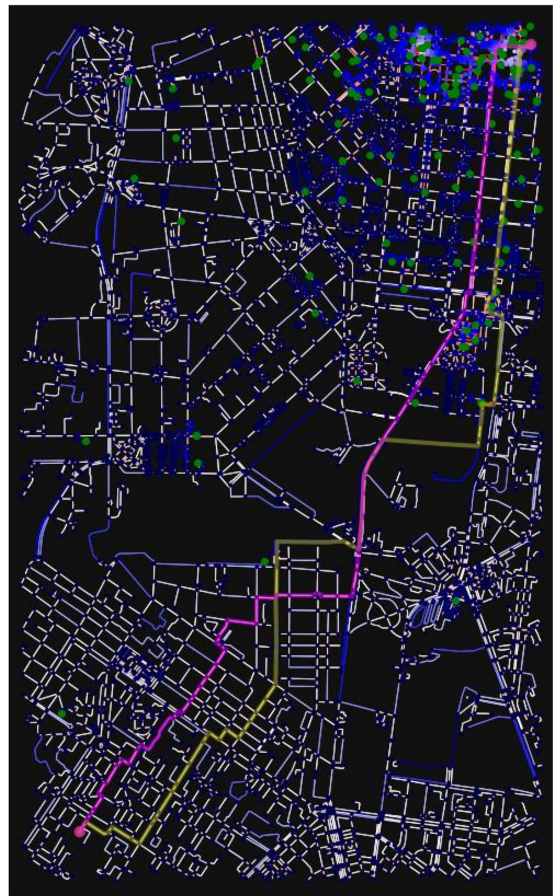

Figure 4.2.2.7. Example 3 route image

We can see that on real data the algorithm can give quite good results: the proposed paths pass close to some interesting points and along a fairly fast route.

In Figure 4.2.2.3. we can see that the algorithm led through some interesting places, even if the shortest route was along other streets.

We can see the same thing in Figure 4.2.2.5. The path to the destination turned out to be a little longer, but it allowed us to walk along Deribasovskaya Street and look at the Odessa Opera House, which are one of the main attractions of the city.

Figure 4.2.2.7. clearly shows that where there is a concentration of attractions (e.g. city center, upper left corner), the algorithm marks the area with higher interest (we can see it as a bright spot there).

So, as we can see, the algorithm can work even with fairly large graphs and work not only with individual points, but also with the areas that they create.

We can see that in almost the same places the shortest streets are of greater value, “more interesting” than the longer ones. This is because the modification algorithm includes street lengths in the final weight calculation.

4.3. The case of circular route with points to be avoided

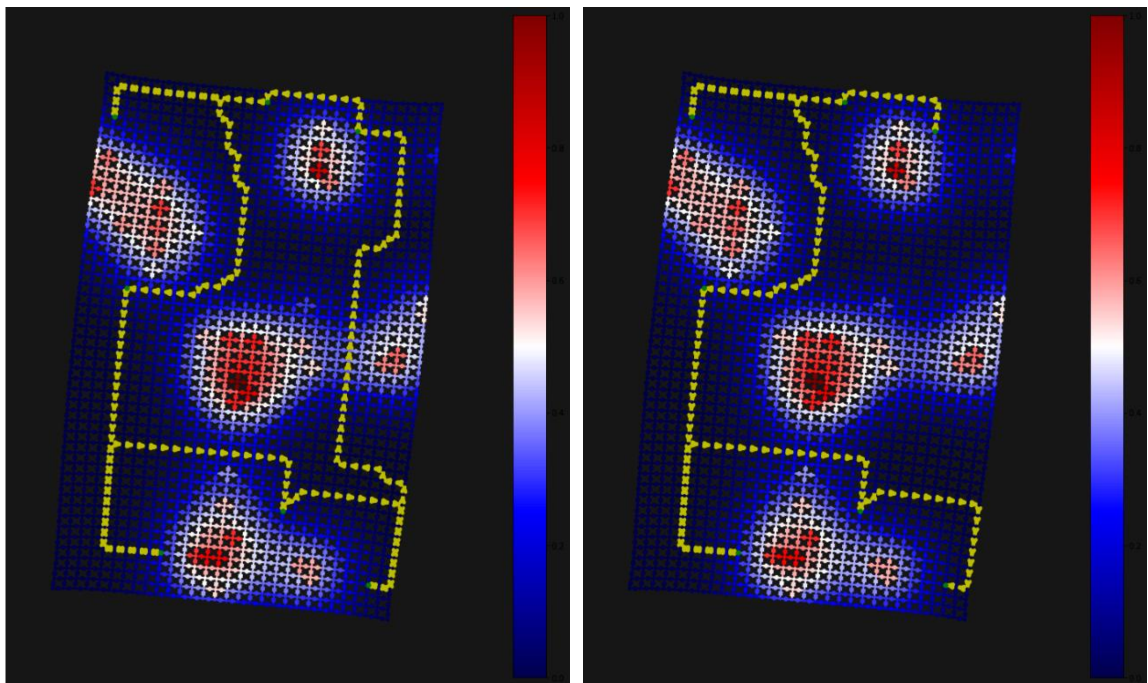

Figure 4.3.1. Circular route example 1 Figure 4.3.2. Circular route example 2

The left figure above. shows the result for the circular route case: e.g. a ship must deliver cargo to several locations and return to port. The right figure shows an example where the route may not be closed. E.g. if there is no need to return to port. We can

simply delete the path leading to the starting point. Or the path with the highest value, if it is not important (it is unknown) where to start. The implementation also allows to choose a starting point. As we can see, modification of the graph allows us to successfully take into account not only the distance, but also the danger criteria. As we can see, the algorithm tries to minimize being in dangerous areas, and also tries to minimize the overall path length. What was his immediate goal. And we didn’t even have to recalculate the graph values in order to solve another problem using a different algorithm.

4.4. The case of circular route with points to be visited

The figure below shows an example of a circular route in Odessa. This is the case when you want to walk a little and get acquainted with the area around. For example. you have a free evening at the hotel. Figure 4.4.2 shows an example when the route may not be closed.

Figure 4.4.1. Circular route example 1

Several points were selected and the algorithm laid out a route to them. In both cases, we see that the algorithm has built a route and it also passes through streets with a high value.

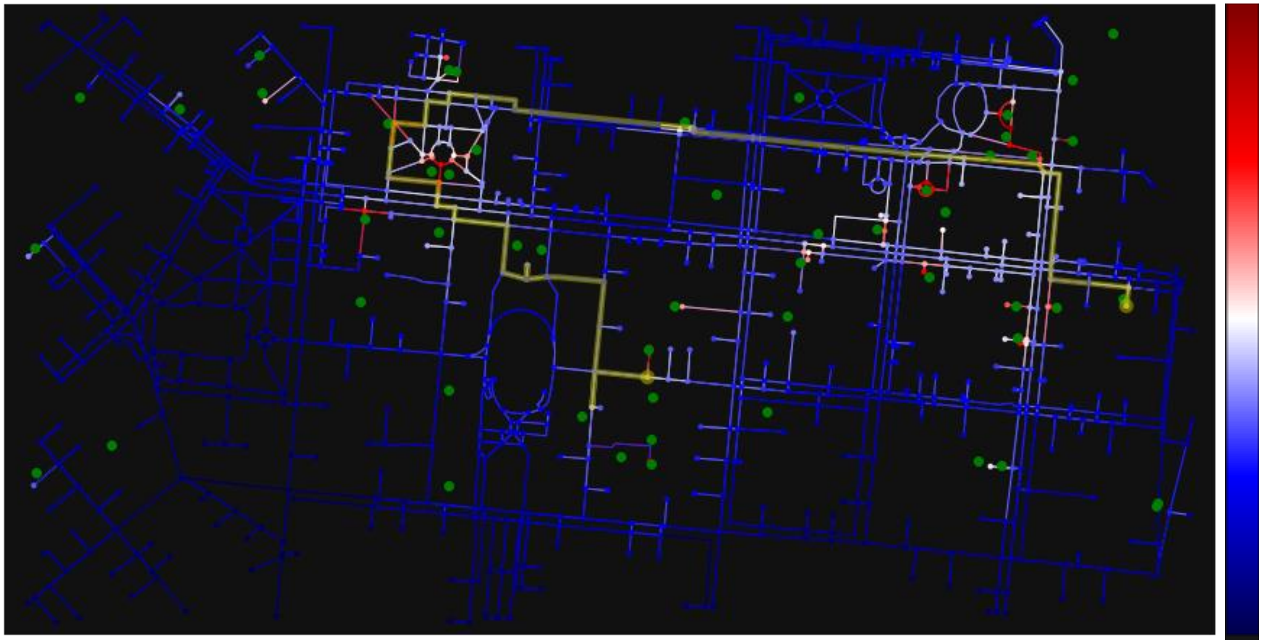

Figure 4.4.2. Circular route example 2

But in Figure 4.4.2. we can see (upper left corner of the path) that the algorithm did not decide to go to the square, which has a fairly high value. This may be due to a peculiarity of the algorithm that it prefers longer paths (look at the end of section 2.3). And sometimes, as in the case in Figure 4.4.2., this may not have such a positive effect.

We also see that the algorithm creates a route to some points, to which the path may be too long. And at the same time, it ignores some points that are not too far from the path and can be visited. Perhaps this is because some points are selected as necessary to visit the TSP, while others are not. So, for the case where not all points are required, it may be useful to use a point filter function which can decide which points to visit: discard those that are too far away, not interesting for a person, etc., and include others that are not close to the probable route, are very interesting, etc. This can be nice when using algorithms that take some given points as must to be visited. Another approach is to use algorithms that can decide on the fly which points can be visited. For example, some greedy algorithms and so on. Or modify some algorithms for this. But despite this, it built a route and visited some attractions.

In this case, the interestingness of the edges depended specifically on the points, but it could be different, depending on various parameters, such as the own interestingness of the streets or traffic, and so on.

CONCLUSIONS

In this article, we looked at how to modify graphs to pay attention to intermediate points, their values, attributes, and so on. A way has been proposed to change the graph to pay attention to the attributes we are trying to take into account, and then many known algorithms can be run to solve specific problems.

This approach has been considered for several cases: a directed route case with subcases of points to be avoided and to be visited; and the case of a circular route, also for points to be avoided and to be visited. The example results show the effectiveness of the algorithm and that it can be quite universal for different tasks — it can be used just as a pre-processing before some other algorithm. Moreover, this preprocessing may have a very little impact on the performance of the main algorithm: it can be calculated once, and then the results can be used. The time for the first calculation, or recalculation if necessary, can be $O(NP)$ in the worst case, as stated in section 2.1. But in practice it can be much less if we take for calculation only the nodes that will be affected.

A convenient map display can make the results easier to understand. The results can be provided as a sequence of nodes or edges, or saved in the GeoDataFrame format, which is easily displayed by some services.

For both cases and their subcases, we obtained fairly good results. The algorithm tries to minimize or minimize the overall value of the criterion (in the examples: danger and interest), taking into account the properties of intermediate points and edges, and at the same time tries to minimize the second one (total path length). Which was it’s goal.

For the considered algorithms, the modification approach gave slightly better results in cases where it was necessary to avoid points. Perhaps this is due to the fact that the algorithms try to minimize the values, which was necessary in this case, and adding even a small inverted value as “uninterestingness” is still an increase, although less than the others. For cases of visiting points (or other cases where the final algorithm is trying to minimize/maximize a value that should be

maximized/minimized), it might be a good idea to not try to invert the values, making them non-positive or otherwise, but to try to change the algorithms so that they can use the appropriate (negative, positive or other properties for the corresponding algorithms) values if their default versions cannot (as so was in the C++ test example in section 4.2.1). Of course, if they can, we can just try to use the node or edge values as they are. Also, modification of the final algorithm can allow for more flexible and in-depth settings and allow to influence the intermediate results of the algorithm, which can lead to better results (as, again, was the case in section 4.2.1).

In conclusion, we can highlight some pros and cons of this graph modernization approach.

Advantages:

- The graph modification time is polynomial and can be quite small in practical problems both for cases of calculations and for repeated calculations if necessary.
- • This approach is quite easy to modify for a specific problem (for example, as is the case when we can add a third Gaussian function to consider another type of region of influence) and it can theoretically take into account many properties of points and edges.
- Changing the graph can be done as a pre-processing before using the main algorithm and can be done independently.
  - This is why it can be used without repeated calculations, saving time.
  - This approach is quite universal and we can use it for a fairly wide range of tasks.

Disadvantages:

- It may favor the longest edges over the shortest ones when they are equivalent or even when the shortest ones are preferred. Perhaps this can be corrected by counting the length of the edge as a weight or in some other way. But even so, we got pretty good results.

Thus, the developed solution can provide the ability to pay attention to various properties of intermediate points and can be performed as a pre-processing before using other algorithms, so it is quite general and can work quite quickly for various tasks.

For future research, there is a lot that can be studied: ways to optimally select coefficients and other parameters (probably this can be done, taking into account the average value of the distances from points to nearest nodes, and so on), consider several attributes of points and edges and their combinations, possible combinations of attributes like each other's weights (their influence on each other), ways to correct the preference for long edges over short ones, and so on. The distance metric in the Gaussian function may not be the square of the Euclidean metric distance, but it may be something else. Combinations of enhancing and reducing properties may also be interesting. Or a combination of different methods of movement: e.g., first on foot, then take transport to another area, walk there, and so on. It might also be interesting to see how this approach could work with negative (or others) values or retrofit some algorithms to work with them. This can more accurately model some interesting and logical situations where it may be preferable to move in the opposite direction of the destination. Or even go to a node and return the same way. For example, to see some tourist attraction. Also in the case of a tourist route, it may be interesting to consider other factors that may influence, such as where the attraction can be seen, and so on.

But even the resulting solution can be a good way to take into account some intermediate points and their properties for a fairly wide range of problems and existing algorithms.

Many results were presented at conferences. References to proceedings from them are in the literature list [14][15][16][17].

ВИСНОВКИ

У цій статті ми розглянули, як модифікувати графи, щоб приділяти увагу проміжним точкам, їх значенням, атрибутам тощо. Запропонували спосіб змінити графік, щоб звернути увагу на атрибути, значення яких ми намагаємося врахувати, а потім можна запускати багато відомих алгоритмів для вирішення конкретних завдань.

Цей підхід розглядався для кількох випадків: випадок спрямованого маршруту з випадками точок, які слід уникати та які слід відвідати; і випадок кругового маршруту також для точок, які слід уникати та які слід відвідати. Результати прикладу показують ефективність алгоритму і те, що він може бути досить універсальним для різних завдань — його можна використовувати просто як попередню обробку перед іншим алгоритмом. Причому ця предобробка може дуже незначною мірою вплинути на продуктивність основного алгоритму: її можна обчислити один раз, а потім використовувати результати. Час першого розрахунку або перерахунку, якщо необхідно, у найгіршому випадку може становити O(N*P), як зазначено у розділі 2.1. Але на практиці він може бути набагато меншим, якщо брати для розрахунку тільки ті вузли, які можуть бути зміненими.

Зручне відображення мапи може полегшити розуміння результатів. Результати можуть бути надані у вигляді послідовності вузлів чи ребр або збережені у форматі GeoDataFrame, що легко відображається деякими сервісами.

Для обох випадків та їхніх підвипадків ми отримали досить добрі результати. Алгоритм намагається мінімізувати або мінімізувати загальне значення критерію (у прикладах: небезпека та інтерес), враховуючи властивості проміжних точок та ребр, і одночасно намагається мінімізувати другий (загальна довжина шляху). Що й було його метою.

Для алгоритмів, що розглядаються, модифікаційний підхід давав дещо кращі результати в тих випадках, коли необхідно було уникати точок. Можливо, це пов'язано з тим, що алгоритми намагаються мінімізувати значення, що й було

необхідно в даному випадку, і додавання навіть невеликого інвертованого значення як «нецікавість» — це все одно збільшення, хоча й менше, ніж в інших випадках. Для випадків відвідування точок (або інших випадків, коли остаточний алгоритм намагається мінімізувати/максимізувати значення, яке має бути максимізоване/мінімізоване відповідно), може бути гарною ідеєю не намагатися інвертувати значення, роблячи їх непозитивним чи іншим чином, а спробувати змінити алгоритми, щоб вони могли використовувати відповідні значення (негативні, позитивні або інші властивості для відповідних алгоритмів), якщо їх версії за замовчуванням не можуть (як це було в прикладі тестової версії C++ у розділі 4.2.1). Звичайно, якщо вони можуть, ми можемо просто спробувати використовувати значення вузлів або ребр такими, якими вони є. Також модифікація фінального алгоритму може дозволити забезпечити більш гнучке та глибоке налаштування та дозволити впливати на проміжні результати роботи алгоритму, що може призвести до кращих результатів (як знову було в розділі 4.2.1).

Можемо виділити деякі плюси та мінуси такого підходу до модернізації графа.

Переваги:

- Час модифікації графа поліноміальний й у практичних завданнях може бути дуже малим як для випадків обчислень, так і для повторних обчислень в разі потреби.
- Цей підхід досить легко модифікувати для конкретного завдання (наприклад, як у випадку, коли ми можемо додати третю функцію Гауса для розгляду іншого типу області впливу) і теоретично він може враховувати багато властивостей точок та ребр.
- Зміна графа може виконуватися як попередня обробка перед використанням основного алгоритму та може бути виконана незалежно.

  - Саме тому його можна використовувати без повторних обчислень, що економить час.
  - Цей підхід є досить універсальним і ми можемо використовувати його для досить широкого кола завдань.

Недоліки:

- Довгим ребрам може віддаватися перевага перед короткими, якщо вони еквівалентні або навіть коли краще найкоротші. Можливо, це можна виправити, враховуючи довжину ребра вагою або якимось іншим способом. Але навіть у цьому випадку ми отримали досить добрі результати.

Таким чином, розроблене рішення може забезпечити можливість брати до уваги різні властивості проміжних точок і може виконуватися як попередня обробка перед використанням інших алгоритмів.

Для подальших досліджень є багато чого можна вивчати: способи оптимального підбору коефіцієнтів та інших параметрів (ймовірно, це можна зробити з урахуванням середнього значення відстаней від точок до найближчих вузлів, тощо), розглянути кілька атрибутів точок та ребр, їх комбінації, можливі комбінації атрибутів, такі як у вигляді ваги один одного (їх вплив один на одного), способи виправлення переваги довгих ребр над короткими і так далі. Метрика відстані в функції Гауса може бути не квадратом відстані евклідової метрики, а може бути чимось іншим. Також цікавими можуть бути комбінації покращувальних та знижувальних властивостей. Або поєднання різних способів руху: наприклад, спочатку пішки, потім їхати транспортом в інший район, гуляти там і так далі. Також може бути цікаво подивитися, як цей підхід може працювати з негативними (або іншими) значеннями чи модифікувати деякі алгоритми для роботи з ними. Це може більш точно змоделювати деякі цікаві та логічні ситуації, в яких може бути краще рухатися у напрямку, протилежному до пункту призначення. Або навіть піти до вузла і повернутися тим самим шляхом.

Наприклад, щоб побачити якусь туристичну пам'ятку. Також у разі туристичного маршруту може бути цікаво розглянути інші фактори, які можуть вплинути, наприклад, де можна побачити визначну пам'ятку тощо.

Але навіть отримане рішення може стати хорошим способом врахувати деякі проміжні точки та їх властивості для широкого кола завдань та існуючих алгоритмів.

Багато результатів було представлено на конференціях. Посилання на матеріали з них є у списку літератури [14][15][16][17].

LITERATURE

1. Andrew Weinreich — Inspirock: vacation itineraries made easy — 10.08.2015— URL: https://www.andrewsroadmaps.com/inspirock-roadmap-video
2. Egor Smirnov— Гуляем по городу с умом: как я делал сервис для построения интересных пешеходных маршрутов—21.06.2018—URL: https://habr.com/ru/post/414433/
3. Egor Smirnov, Sergei Kudinov—Using a Genetic Algorithm for Planning Interesting Tourist Routes in the City on the Basis of Open Street Map Data—28.06-01.07.2021—URL: https://ieeexplore.ieee.org/abstract/document/9504741
4. Seznam.cz web portal—Mapy.cz service—08.12.2023—URL: https://en.mapy.cz/
5. Stuart J. Russell., Peter Norvig —Artificial Intelligence: A Modern Approach Third Edition — 2009—page 1010, section 25.4.3 Modified cost functions— URL: https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf
6. Amit Patel— Amit's A* Pages —15.10.2023—URL: https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
7. Weisstein, Eric W.—Gaussian Function from MathWorld—06.12.2023— URL: https://mathworld.wolfram.com/GaussianFunction.html
8. Tolga Bektaş, Luis Gouveia—Requiem for the Miller–Tucker–Zemlin subtour elimination constraints? —01.08.2014—URL: https://www.sciencedirect.com/science/article/abs/pii/S037722171300619X?via%3Dihub

9. Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010—Asadpour, Arash; Goemans, Michel X.; Madry, Aleksander; Gharan, Shayan Oveis; Saberi, Amin—An O(log n/ log log n)-approximation Algorithm for the Asymmetric Traveling Salesman Problem—17-19.01.2010— URL: https://dspace.mit.edu/handle/1721.1/60990
10. Geoff Boeing— OSMnx user reference—08.12.2023— URL: https://osmnx.readthedocs.io/en/stable/user-reference.html#
11. NetworkX Developers—asadpour_atsp implementation— 08.12.2023— URL:https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.approximation.traveling_salesman.asadpour_atsp.html
12. GeoPandas developers—Documentation— 08.12.2023— URL: https://geopandas.org/en/stable/docs.html
13. Kyryl Veremiov — GitHub repository Route_constructing_considering_intermidiate_points —08.12.2023— URL: https://github.com/KyrylVeremiov/Route_constructing_considering_intermidiate_points
14. Матеріали XXIII Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів «Стан, досягнення та перспективи інформаційних систем і технологій»— Igor Mazurok , Kyryl Veremiov — Optimization of paths, taking into account the significance of intermediate points—20-21.04.2023—pages 95-96 — URL: https://www.ontu.edu.ua/download/konfi/2023/Conference_abstract-IT-21-22-04-23.pdf
15. Матеріали двадцятої всеукраїнської конференції студентів і молодих науковців «Інформатика, інформаційні системи та технології»—

Мазурок І. Є., Веремйов К. В.— Оптимізація шляхів з урахуванням значущості проміжних точок— 28.04.2023— стор. 203-205— URL: https://atl.pdpu.edu.ua/images/doc/inf/2023/Zbirka_tez_IIST-2023_.pdf

16. Матеріали XVI міжнародної науково-практичної конференції «Інформаційні технології і автоматизація – 2023» —I.Mazurok, K.Veremiov — CONSTRUCTING AN ATTRACTIVE ROUTE BY SOLVING THE TRAVELING SALESMAN PROBLEM—19-20.10.2023— pages 67-68— URL: https://ontu.edu.ua/download/konfi/2023/Collection-of-abstracts-of-the-conference-ITIA-2023.pdf
17. Матеріали III міжнародної науково-практичної конференції «Проектний та логістичний менеджмент: нові знання на базі двох методологій»— Igor Mazurok, Kyryl Veremiov — Constructing a safety-speed balanced route—9-10.11.2023— in the process of issuing — URL: https://pm-onmu.org.ua/archieve/

ADDITION A

`Points_to_be_avoided_ship_in_reef_zone_route.ipynb`

This file contains code for implementing algorithms for the case of using a ship in a dangerous area. It presents the initialization of the starting graph, its modification using Gaussian functions, and the running of algorithms for solving pathfinding and TSP problems (for both directional and circular routes).

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
nx.__version__

default_safe_value=0
G = nx.grid_graph(dim=[30, 50])  # nodes are two-tuples (x,y)
pos = nx.kamada_kawai_layout(G)

G=G.to_directed()

danger_points=[((5,10),9),((3,10),8),((7,10),8),((7,10),9),((8,9),8),((5,0),8),((16,23),8),((18,23),8),((14,23),5),
        ((12,23),9),((10,23),9),((14,20),7),((14,26),7), ((20,10),6),((26,9),9),((24,14),9),((24,10),8),((24,18),8),
        ((24,16),9),((43,14),9),((40,15),9),((45,15),8),((37,15),8),((28,15),9),((48,15),8),
       ((45,7),9),((46,7),9),((45,8),9),
       ((45,16),9),((45,17),9),
       ((25,2),9),((24,2),9),((25,3),9),
       ((30,15),9),((29,14),9),((28,15),9),
       ((10,30),9),((11,29),9),((9,30),9),((8,28),9),((20,0),9),((19,0),9)]

origin_point_x= 3
origin_point_y= 7
destination_point_x= 45
destination_point_y= 28
```

```
origin_node= (origin_point_x,origin_point_y)
destination_node=(destination_point_x,destination_point_y)

def dist_euclidean(a, b):
    (x1,y1)=a
    (x2,y2)=b
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5
def gauss_filter(x,y,p1,p2, sigma):
    return np.exp(-((p1-x)*(p1-x)+((p2-y)*(p2-y)))/(2*sigma*sigma))
def get_nodes_attributes (G1, danger_points1,name_attr1:str,sigma_local=0.1, sigma_wide=3,
k_local=1,k_wide=1,default_value=0,improving_attr=True):
    attr_dict={n:{name_attr1:default_value} for n in G1.nodes}
    L=list(attr_dict.values())
    for n in G1.nodes:
        for i in range(len(danger_points1)):
            (d_point,d_value)=danger_points1[i]

addition_value=d_value*(gauss_filter(n[0],n[1],d_point[0],d_point[1],sigma_local)*k_local+gauss_filter(n[0
],n[1],d_point[0],d_point[1],sigma_wide)*k_wide)
            if improving_attr:
                attr_dict[n][name_attr1]=attr_dict[n][name_attr1]-addition_value
            else:
                attr_dict[n][name_attr1]=attr_dict[n][name_attr1]+addition_value
    if improving_attr:
        min_attr_dict= min([L[i][name_attr1] for i in range(len(L))])
        for n in G1.nodes:
            attr_dict[n][name_attr1]=attr_dict[n][name_attr1]+abs(min_attr_dict)
    return attr_dict
danger_attr=get_nodes_attributes(G,danger_points,"danger_attr",default_value=default_safe_value,impr
oving_attr=False)
nx.set_node_attributes(G,danger_attr)
def get_edges_attributes(G1,name_attr1,k_length=0.5,k_attr=0.5):
    attr_dict={edge:{name_attr1:0} for edge in G1.edges}
    for edge in G1.edges:
```

```
        # IF DIGRAF
        #
attr_dict[edge][name_attr1]=G1.nodes[edge[1]]["danger_attr"]*k_attr+k_length*G1.edges[edge]["length"]
        # IF NOT DIRECTED GRAF:
        max_val=max(G1.nodes[edge[1]]["danger_attr"],G1.nodes[edge[0]]["danger_attr"])
        attr_dict[edge][name_attr1]=max_val*k_attr+k_length*G1.edges[edge]["length"]
    return attr_dict
nx.set_edge_attributes(G,{edge:{"length":1} for edge in G.edges})
# If the edge value was not a "weight", the algorithm did not work correctly
danger_attr_edge=get_edges_attributes(G,"weight")
nx.set_edge_attributes(G,danger_attr_edge)

path_orig_dest_nodes=nx.algorithms.shortest_paths.astar_path(G,origin_node,destination_node,heuristic
=dist_euclidean,weight="weight")
path_orig_dest_edges=[(path_orig_dest_nodes[i],path_orig_dest_nodes[i+1])for i in
range(len(path_orig_dest_nodes)-1)]

from matplotlib.pyplot import figure

fig, ax = plt.subplots()
fig.set_size_inches(13,15)
# fig.set_dpi(100)

colormap=plt.cm.seismic
nx.draw(G, pos=pos,
cmap=colormap,node_color=list(nx.get_node_attributes(G,'danger_attr').values()),node_size=10,
        edge_cmap=colormap, edge_color=list(nx.get_edge_attributes(G,'weight').values()),width=2)

sm = plt.cm.ScalarMappable(cmap=colormap)
sm._A = []
plt.colorbar(sm,ax=ax)
fig.set_facecolor('#161616')
nx.draw_networkx_nodes(G,pos=pos,nodelist=[origin_node,destination_node],node_color=["g","r"],node_
size=30)
nx.draw_networkx_edges(G,pos=pos,edgelist=path_orig_dest_edges,edge_color = "y",width=5)
```

```
plt.show()



nodes_to_visit=[(3,7),(46,2),(40,10),(45,20),(20,25),(4,28),(1,15)]
path_circular_nodes=nx.algorithms.approximation.traveling_salesman_problem(G=G,
                                        nodes=nodes_to_visit,
                                                  method=nx.algorithms.approximation.asadpour_atsp,
                                                  cycle=False)
path_circular_edges=[(path_circular_nodes[i],path_circular_nodes[i+1])for i in
range(len(path_circular_nodes)-1)]
from matplotlib.pyplot import figure

fig, ax = plt.subplots()
fig.set_size_inches(13,15)
# fig.set_dpi(100)
colormap=plt.cm.seismic
nx.draw(G, pos=pos,
cmap=colormap,node_color=list(nx.get_node_attributes(G,'danger_attr').values()),node_size=10,
     # edge_cmap=colormap,
edge_color=list(nx.get_edge_attributes(G,'danger_attr_edge').values()),width=2)
     edge_cmap=colormap, edge_color=list(nx.get_edge_attributes(G,'weight').values()),width=2)
sm = plt.cm.ScalarMappable(cmap=colormap)
sm._A = []
plt.colorbar(sm,ax=ax)
fig.set_facecolor('#161616')
nx.draw_networkx_nodes(G,pos=pos,nodelist=nodes_to_visit,node_color=["g"],node_size=30)
nx.draw_networkx_edges(G,pos=pos,edgelist=path_circular_edges,edge_color = "y",width=5)
plt.show()
```

ADDITION B

Points_to_be_visited_tourist_route.ipynb

This file contains the implementation code for the tourist route use case. It contains retrieving the starting graph from the OSM database, retrieving information about the points of interest, modifying the graph using Gaussian functions, inverting the values to non-positive and, again, path finding and TSP algorithms for both directional and circular route use cases.

```
%matplotlib inline
# %matplotlib notebook

import shapely
import geopandas
import networkx as nx
import osmnx as ox
import matplotlib.pyplot as plt
import numpy as np

ox.__version__
class bbox:
  def __init__(self, north,south,east,west):
    self.north=north
    self.south=south
    self.east=east
    self.west=west
# download/model a street network for some city then visualize it
# G = ox.graph_from_place("Одеса, Україна", network_type="walk")
# fig, ax = ox.plot_graph(G)
# ox.io.save_graphml(G, "Odessa")

G=ox.io.load_graphml("Odessa")
fig, ax = ox.plot_graph(G)
```

```
box= bbox(north=46.486292,south=46.480293, west=30.728495, east=30.745258)
G_box=ox.truncate.truncate_graph_bbox(G,box.north,box.south,box.east,box.west)
fig, ax = ox.plot_graph(G_box,show=False, close=False)
features_list_box=ox.features.features_from_bbox(box.north,box.south,box.east,box.west,
{"tourism":True})
features_list_box.head()
features_list_box_points=features_list_box[geopandas.GeoDataFrame(map(lambda x:
type(x)==shapely.geometry.point.Point, features_list_box["geometry"])).values]
features_list_box_points.head()
features_list_box_points_x=features_list_box_points["geometry"].x.values
features_list_box_points_y=features_list_box_points["geometry"].y.values
nodes=ox.distance.nearest_nodes(G,features_list_box_points_x,features_list_box_points_y)
nodes
nodes=list(filter(lambda x: x in list(G_box.nodes),nodes))
features_list_box_points.explore( marker_kwds={"radius": 6})

origin_point_longitude= 46.453293
origin_point_latitude= 30.708495

destination_point_longitude=46.486292
destination_point_latitude= 30.743258

fig, ax = ox.plot_graph(G_box,show=False, close=False)
ax.scatter(origin_point_latitude,origin_point_longitude, c='green')
ax.scatter(destination_point_latitude,destination_point_longitude, c='red')
ax.scatter(features_list_box_points_x,features_list_box_points_y, c='blue')
plt.show()

origin=ox.distance.nearest_nodes(G_box, origin_point_latitude,origin_point_longitude)
destination=ox.distance.nearest_nodes(G_box, destination_point_latitude,destination_point_longitude)

# Negative weights???
path_orig_dest_fast=nx.algorithms.shortest_paths.astar_path(G_box,origin,destination,weight="length")
len(path_orig_dest_fast)
fig, ax = ox.plot.plot_graph_route(G_box,path_orig_dest_fast,show=False, close=False)
```

```
ax.scatter(features_list_box_points_x,features_list_box_points_y, c='blue')
plt.show()
def gauss_filter(x,y,p1,p2, sigma):
  return np.exp(-((p1-x)*(p1-x)+((p2-y)*(p2-y)))/(2*sigma*sigma))
def attribute_value_func(point):
  return 1
def get_nodes_attributes (G1, points1,name_attr1:str,sigma_local=1e-4, sigma_wide=1e-3,
k_local=1e2,k_wide=1e1,default_value=0,improving_attr=True):
  attr_dict={n:{name_attr1:default_value} for n in G1.nodes}
  for n in G1.nodes:
    for i in range(len(points1)):
      point=points1.iloc[i]
      point_x=point.geometry.x
      point_y=point.geometry.y
      addition_value=attribute_value_func(point)*(gauss_filter(point_x,point_y, G1.nodes[n]["x"],
G1.nodes[n]["y"],sigma_local)*k_local
                  +gauss_filter(point_x,point_y, G1.nodes[n]["x"],
G1.nodes[n]["y"],sigma_wide)*k_wide)
      if improving_attr:
        attr_dict[n][name_attr1]=attr_dict[n][name_attr1]-addition_value
      else:
        attr_dict[n][name_attr1]=attr_dict[n][name_attr1]+addition_value
  L=list(attr_dict.values())
  if improving_attr:
    min_attr_dict= min([L[i][name_attr1] for i in range(len(L))])
    for n in G1.nodes:
      attr_dict[n][name_attr1]=attr_dict[n][name_attr1]+abs(min_attr_dict)
  return attr_dict

interestingness_attr=get_nodes_attributes(G_box,features_list_box_points,"interestingness_attr",
improving_attr=False)
nx.set_node_attributes(G_box,interestingness_attr)
print(max([G_box.nodes[n]["interestingness_attr"] for n in
G_box.nodes]),min([G_box.nodes[n]["interestingness_attr"] for n in G_box.nodes]))
def get_edges_attributes(G1,name_attr1,k_length=0.2,k_attr=0.8,improving_attr=True):
```

```
    attr_dict={edge:{name_attr1:0} for edge in G1.edges}
    for edge in G1.edges:
        # IF DIGRAF
        attr_dict[edge][name_attr1]=k_length*G1.edges[edge]["length"]
        # IF NOT DIRECTED GRAF:
        # max_val=max(G1.nodes[edge[1]]["danger_attr"],G1.nodes[edge[0]]["danger_attr"])
        # attr_dict[edge][name_attr1]=max_val*k_attr+k_length*G1.edges[edge]["length"]
        if improving_attr:
          attr_dict[edge][name_attr1]=  attr_dict[edge][name_attr1]-
G1.nodes[edge[1]]["interestingness_attr"]*k_attr
        else:
          attr_dict[edge][name_attr1]=
attr_dict[edge][name_attr1]+G1.nodes[edge[1]]["interestingness_attr"]*k_attr
            # Attempts to prevent preference for short edges
          # attr_dict[edge][name_attr1]=
((attr_dict[edge][name_attr1])**(1/2))*G1.nodes[edge[1]]["interestingness_attr"]*k_attr
          # attr_dict[edge][name_attr1]=
attr_dict[edge][name_attr1]*(1+G1.nodes[edge[1]]["interestingness_attr"]*k_attr)

    L=list(attr_dict.values())
    if improving_attr:
        min_attr_dict= min([L[i][name_attr1] for i in range(len(L))])
        for edge in G1.edges:
            attr_dict[edge][name_attr1]=attr_dict[edge][name_attr1]+abs(min_attr_dict)
    return attr_dict

interestingness_attr_edge=get_edges_attributes(G_box,"weight",improving_attr=True)
nx.set_edge_attributes(G_box,interestingness_attr_edge)
def dist_euclidean(a, b):
    x1=G_box.nodes[a]["x"]
    x2=G_box.nodes[b]["x"]
    y1= G_box.nodes[a]["y"]
    y2=G_box.nodes[b]["y"]
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5
```

```
path_orig_dest=nx.algorithms.shortest_paths.astar_path(G_box,origin,destination,
weight="weight",heuristic=dist_euclidean)
len(path_orig_dest)
fig, ax = ox.plot.plot_graph_route(G_box,path_orig_dest,show=False, close=False)
ax.scatter(features_list_box_points_x,features_list_box_points_y, c='blue')
plt.show()
# Taking into account negative interestingness may lead to some fair results.
nodes_coord=ox.utils_graph.graph_to_gdfs(G_box, edges=False, node_geometry=False)[["x", "y"]]
nodes_coord.head()

fig, ax = ox.plot.plot_graph(G_box,
                        #
node_color=ox.plot.get_node_colors_by_attr(G_box,"interestingness_attr",cmap='seismic_r'),

node_color=ox.plot.get_node_colors_by_attr(G_box,"interestingness_attr",cmap='seismic'),
                edge_color=ox.plot.get_edge_colors_by_attr(G_box, "weight",cmap='seismic_r'),
                        figsize=(15,15),show=False, close=False)
ax.scatter(features_list_box_points_x,features_list_box_points_y, c='g')

ox.plot.plot_graph_route(G_box,path_orig_dest,route_color="y",ax=ax,show=False, close=False)
ox.plot.plot_graph_route(G_box,path_orig_dest_fast,route_color="magenta",ax=ax)
path_circular=nx.algorithms.approximation.traveling_salesman_problem(G=G_box, weight="weight",
                                                  nodes=nodes[0:1]+nodes[12:17],
method=nx.algorithms.approximation.asadpour_atsp,
                                                         cycle=False)
fig, ax = ox.plot.plot_graph(G_box,

node_color=ox.plot.get_node_colors_by_attr(G_box,"interestingness_attr",cmap='seismic'),
                edge_color=ox.plot.get_edge_colors_by_attr(G_box, "weight",cmap='seismic_r'),
                        figsize=(15,15),show=False, close=False)
ax.scatter(features_list_box_points_x,features_list_box_points_y, c='g')
ox.plot.plot_graph_route(G_box,path_circular,route_color="y",ax=ax)
```