Efficient Dijkstra-based greedy algorithm for cycle-route planning*

Antoni Jaszcz[1,*,†], Szymon Hankus[1,†], Michał Bober[1,†] and Bartosz Bugla[1,†]

[1]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44100 Gliwice, POLAND

Abstract

Planning routes based on actual data is not an easy task, especially when faced with challenging conditions. Finding cycles of fixed-length in directed, weighted graphs is an NP-hard problem. In the field, not enough research has been done on many alterations of this problem, especially in regard to real-world applications. In this paper, we propose a greedy algorithm for generating cyclic routes of the desired length and characteristics to match specific types of bicycles, using real-world data. The proposal is based on a greedy search of sub-routes to find the best-fit bicycle route. The results indicate, that the algorithm performs very well and can be easily adjusted for other similar tasks. The mean absolute percentage error for the total distance of the routes was below the acceptable 5% error, reaching 3.24 MAPE for shorter and 2.74 MAPE for longer routes.

Keywords

heuristic, dijkstra, route planning, OSM

1. Introduction

Route planning on maps is a basic tool in today's world, where GPS is an element that allows us to locate our position and plot the route to a specific point. However, new methods are still needed to increase the effectiveness and efficiency of the techniques. It should be noted that maps are updated quite often due to various road accidents or infrastructure changes. Consequently, the ability to update data is one of the basic elements that should not affect the operation of the algorithm itself, as well as its time complexity. New technologies also influence the operation of various computational techniques, which can contribute to improving efficiency in terms of computational and time complexity, or even saving resources. The problem can be classified as a graph or optimization. In both cases, various techniques can be used. An example is the construction of a waste management model in the Internet of Things [1]. The traveling salesman problem also continues to be solved by new tools as shown in [2]. The authors of the research used the parallel river formulation dynamics optimizer. Another solution is to pay attention to time windows in the problem of finding a specific path [3]. An interesting approach is to use the hybridization of the Dijkstra algorithm with marking for the movement of autonomous robots [4]. The autonomy of robots and cars is one of the main applications of the latest way-finding methods [5, 6, 7, 8]. Another modification of the Dijkstra algorithm is shown in [9], which shows the further need to develop these solutions.

*IVUS2024: Information Society and University Studies 2024, May 17, Kaunas, Lithuania

[1,*] Corresponding author

[†] These author contributed equally.

aj303181@student.polsl.pl (A. Jaszcz); sh303175@student.polsl.pl (S. Hankus); mb300327@student.polsl.pl M. Bober); bb303149@student.polsl.pl (B. Bugla)

In the context of road planning, the basic algorithms are graph algorithms, which are quite often greedy. Which results in high computational complexity. An alternative to this group of algorithms is heuristics, which are algorithms that return an approximate solution but with low computational and time expenditure. An example of one of the latest heuristic algorithms is a technique inspired by the behavior of penguins [10]. It should also be noted that older algorithms are subject to hybridization, an example of which is the k-nearest neighbors algorithm for the clustering problem [11]. Such algorithms allow for the possibility of improving machine learning techniques, as shown in [12, 13], where algorithms inspired by nature were used as an optimizer, or to detect features in images. Again in [14], the authors focused on improving the stitching algorithm by applying the selected heuristic. The enormous use of heuristics is reflected in the possibility of using them as optimization methods in drones [15]. Generally, the optimization problem is quite known and can quite often be solved by finding an approximate solution. An example are navigations and different vessel applications [16, 17, 18]. In [19] the authors proposed a heuristic approach for generating cycles of fixed length in undirected graphs. The presented approach was tested on a graph obtained from real-world data.

Based on the literature analysis, the need for new methods and techniques in road planning can be noticed. In this paper, we present a solution for generating cycles of desired length and of certain edge attribute properties in a real-world environment, by introducing a dedicated algorithm. A practical use of finding a cyclic route suited for certain bicycle types is also presented and analyzed. The main contributions of the paper can be listed as:

- New dedicated algorithm for finding cyclic routes with desired total length and edge attribute properties.
- Practical application of the proposed approach in the task of finding the best bicycle route.

2. Methodology

In this section, the proposed approach along with all the techniques used are described.

2.1. Great-circle distance

The distance function used in the algorithm is the Earth's Orthodrome (the shortest path between two points on the surface of a sphere running along its surface), commonly called great-circle distance. It is calculated from the coordinates of the starting vertex 'a' and the destination vertex 'b', in the UTM system. It is most often calculated using the haversine formula (Eq. 1), where r is the Earth's radius, lat_a and lat_b are the latitudes of the two points (in radians), Δlat and Δlon are the differences in coordinate values between the two points (in radians).

$$d = 2r \times \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\text{lat}}{2}\right) + \cos(\text{lat}_1) \times \cos(\text{lat}_2) \times \sin^2\left(\frac{\Delta\text{lon}}{2}\right)}\right) \tag{1}$$

Algorithm 1: Proposed greedy route-finding algorithm

Input: Graph representation G, start node id a, estimated number of sub-routes n, desired total distance D, number of acceptable attempts K, acceptable distance error Er

```
path = [a]; total_distance = 0; part = D/n; part_previous = part;
b = a; p = b; acceptable_attempts = 0; i = 0;
Set the 'visited' attribute for all edges in the G to False;
while acceptable_attempts < K do
    part_path = find_partial_route (a, p, part, part_previous);
    part_distance = path_weight (G, part_path, 'length');
    back_path = dijkstra (G, b, heuristic = dist_func, weight = weight_func());
    back_distance = path_weight (G, back_path, 'length');
    if total_distance + part_distance + back_distance > D · (1.0 + Er) then
        back_path_previous = dijkstra (G, a, heuristic = dist_func, weight = weight_func());
        back_distance_previous = path_weight (G, back_path_previous, 'length');
        if total_distance + back_distance_previous > D · (1.0 - Er) then
            path += back_path_previous[1:];
            total_distance = path_weight (G, path, 'length');
            return path, total_distance;
        else
            part /= 2;
            acceptable_attempts += 1;
            continue;
    else if total_distance + part_distance + back_distance > D · (1.0 - Er) then
        break;
    if i >= n then
        break;
    part_previous = part;
    i += 1;
    mark_as_visited (G, part_path);
    path += part_path[1:];
    total_distance = path_weight (G, path, 'length');
    p = a; a = b;
path += part_path[1:] + back_path[1:]; total_distance = path_weight(G, path, 'length');
return path, total_distance;
```

2.2. Weighting function

The weighting function used in the Dijkstra algorithm takes into account its edges' actual length, surface type, and (if the algorithm is looking for a return route) whether the edge has already been visited. Each considered surface type can be assigned a certain positive, non-zero value, acting as a modifier of the edge's actual length. This makes the algorithm prefer favorable surface types. If the 'visited' edge attribute is set as 'True', the function returns infinity, as its goal is to avoid already visited edges. However, if the path can not be completed without going through such an edge, it can still be included in the path for the second time.

Algorithm 2: Finding partial route algorithm

Input: start node id *a*, previous node id *p*, desired distance from start node *distance*, desired distance from previous node *distance_previous*

```
dijkstra_distances, dijkstra_paths = single_source_dijkstra (G, b, weight =
  weight_func(), cutoff = distance);
possible_nodes = dijkstra_distances.keys();
best_score = ∞;
best_b = a;
for i, k in enumerate (possible_nodes) do
    d = dist_func (a, k);
    d_prev = dist_func (previous, k);
    score = (d - distance)^2 + (d_prev - distance_previous)^2;
    if score < best_score then
        best_score = score;
        best_b = k;
return dijkstra_paths[best_b];
```

2.3. Dijkstra algorithm

Dijkstra's algorithm has proven to be one of the most significant algorithms in graph theory. The algorithm is used to find the shortest paths from a single source vertex to all other vertices in a weighted graph, therefore, finding the definitive shortest path between two nodes 'a' and 'b'. There are many variations of the standard Dijkstra's algorithm, including distance cut-off modification, which was incorporated in the part of the proposed algorithm, responsible for finding sub-route of the desired length (presented in Alg. 2). By introducing a maximum allowable distance from the source vertex, the algorithm seeks all paths that meet the allowable distance requirement. The allowable distance can be affected by a custom weighting function, as described in Sec. 2.2. It is worth pointing out, that if it is desired (for example, by not only taking the length aspect of the edges but other attributes, such as the mentioned surface type), this can result in the found paths being longer than the allowable distance.

2.4. Path-finding algorithm

In this section, the proposed algorithm is described in detail. The algorithm was created for the real-life scenarios. Its objective is to find a directed cycle whose total length is within the permissible range and which has optimal edge attributes. The proposed algorithm is greedy, as it divides the task of finding the whole route into smaller n sub-routes, where it tries to maximize certain objective.

2.4.1. Short overview

The operating principle of the algorithm can be briefly described as follows: The algorithm will search for a route by determining smaller sub-routes. The estimated length of each sub-route is

equal to the quotient of the distance and the number of sub-routes. Starting from the starting vertex, the algorithm searches locally for the best vertex to go to. After one, the algorithm looks for a route to it and adds the distance to the total. After determining the sub-route, the algorithm considers whether it should choose a route back to the start vertex (completing the journey) or continue searching for sub-routes.

2.4.2. Finding sub-routes

Finding a sub-route is presented in detail in Alg. 2. Starting from the starting vertex 'a', the algorithm performs a search of routes to all vertices to which the shortest path (taking into account the weight function) is less than or equal to the estimated length of the sub-route (by the Dijsktra's algorithm with cut-off distance). After finding vertices that meet the condition, the best node is chosen using the following metric:

$$score(a, b, p) = (d(a, b) - R_a)^2 + (d(p, b) - R_p)^2 \tag{2}$$

Where:

- a, b, p are respectively: starting, currently considered and previously chosen nodes,
- d is the distance function (see Sec. 2.1),
- R_a, R_p is the estimated distance from the nodes a and p.

The Dijsktra's algorithm used to find the shortest path between nodes in a graph by iteratively exploring the consecutive neighboring nodes and updating the shortest known distance from the start node to every other node met, until the shortest path is found. The Dijkstra's algorithm is a greedy algorithm.

In order to introduce randomness to the algorithm, the finding sub-routes part can be altered to keep the score for every node found by the Dijkstra algorithm. Having calculated all the scores and sorted them accordingly, a random node from among the top x nodes can be chosen, therefore making the choice of sub-path random, while still selecting a close-to-optimal path. However, this change significantly increases the computational complexity of the algorithm due to the sorting required.

2.4.3. Main algorithm

In the main loop of the algorithm, after finding the local sub-route, the algorithm looks for the shortest return route (using Dijkstra's algorithm) to the starting node. If the route found so far has a length whose total distance has an error within $\pm Er$ of the desired distance D, the algorithm returns this route. If the route is too short, it continues the search. If it is too long, (while the route that would contain the return path from the current vertex 'a' is too short), the algorithm tries again to find vertex 'b', this time with half of the R_a range from the previous iteration of the sub-route search. In case the algorithm fails to find a route so that its total length is within the intended error, during K a such number of iterations (during which R_a is halved each time), the algorithm returns a route with a return path from vertex 'a', thus the length of such a route is less than $D \cdot (1 - Er)$ of the intended length. This acts as a safe mechanism, preventing the algorithm from looping infinitely. Even though this is unlikely to happen, the

algorithm was created in mind of real-world application, in which such a situation can occur (for example, if there are multiple dead-ends or there are not enough roads and the graph is simply too sparse).

3. Experiments

In the experiments, we considered the task of finding a loop for a biking trip, considering the desired length of the trip and the bike type.

3.1. Graph data

In the experiments, we used publicly available OpenStreetMap (OSM) data. OSM is a collaborative project aimed at creating a free, editable map of the world. It is built by a community of contributors who add and update geographical information. OSM is widely used in many applications, such as navigation, urban planning disaster response and even environmental analysis. It provides a great source of information for traffic management, including bicycles. By using its API, we are able to convert the real-world part of a map into a representative graph, as the OSM is built as such graph, containing vertices and edges with certain attributes. There are, however certain limitations. Some edges may be mislabelled or may miss certain attributes.

3.2. Bicycle types and surface attributes

The main task revolves around finding a good path, that will be favorable to the type of bicycle chosen. For example, if we consider racing bicycles, a desired path should be level and related to fast traffic. On the other hand, mountain bikes, and dirt roads should be chosen over the city's streets. To achieve that, the weighting function described in Sec. 2.2 was introduced. As the OSM agenda allows for multiple surface types as edge attributes (in fact, it can be anything), to facilitate the task of assigning multipliers for surface types, we considered three separate categories, to which different types of surface may apply: 'Road' - a surface characterized by a level surface, for high-speed driving, like asphalt, 'Off-road' - a surface characterized by uneven ground, for off-road riding, like dirt and 'Neutral' - suitable but sub-optimal for the extreme bicycle types. This category includes types found mostly on short distances in cities - such as paving stones. This way, for each bike type, we only need to assign multipliers for those three categories. These settings can be stored and easily modified in the '.yaml' file. In the experiments, we used the following settings for bike types and surface categories:

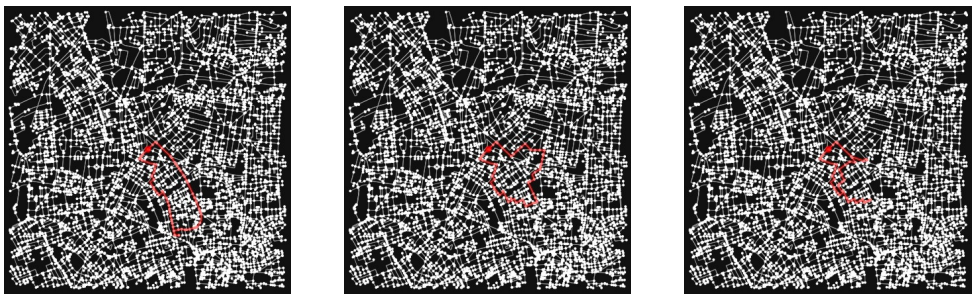
```
Bikes:
    Road:
        {'Road_paths': 0.25, 'Off_road_paths': 2.0, 'Neutral_paths': 1.25}
    Mountain:
        {'Road_paths': 2.0, 'Off_road_paths': 0.25, 'Neutral_paths': 1.0}
    Neutral:
        {'Road_paths': 1.0, 'Off_road_paths': 1.0, 'Neutral_paths': 1.0}
```

```
Surfaces:
    Road_paths:
        ['asphalt', 'cobblestone', 'paving_stones',
        'sett', 'concrete', 'concrete:plates']
    Off_road_paths:
        ['ground', 'grass_paver', 'fine_gravel', 'gravel', 'earth',
        'dirt', 'unpaved', 'pebblestone', 'grass', 'grass_paver']
    Neutral_paths:
        ['paved', 'unhewn_cobbletone', "wood", "compacted", 'grass_paver']
```

The occurring problem we faced was the ambiguity connected with the surface types. Some surface types, such as 'paved' could represent both paths paved with paving stones or just hardened soil.

3.3. Experimental settings

As previously mentioned in Sec. 2.4.3, the main goal of introducing the proposed algorithm was the ability for the user to create a route of desired length and attribute quality, that start and end positions were the same. In the conducted experiments, we have found that the n number of sub-routes equal to 5 yields the most satisfactory results for the user. The routes created with lower n parameter tended to yield straight routes, going just out and back to the start, while the greater n made the route cross over itself and twist. This can be observed in Fig. 1. There was no other simple way to select this parameter, other than manually, as there is no simple way of measuring 'roundness' but by visual observation.



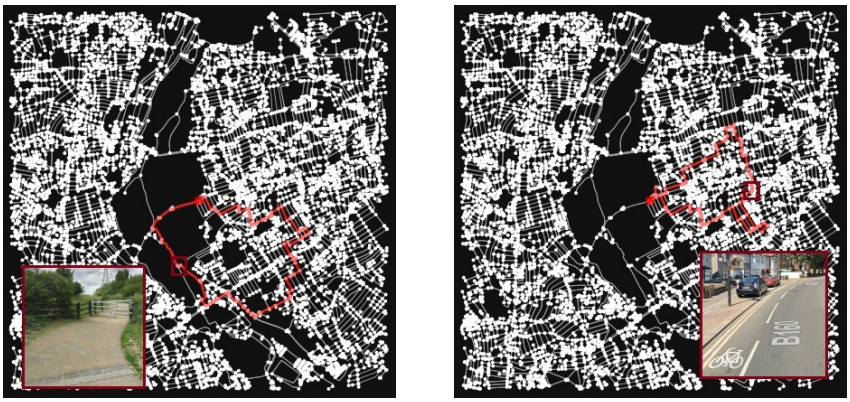
(a) n = 3 (b) n = 5 (c) n = 7

Figure 1: Samples of 4km routes created in London (Islington) for racing bike-type with different number of n sub-routes

The error of ±5% was chosen for the desired distance. Such error gives the algorithm some flexibility in finding good sub-routes while maintaining the total length of the route objective.

4. Results

In this section obtained results are presented. We tested the algorithm with different desired lengths and bike types. In Fig. 2, route comparison of example 9km routes between two opposite bike-types are presented. The starting position was chosen near Walthamstow Wetlands, London, as the London roads are well documented in OSM and the nearby park allowed the algorithm to choose paths further from busy roads (while still prioritizing the 9 km-length objective). As can be observed in the sub-figures, provided examples of surfaces in the two distinct routs differ greatly, as they fit their specific bike type. Route chosen in Fig. 2a leads through green areas, with rough surfaces, while on the contrary, Fig. 2b presents a part of an asphalt bike-lane in the city's busier area.



(a) 'Mountain' bicycle (b) 'Racing' bicycle

Figure 2: Comparison between obtained 9km routes for extreme bike-types. For better insight, sample areas of the route were demonstrated using Google Street-View.

Table 1
Summary of edges' surface types in routes generated in Fig. 2. The 'NaN' indicates types not considered by any of the categories or an unlabelled edge.

Surface-category	A Mountain bike	Racing bike
Road	61	76
Off-road	7	0
Neutral	2	1
NaN	34	50

The algorithm was tested for the mean absolute percentage error of the obtained routes. To do so, two tests were performed, one for short and the other for long routes. The former measured the MAPE of 11 routes, ranging from 2km to 6km, with an interval of 0.4 km. The resulting error was 3.24%. The ladder measured routes ranging from 20km to 60km (with 4km intervals). The obtained MAPE was equal to 2.74%. Both of the obtained errors were within the acceptable range (which was set to 5%, as mentioned in Sec. 3.3).

During tests we were challenged with algorithms limitations, mainly involving insufficient or incorrect edge labelling. Furthermore, the amount of possible surface types is very extensive, which proved to be difficult to deal with and due to which we had to limit the considered surface-types. Another draw-back we had faced was finding routes in the secluded areas, where graphs were often disconnected, making it impossible to find the route meeting the criteria or making the algorithm loop infinitely (which we have since secured, as described in Sec. 2.4.3. As our approach is rather unique, as it considers indirect features of the edges in general cycle-finding problem, it was hard to properly asses the method. However, this makes the research more impactful and the obtained results indicate the correctness of the approach.

5. Conclusion

The proposed algorithm can be used to find a directed cyclic path that needs to meet certain edge conditions in real life. In this paper, the task of finding a cycling route for a specific bike type and the assigned total length was presented. Based on the obtained results, the algorithm performs well for the considered tasks, generating routes within the acceptable distance error and adapting to the desired bike type. The limits of the algorithm are mainly due to miss-labeling of the edges and specific surface topology that may differ from location to location. Although these issues can be addressed by fine-tuning the surface categories and hyperparameters of the algorithm, it prevents the method from being fully universal. Nevertheless, the presented approach is flexible and can be applied to similar tasks. Overall, the algorithm performs well and we plan to develop it further in future work.

Acknowledgments

This work was supported by the Rector's mentoring project "Spread your wings" at the Silesian University of Technology.

References

[1] G. Rahmanifar, M. Mohammadi, A. Sherafat, M. Hajiaghaei-Keshteli, G. Fusco, C. Colombaroni, Heuristic approaches to address vehicle routing problem in the iot-based waste management system, Expert Systems with Applications 220 (2023) 119708.

[2] E. Alhenawi, R. A. Khurma, R. Damaševicius, A. G. Hussien, Solving traveling salesman problem using parallel river formation dynamics optimization algorithm on multi-core architecture using apache spark, International Journal of Computational Intelligence Systems 17 (2024) 1–14.

[3] M. Ogiołda, The use of clonal selection algorithm for the vehicle routing problem with time windows, in: Symposium for Young Scientists in Technology, Engineering and Mathematics, 2017, pp. 68–74.

[4] S. Sundarraj, R. V. K. Reddy, B. M. Babu, G. H. Lokesh, F. Flammini, R. Natarajan, Route planning for an autonomous robotic vehicle employing a weight-controlled particle swarm-optimized dijkstra algorithm, IEEE Access (2023).

[5] W. Wei, F. Gao, R. Scherer, R. Damasevicius, D. Połap, Design and implementation of autonomous path planning for intelligent vehicle, Journal of Internet Technology 22 (2021) 957–965.
[6] M. Yao, H. Deng, X. Feng, P. Li, Y. Li, H. Liu, Improved dynamic windows approach based on energy consumption management and fuzzy logic control for local path planning of mobile robots, Computers & Industrial Engineering 187 (2024) 109767.
[7] M. Reda, A. Onsy, A. Ghanbari, A. Y. Haikal, Path planning algorithms in the autonomous driving system: A comprehensive review, Robotics and Autonomous Systems (2024) 104630.
[8] J. Yu, C. Chen, A. Arab, J. Yi, X. Pei, X. Guo, Rdt-rrt: Real-time double-tree rapidly-exploring random tree path planning for autonomous vehicles, Expert Systems with Applications 240 (2024) 122510.
[9] X. Zhou, J. Yan, M. Yan, K. Mao, R. Yang, W. Liu, Path planning of rail-mounted logistics robots based on the improved dijkstra algorithm, Applied Sciences 13 (2023) 9955.
[10] O. W. Khalid, N. A. M. Isa, H. A. M. Sakim, Emperor penguin optimizer: A comprehensive review based on state-of-the-art meta-heuristic algorithms, Alexandria Engineering Journal 63 (2023) 487–526.
[11] K. Prokop, Grey wolf optimizer combined with k-nn algorithm for clustering problem, IVUS 2022: 27th International Conference on Information Technology (2022).
[12] D. Połap, Neuro-heuristic analysis of surveillance video in a centralized iot system, ISA transactions (2023).
[13] S. Z. Kurdi, M. H. Ali, M. M. Jaber, T. Saba, A. Rehman, R. Damaševičius, Brain tumor classification using meta-heuristic optimized convolutional neural networks, Journal of Personalized Medicine 13 (2023) 181.
[14] K. Prokop, D. Połap, Heuristic-based image stitching algorithm with automation of parameters for smart solutions, Expert Systems with Applications 241 (2024) 122792.
[15] S. Darvishpoor, A. Darvishpour, M. Escarcega, M. Hassanalian, Nature-inspired algorithms from oceans to space: A comprehensive review of heuristic and meta-heuristic optimization algorithms and their potential applications in drones, Drones 7 (2023) 427.
[16] W. Kazimierski, N. Wawrzyniak, M. Wlodarczyk-Sielicka, T. Hyla, I. Bodus-Olkowska, G. Zaniewicz, Mobile river navigation for smart cities, in: Information and Software Technologies: 25th International Conference, ICIST 2019, Vilnius, Lithuania, October 10–12, 2019, Proceedings 25, Springer, 2019, pp. 591–604.
[17] M. Włodarczyk-Sielicka, N. Wawrzyniak, Problem of bathymetric big data interpolation for inland mobile navigation system, in: Information and Software Technologies: 23rd Interna- tional Conference, ICIST 2017, Druskininkai, Lithuania, October 12–14, 2017, Proceedings, Springer, 2017, pp. 611–621.
[18] I. Garczyńska, A. Tomczak, G. Stępień, L. Kasyk, W. Ślączka, T. Kogut, Applicability of machine learning for vessel dimension survey with a minimum number of common points, Applied Sciences 12 (2022) 3453.
[19] R. Lewis, P. Corcoran, Finding fixed-length circuits and cycles in undirected edge-weighted graphs: an application with street networks, Journal of Heuristics 28 (2022) 259–285. URL: https://doi.org/10.1007/s10732-022-09493-5. doi:10.1007/s10732-022-09493-5.