information

Article

Property Graph Framework for Geographical Routes in Sports Training

Alen Rajšp * and Iztok Fister, Jr.

Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška Cesta 46, 2000 Maribor, Slovenia; iztok@iztok-jr-fister.eu
* Correspondence: alen.rajsp@um.si

Abstract: Presenting real-world paths in property graphs is a complex challenge of identifying and representing the properties of routes and their environments. These property graphs serve as foundational datasets for generating smart sports training routes, where route features such as terrain, bends, and hills critically influence the route design. This paper outlines a method for identifying key parameters of real-world paths and encoding them into property graphs. The proposed method has significant implications for sports event planning, particularly in designing route-based training that meets specific athletic challenges. The research concludes by presenting a case study in which a property graph that enables cycling route generation was created for the country of Slovenia, and a sample training route was generated.

Keywords: property graph; geographical maps; smart sports training; data mining; data fusion; OpenStreetMap; digital elevation model (DEM)

check for updates

Academic Editors: Barbara Pes and Andrea Loddo

Received: 12 November 2024
Revised: 20 December 2024
Accepted: 31 December 2024
Published: 7 January 2025

Citation: Rajšp, A.; Fister, I., Jr. Property Graph Framework for Geographical Routes in Sports Training. *Information* **2025**, *16*, 30. https://doi.org/10.3390/info16010030

1. Introduction

Humans have been creating maps and map-like structures for thousands of years [1]. The purpose of a map is to serve as a tool that helps people perceive the universe in which they live, and maps were originally used to explain to others the location or space experienced by the person who created the map [2]. In modern times, solutions like GPS (global positioning system) have allowed us to combine knowledge about map making with knowledge about our current location to allow a modern approach to navigation in which we can know our precise location and receive instructions on where to move in real time [3].

Maps have been tailored to the understanding of their end users, who used them for path-finding between two locations [1]. The problem of pathfinding can be divided roughly into three sub-problems. In the first step, we need to have knowledge about the environment and represent an abstraction of this environment in some form. In the next step, we must find a path that is a solution to some predetermined goal. The last step is utilizing this solution, which involves how the end user will use this route and how it will be represented to him.

Computers are now used to calculate and generate routes, and their conception of location has been fundamentally different. These routing techniques depend on the domain of data that can be used, and they include the following structures for their representation:

- Graphs [4]—are representations of concepts described by a set of nodes and edges between them.

- Property graphs [5] are directed, labeled multi-graphs whose specialty is that each node or edge can have a set of property–value pairs, which describe them in detail.
- Trees [6] are hierarchically organized data structures in which each node except for the root node has exactly one parent node; the root node has none. File systems on computers are mostly organized in tree structures.
- Grids [7] are structures that divide the environment into a grid of equally sized cells.

Pathfinding is a well-known problem that has been solved algorithmically since the 19th century [8]. Pathfinding algorithms are used in problem-solving in robotics [9], logistics [10], and the everyday use of global navigation systems [11]. Using navigation is one of the most common activities on mobile phones, with 48% of all smart mobile device users using a mobile phone to help them navigate at least once a month in 2022 [12]. The goal of pathfinding is addressed most commonly to finding the shortest route [13] in a given search space, for which algorithms like Dijkstra [14], A* [15], Bellman–Ford [16], and Floyd–Warshall [17] are used on a weighted graph. Other known problems based on pathfinding are the vehicle routing problem [18], the traveling salesman problem [19], the Chinese postman problem [20], the Network flow problem [21], the multi-objective shortest path problem [22], and others.

Sometimes, we do not want to find the shortest route within a search space but a route that fits several parameters, as is often the case in sports training planning. The preparation of the path to be taken by an athlete for the purpose of sports training belongs to the sports training planning phase [23].

This paper is, however, not concerned with finding solutions to pathfinding; it provides a method for generating property graph datasets that can be used in such pathfinding methods, particularly for generating cycling training routes. The developed solution is concerned with the identification of individual path properties and how to represent them in a property graph.

The scope of the problem that this paper tackles is concerned primarily with data collection, as well as the data preprocessing phase of the data analysis process, as illustrated by the green brackets in Figure 1, which are sometimes regarded as the two most important steps of data analysis [24]. The property graph serves as a precursor to graph-based pathfinding algorithms. In real-world environments, such approaches based on various forms of graphs have been recently used in training planning for cycling [23] and running [25], as well as travel distance estimation [26].

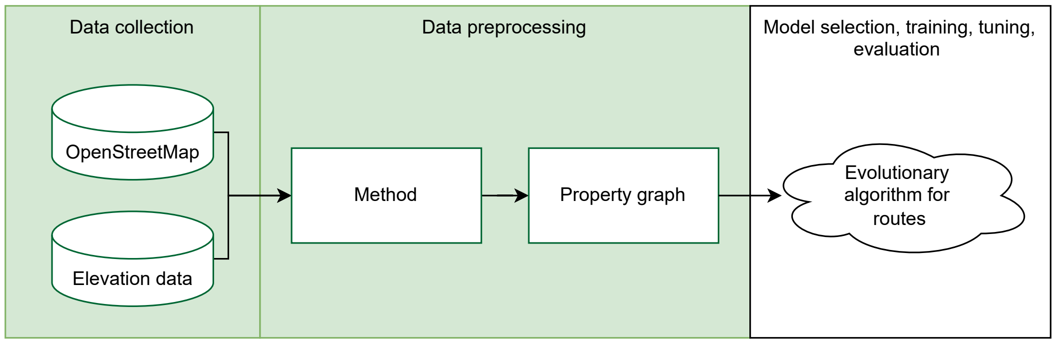

Figure 1. Scope of research.

Data from such graphs can enable the discovery and generation of carefully tailored routes to best replicate the real environments the athletes face during real competitions. From such data, athletes, and trainers can find environments similar to profiles of different cycling races [27] and marathons for which the planned route data are publicly available, meaning that elevation, curviness, and surface types can be carefully replicated in preparation for competitions.

The main contributions and unique values of this study are as follows: (1) presenting a new approach to generating complex geographical properties in the form of a property graph and (2) a method that can form a prerequisite to multi-parameter sports training generation in real-world environments.

The remainder of this paper is organized as follows: Section 2 describes the data collection process for property graph generation, including an analysis of data sources and selection criteria. Section *Data Processing Tools* presents the tools used for data preprocessing and manipulation, including APIs and databases for handling geographical data. Section 3 outlines the method for generating property graphs, starting with a summary of the original method and continuing with the detailed extensions introduced in this study (Section *Extending the Original Method*).

Section 4 explains how to represent path and intersection features within the property graph framework. The results of the proposed method, including a case study focused on Slovenia and the generation of sample cycling routes, are presented in Section 5. Section 6 provides a discussion of the findings, including the advantages and limitations of the approach, as well as its potential applications. Finally, Section 7 concludes the paper and suggests future research directions.

2. Data Collection for Geographical Based Property Graph Generation

To create a geographical property graph from real-world paths, a map with all the paths and their intersections is needed, as well as any additional data that can explain these roads further. Tools also have to be identified that allow the processing and manipulation of these data.

Data sources that offer geographical data were identified in the first step. The analysis of sources was focused on the data that the resources offered, as well as their licensing. In the first step, the base dataset was selected, with the requirement that it had roads and their intersections, that it was licensed in such a way that a dataset could be derived from it freely, and that it had the best data possible.

The analyzed solutions that were fit for use and satisfied the requirements were as follows: OpenStreetMap [28], US Geological Survey (USGS) National Map [29], and Natural Earth [30], as shown in Table 1. It was determined that only OpenStreetMap [28] offered geographical data that were available worldwide, which was why it was selected for property graph generation.

Table 1. Analyzed sources for roads and intersections.

Source	License	Data
OpenStreetMap	ODbL	Worldwide
US Geological Survey National Map	public domain	United States
Natural Earth	public domain	Worldwide, roads only US

The OpenStreetMap is an open-source project that offers and maintains geographical data, and it is maintained by volunteers. The base building blocks of the OpenStreetMap data are nodes, ways, relations, and tags. Nodes are individual points of latitude and longitude with a defined id (as shown in Figure 2 by small dots). Nodes are connected in ways that are ordered lists of 1 to 2000 nodes that are used to represent linear features, such as roads, rivers, buildings, and areas (as shown in Figure 2 by differently colored and shaded parallel lines demonstrating different ways that are composed of nodes), although an individual node can be a part of more than one way (e.g., a road intersection).

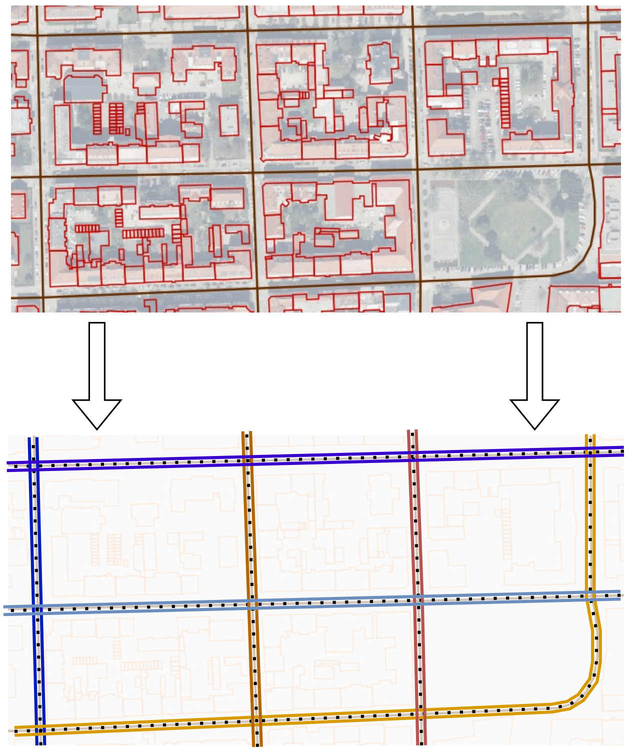

Figure 2. OpenStreetMap data model illustration (background source of roads and buildings: [31,32].

When more than 2000 nodes are needed to describe a feature, ways are connected with relations; for example, a list of ways could describe a named road. All of these elements can then be described further by tags, which are key–value elements.

One part of the missing data in databases such as OpenStreetMap is the elevation of individual nodes. These data can be retrieved from DEMs (digital elevation models), which are mostly square grid representations of the elevation of geographical areas (Figure 3), but they can also be represented by triangulated irregular networks [33]. In DEMs, elevation is measured in terms of terrain elevation and not surface elevation (e.g., trees or buildings) [34]. The resolution greatly influences errors of measurement, which is very important in hills and mountains, where the elevation of two nearby points may vary greatly.

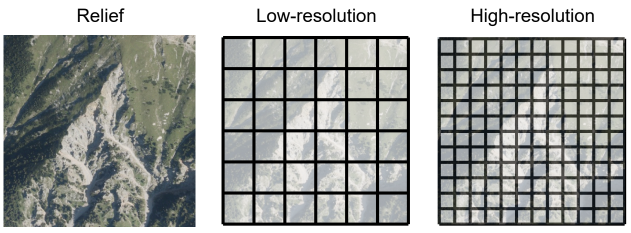

Figure 3. Illustration of regular grid DEM (background source: [31]).

Multiple sources of such elevation models exist, so which one to choose depends on the area the researcher is trying to investigate and generate a property graph for. The following datasets were identified and analyzed: ASTER [35], ETOPO [36], Copernicus [37], EU-DEM [38], NED [29], NZ-DEM [39], and SRTM [40]. The datasets differ in their resolutions and coverage areas, as shown in Table 2. For our purpose, we selected the EU-DEM [38] model since we were to create a property graph of Slovenia (a country in the European Union), so the model was deemed best suited for our purpose.

Table 2. Analyzed sources for digital elevation models.

#	Coverage Area	Resolution (arc s)	Resolution (m^2)
ASTER	Worldwide	1	~30
ETOPO	Worldwide	15 and 30	~450 and ~900
Copernicus	Worldwide	1	30 and 90
EU-DEM	European Environment Agency members	1	~25
NED	North America and Mexico, high-res for the US without Alaska	1 and 1/3	~30 and ~10
NZ-DEM	New Zealand	~0.26	8
SRTM	Worldwide	1	~30

Data Processing Tools

The tools needed refer to the manipulation and extraction of data from the two selected datasets, notably the path and road data from OpenStreetMap and elevation data from EU-DEM. The identified and presented solutions were required to be free and open-source, allowing users to host them on their own servers. Additionally, they needed to have a REST (representational state transfer) interface to facilitate API requests.

The OpenStreetMap data are stored in the XML format, which is compressed further into protocol buffers to decrease their size [41]. To query the XML files, a specialized API (application programming interface), Overpass API [42], can be used. This is an API for querying and retrieving data from the OpenStreetMap database. The querying is done using the Overpass QL (Overpass Query Language). The Overpass QL was used to identify routes and intersections, as well as data about the routes, such as the type of road, traffic light signalization, intersections between the routes, and coordinates of routes along the path.

To retrieve elevation data, two well-established API solutions exist: Open Topo Data [43] and Open Elevation API [44]. The solutions allow for querying latitude and longitude pairs and retrieving the elevation at said points using DEM as the source. Both are suitable for use, and we would recommend that researchers use whichever solution they are more familiar with.

3. Comparison with the Original Method for Property Graph Generation

This property graph generation method is an improved version of the previously presented and developed approach [45]. This section summarizes and extends the original method to include missing complex properties.

The two key instruments for representing and understanding routes in a digital environment are graphs and property graph structures. In layman's terms, a graph is a mathematical structure with a set of objects in which some pairs of objects are related [46].

In the domain of computer science, the presentation, manipulation, and querying of graph and graph-like data are conducted with graph database systems [5].

The original method [45,47] used the OpenStreetMap [28] and EU-DEM [38] datasets to create a property graph that represents the intersecting paths between them. The method works roughly as follows: All the intersections of a given area are identified in the first step. In the next step, roads between the intersections are identified, and bidirectional graph edges are created in the last step. The resulting property graph represents all intersections as nodes and roads between them as edges, as shown in Figure 4. The edges are directed and created twice because traveling from N_x to N_y is not the same as traveling from N_y to N_x.

The processed properties of edges are shown in Table 3. The **ascent** and **descent** were processed using the combination of latitude–longitude pairs of the path and DEM data. The road **type** refers to OpenStreetMap tag *highway*.

Table 3. Path/edge properties.

Attribute	Description	Example Value
Ascent	Ascent in meters	71
Descent	Descent in meters	7
Distance	Distance in meters	328.22
Type	Road type	Track
Intersection-AB	Edge from - to node ids	4893616681–4893616518

The extracted properties of intersections are shown in Table 4. The **node_id** and **way ids** refer to the OpenStreetMap properties of the nodes.

Table 4. Intersection/node properties.

Property	Description	Examples
Latitude	Latitude	465592238
Longitude	Longitude	156378701
Node id	node id	305882164
Way ids	Way ids the node belongs to	[27859724, 247410448]

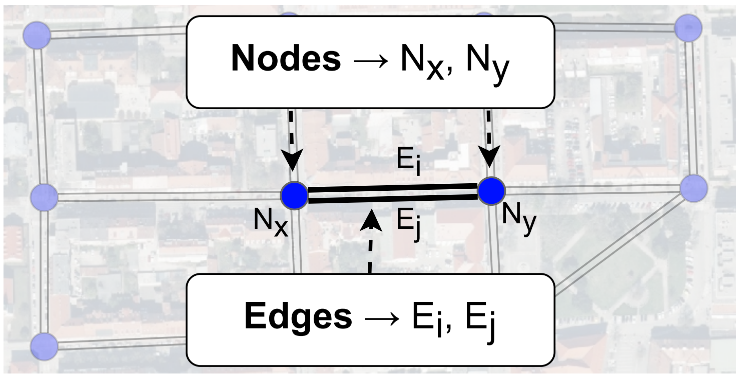

Figure 4. Property graph of intersections (nodes) and roads (edges) (background source: [31]).

The flaws of the (original) method are that the properties of the edges do not include any data regarding the surface of the road (e.g., asphalt, tiles, . . .), the curvature of the road (e.g., how much the road turns), and how hilly it is, which means whether the ascent occurs in a continuous fashion or whether there is a large hill and then a flat expanse. One serious drawback is also that it does not take into account the bends at intersections because crossing the intersection is usually not the same as making a turn. Also, left turns (in right-hand-side driving countries) are harder to achieve due to oncoming traffic than right turns.

Extending the Original Method

The original method allows for the creation of a basic property graph, which was extended greatly in this proposal. This section will summarize the differences between the original method and show which new properties were introduced in the property graph. The original property graph contained only the intersection properties presented in Table 4 and the path properties presented in Table 3. The properties that remained unchanged in the nodes (intersections) were latitude, longitude, and node id. Only the ascent and descent properties remained unchanged in the property graph's edge (path) elements.

The nodes were updated, and individual intersections were split into multiple nodes to include the intersection angles of the nodes. The properties that were introduced to nodes were node elevation and whether the intersection contained traffic signals or not; the final version of the intersection element is shown below in Table 5, displaying the newly added properties.

Table 5. Intersection/node properties of the updated method.

Property	Description	New
Latitude	Latitude	−
Longitude	Longitude	−
Node id	OpenStreetMap node id	−
Elevation	Elevation in meters	+
Traffic signals	Does the node contain traffic lights	+

The calculation of edge distance was updated to increase accuracy, and new properties of curviness, hilliness, and the number of traffic signals on the paths were introduced, as shown in Table 6 below.

Table 6. Path/edge properties of the updated method.

Attribute	Description	New
Ascent	Ascent in meters	−
Descent	Descent in meters	−
Distance	Distance in meters	−
Type	Road type	−
Intersection-AB	Edge from - to node ids	−
Surface	Surfaces type the path is composed of if available	+
Total angle	Sum of changes in direction in degrees over the whole path	+
Curviness	Ratio between total angle and length of the path	+
Traffic lights	Number of traffic lights on the path	+
Hill flat	Meters of path that have a hill gradient 0–1%	+
Hill gentle	Meters of path that have a hill gradient 1–3%	+
Hill moderate	Meters of path that have a hill gradient 3–6%	+
Hill challenging	Meters of path that have a hill gradient 6–9%	+
Hill steep	Meters of path that have a hill gradient 9–15%	+
Hill extremely steep	Meters of path that have a hill gradient +15%	+
Bicycle access	Indicates whether a route is accessible by bicycle	+
Foot access	Indicates whether a route is accessible by foot	+
Car access	Indicates whether a route is accessible by car	+

4. Representing Path and Intersection Features in a Property Graph

The first step of representing path properties is identifying which features should be represented in a property graph and their transformation into presentable features. It should be noted that only intersection nodes are saved in the property graph. Other OpenStreetMap nodes are processed, and edges are created from them.

In the following equations for different properties, the node $(n_1 = n_x)$ represents the starting node of the edge, and node $(n_N = n_y)$ represents the ending node of the edge.

4.1. Intersection Elevation

The feature relates to the elevation attribute of each node (intersection). These data can be mined from the latitude–longitude pairs of nodes and their corresponding elevation on the DEM model. The attribute is retrieved by querying the DEM API (either OpenTopoData or Open Elevation) for each of the graph nodes.

4.2. Ascent and Descent

The feature relates to the total ascent and descent that the person achieves when traveling between two nodes. It is not enough simply to check the starting node (intersection) and ending node (intersection) elevation since it may be that the elevation could be roughly the same, as shown in Figure 5, while there is a hill between the points.

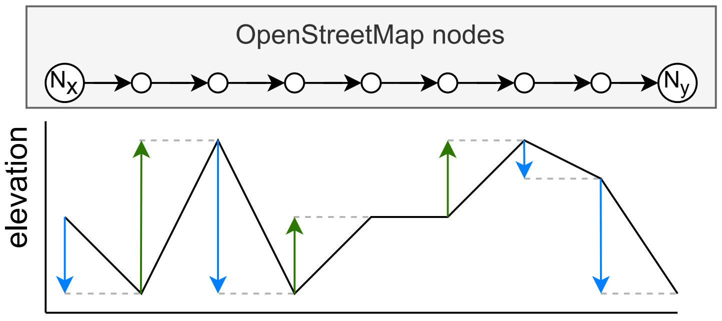

Figure 5. Property graph of intersections (nodes) and roads (edges).

The ascent between nodes n_x and n_y is calculated as a sum of all positive changes in elevation, as shown in (1). This means that, if you move from node n_i to node n_{i+1}, the change in elevation is calculated as $H(n_{i+1}) - H(n_i)$, or 0 if there was a descent between the two nodes.

$$Ascent(n_x, n_y) = \sum_{i=1}^{n-1} \max(0, H(N_{i+1}) - H(n_i)) \tag{1}$$

The descent between nodes n_x and n_y is, therefore, calculated as the sum of all the negative changes in elevation, as shown in (2). This means that, if you move from node n_i to node n_{i+1}, the change in elevation equals $H(n_i) - H(n_{i+1})$, or 0 if there was an ascent between the two nodes. $H(N_i)$ represents the elevation value at the node N_i.

$$Descent(N_x, N_y) = \sum_{i=1}^{N-1} \max(0, H(N_i) - H(N_{i+1})) \tag{2}$$

4.3. Distance

This feature represents the measure of the real-world length of the path connecting two nodes. The distance between nodes n_x and n_y is calculated as (3), where the distance is equal to the sum of all distances for all the succeeding nodes between n_1 and n_n.

$$Distance = \sum_{i=1}^{N-1} d(n_i, n_{i+1}) \tag{3}$$

The individual distance ($d(n_i, n_{i+1})$) between two nodes (n_i and n_{i+1}) with known latitude (ϕ_i, ϕ_{i+1}) and longitude (λ_i, λ_{i+1}) can be calculated using two approaches. The first approach uses the Haversine formula, known commonly as the great-circle distance formula, proposed by [48], and the second method uses the Pythagorean theorem. The drawback of the first method is that the calculated distances assume that both points are on the same elevation, while the drawback of using the Pythagorean theorem is that it does not take the curvature of the Earth into account.

In the following equations for distance calculation, r represents the radius of the Earth, which is approximated to 6378.1 km, as proposed by [49]. $\Delta\lambda$ represents the difference in longitudes between two points ($\lambda_i - \lambda_{i+1}$), and $\Delta\phi$ the difference in latitudes between the two points ($\phi_i - \phi_{i+1}$).

The first approach to using the Haversine Formula (4) is calculated following the approach used in [50]. It should be noted that, when using the Haversine formula, the latitudes and longitudes must be expressed in radians and not degrees.

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{4}$$

The second approach to using the Pythagorean theorem is calculated following the approach used in [50], but it is extended with the measure of elevation. The full equation for the distance between two points is (5).

$$d = \sqrt{\Delta x^2 + \Delta y^2 + \Delta h^2} \tag{5}$$

where Δx is expressed as

$$\Delta x = \Delta\lambda \times r \times \cos\left(\frac{\phi_1 + \phi_2}{2}\right) \tag{6}$$

and Δy is expressed as

$$\Delta y = \Delta\phi \times r. \tag{7}$$

The resolved full and long expression for distance can be resolved as

$$d = \sqrt{\left(\Delta\lambda \times R \times \cos\left(\frac{\phi_1 + \phi_2}{2}\right)\right)^2 + (\Delta\phi \times R)^2 + (\Delta h)^2}. \tag{8}$$

4.4. Splitting the Nodes (Intersections)

When navigating in intersections, where the person must actually go is very important. When traveling over a four-way intersection, the person may choose three different outcomes: the person may turn left, turn right, or cross the intersection, and the choice he/she makes will impact the speed. This means that a simple property graph (Figure 6, where each intersection is represented by only one node and connected with surrounding intersections by double-directed edges) is insufficient in representing the information about real-world turning/changing direction right of way on a property graph path.

A way to resolve individual nodes and their edges must be proposed if we want to model the properties for each of the options the person has when traversing an intersection.

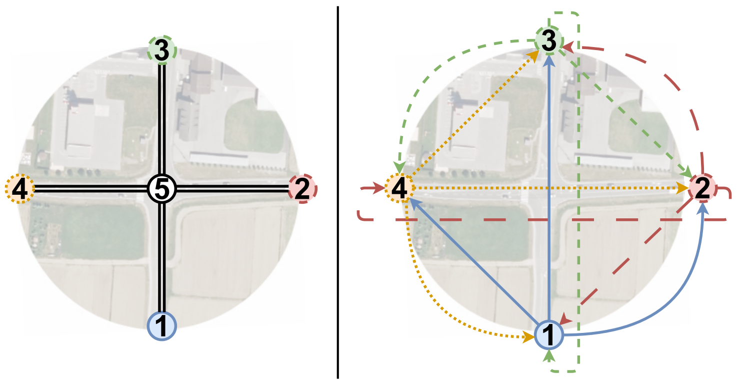

Figure 6. Intersections on a property graph (background source: [31]).

A way to do this is to split the nodes instead of representing an intersection as a single node. We can split a single node into each of the incoming edges to the intersection node, each representing a specific direction or part of the intersection (Figure 7). This allows us to prevent unnecessary information loss when transforming the OpenStreetMap data into a property graph.

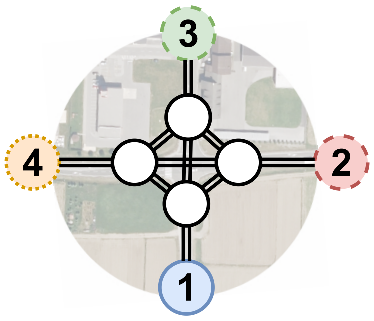

Figure 7. Splitting the nodes (background source: [31]).

This, of course, increases the complexity of the graph, and for each node (intersection), the number of nodes needed for a replacement of an intersection equals the number of nodes it is connected to (N), which means that a four-way intersection will remove one node and add four new nodes. These new nodes need to be connected with edges to represent the choices the person has. The number of new connections can be determined from the maximum number of edges theorem [51], which, applied to our case, results in N_{new1} number of new edges for an undirected graph, as shown in

$$E_{new1} = \frac{N(N-1)}{2} \tag{9}$$

and in an E_{new2} number of new edges for a directed graph (which applies to our property graph of roads and intersections), as shown in

$$E_{new2} = N(N-1). \tag{10}$$

4.5. Calculating Intersection Angles

The calculation of intersection angles can only be performed on previously split nodes (see Section 4.4). The goal of this property is to know the turn that the person has to make when crossing the intersection. Calculating this property is done in two steps. The vector of each of the intersection paths has to be calculated in the first step.

Let us consider three nodes (intersections) called n_a, n_i, n_b that are linked with edges $e_{a,i}$ and $e_{i,b}$. The edges are already constructed relationships, but we also know that each edge is constructed from OpenStreetMap nodes and ways. Assuming that the edge $e_{a,i}$ is constructed from nodes $a_1, a_2, \ldots, a_n$ and edge $e_{i,b}$ is constructed from nodes $b_1, b_2, \ldots, b_m$, and since $e_{a,i}$ terminates in a_n and $e_{i,b}$ starts in b_1, the following is true $a_n = b_1$, as shown in Figure 8.

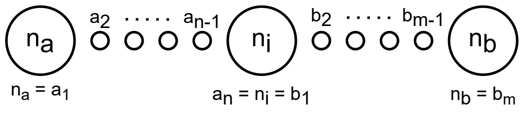

Figure 8. The OpenStreetMap nodes of three intersections and two edges.

We are only interested in the direction of both routes at the intersection. At the intersection node n_i, the direction of the route represented by edge $e_{a,i}$ is determined by the vector from a_{n-1} to a_n, and the direction of the edge $e_{i,b}$ is determined by the vector from

b_1 to b_2. We must first convert the latitude and longitude pairs into Cartesian coordinates (x_E, y_E, z_E). This is done using the approach presented by [52], which states the following:

$$\begin{bmatrix} x_E \\ y_E \\ z_E \end{bmatrix} = \begin{bmatrix} R\cos\phi\cos\lambda \\ R\cos\phi\sin\lambda \\ (1-e^2)R\sin\phi \end{bmatrix}. \tag{11}$$

where R is the Earth's average radius, ϕ represents the geodetic latitude, λ is the geodetic longitude, and e^2 represents the first eccentricity squared. We can simplify the equation further by assuming the Earth is a perfect square, which means the eccentricity equals 0. This gives us the following equation:

$$\begin{bmatrix} x_E \\ y_E \\ z_E \end{bmatrix} = \begin{bmatrix} R\cos\phi\cos\lambda \\ R\cos\phi\sin\lambda \\ R\sin\phi \end{bmatrix}. \tag{12}$$

Once we have the Cartesian coordinates of all nodes, we can show the vectors representing the directions as

$$V_{a,i} = (x_{a_n} - x_{a_{n-1}}, y_{a_n} - y_{a_{n-1}}) \tag{13}$$

for the first edge and

$$V_{i,b} = (x_{b_2} - x_{b_1}, y_{b_2} - y_{b_1}) \tag{14}$$

for the second edge.

To calculate the turning angle at the intersection n_i, we need to find the angle between these two vectors. This can be done using the dot product [53] formula, which is applied to our case and is seen below

$$\theta = \arccos\left(\frac{V_{a,i} \cdot V_{i,b}}{|V_{a,i}||V_{i,b}|}\right) \tag{15}$$

where θ is the angle between the vectors, $V_{a,i} \cdot V_{i,b}$ is the dot product of $V_{a,i}$ and $V_{i,b}$, and $|V_{a,i}|$ and $|V_{i,b}|$ are the magnitudes (lengths) of these vectors.

4.6. Curviness of the Road

This property refers to the shape of the road the user must pass when traveling over the edge between two intersections. This property is important since it affects the person or vehicle dynamics, safety, and speed. To put it in simpler terms, a path that has a lot of curves may not allow such high speeds as a road that is much less curvy, which means that it contains fewer bends or that these bends are less sharp.

There are multiple ways of defining and calculating the curviness of the road, and we present two solutions: (1) the length ratio and (2) the ratio of summed vector angles to the distance.

The length ratio method (1), in our case, would involve calculating the distance of the real-world path (shown as solid lines between individual nodes) versus the distance of the straight-line path (shown as a dashed line) between two points, as shown in Figure 9.

Let us suppose that we have two nodes (intersections) n_a and n_b, that are connected in both ways via edges e_x and e_y. It does not matter whether we want to calculate the ratio for e_x or e_y because they will have the same ratios. The curviness ratio, in this case, would be expressed as a ratio between the straight-line distance ($d_{straight}$) and actual length (d_{actual}) of the path between both nodes. The distance calculations can be done using either the great-circle distance or the Pythagorean theorem, as shown in Section 4.3.

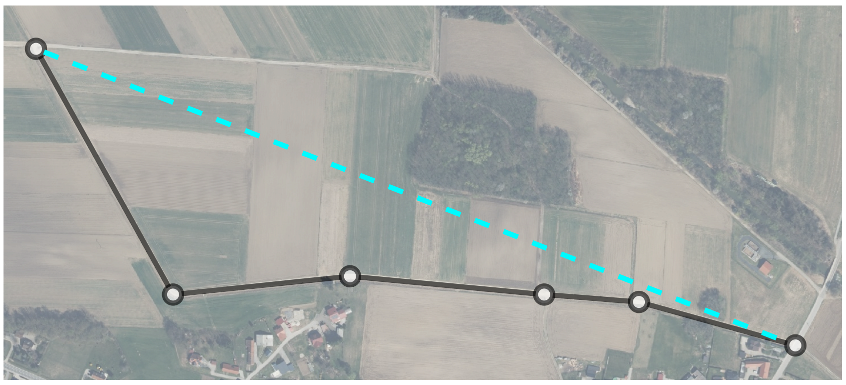

Figure 9. Distances between OpenStreetMap nodes versus straight-line distance between two nodes (intersections) (background source: [31]).

This means that the curviness (C_1) of the edge can be expressed as

$$C_1 = \frac{d_{actual}}{d_{straight}}. \tag{16}$$

This means that an edge with a curviness ratio of exactly one would represent a perfectly straight road in real-world conditions, and an edge with a curviness ratio of two would represent a road that is twice as long as the straight-line distance between two nodes. However, this approach, while really simple and easy to calculate, involves a limitation: a high curviness ratio does not necessarily mean that a road has numerous turns and bends. Instead, it could simply mean that the road follows a lengthy, circuitous route without significant changes in direction, which brings us to the next approach.

The second way (2) to represent curviness is summing the vector angles and dividing them over distance. To do this, we use a similar approach to the approach used in Section 4.5. This means that the angle between each of the three OpenStreetMap nodes can be calculated using the dot product.

Let us again suppose that we have two nodes (intersections), n_x and n_y, that are connected in both ways via edges e_x and e_y. The edge e_x was constructed from OpenStreetMap nodes ($n_1, n_2, n_3, \ldots n_N$) with a known latitude and longitude. In the first step, we must convert all nodes' latitudes and longitudes into Cartesian coordinates, which can be done using the approach presented in (12). Next, the angle must be calculated for all three consecutive nodes n_i, n_{i+1}, n_{i+2}, where $1 <= i + 2 < N$. This is done using the dot product formula, mentioned previously in (15), this gives us angles $\theta_{1,2,3}, \theta_{2,3,4}, \ldots \theta_{n-2,n-1,n}$. The curviness of the road (C_2) can then be defined as a sum of all angles divided by the actual distance the road takes, as shown below

$$C_2 = \frac{\sum_{i=1}^{N-2} \theta_{i,i+1,i+2}}{d_{actual}}. \tag{17}$$

If needed, we can save just the sum of all angles, which would omit the total distance, as shown below

$$C_2 = \sum_{i=1}^{N-2} \theta_{i,i+1,i+2}. \tag{18}$$

This is useful if we want, potentially, to combine the angles with the intersection angles in order to show the total angle change of a route generated from the graph data.

4.7. Hilliness of the Road

The property refers to the intensity and length of the hills that the edge (road) has between two nodes (intersection). This property is needed because the ascent and descent properties do not sufficiently describe whether the path's hills are steep or gradual. The steepness influences the maximal mean power values of cyclists [54] severely and the energy expenditure of runners [55]. The incline of the slope is most usually expressed in gradient/slope and most commonly expressed in %. As such, an x% slope is represented by an x increase in elevation meters ($m_{elevation}$) over $100x$ horizontal meters ($m_{horizontal}$), as defined by [56] and shown in the equation below:

$$slope\ (\%) = \frac{m_{elevation}}{m_{horizontal}}. \tag{19}$$

Because hills vary by intensity, the best way is to introduce multiple categories and properties for hills with different gradients. So, in the first step, we need to categorize the hill gradients into intervals. The hills can, theoretically, have gradients between $-\infty$ for downhill and ∞ for uphill. We must select a suitable number of categories. More categories will allow for finer distinctions between different gradients, which is beneficial for detailed analysis. However, too many categories can lead to a complex system that uses many resources for creation and use.

Suppose we want to have N categories with intervals $(I_1, I_2, I_3, \dots I_N)$ for hill gradients (g). Each of the gradients can, therefore, be classified into one of the categories, where the first interval (20) occupies the space between negative infinity and X_1, and the last interval (23) occupies the space between X_{N-1} and positive infinity. All the other intervals in between (e.g., (21) and (22)) have both maximum and minimum bounds.

$$I_1 = \{g \in \mathbb{R} \mid g < X_1\} \tag{20}$$
$$I_2 = \{g \in \mathbb{R} \mid X_1 \leq g < X_2\} \tag{21}$$
$$I_{N-1} = \{g \in \mathbb{R} \mid X_{N-2} \leq g \leq X_{N-1}\} \tag{22}$$
$$I_N = \{g \in \mathbb{R} \mid g > X_{N-1}\} \tag{23}$$

Following the approach proposed by [57], we classified the hill gradients into six categories, as outlined in Table 7.

Table 7. Cycling hill gradient categories.

Gradient	Category Name	Description
0–1%	Flat terrain	Level terrain offering minimal resistance.
1–3%	Gentle climb	Mild incline, easy for most cyclists.
3–6%	Moderate climb	Noticeable incline may lead to gradual tiredness.
6–9%	Challenging climb	Difficult for regular riders, quite hard for beginners.
9–15%	Steep climb	Intensely challenging, causes significant strain over time.
15%+	Extremely steep climb	Extremely demanding for all, with sustained difficulty.

The six categories are as follows: (1) **flat terrain**, which indicates essentially level ground; (2) **gentle climb**, representing a mild incline that is generally easy to climb for most cyclists; (3) **moderate climb**, representing an incline that could lead to gradual fatigue; (4) **challenging climb**, denoting climbs that are difficult even for regular cyclists and hard for beginners; (5) **steep climb**, which represents a very challenging gradient causing significant strain; and (6) **extremely steep climb**, describing gradients that are exceedingly demanding for all cyclists and very painful to sustain. This means that each edge will have six additional numerical properties.

But how do we know in which category we classify a single hill? Figure 10 shows the issue roughly by defining the hills based on their appearance. The X-axis shows the distance traveled and the Y-axis the elevation. In the case of *(a)*, this is fairly simple since we can determine that the hill started when the ascent started and ended when the road flattened out; we also have a nice, continuous gain in elevation, making the hill gradient fairly simple to calculate. This, however, changes when a hill of type *(b)* is considered. We can see a fairly gentle rise in elevation, which then changes into a steep elevation change and then plateaus into a gentle hill again. We might even arrive at scenario *(c)*, which starts off as a gentle elevation gain and then plateaus and continues with a steep gradient; sometimes the road may even go down a bit, like in case *(d)*, or not the actual road but the data behind it, since the DEMs may involve some small errors.

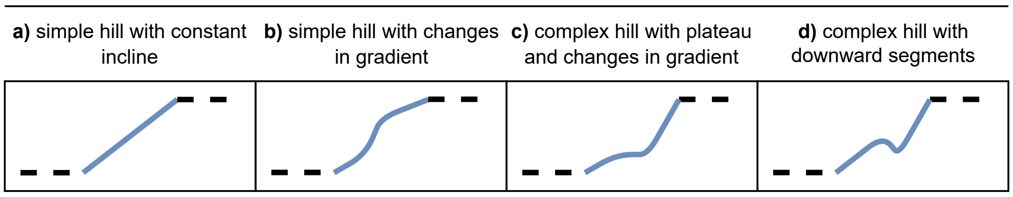
a) simple hill with constant incline
b) simple hill with changes in gradient
c) complex hill with plateau and changes in gradient
d) complex hill with downward segments

Figure 10. Scenarios of different hill types.

Other issues resulting from the DEM that must be taken into account are as follows: (1) their precision, which, in most cases, is 1 m, which means that, in non-steep hills, the increase in elevation may appear to occur step-wise (e.g., 1 1 2 2), while, in reality, it is not; and (2) their resolution in cases of very steep hills, where, even inside the squares, the elevation may be severely different.

4.8. Traffic Signals

Some intersections may contain traffic lights, which means they may be less suitable for travel since the user may lose some time traveling over them waiting for the green light. The traffic signals are often saved in OpenStreetMap nodes [58] using the *tag* with a key–value pair of $highway : traffic\_signals$. The inclusion of traffic signals in property graph data has to consider that the traffic lights may occur somewhere along the road where there are no intersections, e.g., a pedestrian crossing with traffic signals along a major road, or at actual road intersections. This means the property must be included in both nodes and edges. This means we will include the property *signals* in both nodes and edges. In an intersection, this will be calculated as (24), where, n_i represents the intersection node. As shown, there may be a traffic light or not, which means the property's value will be either zero or one.

$$n_i[signals] = \begin{cases} 0 & \text{if node doesn't have a traffic light,} \\ 1 & \text{if node has a traffic light.} \end{cases} \tag{24}$$

In the case of the edge e_i composed of nodes n_1 to n_N, the value of the *signals* property will be the sum of the *signals* properties of all the nodes in the edge, except for the starting and ending node, since they are already counted in node *signals* property. This can be represented as

$$e_i[signals] = \sum_{k=2}^{N-1} \left(\begin{cases} 0 & \text{if } n_k \text{ doesn't have a traffic light,} \\ 1 & \text{if } n_k \text{ has a traffic light.} \end{cases} \right) \tag{25}$$

where n_k represents the nodes of the edge, and k starts with two, since the first node n_1 is excluded, and the sum continues to $N-1$, since the N represents the final node of the edge. This approach ensures that only the traffic signals located between the first and last nodes of the edge are counted.

4.9. Road Type

Over 28 different road types (e.g., motorway, trunk, primary, secondary, tertiary, residential, . . .) exist in OpenStreetMap data, and they are specified in the OpenStreetMap *way* objects as *tags* with the key *highway* [59]. An individual edge on the graph can be composed of multiple ways (if the edge contains more than 2000 nodes). We are interested in what type of road it is. Is it a motorway and not suitable for pedestrians or cyclists? The roads can also be of primary, secondary, or tertiary type, which ranks their importance in the country's road system and may influence traffic. In our case, we want to assign the *highway* tag to each of our edges. Because a road may change between intersections, one edge may have more than one road type.

Suppose we have two intersections, n_x and n_y, connected via edges e_x and e_y. The edge e_x is created from nodes n_1 to n_N, and edge e_y is created from nodes n_N to n_1. Each of these nodes, n_i, where $1 \leq i \leq N$ can be assigned at least one OpenStreetMap way that the node was originally taken from—at least one because, when one way ends and another starts, they can terminate and start in the same node. Otherwise, the road would be unconnected.

We can solve the issue of assigning the road type in two ways: (1) by assigning a list of all types the edge travels over and (2) by selecting the most common road type on an individual edge, with the most common meaning the road type with the largest cumulative distance.

The (1) approach can be conducted by finding all the ways in which the nodes are part of it. Let us say that n_i belongs to a set of ways, W_i. We aim to identify a subset of ways, $W_{actual} \subseteq W_{potential}$, traversed in the path. Here, $W_{potential}$ represents the union of all W_i for $i = 1, 2, \ldots, N$.

The algorithm to determine W_{actual} applies the following order: In the first step, we Initialize W_{actual} as an empty set. In the next step, we compute the intersection of each pair of consecutive nodes' (n_i, n_{i+1}) sets of ways $(I = W_i \cap W_{i+1})$ and update the set W_{actual} by adding the intersection of ways, I, to it $(W_{actual} \cup I)$. Mathematically, this can be represented as

$$W_{actual} = \bigcup_{i=1}^{N-1} (W_i \cap W_{i+1}). \tag{26}$$

where W_i and W_{i+1} are the set of ways that the nodes n_i and n_{i+1} are part of, and $W_i \cap W_{i+1}$ represents the shared ways of two consecutive nodes, n_i and n_{i+1}.

Now that we have a set of all valid ways, as shown below,

$$w_1, w_2, \ldots, w_N \in W_{actual}, \tag{27}$$

We check the *highway* tag $(H(w_i))$ for each of the set elements (w_i), and we create a union set of all highway tags in the resulting edge, as shown in (28).

$$H_{\text{total}} = \bigcup_{i=1}^{N} H(w_i). \tag{28}$$

The (2) second approach can be done in the next way. In the first step, we create a list of all succeeding pairs of nodes $[(n_1, n_2), (n_2, n_3), \ldots (n_N - 1, n_N)]$. From them, we create processed road-type elements, which we will call $(R_{1,2}, R_{2,3}, \ldots R_{N-1,N})$, and save them in set R.

Each $R_{i,i+1}$ element will contain two attributes, distance ($R_{i,i+1}[d]$) and the corresponding *highway* tag ($R_{i+1}[highway]$).

The approach to calculating the distance between two nodes is (4), or (8). The approach to selecting the highway tag for each of the pairs is finding the ways (W_i and W_{i+1}) the first n_i and second n_{i+1} node belong to and computing the intersection which represents the highway tag of the road-type element $I_{i,i+1}$, as shown below

$$I_{i,i+1} = W_i \cap W_{i+1}, \tag{29}$$

The resulting set will, due to the structure of the OpenStreetMap data, always contain only one *way* element. We can then check the highway tag of the way element and add it to the resulting road-type element.

Once we have done this for all of the node pairs to get road-type elements, we must first identify all the present highway tags, H. This is done by creating a union of all the road-type highway sets, as shown below:

$$H = \bigcup_{i=1-1}^{N} R_{i,i+1}[highway]. \tag{30}$$

After this, we need to iterate over all road-type elements in set R, check the highway tag (h) for each segment, and sum the distances for segments that share the same tag to calculate the cumulative distances for each tag ($D(h)$), as shown below

$$D(h) = \sum_{R_{i,i+1} \in \mathrm{R} \ R_{i,i+1}[highway]=h} R_{i,i+1}[d] \quad \forall h \in H. \tag{31}$$

After that, we select the highway tag with the highest value and assign it to the edge.

4.10. Surface

This attribute provides the physical surface of the edge. It is derived from the surface OpenStreetMap tag [60], which provides over 40 values for different values. The *surface* tag is not necessarily present in each OpenStreetMap way, so the value may remain empty. The tag values are categorized roughly into paved, unpaved, and special surfaces. Common values of the attribute are asphalt, paved, concrete, unpaved, gravel, etc.

The issue of an edge possibly containing different surface types is similar to the issue that an edge may contain different road types. So, the transformation into property graph attributes can be performed in the same fashion as in Section 4.9 in the road type subsection.

4.11. Accessibility and Usage Restrictions of Road Types in OpenStreetMap

There are generally three main modes of transport that can travel over a certain path: pedestrians, cyclists, and cars. Not all paths are suitable for each of them; e.g., a pedestrian cannot travel on a motorway, and a car cannot travel on a cycleway. The road type itself can sometimes determine these limits, but additional tags have to be inspected sometimes. Table 8 presents the different road types and their accessibility to cars, cyclists, and pedestrians. In some cases, the accessibility depends on local or national laws, where the tilde ($\sim$) symbol is used.

Table 8. Road types and allowed usage for cars, cyclists, and pedestrians according to OpenStreetMap data.

Road Type	Cars	Cyclists	Pedestrians
Motorway [61]	+	−	−
Trunk [62]	+	∼	−
Primary [63]	+	+	∼
Secondary [64]	+	+	∼
Tertiary [65]	+	+	+
Unclassified [66]	+	+	+
Residential [67]	+	+	+
Service [68]	+	+	+
Pedestrian [69]	−	+	+
Footway [70]	−	+	+
Cycleway [71]	−	+	+
Path [72]	−	+	+
Track [73]	∼	+	+
Living Street [74]	+	+	+

Another thing is that, although a road may normally be used by one mode of transport, sometimes local traffic signalization prohibits its use. For cycling [75], this is shown as the following tag value pairs, *bicycle:dismount* and *bicycle:no* and for pedestrians as *foot:no* [76]; similar rules exist for cars and other modes on transport.

On the property graph, this will be expressed as three additional properties to each path, called *bicycle_access*, *foot_access*, and *car_access*, with each assigned a true or false value.

4.12. Directionality

This property refers to the directionality of the path (edge). This property is found in OpenStreetMap under the *oneway:yes* tag [77]. It signifies that the path is only passable in the same direction in which the nodes are ordered, and not in the opposite direction. This attribute is not calculated, but it actually serves to determine which edges are generated and which are not. When generating a property graph, it is important to determine for which users we are generating routes since directionality only influences vehicles and can be disregarded for pedestrians. Since an edge can be composed of one or multiple ways, the route can be considered one-way if at least one way contains the tag *oneway:yes*.

5. Results

The proposed method was tested and presented in a case study that presents the generation of a property graph based on the data available for the area of Slovenia, which covers roughly 20,271 km^2 [78]. The boundary of the property graph for generation was based on the Geofabrik area of Slovenia [79]. This study aimed to investigate the approach's feasibility and the presentation of a sample property graph generated through the proposed approach. The process can also be adapted easily to other geographical areas.

The process followed the approach proposed in [45,47], changed by several optimizations and improvements, which were a result of newly constructed properties defined in previous chapters, as well as ways to increase the construction speed of the property graph.

This is concluded by using the presented property graph and using it on an existing evolutionary-based pathfinding algorithm [23] to generate a few sample routes, where differences between old and newer methods are also highlighted.

5.1. Generated Property Graph

The generated property graph had **1,486,886 nodes** (split intersections), which were the result of 263,454 actual intersections and **2,973,478 edges**, of which 2,229,886 were intersection-based edges, and 743,592 were actual paths.

5.2. Process

The process of generating a property graph was composed of the following steps, which are described below:

1. Dataset download.
2. Server deployment.
3. Data preprocessing.
4. Node (intersection) identification.
5. Node processing.
6. Edge (paths) construction.
7. Edge (paths) merging.
8. Removing unused nodes.
9. Splitting intersections.
10. Property graph solutions.

5.2.1. Dataset Download

The first step was downloading the right data for the property graph generation. This means that DEM and OpenStreetMap data needed to be downloaded. Since we used Open-Elevation API [44], the dataset for Elevation Data could be split into 1 × 1 degree tiles. The dataset location for EU-DEM [80] was (https://opendem.info/opendemeu_download_4258.html, accessed on 1 August 2024), from which **N45E014**, **N45E015**, **N45E016**, **N46E013**, **N46E014**, **N46E015**, and **N46E016** tiles were downloaded; the available dataset follows the convention described in the Table 9 below for naming files.

Table 9. Explanation of Code N45E014.

N	45	E	014
North	45°–46°	East	14°–15°

The OpenStreetMap data was downloaded from the Geofabrik download servers (https://download.geofabrik.de/europe/slovenia.html, date of download: 12 August 2024). The data are updated daily, and at the time of the download, the latest version (*slovenia-latest.osm.pbf*) was downloaded.

5.2.2. Server Deployment

The servers were established in the second step. The architecture of the solution was split into servers and a local executer program that accesses this server. To ensure portability, the servers were prepared as a set of Docker [81] containers. Two groups of Docker containers were established, notably APIs and databases, as shown in Figure 11.

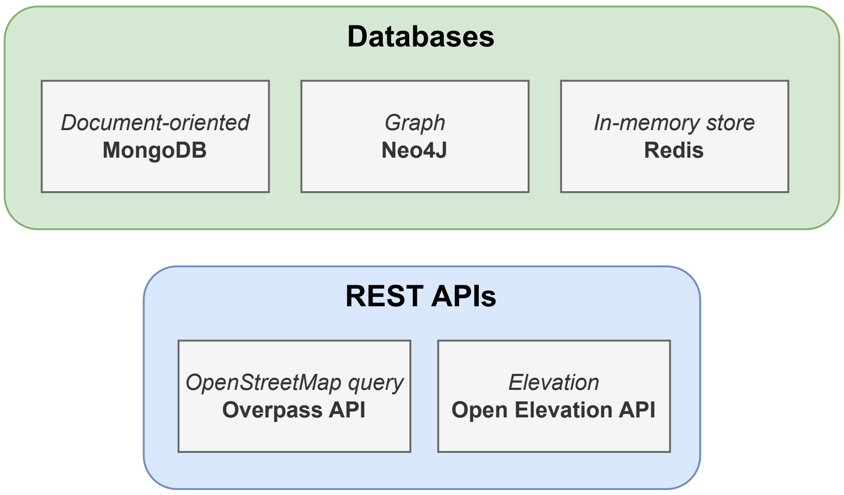

Figure 11. Overview of the server deployment structure.

For APIs, two containers were set up, notably the Overpass API [42] and Open Elevation API [44]. For data containers, three databases were set up. The document-oriented MongoDB served as the storage for all the calculated attributes, and its data were later imported into Neo4j. The remaining Redis database was used to help track which roads and pathways had already been processed and which had not.

5.2.3. Data Preprocessing

Data preprocessing techniques were used to speed up the generation of the graph. This was done by eliminating the unnecessary data from the OpenStreetMap and preparing the data for easier retrieval.

In this step, only ways with tag *highways* (paths) and their nodes were extracted from the data. This was done using the Pyosmium library [82]. The resulting file size was reduced significantly (the updated file size was 17% of the original), as shown in Table 10. The file was also produced in the slightly less compressed version (*.bz2 - bzip2) because Overpass API does not work with *.pbf (protocol buffer) files.

Table 10. Comparison of original and updated OSM files.

File	Nodes	Ways	Format	File Size
Original	40,171,270	2,621,093	osm.pbf	~279 MB
Updated	5,751,333	430,308	osm.pbf osm.bz2	~49 MB ~ 82 MB

This was continued by using Pyosmium to save all the nodes and their ways, as well as ways with their nodes inside a Mongo database, where each intersection was saved into a *helper* collection, and each way was saved into a separate *helper* collection. This was done to allow nodes (in later steps) to be queried directly from the Mongo database. To speed up the later retrieval further, the indexes of documents in both collections were selected as node and way IDs.

5.2.4. Node (Intersection) Identification

The intersection identification was concerned with identifying all the intersections of the specified area. This was done following the original approach [47], where a map is split into small latitude–longitude squares, and then each succeeding square is queried using Overpass API. Nodes that are connected to three or more nodes are identified in the query.

The resulting collection featured 391,087 intersections. The identified nodes were saved in the *intersections* collection, and each entry contained the following attributes:

- **_id**: OpenStreetMap id of the node;
- **lat**: Latitude of the node;
- **lon**: Longitude of the node;
- **tags**: any OpenStreetMap key/value pairs the node contained.

5.2.5. Node Processing

Each node entry in the intersections collection was extended in the node processing phase. This was done by adding elevation data and querying the elevation API. Each node was also checked to see whether it contained the traffic signal tag and the ways in which the node was part of it. The resulting entry for intersections was, therefore, **_id**, **lat**, **lon**, and **tags** from the previous stage, as well as the following additional attributes:

- **elevation**: elevation above sea level in meters.
- **traffic_signals**: is there a traffic signal at the node location with a value of true or false?
- **way_ids**: which (OpenStreetMap) ways is the node part of?

5.2.6. Edges (Paths) Construction

After all the intersections had been identified in previous stages, now, connections had to be established between intersections. The proposed entries for edges had the following attributes:

- **start_node**: the starting node of the connection;
- **end_node**: the ending node of the path;
- **distance**: the total distance in meters using the Pythagorean approach;
- **ascent**, **descent**: the total ascent and descent in meters;
- **path_type**: the array of highway types the path is built on;
- **surface**: the list of surface types, if available (e.g., asphalt);
- **total_angle**: the total change in direction in degrees;
- **curviness**: the degrees per meter traveled, using the ratio of summed vector angles to the distance approach;
- **traffic_lights**: the number of traffic lights on the path (not including traffic lights in an intersection);
- **hill_flat**, **hill_gentle**, **hill_moderate**, **hill_challenging**, **hill_steep**, and **hill_extremely_steep**: the number of meters traveled on the hill of a given grade;
- **backward**: whether an equivalent reverse route exists;
- **bicycle_access**, **foot_access** and **car_access**: access to the route for given modes of transport.

The values of attributes were generated according to previously established and presented methods for their calculation. Overall, 998,714 connections were established.

5.2.7. Edge (Paths) Merging

Before splitting intersections, the unnecessary intersections had to be eliminated, and paths had to be merged. This was done to make the graph as simple as possible without losing too much quality. Suppose we have a case of three nodes (N_X, N_Y, and N_Z), where the middle node (N_Y) is only connected to two nodes (N_X and N_Z), as shown in Figure 12 below.

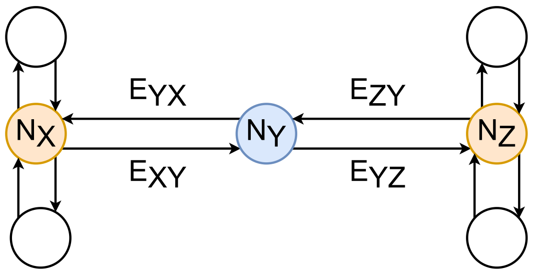

Figure 12. Potential edge merging scenario.

It can be argued that we could eliminate the node (N_Y) and merge the edges (E_{XY}, E_{YZ} and E_{ZY}, E_{YX}) so that the complexity of the graph would be reduced. This process was performed iteratively until no such nodes existed. This was due to the fact that a merged edge can result in the existence of a new edge that has the potential for merging, as demonstrated in Figure 13, where the merging has to happen in two iterations to eliminate unnecessary edges.

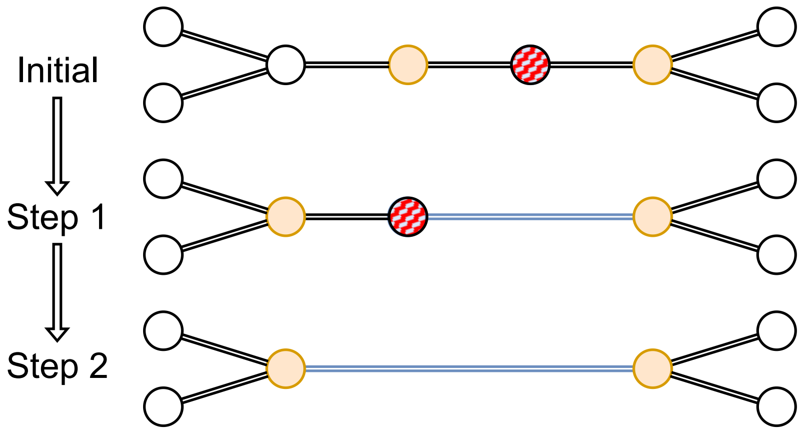

Figure 13. Edge (paths) iterative merging steps.

In our case, the initial graph had 998,714 edges (paths) and 391,087 nodes (intersection); through the process of iterative merging, this was reduced to 743,597 edges (paths) and 263,454 nodes (intersections) in six iterations, as shown in Table 11 below.

Table 11. Total edges and nodes across iterations.

Iteration	Total Edges	Total Nodes
Initial	998,714	391,087
1	816,935	300,133
2	758,099	270,705
3	745,851	264,581
4	743,823	263,567
5	743,613	263,462
6	743,592	263,454
Eliminated	255,122 (~25.5%)	127,633 (~32.6%)

Overall, this led to the elimination of roughly a quarter of edges and a third of all nodes. A total of 255,122 edges and 127,633 nodes were eliminated.

5.2.8. Removing Unused Nodes

It is possible that unused nodes (intersections) exist in the graph. This can rarely happen near the edges of the OpenStreetMap file that was processed; e.g., an intersection was identified, but the ways did not lead anywhere because they were located in another country. These intersections were removed to reduce graph complexity. Before this step, there were 263,454 nodes, which were reduced to 263,322 nodes. This presented a very small reduction, but it was still conducted to ensure best practices.

5.2.9. Splitting Intersections

The splitting intersections phase followed the approach proposed in Section 4.4. The goal was to create new edges and nodes for the better modeling of curves on intersections. A total of 2,229,886 new paths were created. The resulting graph had a total of 1,486,886 intersections and 2,973,478 paths.

5.2.10. Property Graph Solutions

The prepared collections of intersections and paths were then processed and inserted into property graph solutions. This process was conducted for the Neo4j graph database [83], which was selected based on its ubiquity, as Neo4j is the most popular graph-based database management system [84].

5.3. Sample Route Generation

To demonstrate the approach feasibility in route generation, an evolutionary algorithm for generating cycling training routes [23] was used on the property graph where two cases of routes were generated; the routes were then plotted on the OpenStreetMap [28] overlay, and cumulative numerical features are presented for each route, which were derived from the property graph data. Only relationships that were suitable for bicycles were included based on the *bicycle_access* attribute. The first route (A) is thoroughly described, and the remaining route is described in a condensed manner to avoid repetitiveness.

Route A

The algorithm was first tasked with finding and generating a 50-kilometer route with an ascent of 1000 m between Ruše and Lenart. The resulting route (shown in Figure 14) was 52 km long and was of mixed hills and flat terrain.

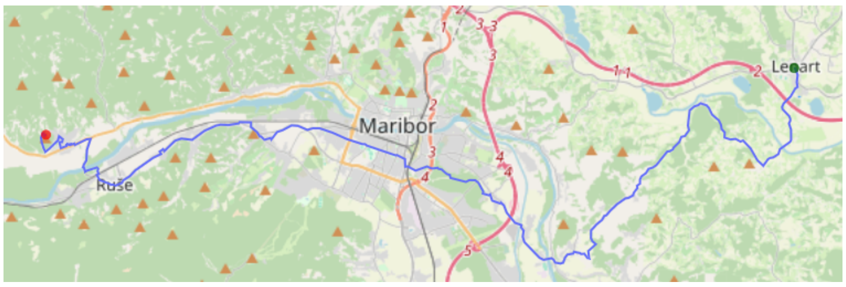

Figure 14. OpenStreetMap projection of Route A.

The computed route was composed of 781 segments (relationships). The property graph data allowed us to thoroughly analyze the route features, as shown in Table 12, and continued in the presented road type and surface type analysis.

Table 12. Route A summary.

Route A			
Distance	52,246 m		
Ascent	1013 m	Descent	875 m
Total angle	13,544°	Curviness	0.259
Traffic lights	9	Bicycle access	true
Flat	30,505 m	Hill gentle	7288 m
Hill, moderate	5796 m	Hill challenging	3354 m
Hill, steep	2374 m	Hill extremely steep	2925 m

The analysis of the property graph features revealed the distribution of road types within the generated route. The most prevalent categories were secondary roads, accounting for 21.70%, and unclassified roads, representing 21.64% of the total route. Residential roads constituted 17.75%, while tertiary roads comprised 13.52%. Other notable road types included paths (7.76%), cycleways (4.32%), footways (4.10%), service roads (3.70%), and tracks (3.31%). Less common categories included primary roads (2.03%) and living streets, which represented only 0.19% of the total route.

The surfaces were distributed as follows: asphalt covered 46.46% of the total distance, making it the most common surface. Unknown surfaces accounted for 48.08%, indicating that a significant portion of the OpenStreetMap data lacks precise surface information. Compacted surfaces contributed 3.59%, while ground surfaces covered 1.71%. Less frequently, gravel represented 0.14%, grass made up 0.02%, and unpaved areas constituted 0.01%.

Route B

The algorithm was tasked with generating a short 13-km route between Murska Sobota and Gančani with as little ascent as possible. The resulting route (shown in Figure 15 was 12,875 m long with a total ascent of 28 m.

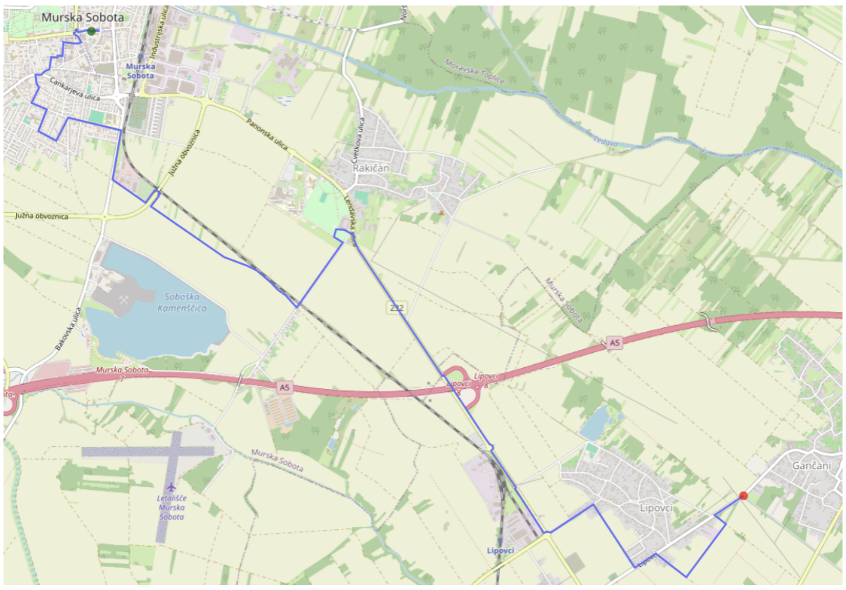

Figure 15. OpenStreetMap projection of Route B.

The analysis of property graph data allowed us to display the route features shown in Table 13.

Table 13. Route B summary.

Route B			
Distance	12,875 m		
Ascent	23 m	Descent	38 m
Total angle	2908°	Curviness	0.225
Traffic lights	2	Bicycle access	true
Flat	11,844 m	Hill gentle	856 m
Hill, moderate	93 m	Hill challenging	43 m
Hill, steep	29 m	Hill extremely steep	7 m

In the original property graph, only the distance, ascent, descent, and road type data could be extracted.

6. Discussion

The proposed approach, which is an extension of the original approach [47], fills the knowledge gap of route property graph generation. The additional computed properties improved the original property graph method greatly; each node had only four usable properties and each edge (path) only six. This was extended to five properties in each node (intersection) and 20 usable edge (path) properties. This can improve the existing approaches [23] to route generation greatly, and these approaches were already applied to the data generated from the previous method.

The key drawbacks of the previous property graph were identified and improved upon, mainly by providing additional data regarding the directionality of paths, which was ignored previously. Another key improvement is expanding the elevation data by providing ascent and descent and the actual grade of hills from which the ascent was calculated. The curviness of roads and angle data were also added to edges, which allowed for even finer refinement when identifying potential routes in real-world environments.

The upgraded approach also followed the best practices of replicability and transparency by establishing how individual properties are generated scientifically and sharing the repository for data creation, which allows for the verification of the approach and the quick usage of the approach in other areas.

The property graph potential usage was also expanded to other users (pedestrians and cars), not only cyclists.

The *drawback* of the improved approach is that the complexity and size of data are much larger, which can impact the speed and resources needed for their actual usage. This can be solved by reducing the area of the property graph, which would reduce the number of nodes and edges, or the researchers may opt to skip the step of intersection splitting, which would reduce the number of total edges and intersections vastly.

The large size of the data needs to be considered since it can also limit suitable software for manipulating said data. Multiple benchmarks [85–87] of popular graphing libraries exist, so the researchers must take into consideration what area they will work and what data they will use, as well as the ease of use of each of them. We would not recommend any particular library, and would leave the choice to researchers using this method.

7. Conclusions

The work presented in the paper presents a comprehensive method for creating complex geographical-based property graphs based on real-world geographical route data, which are particularly useful for sports training applications where advanced elevation, curviness, and track data are needed.

The approach also presents individual properties and identifies ways to calculate them mathematically from available geographical data. In particular, a new way to tackle the limitations of property graphs in interpreting intersections, by splitting notes, was presented, allowing for the incorporation of individual intersection angle data in property graphs.

Future research includes improving the evolutionary method for generating cycling routes [23], which will utilize the new knowledge of the property graph, and prepare a meta-method that will allow for adding new properties across all components of the property graph. Another aspect that could be explored is including the context of the edges where building and other scenic features could be incorporated as mentioned in [88].

Author Contributions: Conceptualization, A.R. and I.F.J.; methodology, A.R.; software, A.R.; validation, A.R. and I.F.J.; formal analysis, A.R.; investigation, A.R. and I.F.J.; resources, A.R.; data curation, A.R.; writing—original draft preparation, A.R.; writing—review and editing, A.R. and I.F.J.; visualization, A.R.; supervision, I.F.J.; project administration, I.F.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Slovenian Research Agency, Research Core Funding No. P2-0057.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The proposed solution is available publicly under the MIT License. The up-to-date version is available at https://github.com/alenrajsp/GeoGraphGen (accessed on 1 November 2024) GitHub repository.

Acknowledgments: The backgrounds of Figures 2–4, 6, 7 and 9 were retrieved from the aerial photography, as well as the real estate cadastre, of the Surveying and Mapping Authority of the Republic of Slovenia, and they are publicly available permission of use: https://www.e-prostor.gov.si/en/access-to-geodetic-data/?acitem=1934-1931 (accessed on 1 November 2024) under Creative Commons 4 license.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

A*	A-star search algorithm
API	Application programming interface
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer
DEM	Digital elevation model
ETOPO	Earth topography
EU-DEM	European Union Digital Elevation Model
GPS	Global positioning system
NED	National Elevation Dataset
NZ-DEM	New Zealand Digital Elevation Model
OSM	OpenStreetMap
QL	Query language
SRTM	Shuttle Radar Topography Mission
USGS	United States Geological Survey
XML	Extensible Markup Language

References

1. Wolodtschenko, A.; Forner, T. Prehistoric and Early Historic Maps in Europe: Conception of Cd-Atlas. *Int. Web J. Sci. Technol. Affin. Hist. Cartogr. Maps* **2007**, *2*, 114–116.
2. An, M. Introduction. In *Advances in Cartography and Geographic Information Engineering*; Chapter History of Cartography; Springer: Singapore, 2021; pp. 601–602. [CrossRef]
3. McNeff, J. The global positioning system. *IEEE Trans. Microw. Theory Tech.* **2002**, *50*, 645–652. [CrossRef]
4. Diestel, R. Graphs. In *Graduate Texts in Mathematics*; Chapter The Basics; Springer: Berlin/Heidelberg, Germany, 2017; pp. 1–4. [CrossRef]
5. Angles, R. The Property Graph Database Model. In Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, 21–25 May 2018; CEUR Workshop Proceedings; Volume 2100, pp. 1–12.
6. Subero, A. Trees. In *Codeless Data Structures and Algorithms: Learn DSA Without Writing a Single Line of Code*; Chapter Tree Data Structures; Apress: Berkeley, CA, USA, 2020; pp. 31–34. [CrossRef]
7. Wellhausen, C.; Clemens, J.; Schill, K. Efficient Grid Map Data Structures for Autonomous Driving in Large-Scale Environments. In Proceedings of the 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), Indianapolis, IN, USA, 19–22 September 2021; pp. 2855–2862. [CrossRef]
8. Schrijver, A. On the History of the Shortest Path Problem. *Doc. Math.* **2012**, *17*, 155–167.
9. Gasparetto, A.; Boscariol, P.; Lanzutti, A.; Vidoni, R. Path Planning and Trajectory Planning Algorithms: A General Overview. In *Motion and Operation Planning of Robotic Systems*, 1st ed.; Carbone, G., Gomez-Bravo, F., Eds.; Springer: Cham, Switzerland, 2015; pp. 3–27. [CrossRef]
10. Zhang, X.; Zhang, Z.; Zhang, Y.; Wei, D.; Deng, Y. Route selection for emergency logistics management: A bio-inspired algorithm. *Saf. Sci.* **2013**, *54*, 87–91. [CrossRef]
11. Haria, V.; Shah, Y.; Gangwar, V.; Chandwaney, V.; Jain, T.; Dedhia, Y.; Surendra Modi, A. The Working of Google Maps, and the Commercial Usage of Navigation Systems. *Int. J. Innov. Res. Technol.* **2019**, *6*, 184–191.
12. Statista. Leading Smartphone Users Activities Worldwide from July 2021 to June 2022. 2022. Available online: https://www.statista.com/statistics/1337895/top-smartphone-activities/ (accessed on 1 November 2024).
13. Yu, G.; Yang, J. On the Robust Shortest Path Problem. *Comput. Oper. Res.* **1998**, *25*, 457–468. [CrossRef]
14. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
15. Hart, P.E.; Nilsson, N.J.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
16. Ford, L.R. *Network Flow Theory*; Technical Report; RAND Corporation: Santa Monica, CA, USA, 1956.
17. Hougardy, S. The Floyd–Warshall algorithm on graphs with negative cycles. *Inf. Process. Lett.* **2010**, *110*, 279–281. [CrossRef]
18. Dantzig, G.B.; Ramser, J.H. The truck dispatching problem. *Manag. Sci.* **1959**, *6*, 80–91. [CrossRef]
19. Flood, M.M. The Traveling-Salesman Problem. *Oper. Res.* **1956**, *4*, 61–75. [CrossRef]
20. Minieka, E. The Chinese Postman Problem for Mixed Networks. *Manag. Sci.* **1979**, *25*, 643–648. [CrossRef]
21. Edmonds, J.; Karp, R.M. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* **1972**, *19*, 248–264. [CrossRef]
22. Ghoseiri, K.; Nadjari, B. An ant colony optimization algorithm for the bi-objective shortest path problem. *Appl. Soft Comput.* **2010**, *10*, 1237–1246. [CrossRef]
23. Rajsp, A.; Fister, I. A Modified Evolutionary Algorithm for Generating the Cycling Training Routes. *IEEE Access* **2022**, *10*, 109743–109759. [CrossRef]
24. Chicco, D. Ten quick tips for machine learning in computational biology. *Biodata Min.* **2017**, *10*, 35. [CrossRef]
25. Loepp, B.; Ziegler, J. Recommending Running Routes: Framework and Demonstrator. In Proceedings of the ComplexRec '18: Proceedings of the 2nd Second Workshop on Recommendation in Complex Scenarios, Vancouver, BC, Canada, 7 October 2018; pp. 1–4.
26. Mariescu-Istodor, R.; Fränti, P. Fast travel-distance estimation using overhead graph. *J. Locat. Based Serv.* **2021**, *15*, 261–279. [CrossRef]
27. Lustig, H. Tour de France 2024 Route Stage 1: Florence—Rimini. 2024. Available online: https://www.cyclingstage.com/tour-de-france-2024-route/stage-1-tdf-2024/ (accessed on 15 December 2024).
28. Openstreetmap Foundation. OpenStreetMap. 2024. Available online: https://www.openstreetmap.org/ (accessed on 1 November 2024).
29. U.S. Geological Survey. The National Map. 2024. Available online: https://www.usgs.gov/core-science-systems/ngp/tnm-delivery (accessed on 19 January 2024).
30. Natural Earth. Available online: https://www.naturalearthdata.com/ (accessed on 19 January 2024).

31. Surveying and Mapping Authority of the Republic of Slovenia. Public Geodetic Data: Ortophoto DOF025. Creative Commons Attribution 4.0 International License, Geodetska Uprava Republike Slovenije, Data from Ortophotography. 2024. Available online: https://eprostor.gov.si/imps-test/srv/api/records/1e0f54cf-9303-4140-8c20-352d5468c0b3 (accessed on 4 October 2024).
32. Surveying and Mapping Authority of the Republic of Slovenia. Public Geodetic Data: Realestate Cadastre. Creative Commons Attribution 4.0 International License, Geodetska uprava Republike Slovenije, Data from Ortophotography. 2024. Available online: https://eprostor.gov.si/imps/srv/api/records/9a8fd241-9162-407c-94e7-c98e05766881 (accessed on 4 October 2024).
33. Wilson, J.P.; Gallant, J.C. *Digital Elevation Data Sources and Structures*; Chapter Digital Te; John Wiley & Sons: Nashville, TN, USA, 2000; pp. 3–5.
34. Polidori, L.; El Hage, M. Digital Elevation Model Quality Assessment Methods: A Critical Review. *Remote Sens.* **2020**, *12*, 3522. [CrossRef]
35. NASA; METI; AIST; Japan Spacesystems; U.S./Japan ASTER Science Team. ASTER Global Digital Elevation Model V003 [Dataset]. 2019. Available online: https://lpdaac.usgs.gov/products/astgtmv003/ (accessed on 4 October 2024).
36. NOAA National Centers for Environmental Information. ETOPO Global Relief Model. 2022. Available online: https://data.noaa.gov/metaview/page?xml=NOAA/NESDIS/NGDC/MGG/DEM//iso/xml/etopo_2022.xml&view=getDataView&header=none (accessed on 4 February 2024).
37. European Space Agency (ESA). Copernicus Digital Elevation Model. 2023. Available online: https://dataspace.copernicus.eu/explore-data/data-collections/copernicus-contributing-missions/collections-description/COP-DEM (accessed on 4 February 2024).
38. European Environment Agency. EU-DEM 1.1. 2017. Available online: https://www.eea.europa.eu/data-and-maps/data/copernicus-land-monitoring-service-eu-dem (accessed on 19 January 2023).
39. Land Information New Zealand (LINZ). NZ 8m Digital Elevation Model. 2012. Available online: https://data.linz.govt.nz/layer/51768-nz-8m-digital-elevation-model-2012/ (accessed on 4 February 2024).
40. NASA Jet Propulsion Laboratory. Shuttle Radar Topography Mission (SRTM) Global. Distributed by OpenTopography. 2013. Available online: https://portal.opentopography.org/raster?opentopoID=OTSRTM.082015.4326.1 (accessed on 4 February 2024).
41. OpenStreetMap Contributors. PBF Format. 2023. Available online: https://wiki.openstreetmap.org/wiki/PBF_Format (accessed on 5 February 2024).
42. Overpass API Team. Overpass API. 2024. Available online: https://overpass-api.de/ (accessed on 5 February 2024)
43. Nisbet, A. Open Topo Data. Available online: https://www.opentopodata.org (accessed on 5 February 2024)
44. Lourenço, J.R. Open-Elevation: A Free and Open-Source Elevation API. 2024. Available online: https://open-elevation.com/ (accessed on 5 February 2024)
45. Rajšp, A.; Fister, I. Neo4j graph dataset of cycling paths in Slovenia. *Data Brief* **2023**, *48*, 109251. [CrossRef]
46. Yadav, S.K. Basics of Graph Theory. In *Advanced Graph Theory*, 1st ed.; Chapter Graph; Springer International Publishing: Cham, Switzerland, 2023; pp. 4–6.
47. Rajšp, A.; Heričko, M.; Iztok, F.J. Preprocessing of roads in OpenStreetMap based geographic data on a property graph. In Proceedings of the Central European Conference on Information and Intelligent Systems, Varazdin, Croatia, 13–15 October 2021, pp. 193–199.
48. Robusto, C.C. The Cosine-Haversine Formula. *Am. Math. Mon.* **1957**, *64*, 38–40. Available online: http://www.jstor.org/stable/2309088 (accessed on 30 December 2024). [CrossRef]
49. Prša, A.; Harmanec, P.; Torres, G.; Mamajek, E.; Asplund, M.; Capitaine, N.; Christensen-Dalsgaard, J.; Depagne, É.; Haberreiter, M.; Hekker, S.; et al. Nominal values for selected solar and planetary quantities: IAU 2015 Resolution B3. *Astron. J.* **2016**, *152*, 41. [CrossRef]
50. Maria, E.; Budiman, E.; Haviluddin; Taruk, M. Measure distance locating nearest public facilities using Haversine and Euclidean Methods. *J. Phys. Conf. Ser.* **2020**, *1450*, 012080. [CrossRef]
51. Yadav, S.K. Algebraic terms and operations used in Graph Theory. In *Advanced Graph Theory*, 1st ed.; Chapter Some Impor; Springer International Publishing: Cham, Switzerland, 2023; pp. 29–30.
52. Ligas, M.; Banasik, P. Conversion between Cartesian and geodetic coordinates on a rotational ellipsoid by solving a system of nonlinear equations. *Geod. Cartogr.* **2011**, *60*, 145–159. [CrossRef]
53. Axler, S. Inner Products and Norms. In *Linear Algebra Done Right*, 4th ed.; Chapter Inner Product Spaces; Gorkin, P., Sidman, J., Eds.; Springer: Cham, Switzerland, 2024; pp. 182–183. [CrossRef]
54. Valenzuela, P.L.; Mateo-March, M.; Muriel, X.; Zabala, M.; Lucia, A.; Pallares, J.G.; Barranco-Gil, D. Road gradient and cycling power: An observational study in male professional cyclists. *J. Sci. Med. Sport* **2022**, *25*, 1017–1022. [CrossRef] [PubMed]
55. Minetti, A.E.; Moia, C.; Roi, G.S.; Susta, D.; Ferretti, G. Energy cost of walking and running at extreme uphill and downhill slopes. *J. Appl. Physiol.* **2002**, *93*, 1039–1046. [CrossRef]
56. DeYoung, J. Defining slope. In *Forest Measurements—An Applied Approach*, 1st ed.; Chapter Slope; Open Oregon Educational Resources: Gresham, OR, USA, 2016; pp. 4–7.

57. de Neef, M. Gradients and Cycling: An Introduction. Climbing Cyclist. 2013. Available online: https://theclimbingcyclist.com/gradients-and-cycling-an-introduction/ (accessed on 25 August 2024).
58. OpenStreetMap Contributors. Tag:highway=traffic_signals. 2024. Available online: https://wiki.openstreetmap.org/wiki/Tag:highway%3Dtraffic_signals (accessed on 1 March 2024).
59. OpenStreetMap Contributors. Key:highway. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway (accessed on 1 March 2024).
60. OpenStreetMap Contributors. Key:surface. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:surface (accessed on 1 March 2024).
61. OpenStreetMap Contributors. Key:highway#Motorway. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Motorway (accessed on 1 March 2024).
62. OpenStreetMap Contributors. Key:highway#Trunk. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Trunk (accessed on 1 March 2024).
63. OpenStreetMap Contributors. Key:highway#Primary. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Primary (accessed on 1 March 2024).
64. OpenStreetMap Contributors. Key:highway#Secondary. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Secondary (accessed on 1 March 2024).
65. OpenStreetMap Contributors. Key:highway#Tertiary. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Tertiary (accessed on 1 March 2024).
66. OpenStreetMap Contributors. Key:highway#Unclassified. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Unclassified (accessed on 1 March 2024).
67. OpenStreetMap Contributors. Key:highway#Residential. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Residential (accessed on 1 March 2024).
68. OpenStreetMap Contributors. Key:highway#Service. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Service (accessed on 1 March 2024).
69. OpenStreetMap Contributors. Key:highway#Pedestrian. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Pedestrian (accessed on 1 March 2024).
70. OpenStreetMap Contributors. Key:highway#Footway. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Footway (accessed on 1 March 2024).
71. OpenStreetMap Contributors. Key:highway#Cycleway. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Cycleway (accessed on 1 March 2024).
72. OpenStreetMap Contributors. Key:highway#Path. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Path (accessed on 1 March 2024).
73. OpenStreetMap Contributors. Key:highway#Track. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Track (accessed on 1 March 2024).
74. OpenStreetMap Contributors. Key:highway#Living_Street. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:highway#Living_Street (accessed on 1 March 2024).
75. OpenStreetMap Contributors. Bicycle. 2024. Available online: https://wiki.openstreetmap.org/wiki/Bicycle (accessed on 16 July 2024).
76. OpenStreetMap Contributors. Key:foot. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:foot (accessed on 16 July 2024).
77. OpenStreetMap Contributors. Key—Oneway. 2024. Available online: https://wiki.openstreetmap.org/wiki/Key:oneway (accessed on 16 July 2024).
78. Britannica, The Editors of Encyclopaedia. Slovenia Summary. 2020. Available online: https://www.britannica.com/summary/Slovenia (accessed on 20 August 2024).
79. Geofabrik GmbH. Download Server for Slovenia Data. 2024. Available online: https://download.geofabrik.de/europe/slovenia.html (accessed on 15 March 2024).
80. European Environment Agency (EEA). European Digital Elevation Model (EU-DEM), Version 1.1. 2019. Available online: https://land.copernicus.eu/imagery-in-situ/eu-dem/eu-dem-v1.1 (accessed on 4 February 2024).
81. Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *2014*, 2.
82. OpenStreetMap Community. Pyosmium: Python Bindings for Libosmium. Available online: https://osmcode.org/pyosmium (accessed on 8 May 2024).
83. Neo4j, I. Neo4j Graph Database Management System. 2023. Version 5.12. Available online: https://neo4j.com/deployment-center/ (accessed on 10 January 2024).

84. DB-Engines. DB-Engines Ranking of Graph DBMS. 2024. Available online: https://db-engines.com/en/ranking/graph+dbms (accessed on 20 August 2024).
85. Peixoto, T.P. Graph-Tool: Performance Comparison. 2023. Available online: https://graph-tool.skewed.de/performance.html (accessed on 21 August 2024).
86. Lin, T. Benchmark of Popular Graph Network Packages—V2. 2022. Available online: https://www.timlrx.com/blog/benchmark-of-popular-graph-network-packages-v2 (accessed on 21 August 2024).
87. Treinish, M.; Carvalho, I.; Tsilimigkounakis, G.; Sá, N. Rustworkx Benchmarks. 2023. Available online: https://www.rustworkx.org/dev/benchmarks.html (accessed on 12 October 2024).
88. Mariescu-Istodor, R.; Fränti, P. Context-aware similarity of GPS trajectories. *J. Locat. Based Serv.* **2020**, *14*, 231–251. [CrossRef]