IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 24 August 2022, accepted 10 October 2022, date of publication 14 October 2022, date of current version 20 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3214997

RESEARCH ARTICLE

A Modified Evolutionary Algorithm for Generating the Cycling Training Routes

ALEN RAJŠP AND IZTOK FISTER JR. (Member, IEEE)
Faculty of Electrical Engineering and Computer Science, University of Maribor, 2000 Maribor, Slovenia
Corresponding author: Alen Rajšp (alen.rajsp@um.si)

This work was supported by the Slovenian Research Agency under Grant No. P2-0057.

ABSTRACT We present a novel method for generating cycling training routes from geographical property graphs based on an Evolutionary Algorithm. The algorithm operators of crossover and mutation are adjusted for use in the Property Graph domain. Data fusion of geographical data from the OpenStreetMap, EU-DEM digital surface model, and existing training records is performed as a basis of the intersections-paths property graph. The proposed approach allows route generation based on their starting and ending points in the property graph and their distance and ascent. A property graph of all intersections and cycling roads is shown and generated for the regions of Podravje and Pomurje. The property graph used in the proposed algorithm's feasibility demonstration is shown. This is done by presenting four different cases of routes generated with our algorithm. The algorithm allows for generating classic cycling routes of A to B nature, routes with more than two fixed points, and cyclical training routes. The research is concluded by offering further directions on route generation research.

INDEX TERMS Data mining, sports training, automatic generation of sports training, evolutionary algorithms.

I. INTRODUCTION

Our lifestyles are changing and constantly evolving in modern society. The annual working hours have decreased gradually in the OECD countries and have dropped by ≈11 % since the 1970s till now [1]. This additional free time has enabled us to pursue additional time for leisure. This time has been spent not only on entertainment, using Information and Communications Technology, but also on sports. According to the European Commission [2] 42.8 % of Europeans Union citizens perform sports activities at least once weekly.

Sports remain an integral part of daily human lives. People have engaged n them on both amateur and professional levels. Nowadays, each professional sport has organizations that define rules and organize competitions. Athletes and teams are listed in leagues, scoreboards, divisions, and tournaments to promote competitiveness [3]. The goal of each team or athlete is to win and be the best. To do this, the athlete must train and improve his performance.

The associate editor coordinating the review of this manuscript and approving it for publication was Gang Mei.

Sports training can be organized into four distinct interconnected phases, namely, planning, realization, control, and evaluation, during which the athlete is monitored closely by his coach, and adjustments are made at each iteration of sports training to reach his highest potential [4]. Because competitiveness in sports is high, training is an essential part of an athlete's routine. The training has evolved with the individual sports, where each improvement to the training routine can change the outcome.

One of the advances in sports training has been due to the inception of digital technologies and approaches in every step of sports training. Such approaches, broadly characterized as Smart Sports Training, have been found in at least 32 different sports [5]. Sports training can be a highly organized routine between an athlete and a coach. One of the primary roles is facilitation, supervision, and monitoring of the training session to achieve the set objectives [6]. Since coaches need to be paid, they provide additional costs to the athlete, who may find this a barrier to progressing within his discipline. This is particularly true for amateur athletes, who do not receive any monetary compensation for their endeavors, making it an

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

even more significant issue to pay for coaches. Such issues will be easily solvable in the future by incorporating Artificial Sports Trainers into the training process. Such a solution can provide a trainer to athletes who do not have access to one and also aid in the decision-making process of real trainers [7].

While sports such as athletics, football, volleyball, and soccer are performed and trained mainly in gyms and stadiums, which can be heavily monitored and predictable environments, sports such as cycling and running are trained primarily in the real-world outdoor environments of public roads, tracks, and paths. On the one hand, this can be a problem since it is much harder to plan individual training in detail if the environment is constantly changing, is not monitored as closely, and is much more unknown to the trainee and the trainer. On the other hand, this is a vast opportunity for development and research on training generation methods that can provide constantly changing and challenging training routines that will not be seen as repetitive to the trainee.

Sports training is a constantly evolving research field, where long-term success is measured in the competition results of athletes training under different training regimes/approaches. The existing approaches to training planning, as identified in a literature review [5], are related mainly to using historical training data of athletes (e.g. [8], [9], [10]) to improve their results. Another thing familiar with such approaches is that the algorithms and methods present the athlete with a spreadsheet of parameters he / she should achieve to ensure better results and progress. In contrast, this is a straightforward and quickly followed plan in sports based on repetitive routines (e.g., running, athletics, weight lifting). This approach has proved severely lacking in sports such as cycling. Even if the cyclist is presented with a spreadsheet training plan, he / she must know how and where to achieve it. The physical parameters of heart rate, cadence, and speed that the cyclist must follow in a training session must also be combined with the environmental parameters of distance, terrain, and altitude. Moreover, while approaches for generating the raw numbers of what the training should look like already exist and are used widely, the environment requirements are still an unknown domain.

To generate cycling training of high quality, the existing approaches for training generation, notably those proposing the environment and physical parameters, must be used as inputs for a computational intelligence method incorporating the real-world environment's geographical data to generate actual training routes.

An approach that allows an extensive and diverse set of data is needed to solve the identified problem and generate specialized and discrete training routes. Evolutionary Algorithms are the most appropriate for the task. They allow generating solutions to complex real-world problems, where it is impossible to use heuristic solutions [11]. Another constraint in the generation of routes is that the problem can be multi-objective. The input parameters of the route generation ascent and distance often conflict with each other, and improving one parameter may deteriorate the other; this is another field where Evolutionary Algorithms are highly suitable [12].

This previously unaddressed need for generating actual training routes is the key focus of our paper. We solve this problem by proposing a novel Evolutionary Algorithm based method for training track generation in the sport of cycling by combining different sources of real-world geographical data.

The contributions of this paper are:

- proposed data fusion method of geographical data-based property graph and previous training knowledge
- comparison and analysis of existing approaches for cycling training generation
- an evolutionary algorithm-based method for generating sports training tracks in cycling
- practical evaluation of the proposed algorithm.

The paper is structured as follows, Section II presents existing solutions for the generation of cycling training routes, Section III presents the data collection and fusion for the generation of a property graph and the novel Evolutionary Algorithm for the generation of cycling training routes. Section IV presents the results and basic optimization of the input parameters for the proposed algorithm. Section V discusses the domain of route generation and places our approach in the domain. The final Section VI summarizes our findings and suggests future research in the field of Cycling Training Generation.

II. GENERATING CYCLING TRAINING SESSIONS AND ROUTES

Computational intelligence, meaning Artificial Neural Networks, Evolutionary Algorithms, Fuzzy Systems, and Bayesian Networks, has been used heavily in generating cycling training and routes. It has been adapted heavily to solving and finding approximate solutions to problems that are difficult to solve in other ways [13].

Automatizing and enriching cycling training by the use of Computational Intelligence can be divided roughly into two parts, as presented in this section. The first part of this section is concerned with the generation of training. In this approach, previous training may be evaluated to propose improvements in the future, and optimum training strategies may be proposed. This can concern the physiological parameters of the athlete, such as data collected by power meters, heart-rate sensors, personal feelings, and environment recordings such as distance traveled, speed, ascent, descent, and total duration, as seen in Fig. 1. This is a lot of data that can typically be recorded from a training session and can be predetermined precisely by the training plan. By selecting a suitable route the cyclist must travel on, we can accurately predict and adjust the distance, ascent, and descent the cyclist will cover in a training session.

Sports training may include the generation of actual routes that the cyclist travels and training parameters such as target heart rate, cadence, and speed. The first part of this section presents approaches to sports training generation that include

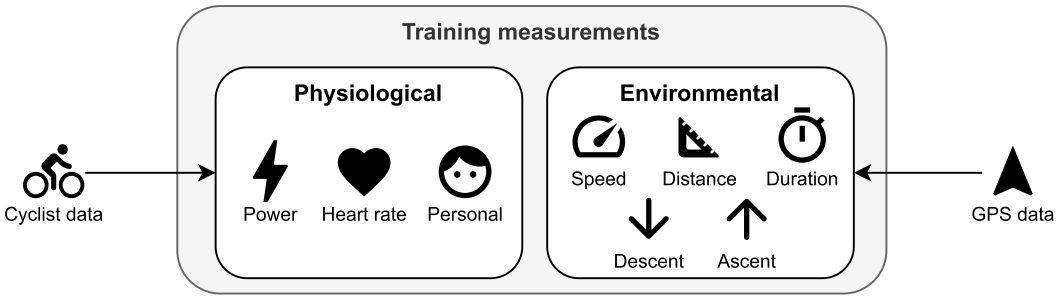

FIGURE 1. Data typically recorded in a training session.

mainly only the raw target parameters of the athletes. In the second part of this section, we present existing solutions that address generating cycling routes solely and are not connected with the generation of actual sports training plans. An overview of the Computational Intelligence approaches used in planning bike training was presented in [14], where the use of approaches using GPS devices, power and cadence meters, and heart-rate wearables are identified. Existing approaches of Computational Intelligence used in cycling training are presented in [5].

A. SPORT TRAINING GENERATION

The process of sports training generation is tailored to generate the optimum training plans for athletes to reach the highest levels of performance. Sports training must be done carefully to avoid the pitfalls of inadequate training plans, such as overtraining, plateau effects, and monotony, which negatively impact athletes' performance [15]. Some approaches generate generalized sports training plans for athletes [15], [16], [17], [18] where target power-meter, heart rate, and duration are proposed for individualized sports training sessions. How and where the trainee will execute his goals are left to his own will; some approaches do not generate actual sports training but aid in the training process (e.g. [19]), and some of them generate actual sports training routes that the cyclist must complete (e.g. [20]). Predicting a cyclist's heart rate at a given workload, activity duration, fatigue, exhaustion, and recovery were investigated in [16]. The study was done on nine well-trained cyclists. The proposed model could predict the heart rate to about 3bpm.

Three models for predicting average power on an individual cycling sports training were constructed in [21]. The three models were of a for-all approach, athlete-based, and cluster-based type. The study found that athlete-based models were the most suitable for power prediction. Their results show that regression models can predict the average power of new cycling activities within 20W of the actual effort. This is very useful because an average cycling enthusiast may not have access to expensive equipment such as power meters.

The relation between the heart rate and cycling cadence and the heart rate in the future second were researched in [19]; the result of the study was a Feedforward Neural Network model that predicted heart rate to an average error of 2.43 beats per minute for the next second.

The relationship between sports training load and performance in cycling was investigated in [17], and an Evolutionary Algorithm-based model was proposed for generating training plans. The main goal of the generated training plan was to avoid over-training and plateau effects.

Zahran et al. [22] proposes a conceptual framework for generating adaptive weekly sports training plans based on internal training load, physiological constraints, and behavioral-change features. The drawback is that the approach has not been used in practice outside of the preexisting training data used to construct it.

Study [18] proposed planning individual interval sport training sessions using stochastic population-based nature-inspired metaheuristics. The Bat algorithm was used and tested on data from previously recorded interval training sessions.

Cycling training route generation was attempted in [20], where a method was presented for generating training sessions with routes in cycling based on data from power meters and previously recorded training routes. The altitude data was extracted from the training records. The routes generated following this approach can only be compiled from segments of previously completed and recorded routes. The route generation was completed using an Evolutionary Algorithm, which used Dijkstras' algorithm [23] and Tabu search [24] for route initialization. The weights for the route initialization were the distances between the nodes.

Kumyaito1 et al. [15] was concerned with generating eight-week sport training plans for athletes with known resting and maximum heart rates by generating a plan which determined the average heart rate and duration of each training session by using a Particle Swarm Optimization algorithm. The Particle Swarm Optimization algorithm-based sports training plan outperformed the standard plan based on a training plan from British Cycling, while satisfying all the physiological constraints of monotony, chronic training load ramp rate, and daily training impulse.

Similar research by Kumyaitol et al. [25] used a Genetic Algorithm to develop an eight-week aerobic cycling training plan, and found a Genetic Algorithm capable of creating

training plans over-performing the standardized British cycling training plan.

B. ROUTE GENERATION

The problem of generating cycling routes can be solved from different perspectives, as presented in Table 1. The Table columns and their meanings are the following:

- Approach - Citation to the approach.
- Description - A summary of the presented approach.
- Supporting algorithms - Which algorithms were used for generating the routes?
- Generated from - From which data were routes generated?
- Start - Does the solution allow the selection of the route starting point?
- Via - Does the solution allow the selection of mid-points that the route must cover?
- End - Does the solution allow the selection of the route ending point?
- Optimization parameters - What parameters and in which way are selected in route generation?
- Type - Is the solution presented in a research paper, or is it a commercial solution?

Each solution is presented further following the Table or in the previous section.

It can be approached as a simple routing problem, where the goal is to find the shortest route between points A and B. This is a suitable approach for travel purposes where the person needs to spend the least time and effort traveling between two destinations. Such an approach was studied in [26], where analysis was performed of recorded GPS data of cycling routes. The approach splits the routes into zones of 1000×1000, 350×350, and 100×100 meters, and tries to find the optimized shortest route between the starting and ending points. The algorithm for finding the shortest routes was the Dijkstra algorithm [23]. While the work presents research on route planning in cycling, it was unsuitable for our approach since it did not consider the altitude, and was proposed regarding the shortest route.

Commercial solutions also exist for generating cycling routes. Naviki [27] allows choosing the distance and bicycle type and a starting point, and generates a route where the user can see the total distance, ascent, and descent meters. Routeshuffle [28] allows the generation of circular cycling routes where the target distance can be configured, but there is no way to determine the number of elevation meters.

The Bicycle route generation was presented in [31], where OpenStreetMap [32] and Shuttle Radar Topography Mission [33] elevation data were combined. The presented solution allowed the generation of routes between two points tailored to the cyclist's parameters: quietness, flatness, comfort, and travel time. The approach allows the selection of only one criterion (e.g., only flatness).

Other solutions that aid in creating bike routes, where the user inputs the points to visit and receives a bike route with calculated distance, ascent, and descent, include Ride With GPS [30] and Bikemap [29]. These tools only help create the routes and calculate the ascents/descents, but cannot create routes where the user can also choose the ascent/descent parameters.

As can be seen, the only solutions that allow selecting the route's start, mid, and ending points are [27], [29], [30], all of them being closed-source commercial solutions. Another limitation of these three mentioned solutions is that they either find the shortest route ([29], [30]) or only allow choosing the distance while not allowing the selection of the ascent parameter ([27]).

The consequence is that the distance of the route must be approximated by the user, who must choose the appropriate starting and ending points of the routes using trial and error compared to our approach.

III. PROPOSED METHOD

The goal of the proposed method was to generate viable cycling routes based on elevation and distance constraints between two or more fixed predetermined points, generated from a pre-processed geographical data property graph.

The method consists of the following steps:

- Data collection
- Property graph generation
- Data fusion
- Generation of training supported by an evolutionary algorithm model

All the steps are outlined in detail in the following subsections. All the original algorithms were implemented in the Python 3.9 [34] programming language. The execution environment of all the non-original algorithms is described where they are used.

A. DATA COLLECTION

The proposed method depends on OpenStreetMap (OSM) [32] geographical data, EU-DEM [35] elevation data and previously recorded training activities which were collected from [36] and [37]. From all the mentioned knowledge, the property graph was constructed and utilized as illustrated in Fig 2.

1) OpenStreetMap

OpenStreetMap [32] is a worldwide open-source project for creating a geographic database of the whole planet. The data are stored in XML format. The geographical data are represented by nodes, ways, relations, and tags. The nodes represent longitude and latitude pairs, and are, together with tags, the smallest building blocks of the map. Ways are ordered lists of nodes representing map features, for example, roads, buildings, and administrative boundaries. Multiple ways are connected with relations when a feature needs more than 2000 nodes to be represented. Because nodes, ways, and relations are just individual nodes or (connected) lists of them, Tags are used to give them context. They consist

TABLE 1. Comparison of existing cycling route generation approaches.

Approach	Description	Supporting algorithms	Generated from	Start	Via	End	Optimization parameters	Type
[20]	Generation of sports training sessions for individual athletes based on existing training sessions of individual athletes.	Evolutionary algorithms, Dijkstra's algorithm, Tabu search	Topology based on existing activities	✗	✗	✗	Distance, ascent, average power	Research paper
[26]	Generating synthetic bicycle routes from heat-map of GPS cycling data	Dijkstra's algorithm	Aggregated heat-map of existing cycling data	✓	✗	✓	Distance (shortest)	Research paper
[27]	Solution for generation of bicycle routes. Allows generation of circular (customizable distance) and A-B (non-customizable distance) routes	N/A	N/A	✓	✓	✓	Distance (for circular routes), shortest distance (for A-B routes)	Commercial solution
[28]	Random circular route generator for cycling, running, and walking.	N/A	N/A	✓	✗	✗	Distance	Commercial solution
[29]	Route generator from multiple points.	N/A	N/A	✓	✓	✓	Shortest route	Commercial solution
[30]	Route generator from multiple points.	N/A	N/A	✓	✓	✓	Shortest route	Commercial solution
[31]	Route generator from user points (A-B)	A*	OpenStreetMap (paths), Shuttle Radar Topography Mission (elevation)	✓	✗	✓	Shortest route, flattest route, quietest route. Only one criterion is possible for selection.	Research article
Our proposal	Route generator from user points (A-B) and ascent and distance adjustment	Dijkstra's algorithm, evolutionary algorithms	OpenStreetMap (paths), EU-DEM (elevation), Existing training records (verification)	✓	✓	✓	Distance, ascent	Research article

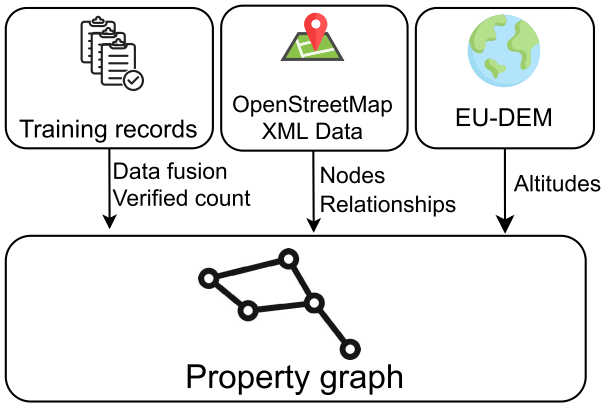

FIGURE 2. Altitude integration from OSM and EU-DEM.

of key-value pairs describing nodes' metadata, ways, and relations. To retrieve data from OSM, Overpass API [38] was used, which is a read-only API for querying selected parts of OSM geographical data.

2) EU-DEM

EU-DEM [35] is a digital surface model (elevation model) of 33 EEA (European Economic Area) members and six cooperating countries. The model allows retrieval of elevation data from latitude and longitude, has a vertical accuracy of 2.9 RMSE (Root Mean Square Error), and is, on average, 0.56 meters accurate [39]. The elevation data are sampled at 25 x 25 meters squares. The surface model was used by the self-hosted service Open Elevation API [40], which allows importing and querying digital surface model maps and returning elevation data for given latitudes and longitudes.

3) PROPERTY GRAPH

The Property graph was generated from the presented OpenStreetMap [32] and EU-DEM [35] data. So, from a directed graph perspective, as seen in (1), a property graph (G) is defined by a set of vertices (V), which represent actual intersections, and edges (E), which represent routes between them.

TABLE 2. Selected training records.

		Selected	
Dataset	**Athletes**	**Athletes**	**Records**
[36]	2 mountain biking 1 triathlon 1 road cycling	1 road cycling	462 (tcx)
[37]	6 cycling 2 multi-sport 4 mountain biking 1 running 2 triathlon	6 cycling	2189 (gpx)
Total	**19**	**7**	**2651**

The edges are directed, which means that their order matters. The edges exist only between the vertices connected in the real-world environment.

$$\begin{aligned} G &= (V, E) \\ V &= \{v_1, v_2, \ldots v_n\}, \\ E &= \{(v_1, v_2), (v_2, v_1), \ldots (v_x, v_y), (v_y, v_x)\} \end{aligned} \tag{1}$$

The procedure for generating the used graph is presented in [41]. The only difference that we implemented in relation to our previously proposed approach is that we used the Neo4J [42] for saving graphs instead of RedisGraph [43]. The change was done because the RedisGraph offers limited functionalities related to exporting and importing the generated property graphs.

4) SPORTS TRAINING ACTIVITY DATASETS

The collected sports training activity datasets were in the TCX (Training Center XML) and GPX (GPS Exchange Format), which are file formats used by training recording software [44]. All training activity files were composed of recorded sports training, which consisted of succeeding points. Each point had a recorded geographical longitude and latitude, timestamp, and heart rate. Two relevant datasets were identified and used [36], [37]. Only road and normal cyclists (not riding on mountain trails) were selected, and a total of 2651 training records were selected, as seen in Table 2.

The inclusion of only regular and road cycling was done because of the specifics of mountain biking, where training happens on sometimes unrecorded pathways, and high elevation differences in training. Only the road cycling and general cycling athletes were selected. Other sports (e.g., triathlon) were naturally eliminated since they were outside the scope of our research.

B. PROPERTY GRAPH

The generated Property graph is a graph featuring **nodes / vertices**, which are abstractions of actual *intersections* on the OpenStreetMap and **relationships / edges** which are abstractions of *roads / ways* between them. Each node has the following attributes: latitude, longitude, node id, and way ids, as seen in Table 3.

TABLE 3. Data stored in a property graph node (intersection).

Attribute	Value type	Description
latitude	float (-90 to 90)°	Geographical latitude on Earth.
longitude	float (-180 to 180)°	Geographical longitude on Earth
node_id	integer	Id of the node containing the intersection in OSM
way_ids	integer array	Connected ways in OSM

TABLE 4. Data stored in a property graph relationship (path).

Attribute	Value type	Description
ascent	float	Ascent on the given section
descent	float	Descent on the given section
distance	float	Total distance between the two nodes
type	string array	List of types road types the path traverses

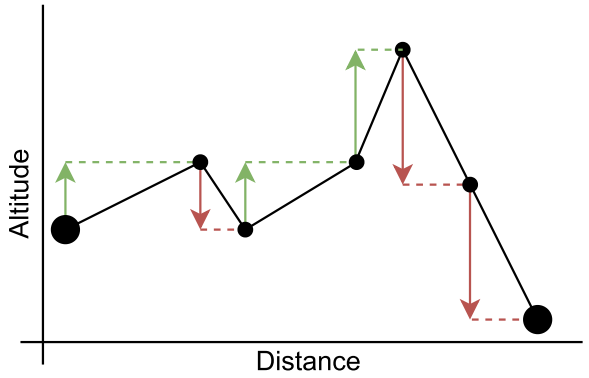

FIGURE 3. Altitude integration from OSM and EU-DEM.

Each node is connected to other nodes via relationships named paths in our case, as seen in table 4. Each path contains the distance between the two connected nodes, ascent and descent.

The path can contain a positive value of both ascent and descent concurrently. This can happen because the ascent and descent are indexed at each node of the way in the OpenStreetMap. Hence, a path that traverses up and downhill will contain both, even if the ending altitude is the same as the starting point.

An example of such calculation is shown in 3 where two connected intersections (big black dots) are shown. They are connected by five nodes which are shown by the smaller dots. For each of the nodes, their altitude is determined based on their latitude and longitude, with upwards arrows showing the positive gain of altitude and downward arrows showing the negative gain of altitude. The positive altitude gains are summed to determine the ascent gain of a relationship. The negative gains are summed to calculate the descent gain.

Each pair of nodes connected in an intersection is done so in a two-way fashion, as seen in Fig. 4. That is done because traveling from A-B is due to ascent and descent properties different from traveling from B-A. Therefore, the ascent and descent values are switched when traveling in the opposite direction.

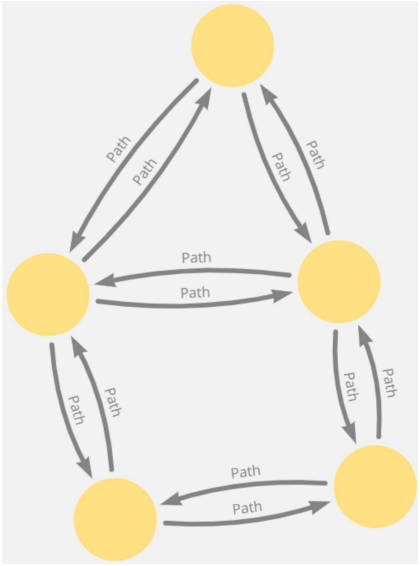

FIGURE 4. Property graph example of cycling intersections and pathways.

The method of generation of the aforementioned property graph has been described thoroughly in [41].

1) LIMITATIONS OF THE PROPERTY GRAPH

Our process is limited in that we only queried connections from the OpenStreetMap, and some roads may be one-way roads in the real-world environment, but we did not check the directionality, and simplified the process of generating a two-way connection from each road. However, this is rarely a problem because roads, where cyclists ride are, in the majority, two-way roads.

C. DATA FUSION

The sports training data from [36], [37] and the intersection data in the property graph were in different formats. This problem was solved using data fusion, which is the process of fusing multiple records representing the same real-world object into a single, consistent, and clean representation [45].

Data fusion was applied to the property graph to fully utilize the combined knowledge to improve quality. This was done by reading every training record, and identifying which recorded intersections it passed. These data were then added to the property graph as a verified count, where each intersection recorded the number of times it was driven over.

The verification procedure (Algorithm 1) receives a list of training records and a list of all intersections as parameters (**line 1**). The training records were parsed using The minimalistic toolbox for extracting features from sport activity files [44]. The outer for loop ensures that every training activity was processed (**line 3-17**), and the inner for loop (**line 4-16**) processed every individual recorded point of the individual training activity (GPX/TCX point). The next innermost for loop (**line 6-11**) calculates the distance between the recorded training activity point and each of the saved intersections, and saves it to the list of potential verified intersections (**line 8-9**). Suppose at least one intersection closer than 20 meters was identified (**line 12**). In that case, the potentially verified intersections are first sorted by their distance in an ascended list (**line 13**), and the closest intersection is added to the list of verified intersections (**line 14**). In the end, the verified intersections are returned (**line 18**), and for each occurrence of a verified intersection, its verified property in the property graph is increased by one.

Algorithm 1 Verified Intersections Identification

```
1:  procedure verifyIntersections(intersection_list, training)
2:      verified_intersections = []
3:      for each training in trainings do
4:          for each point in training do
5:              intersections = []
6:              for each intersection in intersection_list do
7:                  intersection_point = geodesic_d(intersection, point)
8:                  if intersection_point < 20 then
9:                      intersections+=intersection_point
10:                 end if
11:             end for
12:             if intersections.length>0 then
13:                 intersections.sort()
14:                 verified_intersections+=intersections[0]
15:             end if
16:         end for
17:     end for
18: return verified_intersections
19: end procedure
```

This process was applied for every of the 2651 training records. This ensured that a verify count could be determined for each intersection.

D. EVOLUTIONARY ALGORITHM FOR PLANNING CYCLING ROUTES

The purpose of the proposed algorithm was to generate a suitable route in terms of starting intersection, ending intersection, ascent, and distance parameters. The novel approach for generating cycling routes was based on the Genetic Algorithms [46], which are a group of Evolutionary Algorithms. The variation operators in Genetic Algorithms are selection, crossover, and mutation [47].

The proposed algorithm followed a similar procedure as shown in Fig. 5, which demonstrates the steps followed in the algorithm. The algorithm needs the combined knowledge represented in the form of a Property graph and the algorithm as well as route parameters to start with the optimum route generation. In the first step, initial routes are initialized, and then a subset of them is used for crossover and mutations. Following these two steps, the selection of the best routes is performed, and then the algorithm continues until termination

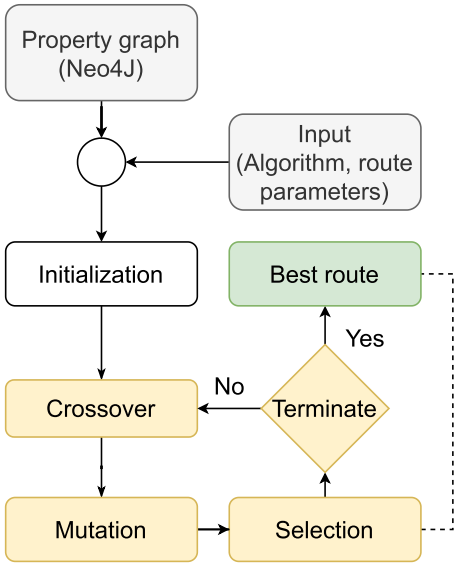

FIGURE 5. Algorithm execution flow.

TABLE 5. Route parameters.

Parameter	Value	Name
Start point id	integer	start_id
Midpoints id*	integer array	mid_points
Endpoint id	integer	end_id
Maximum ascent (m)	positive float	max_ascent
Minimum ascent (m)	positive float	min_ascent
Maximum distance (km)	positive float	max_km
Minimum distance (km)	positive float	min_km

conditions are reached. The best route in the end set of routes is determined using the selection evaluation function. All the concrete steps are further described in the following subsubsections.

1) PARAMETERS

The input parameters for this algorithm were divided into five categories, algorithm parameters, evaluation weights, route parameters, route generation parameters, and penalty parameters.

The route parameters refer to the conditions we want the route to contain (Table 5). The required route has a defined location of the start (*start_id*), the end (*end_id*) and midpoints* (*mid_points*) Midpoints are explained in the subsection III-D6. If we want to create a circular route, we just select the same node of the end intersection as the start intersection, and add at least one node as a midpoint. The route parameters also included their minimum (*min_km*) and maximum distance (*max_km*) in kilometers, and their minimum *min_ascent* and maximum ascent (*max_ascent*) in meters.

The algorithm parameters in Table 6 define the chance that each route will be selected for crossing over (*crossover_r*) in each iteration, and the chance that the route crossed over will be mutated (*mutation_r*). It also determines the number of iterations to run, which signifies the number of generations

TABLE 6. Algorithm parameters.

Parameter	Value	Name
Crossover rate	0-1	crossover_r
Mutation rate	0-1	mutation_r
Number of generations	positive integer	i
Population size	pos. integer	n

TABLE 7. Route generation parameters.

Parameter	Value	Name
Used path weight	pos. float	used_w
Path generation weight adjustment	pos. float	p_gen_w
Path generation weight chance	0-1	p_gen_c

to create (i) in genetic-like algorithms, population size (n), which was the number of starting routes, and the number of routes to select in each selection step.

2) INITIALIZATION

In the first initialization step, the individual routes are generated from the previously presented property graph (1) based on the start and end node id from the route parameters (Table 5).

We investigated and tried using the Yen's K-Shortest Paths algorithm [48], which generates the K number of shortest paths between nodes, but decided against it since the time complexity increased exponentially with longer routes. We based our path generation on Dijkstra's algorithm [23], which is implemented inside the Neo4J [42] database. Additional steps were added to prevent the generation of the same shortest paths over and over again. We have introduced three new parameters, as seen in Table 7 called Route generation parameters.

Another vital thing to note and define is what we want the shortest route to be in Dijkstra's algorithm. We do not want the shortest routes to be the routes of the smallest distance, but want the newly generated routes to be different from previous ones, so that the diversity of routes helps to reach the optimized route. To enable this behavior in Dijkstra's algorithm [23], the algorithm's cost calculation is modified, as shown in (2). In a typical case (real (unmodified) cost $\rightarrow r$), the cost would either be the distance or ascent between the two by path-connected nodes. A cost adjustment modifier between A-B is introduced to ensure that paths in each generation will be different from the routes of the previous generation ($Cost(path_{A-B})$). It is equal to the sum of a chosen cost parameter, either distance or ascent (r) between A-B, and the chosen cost parameter (r) multiplied by the usage count (*usage_c*) of the path, that is, how many times did the selected path appear in the previous generation of routes, and multiplied by a constant *used_w*. The constant can be selected arbitrarily and is subject to further optimization. This cost is calculated for each path between the nodes, and done for each generation of routes, so new paths are explored. The purpose is to penalize the heavily used routes and increase the relative cost of traveling them for other costlier but new

routes to become viable.

$$r \epsilon \mathbb{R},\ r \epsilon \text{ distances} \vee r \epsilon \text{ ascents}$$
$$\text{Cost}(\text{path}_{\text{A}-\text{B}}) = r + (r \cdot \text{usage\_c} \cdot \text{used\_w}) \quad (2)$$

The second part of the Route generation parameters relates to the adjusting costs in generating paths, as seen in the Algorithm 2.

In the route initialization step a n number of paths are required between the start *start_id* and end *end_id* nodes. The algorithm additionally needs the generated property graph (G), *costs* dictionary, where the identity key of each path between two intersections has a defined travel cost and the Path generation weight adjustment *p_gen_w* and chance (*p_gen_c)* explained in Table 7. Routes array (*routes*) in **line 2** collects all the generated paths. Routes are generated inside the while loop (**line 3-10**), which continues until n number of routes are generated. In the first step, the shortest path is generated (**line 4**) using Dijkstra's shortest path algorithm [23], and then added to the list of generated routes (**line 5**).

After that, a determined percentage *p_gen_c* of paths (edges) of the generated route are selected (**line 6**). For the selected / sampled paths the internal cost (for the Dijkstra's algorithm) is increased by a ratio of Path generation weight adjustment (*p_gen_w*) (**line 7-9**). The cost increases generated and appended in each route generated are non-persistent and affect only the generation of routes in the current execution of the procedure. After n routes are generated, the procedure terminates and returns them (**line 10-12**).

Algorithm 2 Generate N Routes Between a and B

```
1:  procedure generateNRoutes(start_id, end_id, n, G,
    costs, p_gen_w, p_gen_c)
2:      routes=[]
3:      while len(routes) ≠ n do
4:          route = dijkstraShortestPath(start, end, G, costs)
5:          routes+=route
6:          paths = randomSample(route, p_gen_c)
7:          for each path in paths do
8:              costs[path] = costs[path]· p_gen_w
9:          end for
10:     end while
11: return routes
12: end procedure
```

3) SELECTION

The routes were generated fully only in the first step, but a selection evaluation was performed in each of the succeeding steps. The evaluation function depended on the route and evaluation parameters. The route parameters were the starting (*start_id*) and ending points (*end_id*), which were already used in the initialization step. The minimum ascent (*min_ascent*) and distance (*min_distance*) provided the lower boundaries of the generated routes, and maximum ascent (*max_ascent*) and distance (*max_distance*) provided the upper boundaries of the generated routes. All routes not between these boundaries were penalized based on the evaluation function. The algorithm's goal was to find the route with the lowest possible penalty result, which was evaluated for each route in the evaluation function. A penalty scoring type was determined for each of the deviations from the proposed values, as seen in Table 8.

TABLE 8. Penalty parameters.

Parameter	Value	Name
Penalty type	absolute / relative	type
Weight	float	weight

TABLE 9. Verification bonus.

Parameter	Value	Name
Threshold	float (0-1)	threshold
Bonus percentage	float (0-1)	bonus_percentage

Two types of penalty scoring exist: absolute and relative. In the case that the absolute scoring was chosen in the algorithm initialization, and the investigated route parameter (x) was between the proposed minimum (*min*) and maximum (*max*), there was no penalty (value 0). If it was above the maximum, the penalty was equal to the absolute difference (*x-max*) times penalty weight (*weight*), and, if it was below the minimum, it was equal to the difference (*min-x*) between the actual value (x) and the maximum value (*max*) times the penalty weight (3).

$$\begin{cases} (\min - x) \cdot \text{weight} & \text{if } x \leq \min \\ (x - \max) \cdot \text{weight} & \text{if } x \geq \max \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

In relative scoring, the absolute value of the difference between the proposed value and actual value was first divided by the proposed value and then multiplied by the weight ($(min - x)/min \cdot weight$ **or** $(x - max)/min \cdot weight$) as seen in (4).

$$\begin{cases} \dfrac{\min - x}{\min} \cdot \text{weight} & \text{if } x \leq \min \\ \dfrac{x - \max}{\max} \cdot \text{weight} & \text{if } x \geq \max \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Another bonus was inspected and added since data can be combined with the existing training data verification bonus to reduce the penalty. Each time an existing recorded training passes through an intersection, its verified value increases by one (Table 9).

We can then sum the number of verifications for each route (the sum of all verification values in all intersections) to get the amount of verification (v) of a route. In the algorithm we determine the threshold (*v_threshold*) for the maximum verification bonus and bonus percentage (b) a route can receive. The actual bonus (B) is equal to the number of verifications (v) divided by the threshold value of verifications ($v_{threshold}$)

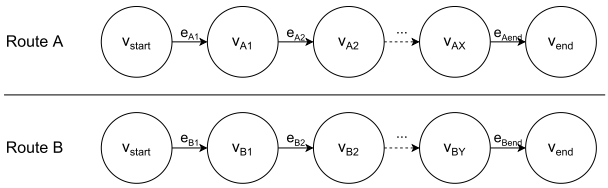

FIGURE 6. Two candidate routes for crossover.

times the maximum verification bonus (b), as seen in in (5).

$$B = v/v_{threshold} \tag{5}$$

If v is larger than $v_{threshold}$ the bonus is capped at the maximum verification bonus (b), as shown in (6).

$$\begin{cases} B = \dfrac{v}{v_{threshold}} \cdot b & \text{if } v \leq v_{threshold} \\ B = b & \text{otherwise} \end{cases} \tag{6}$$

The actual evaluated value (E_{val}) is then the sum of all four penalties: Minimum ascent (P_{min_ascent}), maximum ascent (P_{max_ascent}), minimum distance ($P_{min\_distance}$), and maximum distance ($P_{max\_distance}$) times 1 minus the actual bonus ($1-B$). It is also evident that, if the route is penalized for lower than the minimum boundary of a specific property (ascent or distance), the penalty for the maximum boundary will be zero, and vice versa, as seen in (7).

$$E_{val} = (P_{min\_ascent} + P_{max\_ascent} + P_{min\_distance} + P_{max\_distance}) \cdot (1 - B) \tag{7}$$

4) CROSSOVER

After the evaluation or selection phase, the crossover phase follows. This step selects a percentage (determined by the crossover rate) of the total population for crossover. We cannot simply sample individuals randomly due to the nature of the paths.

We can only crossover ways that share at least three points. Each route has a common start (V_{start}) and endpoint (V_{end}), as seen on Fig. 6. This is in contrast to classic non-domain specific Genetic Algorithms, where any part of the genome can be switched and replaced between the two parents.

However, we must only crossover routes where there is at least another common point, as seen in (8). This means that a non-starting and non-ending vertice of route A ($v_{a'}$) is equal to a non-ending and non-starting vertice of route B ($v_{b'}$), meaning the routes travel through a common vertice (intersection) as seen in (8)

$$\begin{aligned} &v_{a'} \in \text{Route A}; \\ &v_{b'} \in \text{Route B}; \\ &v_{a'} \neq v_{start} \wedge v_{a'} \neq v_{end} \wedge \exists v_{a'} = v_{b'} \end{aligned} \tag{8}$$

The algorithm for crossover selection (Algorithm 3) was provided with the number of needed routes (*needed*) and an array of existing routes (R).

The routes are shuffled randomly in an array in the first step (**line 3**). This is followed by the while loop (**line 4-20**) where previously shuffled succeeding routes ($i = 1, 2, 3, \ldots$) are checked for crossover with the initial route (0). The loop continues while there are fewer routes selected for crossover (*len(pairs)*) than the required number of routes (*needed*), and two or more routes are remaining. The inner while loop (**line 7-16**) checks every remaining route in succession to check the viability by checking (**line 8**) if any intersections suit the criteria listed in (8). When such a route is found (**line 8**), both the initial route 0 and succeeding route i are removed from the route candidates (**line 10-11**) and added to the crossover route pairs (**line 9**). If no routes are found to have a common point with the initial route 0, the route at the start of the array is removed from the candidates, and the next route (the one with index 1) takes its place (**line 17-19**). The main loop continues until no more candidates are identified, or a needed number of pairs are reached. The algorithm returns an array of route pairs (R_y, R_x) saved in the lines' array (instantiated in **line 2**).

Algorithm 3 Crossover Selection

```
1:  procedure crossoverSelection(R, needed)
2:      pairs=[]
3:      R = randomlyShuffleArray(R)
4:      while len(R) ≥ 2 and len(pairs) < needed do
5:          i=1
6:          found = false
7:          while i < len(R) do
8:              if viable(R[0], R[i]) then
9:                  pairs+=(R[0], R[i])
10:                 R.remove(R[i])
11:                 R.remove(R[0])
12:                 found = true
13:             else
14:                 i+=1
15:             end if
16:         end while
17:         if found == false then
18:             R.remove[0]
19:         end if
20:     end while
        return pairs
21: end procedure
```

Once the route pairs are selected it is time to perform the actual crossover. Suppose that we have two routes, A and B, and they have a set (C_p) of crossover points ($v_{p1}, v_{p2}, \ldots v_{pn}$) where (8) which is true. Because the crossover can also happen at the beginning, v_{start} is added to the possible crossover points, forming a set of all possible crossover points $C_{p'}$, as seen in (9).

$$C_p = \{v_{p1}, v_{p2}, \ldots v_{pn}\}; C_{p'} = C \cup \{v_{start}\} \tag{9}$$

Only the starting node is added to the possible crossover vertices since crossover from v_{start} to v_x will generate the same outcome as v_x to v_{end}.

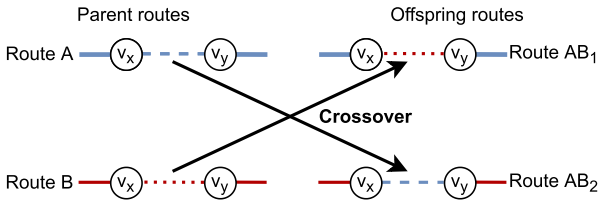

FIGURE 7. Crossover between route A and B with common intersections v_x, v_y.

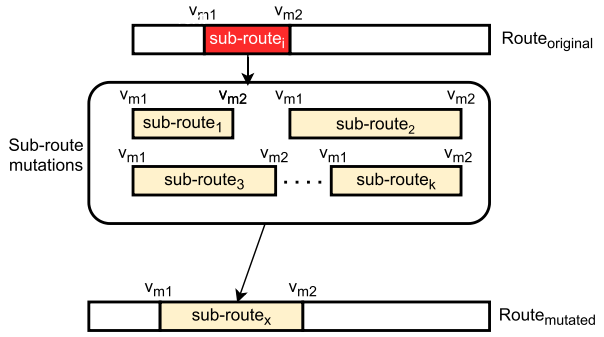

FIGURE 8. Mutation of the route selected for mutation.

To crossover the two routes, two random non-identical vertices (v_x, v_y)) are chosen from the set $C_{p'}$. Then, all the vertices and relationships are switched between the two nodes (v_x, v_y). It should be noted that the switched routes do not need to be of the same number of nodes (Fig. 7).

5) MUTATION

After the crossover is performed the routes are selected for mutation based on the mutation rate parameter (*mutation_r*). The mutation is performed so that any two points of a route are chosen randomly (v_{m1}, v_{m2}). After the points are chosen, all the vertices and paths between them are removed, and new routes are generated between them (Fig. 8). The routes are generated using the Algorithm 2, which is based on Dijkstra's algorithm [23]. The k-value chosen is the number of intersections between the chosen nodes (v_{m1}, v_{m2}) for route mutation.

Once the k-number of routes is generated, one of them is chosen randomly to replace the previous sub-route. One of the generated sub-routes is chosen randomly ($sub - route_x$) and inserted where the previously deleted sub-route ($sub - route_i$) was.

After the mutations of randomly selected routes are performed, each pair of original parent routes (R_1, R_2) has a pair of crossover offspring (R_{AB1}, R_{AB2}), where one or both of them can be mutated (if chosen randomly for mutation). From them, we create sets of four routes, two original and two new, and evaluate them according to the evaluation function previously mentioned. Based on the evaluation function, we select the two routes with the lowest possible values and replace the original two routes in them. Once we perform this evaluation step for all the parent-offspring quartets, we move to the crossover step and repeat it until the required number of generations has been reached.

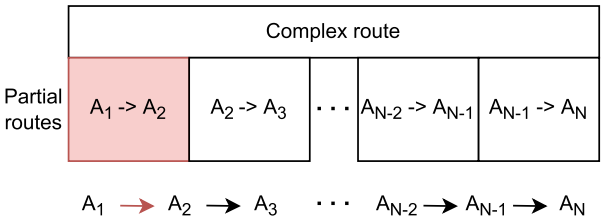

FIGURE 9. Complex route composed of succeeding partial routes.

6) GENERATING A ROUTE BETWEEN MORE TWO POINTS AND CIRCULAR ROUTES

The previously explained algorithm has been extended to enable the generation of complex routes. These are routes that, in addition to the start and end nodes, also contain an ordered list of mid nodes. In our case, this means that the route can have a list of nodes (A_1, A_2, A_3, $\ldots A_{N-2}$, A_{N-2}, A_{N-2}). These are the nodes that the route must cover in the following order, as seen in Fig.9 An array of partial routes, therefore, represents a complex route (e.g., $A_1 \rightarrow A_2$) as shown on Fig. 9. To generate a complex circular route, the ending point (A_N) must be equal to the route's starting point (A_1).

The **initial** route generation is performed by generating a population of complex routes. Each partial route is composed of the aforementioned partial paths. This is shown in the Algorithm 4. The procedure *generateNComplexRoutes* receives all the inputs that the Algorithm 2 receives, with the addition of *mid_ids*, which is an array of mid nodes that the route must contain. At the first step (**lines 2-3**), an empty routes array (*cpl_routes*) is initialized, and an array where the start node, mid nodes, and end node are concatenated. Sub-routes are generated in the while loop (**lines 5-9**). First, the start (a) and end (b) nodes are selected (**lines 6-7**) from the *all_nodes* array. This serve as the *start_id* and *end_id* input parameters of the *generateNRoutes* procedure (Algorithm 2). After the while loop is concluded the complex routes (*cpl_routes*) array is returned (**line 10**).

Algorithm 4 Generate Complex Routes Between Node List

```
1:  procedure generateNComplexRoutes(start_id, end_id,
    mid_ids, G, costs, n, p_gen_w, p_gen_c)
2:      cpl_routes = []
3:      all_nodes=[start_id + *mid_ids + end_id]
4:      i=0
5:      while i < len(all_nodes)-1 do
6:          a = all_nodes[i]
7:          b = all_nodes[i+1]
8:          cpl_routes += [generateNRoutes(a, b, n, ...)]
9:      end while
10: return cpl_routes
11: end procedure
```

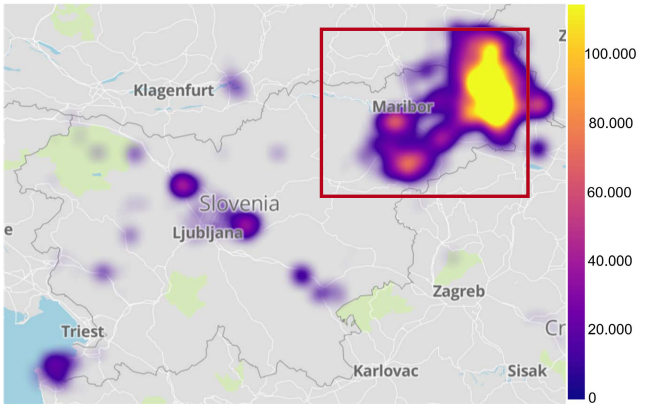

FIGURE 10. Heat-map of recorded training activities.

TABLE 10. Initialisation parameters for the algorithm.

Algorithm parameter	Value
Population size	50
Number of generations	100
Crossover rate (%) Mutation rate (%)	50
K	50

TABLE 11. Initialisation parameters for the penalties.

Algorithm parameter	Value	Type
Minimum kilometres Maximum kilometres	1	absolute
Minimum ascent Maximum ascent	0.1	
Verified bonus	0.5	Bonus 5%

The first index of the *cpl_routes* array is the **sub route** index of an individual complex route, and the second one is the **complex route** index.

This changes the **crossover** phase by allowing the operation only to be done on the partial route of the same start and end nodes. Which partial route is chosen to crossover with another partial route is selected randomly, the same as selecting the two original parent routes.

The **mutation** phase is adjusted so that mutation can only be done on individual partial routes and not the whole route, to maintain the fixed points that must remain in the complex route.

The **evaluation** of routes is performed so that each partial route's total ascent and distance are calculated, and then summed up to receive the complex-route ascent and descent, which can then be graded per the route penalty criteria.

IV. EXPERIMENTS AND RESULTS

We demonstrated the use of our proposed method by designing the requirements for several typical cycling route test cases. The routes were generated based on their parameters. The area of routes was selected based on the training data of the selected athletes, described in Table 2 from the datasets [36], [37]. The heatmap projection of training and the selected area on which the routes were generated is shown in Fig. 10. This was done to enable the use of verified weights of nodes (intersections). As seen in Fig. 10, the highest activity in the viable merged dataset was recorded in northeast Slovenia in the geographical regions of Podravje and Pomurje. To reduce the processing and track generation time, a limited area with the following boundaries: minimum latitude: 46.20, maximum latitude: 46.86, minimum longitude: 15.16, and maximum longitude: 16.37, was selected, as shown by the red rectangle on the heatmap. The resulting property graph had a size of 79,6 megabytes, and the additional data needed for visualizations was 394 megabytes (latitudes and longitude data for each point of a vertice between two edges).

The mentioned area encompasses the places of Celje, Lenart, Ljutomer, Maribor, Murska Sobota, Ormož, Radgona, Rogaška Slatina, and Slovenska Bistrica. In the given area, 40369 intersections were discovered, together with 109298 paths between them. The area is mostly flat in the region of Pomurje (near Murska Sobota and Ljutomer), and offers hilly terrain around the Pohorje mountain (near Maribor, Slovenska Bistrica.

Four routes, one for each scenario, were generated to test the proposed method:

- Case A - Route of length **25** kilometers and an ascent of 400 meters.
- Case B - Route **50** kilometers long and 450 meters of ascent.
- Case C - Route **100** kilometers long and 900 meters of ascent.
- Case D - **Circular** complex route **100** kilometers long and **1000** meters of ascent.

The initialization parameters for the algorithm were the following, as presented in Tab. 10. We tried to generate three typical training routes, short (25 kilometers), medium (50 kilometers), and long (100 kilometers). The crossover rate and mutation rate were selected at 50 %. However, it should be noted that, in our algorithm, only the crossover routes were mutated, so that means that 50 % of crossover routes were mutated in each iteration (25% of the total routes). There were 50 routes in each generation, and the algorithm generated 100 population generations.

The penalties (Table 11) were determined at 1 point deduction for each kilometer under or over the determined value, 0.1 points for each meter over or under the determined ascents. A bonus of up to 5 % was awarded to routes with more verified paths than an average path.

Each of the following subsections presents a case of a generated route (A, B, C, D). A brief description and the generated route image are provided for each case. The algorithm's progress in each case is also shown on charts which show the grade, ascent, and distance of the best route in each generation, as well as the average grade of all routes for each iteration.

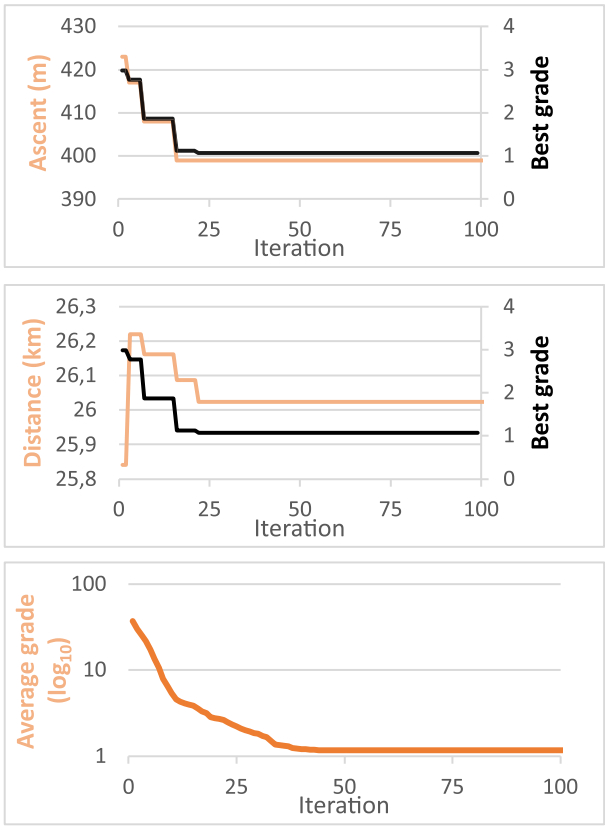

FIGURE 11. Distance in kilometers, ascent in meters, and best grade through the generations for Case A.

A. CASE A

Case A demanded a short route (25 kilometers) and was limited to 400 altitude meters in a hilly environment (starting point: Ptuj, ending point: Ivanjkovci). We also chose Ivanjkovci because conventional pathfinding using Google Maps [49] provided us with a route of 27.7 kilometers for walking and of 32 kilometers long for driving. We were interested in how the algorithm would respond to such a problem. The best route of the first generation had the best grade of 1,06 penalty points. It had a distance of 26023 meters and an ascent value of 399 meters. The route generation process is shown in Fig. 11, where the distance, ascent, and best grade are shown in each iteration, and the average grade of the routes generated.

Interestingly, the best route was found in the 20th iteration, and no better routes were found afterward, as seen in Fig. 11. It can be seen that the main problem was finding a shorter route between the two determined destinations while still conforming to the ascent criteria. The route had a length of 26.02 kilometers and covered 399 meters of ascent, which is 1.02 kilometers over the distance limit and 1 meter under the ascent limit.

The best route of the first generation is shown with a blue line, and the changes in comparison to the final version of the route are shown with a pink line in Fig. 12. It can be seen that modification happened only on the starting part of the route, and no further optimizations were found.

FIGURE 12. Visualization of the best-generated route A (blue line route).

FIGURE 13. Visualization of the best-generated route B.

B. CASE B

Case B presented regular training in a mixed environment, partially flat terrain (from Slovenska Bistrica to Ptuj) and partially hilly (from Ptuj to Lenart). The route had a required length of 50 kilometers and 450 ascent meters. The best route is shown in Fig. 13 where the blue line shows the original best route and the pink line is the final best route (modifications in comparison to the original). For brevity, the parts where changes occurred are marked with a slightly darkened circle marked by a green border. The route optimizations were reached in two parts when comparing the original best route and the ending best route.

This best solution was discovered in the 94th generation, as seen in Fig. 14.

It should be noted that because the route can only start and end in an intersection, the termination parameters should be set to a higher than zero penalty value in the future due to the high improbability of reaching the exact number of meters.

It can be noted that the initial generation best grade route had a grade of 7.10 points (average grade: 95.39), an ascent of 523 meters and a distance of 49.83 kilometers, and the final route had a distance of 50.79 kilometers and an ascent

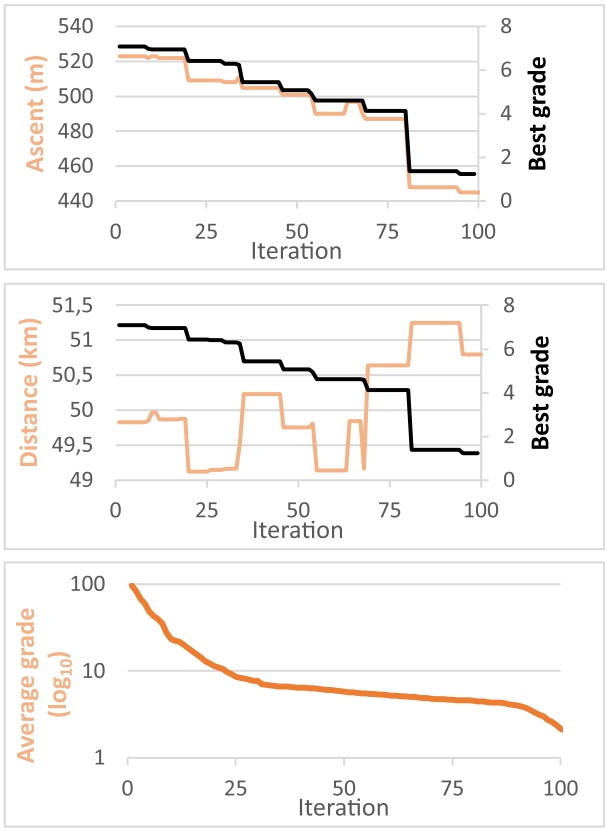

FIGURE 14. Distance in kilometers, ascent in meters, and best grade through the generations for Case B.

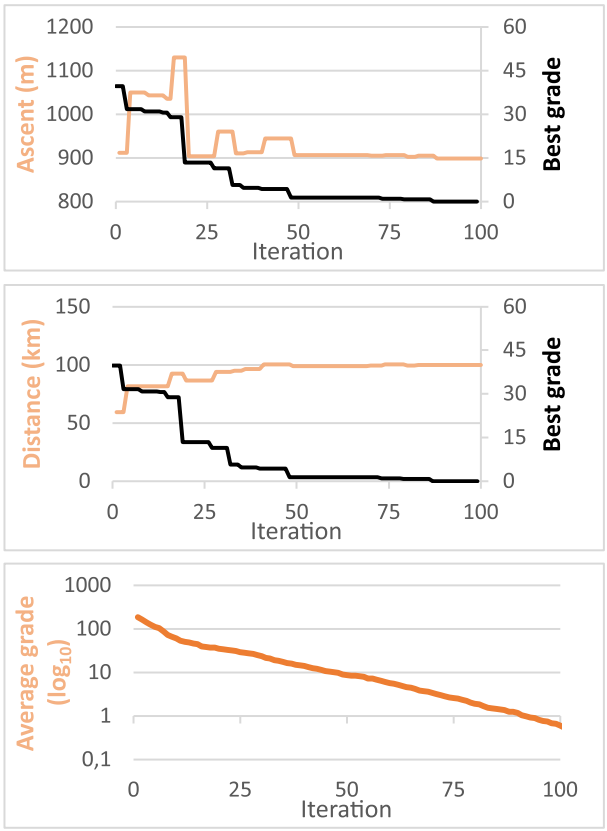

FIGURE 15. Distance in kilometers, ascent in meters, and best grade through the generations for Case C.

of 445 meters, giving it a grade of 1.22 (average grade: 2.11), it was 0.79 kilometers too long and 5 meters under the ascent threshold.

C. CASE C

Case C was a case of a long-range route with a higher number of altitude meters (100 kilometers and 900 meters of ascent) between Slovenska Bistrica and Kungota. In this case, the optimum solution was discovered in the 87th generation of routes, as seen in Fig. 15. The best solution had a penalty value of 0.0975 and covered 99.99 kilometers with 899 ascent meters. It is better to place the two points a bit closer together, so that more different routes can be found between them, within the given parameters.

We can also see from Fig. 15 that the algorithm successfully tried to find a trade-off between distance and ascent to discover the most optimum route. The final route generated was 99.9 kilometers long and had an ascent gain of 899 meters.

The first generation best route is shown, together with the final best route in Fig. 16. The blue line shows the original best route, while the yellow line shows the final version of changes to the original route to reach the optimum solution. The red-bordered circles denote four notable diversions from the original best route.

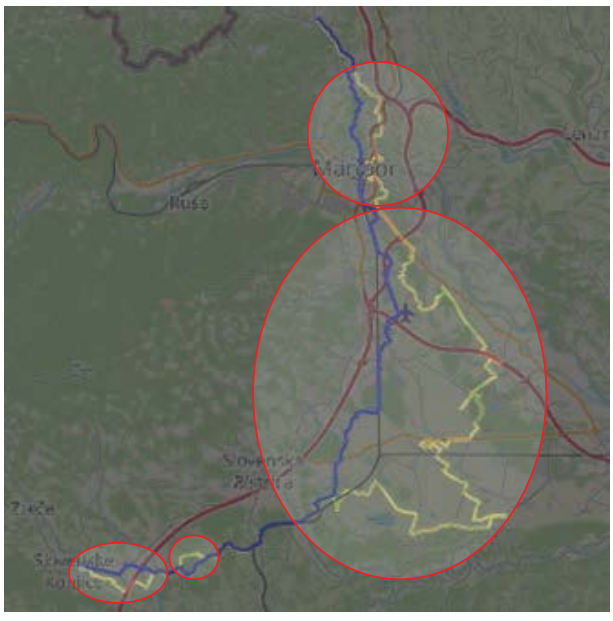

FIGURE 16. Visualization of the best-generated route in case C.

D. CASE D

Case D presents a circular route based on the complex route extension of the original algorithm. The requirements for the route were 100 kilometers and 1000 ascent meters. The

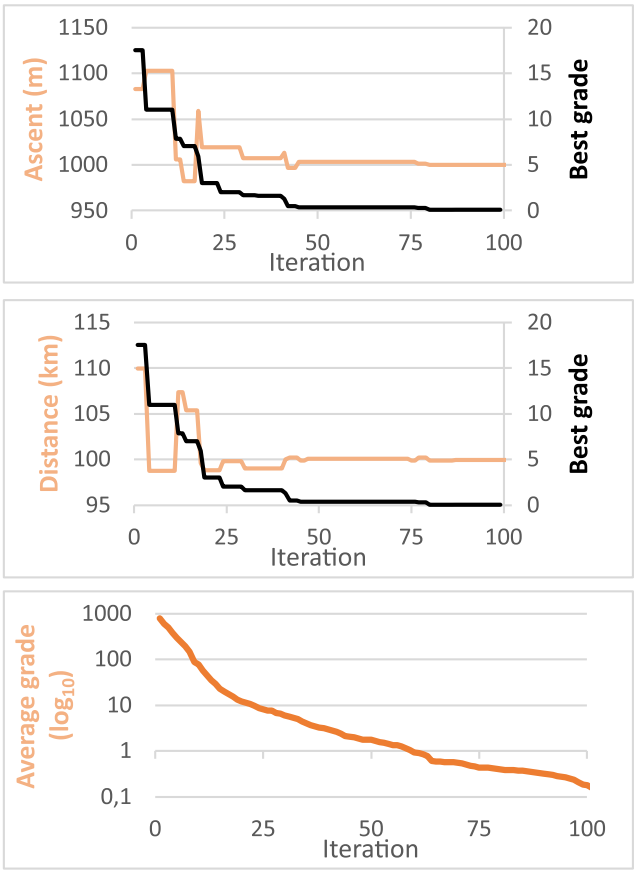

FIGURE 17. Distance in kilometers, ascent in meters, and best grade through the generations for Case D.

complex circular route was generated between the following four intersections in the given order and places:

1) Videm pri Ptuju (start)
2) Pragersko
3) Zgornja Korena
4) Videm pri Ptuju (end)

From the four intersection points, three partial routes were generated, namely: 1) Videm pri Ptuju → Pragersko, 2) Pragersko → Zgornja Korena, 3) Zgornja Korena → Videm pri Ptuju. The goal of the case was to prove the feasibility of the algorithm for generating complex routes that were also circular.

The initial best route had a grade of 17.54, and the best route overall was discovered in the 86th iteration. It had a grade of 0.0767. The route was 80 meters shorter than 100 kilometers and covered precisely 1000 meters of elevation gain. A route with a penalty smaller than one was discovered in the 41st generation, as seen in Fig. 17. The last chart, which represents the average grade of a route, is plotted on a logarithmic scale (base 10).

The best solution can be seen plotted on a map in Fig. 18. The red dot represents the starting and ending point of the route (Videm pri Ptuju), and the leftmost yellow dot (Pragersko) and topmost yellow dot (Zgornja Korena)

FIGURE 18. Visualization of the best-generated route in case D.

represent the two additional fixed points that the route must cover. The red arrows represent the direction of the route.

E. LIMITATIONS

The algorithm initialization parameters were chosen on a trial-and-error basis and were found to be suitable for generating the required routes. The values (namely population size and number of generations) were limited due to the constraints of running the algorithm on a personal computer (CPU: 7700HQ i7, 16 GB ram). The routes were only generated in the bounding map (latitude and longitude limitations) since the most viable training data was available from that area.

V. DISCUSSION

The domain of Route Generation has been studied thoroughly. While widely known algorithms exist for finding the shortest routes, such as Dijkstra [23] and Yen's K-shortest routes [48], there are much fewer possibilities for finding the exact routes, especially if there is more than one parameter that we want to control.

Several geographical domain-specific challenges were also identified, (1) Crossover can only be done between routes with common intersections, (2) Any operation of the algorithm on individual or multiple routes must maintain that each route remains completely connected, and (3) At least the starting and ending point is fixed permanently, which somewhat limits the possibilities for route generation. Each of the identified challenges was addressed in the algorithm.

We have also presented the challenge of generating the routes that need to be controlled by more than one parameter. In our case, we controlled the routes not only for their length but also for their total ascent. We have also presented the

TABLE 12. Comparison between generated routes distance and ascent and their target values.

	Distance (km)		**Ascent** (m)	
	Target	Δ	Target	Δ
Case A	25	+1.02 (+*4.08 %*)	400	-1 (*-0.25 %*)
Case B	50	+0.79 (+*1.58 %*)	450	-5 (*-1.11 %*)
Case C	100	-0.003 (≈ *0 %*)	900	-1 (≈ *0 %*)
Case D	100	-0.08 (≈ *0 %*)	1000	0

extension of the original method to allow the generation of circular routes.

We have demonstrated the feasibility of our algorithm by presenting four different cases: (A) a Short route, (B) a Medium length, (C) a Long route, and (D) A long circular route. In all cases, the generated routes were near the required values, so we can say that the algorithm solved the given problems successfully. When comparing the route ascents and lengths, no final instances of best-generated routes had any of their parameters deviate by more than 4.08 % (distance in Case A), as seen in Table 12.

A thing that has not yet been optimized is the selection of the starting and ending points, which currently need to be selected manually. The only condition to generate valid routes is that the points are not too far apart.

VI. CONCLUSION

We have found a feasible solution to the generation of cycling training routes. Our novel approach offers a way to route generation inspired by the Evolutionary Algorithm approach. We have introduced the implementation of the (1) Initialization, (2) Selection, (3) Crossover, (4) Mutation, and (5) Evaluation stages in the domain of Geographical Maps.

The proposed approach can discover new routes based on route parameters: distance, ascent, start, and end points. The approach could be extended further for additional parameters if the need for them existed. The approach has been tested and demonstrated successfully on the four presented cases of routes (25 kilometers, 50 kilometers, 100 kilometers, and 100 kilometers circular) of varying lengths. Potential further research directions include:

- extending the original property graph to include data for maximum and average slope inclines (degrees) between intersections;
- optimization and sensitivity analysis of the algorithm parameters so that optimum solutions could be found faster and with fewer resources;
- adapting the route generating algorithm to include route generation for running.

REFERENCES

[1] C. Giattino (2020). *Working Hours*. Accessed: Jul. 2, 2022. [Online]. Available: https://ourworldindata.org/working-hours

[2] *Performing (Non-Work-Related) Physical Activities by Sex, Age and Income Quintile*, 2020th ed., Eurostat, Publications Office of the European Union, Luxembourg, Luxembourg, 2020.

[3] R. G. Noll, "The organization of sports leagues," *Oxford Rev. Econ. Policy*, vol. 19, no. 4, pp. 530–551, Dec. 2003, doi: 10.1093/oxrep/19.4.530.

[4] I. Fister, "Fundamentals of sports training theory," in *Computational Intelligence in Sports* (Adaptation, Learning, and Optimization), 1st ed. Cham, Switzerland: Springer, 2019, pp. 105–106, doi: 10.1007/978-3-030-03490-0.

[5] A. Rajšp and I. Fister, "A systematic literature review of intelligent data analysis methods for smart sport training," *Appl. Sci.*, vol. 10, no. 9, p. 3013, Apr. 2020, doi: 10.3390/app10093013.

[6] I. Fister, "Coaching science," in *Computational Intelligence in Sports* (Adaptation, Learning, and Optimization), 1st ed. Cham, Switzerland: Springer, 2019, pp. 105–106, doi: 10.1007/978-3-030-03490-0.

[7] I. Fister, "Design and implementation of an artificial sports trainer," in *Computational Intelligence in Sports* (Adaptation, Learning, and Optimization), 1st ed. Cham, Switzerland: Springer, 2019, pp. 121–129, doi: 10.1007/978-3-030-03490-0.

[8] I. Fister, S. Rauter, X.-S. Yang, and K. Ljubič, and I. Fister, Jr., "Planning the sports training sessions with the bat algorithm," *Neurocomputing*, vol. 149, pp. 993–1002, Feb. 2015, doi: 10.1016/J.NEUCOM.2014.07.034.

[9] K. Brzostowski, J. Drapała, A. Grzech, and P. Świątek, "Adaptive decision support system for automatic physical effort plan generation-data-driven approach," *Cybern. Syst.*, vol. 44, nos. 2–3, pp. 204–221, Mar. 2013, doi: 10.1080/01969722.2013.762260.

[10] T. Skerik, L. Chrpa, W. Faber, and M. Vallati, "Automated training plan generation for athletes," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2018, pp. 3865–3870, doi: 10.1109/SMC.2018.00655.

[11] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *Proc. Int. Conf. Global Trends Signal Process., Inf. Comput. Commun. (ICGTSPICC)*, Dec. 2016, pp. 261–265, doi: 10.1109/ICGTSPICC.2016.7955308.

[12] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, Mar. 2011, doi: 10.1016/J.SWEVO.2011.03.001.

[13] R. Kruse, "1.2 computational intelligence," in *Computational Intelligence* (Texts in Computer Science). London, U.K.: Springer, 2016, pp. 2–3, doi: 10.1007/978-1-4471-7296-3.

[14] S. Rauter, "New approach for planning the mountain bike training with virtual coach," *Trends Sport Sci.*, vol. 25, no. 2, pp. 69–74, 2018, doi: 10.23829/TSS.2018.25.2-2.

[15] N. Kumyaito and K. Tamee, "Intelligence planning for aerobic training using a genetic algorithm," in *Proc. Int. Symp. Natural Lang. Process.* Cham, Switzerland: Springer, 2016, pp. 196–207, doi: 10.1007/978-3-319-70016-8_17.

[16] A. Le, T. Jaitner, F. Tobias, and L. Litz, "A dynamic heart rate prediction model for training optimization in cycling," *Eng. Sport*, vol. 7, pp. 425–433, Jan. 2008.

[17] D. Schaefer, A. Asteroth, and M. Ludwig, "Training plan evolution based on training models," in *Proc. Int. Symp. Innov. Intell. Syst. Appl. (INISTA)*, Sep. 2015, pp. 1–8.

[18] I. Fister, Jr., "Population-based metaheuristics for planning interval training sessions in mountain biking," in *Proc. Int. Conf. Swarm Intell.* Cham, Switzerland: Springer, 2019, pp. 70–79, doi: 10.1007/978-3-030-26369-0_7.

[19] K. Mutijarsa, M. Ichwan, and D. B. Utami, "Heart rate prediction based on cycling cadence using feedforward neural network," in *Proc. Int. Conf. Comput., Control, Informat. Appl. (IC3INA)*, Oct. 2016, pp. 72–76, doi: 10.1109/IC3INA.2016.7863026.

[20] I. Fister, D. Fister, and I. Fister, "Topology-based generation of sport training sessions," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 1, pp. 667–678, Jan. 2021, doi: 10.1007/s12652-020-02048-1.

[21] E. M. Burford, "Predicting cycling performance using machine learning," Ph.D. thesis, Dept. Comput. Sci., Wake Forest Univ., Winston-Salem, NC, USA, 2020.

[22] L. Zahran, "A conceptual framework for the generation of adaptive training plans in sports coaching," in *Proc. Int. Conf. Adv. Intell. Syst. Inf.* Cham, Switzerland: Springer, 2019, pp. 673–684, doi: 10.1007/978-3-030-31129-2_62.

[23] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Oct. 1959.

[24] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, Aug. 1989, doi: 10.1287/IJOC.1.3.190.

[25] N. Kumyaito, P. Yupapin, and K. Tamee, "Planning a sports training program using adaptive particle swarm optimization with emphasis on physiological constraints," *BMC Res. Notes*, vol. 11, no. 1, pp. 1–6, Dec. 2018, doi: 10.1186/s13104-017-3120-9.
[26] S. Huber, "Synthetization of bicycle route data from aggregate GPS-based cycling data and its utility for bicycle route choice analysis," in *Proc. 7th Int. Conf. Models Technol. Intell. Transp. Syst. (MT-ITS)*, Jun. 2021, pp. 1–6, doi: 10.1109/MT-ITS49943.2021.9529316.
[27] Naviki. (2021). *Naviki | World-Wide Bicycle Route Planner and Map*. Accessed: Mar. 2, 2022. [Online]. Available: https://www.naviki.org/
[28] R. Walz. (2021). *Routeshuffle*. Accessed: Jun. 12, 2022. [Online]. Available: https://routeshuffle.com/
[29] Bikemap. *Cycle Route Planner| Bikemap Your Bike Routes*. [Online]. Available: https://www.bikemap.net/en/routeplanner/
[30] Ride with GPS. (2021). *Bike Route Planner Ride With GPS*. Accessed: Jun. 11, 2022. [Online]. Available: https://ridewithgps.com/routes/new
[31] J. Hrncir, Q. Song, P. Zilecky, M. Nemet, and M. Jakob, "Bicycle route planning with route choice preferences," *Frontiers Artif. Intell. Appl.*, vol. 263, pp. 1149–1154, Jan. 2014, doi: 10.3233/978-1-61499-419-0-1149.
[32] Openstreetmap Foundation. (2021). *OpenStreetMap*. [Online]. Available: https://www.openstreetmap.org/
[33] T. G. Farr, "The shuttle radar topography mission," *Rev. Geophys.*, vol. 45, no. 2, pp. 1–33, May 2007, doi: 10.1029/2005RG000183.
[34] G. Van Rossum and F. L. Drake, Jr., *Python Reference Manual*. Amsterdam, The Netherlands: Centrum voor Wiskunde en Informatica Amsterdam, 1995.
[35] European Environment Agency. (2017). *EU-DEM 1.1*. Accessed: Jan. 19, 2022. [Online]. Available: https://www.eea.europa.eu/data-and-maps/data/copernicus-land-monitoring-service-eu-dem
[36] I. Fister, Jr., "A collection of sport activity datasets with an emphasis on powermeter data," Univ. Maribor, Fac. Elect. Eng. Comput. Sci., Maribor, Slovenia, Tech. Rep., 2017. [Online]. Available: http://www.iztok-jr-fister.eu/static/css/datasets/Sport.zip
[37] I. Fister, Jr., "A collection of sport activity datasets for data analysis and data mining 2017a," Univ. Maribor, Fac. Elect. Eng. Comput. Sci., Maribor, Slovenia, Tech. Rep., 2017. [Online]. Available: https://academictorrents.com/details/f2221a292540ff3e6c85025754f775361c7cd886
[38] OpenStreetMap Foundation. (2021). *Overpass API OpenStreetMap Wiki*. Accessed: Dec. 8, 2021. [Online]. Available: https://wiki.openstreetmap.org/wiki/Overpass_API
[39] DHI Gras. (2014). *EU-DEM Statistical Validation Report*. DHI GRAS. [Online]. Available: www.dhi-gras.com
[40] J. R. Lourenço. (2021). *Open-Elevation API*. Accessed: Dec. 8, 2021. [Online]. Available: https://open-elevation.com/
[41] A. Rajšp, "Preprocessing of roads in OpenStreetMap based geographic data on a property graph," in *Proc. Central Eur. Conf. Inf. and Intell. Syst.*, 2021, pp. 193–199. [Online]. Available: http://archive.ceciis.foi.hr/app/public/conferences/2021/Proceedings/IS/IS3.pdf
[42] Neo4j Inc. (2022). *Graph Data Platform | Graph Database Management System | Neo4j*. Accessed: Feb. 8, 2022. [Online]. Available: https://neo4j.com/
[43] Redis Ltd. (2022). *RedisGraph—A Graph Database Module for Redis*. Accessed: Apr. 22, 2022. [Online]. Available: https://oss.redis.com/redisgraph/
[44] I. Fister, L. Lukac, A. Rajsp, I. Fister, L. Pecnik, and D. Fister, "A minimalistic toolbox for extracting features from sport activity files," in *Proc. IEEE 25th Int. Conf. Intell. Eng. Syst. (INES)*, Jul. 2021, pp. 121–126, doi: 10.1109/INES52918.2021.9512927.
[45] J. Bleiholder and F. Naumann, "Data fusion," *ACM Comput. Surv.*, vol. 41, no. 1, pp. 1–41, Jan. 2009, doi: 10.1145/1456650.1456651.
[46] D. Whitley, "A genetic algorithm tutorial," *Statist. Comput.*, vol. 4, no. 2, pp. 65–85, Jun. 1994, doi: 10.1007/BF00175354.
[47] S. Mirjalili, *Genetic Algorithm*. Cham, Switzerland: Springer, 2019, pp. 43–55, doi: 10.1007/978-3-319-93025-1_4.
[48] J. Y. Yen, "Finding the K shortest loopless paths in a network," *Manage. Sci.*, vol. 17, pp. 712–716, Jul. 1971.
[49] Google LLC. *Google Maps (Ptuj to Ivanjkovci)*. Accessed: Jul. 3, 2022. [Online]. Available: https://www.google.com/maps/dir/2250+Ptuj/Ivanjkovci,+2259/@46.4451379,15.940492,12z/data=x!3m1!4b1!4m14!4m13!1m5!1m1!1s0x476f669e5ce0430d:0x400f81c823ff4b0!2m2!1d15.8696884!2d46.4199535!1m5!1m1!1s0x476f452588df10a7:0xe0a43fe235d251a0!2m2!1d16.151144!2d46.4652005!3e2

ALEN RAJŠP received the B.Sc. degree in computer science and the M.Sc. degree in informatics from the University of Maribor, Slovenia, in 2016 and 2018, respectively, where he is currently pursuing the Ph.D. degree in computer science. He is also a Teaching Assistant at the University of Maribor. His research interests include computational intelligence, sports science, and gamification.

IZTOK FISTER JR. (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Maribor, Slovenia. He is currently an Assistant Professor at the University of Maribor. He has published more than 120 research papers in refereed journals, conferences, and book chapters. His research interests include data mining, pervasive computing, optimization, and sports science. He has acted as a program committee member of more than 30 international conferences. Furthermore, he is a member of the editorial boards of three different international journals.