Running Routes For Out-of-Town Runner

Shirui Chen
Northeastern University
Vancouver British Columbia, Canada
chen.shir@northeastern.edu

Mon-Shan Lin
Northeastern University
Vancouver British Columbia, Canada
mon.lin@northeastern.edu

Tao Wu
Northeastern University
Vancouver British Columbia, Canada
wu.tao1@northeastern.edu

Wenjing Yang
Northeastern University
Vancouver British Columbia, Canada
yang.wenji@northeastern.edu

Abstract

This paper presents a customizable route generation system tailored to runners, particularly those navigating unfamiliar urban environments. Unlike traditional route-finding solutions such as Google Maps or existing runner apps like Strava, our approach prioritizes user preferences for distance, elevation, and points of interest (POIs). The system generates multiple candidate routes using a bi-directional A* algorithm, which improves search efficiency by expanding paths from both the start and end points. A priority-based selection mechanism is then employed to choose the optimal route, allowing users to specify preferences for elevation range and minimum POIs while assigning a first and second priority to ensure alignment with their specific goals.

The system integrates real-world geographic data from OpenStreetMap (OSM) and elevation data from the PyHigh Package, enabling dynamic customization for both circular and point-to-point routes. Compared to existing solutions, it uniquely supports customizable start and end points, avoids default circular route constraints, and incorporates detailed POI integration, including visual markers on the map. While the application is currently limited to a web-based platform and predefined geographic regions, future work aims to extend multi-platform compatibility and incorporate real-time data sources to further enhance usability. This solution bridges critical gaps in current route generation tools, providing a highly personalized and adaptive experience that improves the effectiveness and enjoyment of urban running.

ACM Reference Format:
Shirui Chen, Mon-Shan Lin, Tao Wu, and Wenjing Yang. 2024. Running Routes For Out-of-Town Runner. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Runners, especially those in unfamiliar areas, often need more than the shortest path between two points. Key factors like distance, elevation, and preferences for POIs play a significant role in creating enjoyable and effective routes. Traditional route-finding algorithms [15] typically focus on efficiency by finding the shortest or fastest path, but they often overlook the personalized needs of runners. Popular applications like Strava [14] and Runkeeper [2] offer limited customization, focusing mainly on short paths and neglecting important factors like elevation changes or the ability to avoid or include POIs.

To address these limitations, this project develops a dynamic route generation system tailored specifically for runners, capable of creating customizable routes—either circular or point-to-point—that balance distance, elevation, and POIs according to the runner's preferences. The core challenge lies in designing an algorithm that effectively balances these user-defined factors, allowing runners to dynamically generate routes that meet varying preferences such as avoiding or encountering POIs, setting a desired route distance, and maximizing or minimizing elevation changes.

This project seeks to answer the research question: *How can a route generation system optimally balance distance, elevation, and the number of POIs while allowing flexible user input to create personalized routes?* By addressing this challenge, the system will provide a more tailored and enjoyable running experience, better aligning with individual fitness goals.

2 Related Work

Integrating user preferences with real-world constraints is essential for developing a practical and personalized route generation system for runners. Previous research [11, 15] highlights the importance of balancing factors such as POIs, elevation preferences, and route characteristics to create tailored running experiences. Our approach builds on these findings by introducing a priority-based system that enables users to specify the importance of POIs and elevation changes, providing flexibility to adapt to diverse running goals.

2.1 Multi-Objective Optimization in Route Planning

Non-Dominated Sorting Genetic Algorithm II (NSGA-II): NSGA-II [7] has been widely used for multi-objective optimization, particularly for generating *Pareto-optimal solutions*, which balance

competing factors like POI density and elevation without sacrificing one for the other. The computational overhead of NSGA-II, while high, provides a framework for understanding how multiple objectives can be balanced in a static setting. This inspired our priority-based approach to ensure user-defined preferences are methodically addressed.

Ant Colony Optimization (ACO): ACO [3] excels in environments requiring adaptation to changing conditions by mimicking the behavior of ants to find efficient paths. Although its adaptability is noteworthy, its slow convergence in large datasets made it less practical for our application, which prioritizes efficiency in static route calculations. ACO's adaptability highlighted the importance of incorporating user-defined constraints during route selection, informing our approach.

Multi-Objective A* (MOA*): MOA* [4] integrates multiple objectives, such as distance, elevation, and POIs, directly into the pathfinding process by extending the traditional A* algorithm. While MOA* guarantees optimal solutions with an admissible heuristic, its complexity and computational cost led us to develop a simpler, priority-based system. Nonetheless, MOA*'s methodology for balancing multiple objectives guided our design of filtering routes based on user priorities.

2.2 Pathfinding Algorithms

Traditional A* and Bi-Directional A*: Traditional A* [8] is a foundational pathfinding algorithm, but its performance often suffers from excessive pruning and backtracking, particularly under strict constraints like fixed route distances. Bi-directional A* [13] addresses this limitation by expanding the search from both the start and goal nodes, meeting at a midpoint to reduce redundant node expansions and improve computational efficiency. The insights from Pohl's work directly influenced our choice of bi-directional A* for route generation, allowing us to generate approximately 100 valid candidate routes within the specified distance range, ensuring efficient pathfinding.

2.3 Tour Recommendation Systems

Tour recommendation systems, such as those proposed by Gionis et al. [9] and Yoon et al. [16], utilize geotagged data and venue types to create personalized routes for tourists. These systems offer valuable strategies for route generation by integrating POIs into the planning process. However, their focus on static POI recommendations and lack of flexibility for dynamic parameters such as elevation changes limits their applicability for runners. These studies emphasized the need for route systems to adapt to more specific requirements, such as elevation constraints, which influenced our approach to prioritizing user-defined elevation ranges during route selection.

2.4 SLAM and Skyline Operator

SLAM (Simultaneous Localization and Mapping): While SLAM [5] is extensively used in robotics for mapping and navigation in unknown environments, its real-time mapping capabilities were unnecessary for our project, which uses predefined geographic datasets from OpenStreetMap (OSM) [6]. Nonetheless, the concept of SLAM's adaptability reinforced the importance of handling user constraints dynamically within our static system.

Skyline Operator: The Skyline Operator [12] is effective for multi-criteria decision-making by identifying non-dominated solutions across multiple objectives. Although primarily used in environments requiring immediate response, the principles of the Skyline Operator informed our method for filtering and selecting routes based on user-defined priorities, enabling effective decision-making without the need for real-time computation.

2.5 Incorporation into Our Approach

Ultimately, our route planning algorithm does not rely on MOA*, NSGA-II, or ACO due to their high complexity and computational requirements. Instead, a **priority-based system** was developed to simplify multi-objective optimization, which allows users to set their preferences for elevation changes and POIs. The first priority is strictly satisfied during route selection, while the second priority is optimized within the remaining constraints. This approach was inspired by the strengths and limitations of the above algorithms and research, enabling us to create a system that is both efficient and responsive to the nuanced needs of runners.

3 Data Collection and Processing

3.1 Geographic Data Sources

This project utilized geographic data from **OpenStreetMap (OSM) [6]**, a widely recognized open-source geographic database. The dataset focuses on the **British Columbia (BC)** region and includes three key elements essential for constructing running routes:

- **Nodes**: Represent specific geographic points with latitude and longitude, such as intersections or landmarks.
- **Ways**: Collections of nodes forming linear features, like roads, trails, or paths.
- **Relations**: Group nodes and ways to define complex structures, such as routes or administrative boundaries.

These elements form the backbone of our graph-based route planning system, providing a detailed representation of the region's geography.

3.2 Data Processing and Preparation

To generate routes optimized for user preferences, the raw OSM data underwent a structured transformation and enhancement process.

Initially, the OSM XML files were preprocessed using **osm4-routing**[10], a tool specifically designed for graph-based routing applications. This tool extracted relevant geographic elements such as **nodes** and **ways**, along with their attributes like road type and length. A key step in this process was simplifying the graph by merging *non-decision points*, which are nodes connecting exactly two edges without branching, into single edges. This significantly reduced graph complexity while preserving decision-critical points, such as intersections. The resulting data, simplified into nodes and edges, was exported in CSV format for direct use in routing algorithms.

Points of Interest (POIs) were extracted from OSM data based on specific tags to identify relevant locations like parks, monuments, and attractions. However, the raw POI dataset contained significant noise, including duplicate or irrelevant entries, which affected its utility. To address this, the project performed a manual data cleaning

process to improve data quality, reducing the dataset from an initial 7,080 entries to a refined set of 4,247 POIs. This cleaned dataset was integrated into the graph, enabling users to customize routes based on the desired density of POIs.

Elevation data was incorporated using the **PyHigh Package** [1], which retrieves accurate elevation values for geographic coordinates. Each node in the graph was assigned an elevation attribute, allowing the system to consider vertical profiles in route planning. Users can specify their preferred elevation change range, enabling them to select routes with minimal elevation changes for a flatter experience, or routes with significant elevation variation for a more challenging run. The integration of elevation data ensures that both geographic and vertical attributes are harmonized, allowing for precise route optimization tailored to these user-defined preferences.

The final step involved constructing an undirected graph for routing by combining the processed geographic and elevation data. Attributes such as road type, surface type, and edge length were assigned to edges, creating a comprehensive dataset for routing algorithms to consider during optimization. Throughout this process, consistency checks were performed to ensure synchronization between geographic and elevation data, minimizing errors and guaranteeing the reliability of the generated routes.

3.2.1 POI Processing Details. Nodes extracted directly from OSM using specific tags (**leisure**, **historic**, and **tourism**) were flagged as POIs and incorporated into the route graph. This allowed the system to calculate the density of POIs along different routes, enabling users to influence the level of interaction with these points without selecting specific types.

3.2.2 Integration Details. After importing data from the `nodes.csv` and `edges.csv` files into the `nodes` and `edges` tables, two temporary tables are created: `temp_node_elevation` for elevation data and `temp_node_pois` for POI data. These temporary tables are then used to update the `nodes` table by adding elevation values and marking points of interest (`is_poi = true`).

3.3 Importance of Data Quality

The reliability and usability of the generated routes depend heavily on the quality of the underlying data. While OSM is a comprehensive open-source dataset, it contains inconsistencies, missing attributes, and noise, especially in less densely mapped areas. To address this, the project performed extensive data cleaning and validation, ensuring that only accurate and relevant elements were included in the graph. This process enhanced the precision and applicability of the routing algorithm. Additionally, continuous synchronization between geographic and elevation data ensures that each node's attributes are accurate, further bolstering the system's reliability and user-specific route optimization.

4 Algorithm Design

The algorithm for route planning consists of two primary components: **Route Generation** and **Route Selection**. These components work together to generate a diverse set of candidate routes that meet user-defined constraints and select the optimal route based on user priorities.

4.1 Route Generation

Efficient route generation is fundamental to this project. To enhance performance, particularly in dealing with large graphs or complex routing requirements, the **Bi-Directional A* algorithm** is implemented. This advanced version of the traditional A* algorithm initiates search fronts simultaneously from both the starting node and the target node, meeting in the middle to finalize the route. This bidirectional approach significantly reduces computational overhead and optimizes the adherence to user-defined constraints.

The traditional A* algorithm, foundational in pathfinding, balances two essential components:

- The actual cost from the start node to the current node ($g(n)$).
- A heuristic estimate of the remaining cost to the goal ($h(n)$).

While this balance enables the A* algorithm to efficiently explore potential paths, the unidirectional nature of the traditional approach can lead to excessive node exploration. This becomes particularly problematic in scenarios involving extensive graphs or routes with stringent constraints, making the process computationally intensive. Moreover, when complex path constraints are applied, such as specific distance limitations or required path features, the backtracking process may slow significantly. In extreme cases, especially when dealing with very long distances, the computation time might exceed practical limits (e.g., over a minute), potentially resulting in the failure to produce a viable route.

By leveraging the Bi-Directional A* algorithm, our system more effectively manages these challenges, enhancing the efficiency and accuracy of route generation. This method not only speeds up the process but also increases the precision with which it adheres to predefined path constraints, ensuring that the routes generated meet the specific needs and preferences of the users.

Bi-Directional A* algorithm is adopted to address these limitations, where the search is conducted from both the starting node and the goal node simultaneously. The key advantages of this approach include:

(1) **Two Search Fronts:** Instead of expanding outward from a single node, Bi-Directional A* alternates between expanding the forward search from the start node and the backward search from the target node. The two search fronts aim to meet in the middle, significantly reducing the number of nodes explored.
(2) **Distance Constraints:** The algorithm enforces strict distance constraints to ensure that generated routes fall within the range:

$$D_{\min} = 0.85 \times \text{input\_distance}, \quad D_{\max} = 1.15 \times \text{input\_distance}.$$

Routes exceeding $D_{\max}$ are pruned during the search, while those below $D_{\min}$ are filtered out in a post-processing step.
(3) **Path Merging:** When the forward and backward search fronts meet at a common node, their respective paths are combined. The total path length is calculated, and only those paths falling within the valid distance range are retained for further evaluation.

Heuristic Selection. In traditional A*, the heuristic function $h(n)$ is critical for guiding the search toward the goal efficiently. However, this project intentionally set $h(n) = 0$, effectively turning the

algorithm into a uniform-cost search. This decision was made based on the following factors:

(1) **Complexity of Constraints:** Unlike standard shortest-path problems, the algorithm must satisfy additional constraints, such as distance ranges and user-defined preferences for elevation change or POI count. These constraints render typical heuristic functions (e.g., Euclidean distance) less effective, as they do not account for these nuanced factors.
(2) **Empirical Observations:** In practical testing within the project, the use of heuristics did not lead to significant improvements in computational performance or search quality. Specifically, the paths generated using elevation and POI as heuristics showed no substantial difference compared to those generated with $h(n) = 0$ in real-world scenarios.

By combining the dual-front search strategy of Bi-Directional A* with a heuristic-free approach, this algorithm minimizes the number of nodes explored compared to traditional A*, effectively reducing the search space and computational overhead. Additionally, it demonstrates excellent scalability, remaining computationally efficient even when applied to large graphs and accommodating diverse user inputs. These benefits make the Bi-Directional A* algorithm well-suited for the complex requirements of the project.

4.2 Route Selection

Once the candidate routes are generated, the algorithm evaluates them based on user-defined preferences for **elevation change** and **minimum POI count**. Users specify two levels of priority:

(1) **First Priority:** The primary criterion that must be strictly satisfied.
(2) **Second Priority:** A secondary criterion that can be partially sacrificed if necessary.

Elevation Change Calculation. The elevation change of a route is quantified as the total absolute elevation differences between consecutive nodes along the path. Mathematically, for a route consisting of n nodes with elevations $e_1, e_2, \ldots, e_n$, the total elevation change E_{total} is computed as:

$$E_{\text{total}} = \sum_{i=1}^{n-1} |e_{i+1} - e_i|$$

For instance, consider a route with three nodes where the elevations are $e_1 = 50\,\text{m}$, $e_2 = 0\,\text{m}$, and $e_3 = 200\,\text{m}$. The total elevation change is calculated as:

$$E_{\text{total}} = |e_2 - e_1| + |e_3 - e_2| = |0 - 50| + |200 - 0| = 50 + 200 = 250\,\text{m}.$$

This metric provides a numerical representation of the physical effort required to traverse the route, with higher values indicating a more challenging path due to significant elevation variations.

POIs Consideration. To enhance the recreational or cultural value of a route, the number of POIs encountered along the path is considered. The system counts only those POIs that are directly on the route, ensuring relevance to the runner's physical journey. Mathematically, the total number of POIs encountered along a route, P_{total}, can be expressed as:

$$P_{\text{total}} = \sum_{i=1}^{n} p_i$$

Here, n is the total number of nodes on the route, and p_i is a binary indicator for whether a POI is present at node i ($p_i = 1$ if the node has a POI, otherwise $p_i = 0$).

For example, consider a route with 4 nodes, where $p_1 = 1$ (POI present), $p_2 = 0$ (no POI), $p_3 = 1$ (POI present), and $p_4 = 0$ (no POI). The total POI count is calculated as:

$$P_{\text{total}} = p_1 + p_2 + p_3 + p_4 = 1 + 0 + 1 + 0 = 2.$$

This approach ensures precise integration of POIs, directly enhancing the user's experience by focusing on POIs that are encountered along the exact route path.

Filtering Based on First Priority. The algorithm first evaluates all candidate routes against the user's specified first priority. Routes that do not meet the first priority requirement are immediately filtered out. For example:

- If the first priority is an elevation change range (e.g., 400–600m), any route with an elevation change outside this range is discarded.
- If the first priority is a minimum POI count (e.g., at least 10 POIs), routes with fewer than 10 POIs are discarded.

If none of the generated routes satisfy the first priority, the algorithm raises an alert, notifying the user that no valid routes exist under the current constraints. Additionally, the algorithm provides feedback on the achievable range of the first priority based on the available routes. For instance, if all routes have elevation changes between 0–400m, the system informs the user that the achievable elevation change range is 0–400m, suggesting adjustments to the input constraints.

Rank Based on Second Priority. After filtering routes based on the first priority, the algorithm evaluates the remaining routes using the second priority to rank them. The second priority represents a user preference that can be optimized within the constraints of the first priority. Unlike the strict filtering applied for the first priority, routes are ranked based on how closely they align with the user-defined second priority factor. The higher the rank of a route, the closer it matches the user's preference for the second priority, while still adhering to the constraints imposed by the first priority.

To illustrate this process, consider the following scenario: - **First Priority:** Elevation change must be within the range $200\,\text{m} \leq E_{\text{change}} \leq 400\,\text{m}$. - **Second Priority:** The number of POIs along the route, with a user-specified preference for at least 5 POIs.

Suppose there are four candidate routes after applying the first priority filter. The elevation change and POI count for each route are as follows:

	Elevation Change (m)	POI Count	Difference	Rank
Route A	300	3	$\|3-5\| = 2$	2
Route B	350	1	$\|1-5\| = 4$	4
Route C	250	2	$\|2-5\| = 3$	3
Route D	400	4	$\|4-5\| = 1$	1

Here, the POI count for each route is compared to the user preference of 5 POIs, and the absolute difference $|POIs - 5|$ is calculated. Routes with a smaller difference are ranked higher, as they are closer to the user's second priority preference.

In this example:

- **Route D** ranks highest because it has 4 POIs, which is the closest to the user's second priority preference.
- **Route B** ranks lowest as it deviates the most from the desired POI count.

Final Route Selection. The final output of the algorithm is a single route that satisfies the user-defined constraints. This route is selected based on strict adherence to the first priority and rank of the second priority. By presenting only one route, the algorithm ensures that the result is both actionable and aligned with the user's preferences.

5 Implementation and System Architecture

The system architecture for generating optimized running routes is designed to efficiently handle user inputs, process data, generate routes, and display the results interactively. Each component in the architecture plays a crucial role, as illustrated in Figure 1.

The process begins with the Input Form, where users input their preferences for distance, elevation, and Points of Interest (POIs). This input is then processed by the Priority Selector, which organizes these preferences according to their assigned priorities.

The Data Processing block manages the integration and preparation of various data sources necessary for route creation:

- **OSM Data:** Geographic data from OpenStreetMap used for mapping the routes.
- **POI Data:** Points of interest that enhance the route's relevance and appeal.
- **Elevation Data:** Topographical data to match the user's elevation preferences.

This data is converted into a compatible format and stored in the Database System for efficient querying.

In the backend, the Data Fetcher retrieves this processed data, which is then used by the Route Generation component to construct potential routes. The Route Selector evaluates these routes based on the predefined priorities and selects the optimal path.

If a valid route is found, it is displayed to the user through the Result Display. If no suitable route can be found, an Alert Notification is triggered, offering suggestions for alternative preferences or modifications to ensure a route can be generated.

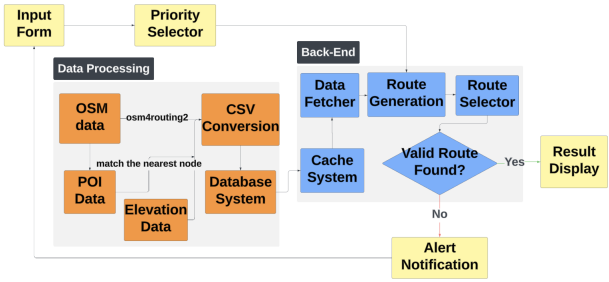


Figure 1: System Architecture Overview

5.1 Data Processing

The first stage of the system involves processing raw data to create a graph suitable for route generation. The project utilizes two primary data sources:

- **OpenStreetMap (OSM)**: Provides detailed geographic data, including roads, paths, and trails, along with POI information.
- **PyHigh Package**: Supplies elevation data for each geographic node, enabling customization of routes based on user preferences for elevation changes.

Data processing begins with extracting nodes, ways, and relations from OSM data using the `osm4routing` tool. This tool simplifies the dataset by reducing non-decision nodes and outputs a graph in CSV format, containing `nodes.csv` and `edges.csv`. POI nodes are filtered based on their tags and matched to the nearest geographic nodes, while elevation data is integrated by associating elevation values with each node in the graph. The processed data, including POI attributes and elevation, is stored in a database for efficient access during route generation.

5.2 User Input and Priority Selector

Users provide inputs through an interactive form, specifying:

- Desired route distance (input distance),
- Elevation change range (e.g., 200–400m),
- Minimum number of POIs,
- Priority (elevation or POIs as the first priority).

The **Priority Selector** processes these inputs to establish a hierarchy of criteria:

- The first priority must be strictly satisfied during route selection.
- The second priority can be optimized after the first priority is met.

5.3 Route Generation and Filtering

The back-end system retrieves the processed graph from the local cache to ensure quick response times. Route generation uses the **bi-directional A* algorithm**, which expands search fronts from both the start and end nodes simultaneously. The algorithm is constrained by the input distance, ensuring that only routes within 15% of the specified distance are considered:

$$D_{\text{min}} = 0.85 \times \text{input\_distance}, \quad D_{\text{max}} = 1.15 \times \text{input\_distance}$$

Routes outside this range are filtered out during or after the search.

Approximately 100 candidate routes are generated, each meeting the distance constraint. These routes are then passed to the **Route Selector** for further filtering based on user-defined priorities.

5.4 Route Selector and Priority-Based Optimization

The **Route Selector** evaluates the candidate routes against the user's first priority:

- For elevation, the total elevation change is calculated as the absolute sum of elevation differences along the route.
- For POIs, the count of POIs along the route is considered.

Routes that do not satisfy the first priority are eliminated. If no routes meet the first priority, an alert is triggered, notifying the user and providing actionable feedback. For example, if the user specifies an elevation change range of 300–500m, but all generated routes fall outside this range (e.g., 0–200m), the system suggests adjusting the range to the achievable values.

Once the first priority is satisfied, the remaining routes are ranked based on the second priority. The route that best meets the second priority is selected as the final route and sent to the front end.

5.5 Front-End Display and Interaction

The optimized route, represented as a sequence of geographic coordinates, is transmitted to the front end and displayed on an interactive map using the **Leaflet library**. The map is initialized with OpenStreetMap as the base layer. The route is rendered as a polyline, while POIs are marked with clickable markers that display details like names and descriptions.

Users can adjust their preferences dynamically. Each update triggers a new request to the back end, which generates and returns a new route. Additional information, such as total distance, elevation gain, and estimated travel time, is displayed alongside the map to provide a comprehensive overview.

5.6 Caching and Performance Optimization

To reduce latency and enhance system performance, the processed graph is stored in a local **cache system**. This avoids the need to repeatedly generate the graph, significantly speeding up the route generation process. The caching mechanism ensures that the system can handle multiple user requests efficiently without compromising response times.

5.7 Alert Notification System

If no routes satisfy the first priority, the system triggers an **Alert Notification** that informs the user of the issue and suggests achievable constraints. For example, if no routes meet the specified POI minimum, the alert will provide the maximum achievable POI count within the current dataset.

This mechanism ensures that users receive clear feedback and can refine their input parameters to generate valid routes.

6 Evaluation and Results

The evaluation of this project demonstrates the effectiveness of the route generation algorithm compared to existing solutions, including Google Maps and popular runner apps like Strava and Runkeeper. Unlike these tools, our system provides customizable and tailored routes based on user preferences for elevation changes, POI inclusion, and specific distance constraints. This evaluation emphasizes the qualitative advantages of our approach, highlighting how it addresses gaps in current applications and provides a more personalized running experience.

6.1 Comparison with Google Maps

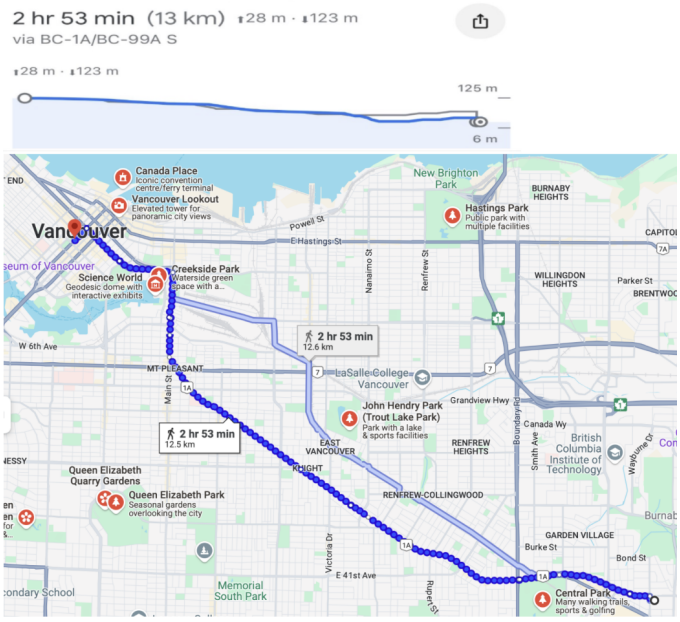
While Google Maps excels at generating the shortest or fastest routes for general navigation, it falls short in catering to the specific needs of runners. It does not allow users to specify preferences for elevation changes or include POIs along their route, which are crucial for creating a more personalized and engaging running experience. Additionally, Google Maps lacks the ability to generate circular routes—a feature often preferred by runners who want to start and finish at the same location. These limitations make Google Maps less suitable for runners seeking customized, scenic, or fitness-oriented routes. Our system addresses these limitations by enabling users to:

- Define specific elevation change preferences, such as elevation gain for a challenging run or for a flatter path.
- Allow users to customize the number of POIs in the route based on user-defined priorities, providing a more scenic and engaging running experience.
- Restrict the route length to within 15% of the user's input distance, ensuring the generated route aligns closely with their desired distance.
- Offer users the ability to customize the number of POIs included in their routes, tailoring routes to individual preferences.

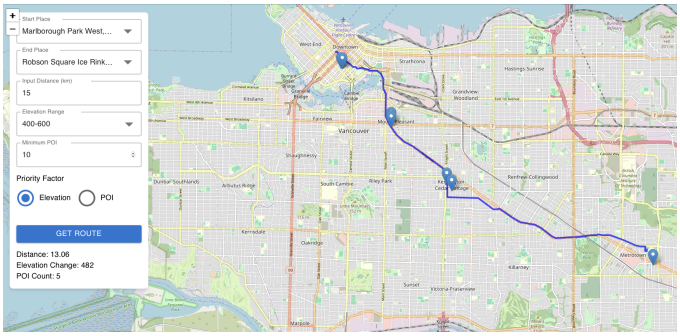
While the Google Maps route prioritizes efficiency, our system optimizes routes based on elevation and POI preferences, showcasing its ability to address runner-specific needs. In Figure 2, a route generated by Google Maps is compared with one created by our system.

Using the same starting and ending points, our system, Run Like a Local, outperforms Google Maps in aligning with user input for distance, elevation range, and POIs. For a user-defined distance of 15 km, Google Maps generated a shorter route of 12.5 km while Run Like a Local produced a route closer to the desired length at 13.06 km. In terms of elevation range, the user specified a range of 400–600 meters. Google Maps provided a significantly lower elevation range of 151 meters, whereas Run Like a Local achieved 482 meters, aligning closely with the user input. Lastly, our system incorporates 5 POIs, offering a more engaging and scenic route than Google Maps, which includes only 3.

This comparison highlights how Run Like a Local better aligns with the user-specified parameters, such as achieving a closer match to the requested distance, elevation range and including more POIs than Google Maps, delivering a more personalized and runner-focused experience.



(a) Route generated by Google Maps.



(b) Route generated by Run Like a Local.

Metric	User	GMaps	RLL	Deviation (%)
Dist (km)	15	12.5	13.06	G: -16.67, RLL: -12.93
Elev Range (m)	400-600	151	482	G: -60.25*, RLL: -3.00*
POIs	10	3	5	G: -70, RLL: -50

Based on midpoint of range (500m).

Figure 2: Comparison of Routes: Google Maps vs. Run Like a Local. The first image shows a route generated by Google Maps. The second image shows a route generated by Run Like a Local. The third image showcases a comparison of user input, Google Maps, and Run Like a Local outputs

6.2 Summary of Quantitative Comparison

The comparative analysis distinctly illustrates that *Run Like a Local* more effectively aligns with runner-specific preferences than Google Maps. Specifically, while Google Maps deviates significantly from user-specified distance, elevation, and POIs—falling short by 16.67% in distance and 70% in the number of POIs—*Run Like a Local* demonstrates a closer adherence to the user's requirements, with only a 12.93% deviation in distance and a 50% deviation in POIs. Most notably, in terms of elevation, *Run Like a Local* achieves an elevation range within 3% of the user's preferred settings, whereas Google Maps only reaches 30.2% of the desired elevation midpoint, underscoring a substantial difference in catering to the elevation preferences crucial for training effectiveness and user satisfaction. These findings underscore the tailored capabilities of *Run Like a Local* in meeting the nuanced needs of runners seeking customized routes, confirming its superiority over Google Maps for specific athletic and recreational applications.

6.3 Comparison with Strava

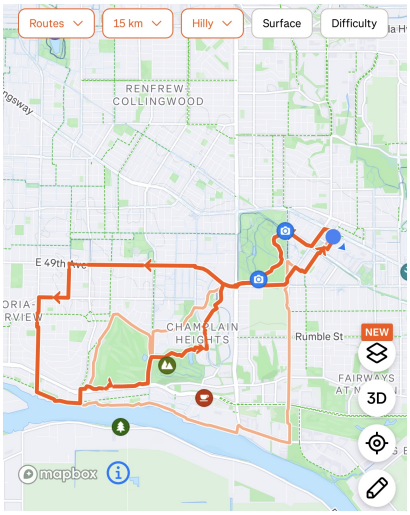
Strava, a popular runner app, offers limited customization options for route generation:

- **Default Circular Routes:** Strava primarily generates circular routes that start and end at the same location. This default setting may not meet the needs of users seeking to plan a one-way route between two custom points. Our system, on the other hand, allows users to specify distinct start and end points, providing greater flexibility in route planning.
- **Lack of POI Integration:** Strava does not incorporate user preferences for Points of Interest (POIs) when generating routes. Additionally, its maps do not provide detailed markers or information about POIs, which limits its ability to create engaging and informative routes. In contrast, our system integrates POIs directly into the route generation process, enabling users to prioritize scenic or cultural points of interest and providing detailed POI markers on the map.

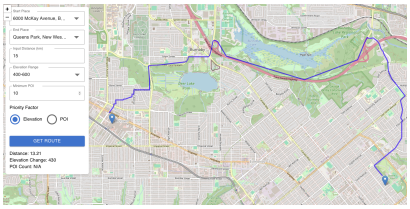
In Figure 3, this compares a circular route generated by Strava with a user-defined point-to-point route from our system, showcasing the additional flexibility and customization options offered by our approach. Strava's route generation is restricted to circular routes, where the starting and ending points are the same. While this may suit users looking for a traditional loop, it fails to address the needs of those who require more diverse routing options, such as point-to-point routes for one-way journeys, urban exploration, or connecting two specific locations for convenience or training purposes.

While we have the same ability to generate the same desired distance and elevation requirement provided by Strava's route, our system goes a step further by offering the capability to generate both circular and point-to-point routes while incorporating a user-specified number of POIs. This unique feature empowers users to design runs that align with their specific preferences and goals, making the experience more personalized and engaging. By enabling users to define distinct start and end points, our system overcomes the limitations of traditional circular-only route planning. This not only enhances versatility but also caters to a broader spectrum of user needs, positioning our system as a more user-centric solution for route planning. Furthermore, the ability to generate customized circular loops adds another layer of flexibility.

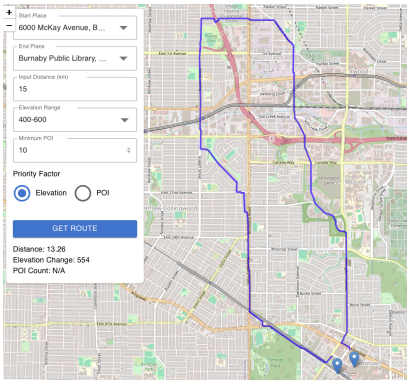
Overall, our system's ability to seamlessly adapt to various route types enhances its appeal to a broader audience. Whether a user prefers a scenic circular route for leisure or a direct point-to-point route for practicality, our approach ensures their needs are met. This adaptability demonstrates our system's commitment to providing a personalized and enriching experience for runners of all levels.



(a) Route generated by Strava.



(b) Point-to-Point route generated by Our System.



(c) Circular route generated by Our System.

Figure 3: Comparison of Routes: Strava vs. Our System. The first image shows a route generated by Strava, typically limited to circular paths. The second image shows a point-to-point route and the third image shows a circular route generated by Our System, allowing for customized start and end points with a tailored experience.

6.4 Qualitative Advantages of Our System

Our system is designed to address key limitations in current solutions by providing a fully customizable and user-centric approach to route generation. The following qualitative advantages highlight its effectiveness:

(1) **User-Defined Distance Constraints:** Unlike Google Maps, which typically searches for the shortest route without offering options to customize the route length, our system allows users to set specific distance constraints by ensuring that the generated route length falls within 15% of the user-specified input distance, providing users with precise control to align their routes with specific training or recreational goals.
(2) **Flexibility in Route Configuration:** While Strava primarily generates circular routes and Google Maps specializes in non-circular routes, our system supports both route types. Users can define specific start and end points, making it ideal for planning point-to-point runs or circular loops. This flexibility ensures a highly personalized running experience, accommodating diverse preferences such as training needs, sightseeing goals, or logistical constraints, providing runners with more creative and adaptable route planning options..
(3) **Elevation Change Customization:** Our system allows users to define an elevation range for their route, catering to preferences for flatter or more challenging terrain. This feature is unavailable in both Google Maps and existing runner apps.
(4) **POI Customization, Integration and Visualization:** By integrating POIs into the route generation process, our system not only provides detailed map markers for each POI but also allows users to customize the number of POIs included in their routes. This approach ensures routes that are more scenic, engaging, and tailored to individual preferences. Unlike current apps, which often lack this level of POI customization, our solution stands out by offering a uniquely enriched user experience that combines functionality with exploration and personalization.

6.5 Limitations

While the project successfully implemented a customizable route generation system tailored to runners' preferences, several limitations highlight areas for improvement:

(1) **Circular Routes with Close Start and End Points:** The current implementation supports generating circular routes; however, if the start and end points are too close to each other (e.g., less than 100 meters apart), the system fails to find a valid circular route and returns an error. This limitation arises from the difficulty of creating meaningful circular paths within such a constrained space. To address this, a minimum distance of 100 meters between the start and end points is currently required for circular routes.
(2) **Platform Limitation:** The application is currently developed as a web-based platform. While it is fully functional on desktop and mobile browsers, it does not have native applications for iOS or Android, restricting its accessibility and usability for users who prefer dedicated mobile apps.

(3) **Real-Time Data Integration:** The system does not integrate real-time data such as traffic, weather, or construction updates, which could influence the suitability of a running route. This limits the adaptability of the application in dynamic environments.
(4) **Static Geographic Coverage:** The current implementation is limited to the geographic data of specific regions (e.g., British Columbia). Managing a larger geographic area requires processing vast amounts of additional data, including maps, routes, and POIs. This increased data load could make the application less efficient, leading to slower response times and a less seamless user experience. Additionally, expanding geographic coverage is time-consuming and labor-intensive. It involves sourcing, integrating, and validating data for new regions, ensuring compatibility with existing systems, and maintaining data accuracy. These tasks demand substantial development time, which could detract from enhancing core features and optimizing the current system.
(5) **Algorithm Efficiency:** While the bi-directional A* algorithm is efficient for most use cases, its performance can degrade when handling extremely large datasets or highly complex user inputs. Further optimization of the algorithm is necessary to ensure scalability.

6.6 Future Work

To address the limitations and further enhance the system, several future improvements are proposed:

(1) **Improving Circular Route Generation:** Enhance the logic for generating circular routes, particularly for scenarios where the start and end points are close together. This could involve dynamically adjusting the pathfinding algorithm to create smaller loops or modifying the distance constraints to handle such cases more effectively.
(2) **Multi-Platform Development:** Develop native applications for iOS and Android to complement the web platform. These apps would provide a more seamless user experience by leveraging device-specific features such as GPS tracking and offline maps.
(3) **Real-Time Data Integration:** Incorporating real-time data sources such as traffic conditions, weather updates, and construction alerts would significantly enhance the system's functionality. By dynamically adjusting routes to reflect current conditions, users can enjoy a safer, more convenient, and seamless running experience.
(4) **Extending Geographic Coverage:** Automating the integration of OpenStreetMap data for new regions would enable the application to expand its geographic coverage beyond British Columbia with minimal manual intervention. Expanding the geographic coverage to include additional regions could significantly enhance the system's scalability and appeal. By broadening the geographic scope, the system could cater to a more diverse user base and accommodate runners in various locations worldwide, fostering greater adoption and usability.
(5) **Algorithm Optimization:** Explore alternative algorithms or enhancements to the bi-directional A* approach to improve performance on large datasets. Techniques such as heuristic tuning, parallel processing, or hierarchical pathfinding could significantly reduce computation times and enable real-time route adjustments.
(6) **Enhancing POI Features:** Expand the functionality of POI integration by allowing users to filter or prioritize specific types of POIs (e.g., parks, historical landmarks or tourist attractions). This would provide greater customization options and improve the overall user experience.

By addressing these limitations and implementing the proposed future work, the system can evolve into a comprehensive multi-platform application that provides highly personalized and adaptive running routes for users across diverse environments.

7 Conclusion

This paper has introduced a customizable route generation system designed specifically for runners, offering a personalized approach to planning routes in unfamiliar urban environments. Unlike existing tools such as Google Maps and runner apps like Strava, our system integrates user-defined preferences for distance, elevation, and points of interest, enabling the creation of routes tailored to individual fitness goals and environmental preferences.

The system generates candidate routes using a bi-directional A* algorithm, ensuring computational efficiency while adhering to user-specified distance constraints. A priority-based selection mechanism refines these candidates by strictly satisfying the user's first priority (elevation or POIs) and optimizing for the second. Leveraging real-world geographic data from OpenStreetMap (OSM) and elevation data from the PyHigh Package, the system supports both circular and point-to-point routes, enhancing flexibility and usability.

Our evaluation highlights the system's ability to address key limitations of existing solutions, including support for customizable start and end points, detailed POI integration, and elevation customization. These features collectively bridge significant gaps in current route planning tools, providing a more personalized and engaging experience for runners.

Looking ahead, future work will focus on addressing identified limitations, such as improving circular route generation for closely located start and end points, expanding geographic coverage, and integrating real-time data for dynamic route adjustments. Additionally, the development of native applications for iOS and Android will extend the system's accessibility and usability across platforms. By building upon the foundation established in this work, the system has the potential to become a comprehensive, multi-platform solution for personalized route generation, catering to the diverse needs of runners worldwide.

References

[1] [n. d.]. PyHigh: Elevation Data Package. https://ardupilot.org/. Accessed: 2024-11-01.
[2] ASICS Digital, Inc. 2024. Runkeeper: Track your runs, walks and more with your iPhone or Android phone. https://www.runkeeper.com. Accessed: 2024-10-04.
[3] M.A. Awadallah, S.N. Makhadmeh, M.A. Al-Betar, L.M. Dalbah, A. Al-Redhaei, S. Kouka, and O.S. Enshassi. 2024. Multi-objective Ant Colony Optimization:

Review. *Archives of Computational Methods in Engineering* (2024). https://doi.org/10.1007/s11831-024-10178-4

[4] Jin Bo. 2021. Multi-Objective A* Algorithm for the Multimodal Multi-Objective Path Planning Optimization. In *Proceedings of the IEEE Conference Publication*. https://doi.org/10.1109/9504943

[5] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. 2016. Past, Present, and Future of Simultaneous Localization And Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics* 32, 6 (2016), 1309–1332. https://doi.org/10.1109/TRO.2016.2624754

[6] OpenStreetMap Contributors. 2024. OpenStreetMap. https://www.openstreetmap.org Accessed: 2024-10-05.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[8] Rina Dechter and Judea Pearl. 1985. Generalized Best-First Search Strategies and the Optimality of A*. *J. ACM* 32, 3 (1985), 505–536. https://doi.org/10.1145/3828.3830

[9] A. Gionis, T. Lappas, K. Pelechrinis, and E. Terzi. 2014. Customized Tour Recommendations in Urban Areas. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining (WSDM)*. 313–322. https://doi.org/10.1145/2556195.2559893

[10] Tristram Gräbener. 2024. osm4routing: OSM Data for Routing. https://github.com/Tristramg/osm4routing Accessed: 2024-10-05.

[11] S. Halder, K.H. Lim, J. Chan, et al. 2022. Efficient itinerary recommendation via personalized POI selection and pruning. *Knowledge and Information Systems* 64 (2022), 963–993. https://doi.org/10.1007/s10115-021-01648-3

[12] Jianxin Li, Chengfei Liu, Rui Zhou, and Wenny Rahayu. 2015. The World of Big Data Skyline: A Survey. In *IEEE Transactions on Knowledge and Data Engineering*. https://doi.org/10.1109/TKDE.2015.2509975

[13] Ira Pohl. 1971. An Efficient Bidirectional Search Algorithm. https://doi.org/10.1145/321662.321674 Accessed: 2024-11-01.

[14] Strava Inc. 2024. Strava: The 1 app for runners and cyclists. https://www.strava.com. Accessed: 2024-10-04.

[15] Ma Xunchi. 2024. A Summary of the Routing Algorithm and Their Optimization, Performance. *arXiv preprint arXiv:2402.15749* (2024). https://doi.org/10.48550/arXiv.2402.15749

[16] H. Yoon, Y. Zheng, X. Xie, and W. Woo. 2012. Social Itinerary Recommendation from User-Generated Digital Trails. *Personal and Ubiquitous Computing* 16, 5 (2012), 469–484. https://doi.org/10.1007/s00779-011-0404-z