Comparative Metaheuristic Approaches to Tourist Itinerary Optimization in Low-Infrastructure Urban Contexts: A Case Study of Tirana

Alketa Hyso[*], Dezdemona Gjylapi
Department of Computer Science, Faculty of Technical and Natural Sciences, University "Ismail Qemali", Albania
E-mail: alketa.hyso@univlora.edu.al, dezdemona.gjylapi@univlora.edu.al
[*]Corresponding author

Keywords: tourist itinerary planning, route optimization, Tirana, genetic algorithm, simulated annealing, brute force, hybrid Greedy–SA algorithm

Received: September 29, 2025

Tourist itinerary planning is a central component of smart tourism, yet it remains challenging in developing cities where computational resources and digital infrastructure are limited. This study examines Tirana, Albania, as a representative case for urban pedestrian-based tourist itinerary optimization. The analysis is carried out using both exact and heuristic optimization techniques, including Brute Force, Genetic Algorithm (GA), Simulated Annealing (SA), and a Hybrid Greedy + SA approach that integrates deterministic initialization with stochastic refinement. Distances between attractions were derived from OpenStreetMap, enabling fully reproducible experiments conducted on multiple datasets representing different sets of attractions with increasing size and under varying conditions, including ideal, noisy, and incomplete information. The results show that while exact computation rapidly becomes impractical as the instance size grows, metaheuristic methods, particularly SA and the hybrid variant consistently deliver high-quality and stable solutions. To evaluate real-world applicability under digital and computational constraints, the hybrid algorithm was implemented as a mobile-ready Progressive Web App and executed entirely on a resource-constrained device, demonstrating near-instantaneous optimization and confirming its feasibility for fully on-device use without reliance on backend servers. Overall, the study shows that lightweight metaheuristics, especially the Hybrid Greedy + SA method, offer a robust, scalable, and mobile-ready approach to urban tourism itinerary planning, suitable for deployment in environments with limited computational and infrastructural resources.

Povzetek: Študija pokaže, da so lahki hevristični algoritmi primerni za hitro načrtovanje turističnih poti tudi tam, kjer so računalniški viri in digitalna infrastruktura omejeni.

1 Introduction

Urban tourism functions as a major driver for city development because visitors seek to experience as much as possible during their brief urban visits. Tourism depends on digital services to let customers reserve accommodations and arrange their travel schedules [1], and Industry 4.0 technologies (AI, IoT, blockchain) create better visitor experiences [2]. However, this digital transformation has not been evenly distributed. In Tirana, the capital of Albania and one of the country's most rapidly developing tourist destinations, the growing number of visitors faces persistent difficulties due to the city's limited digital infrastructure. The city lacks the technological resources needed to support visitors in organizing and optimizing their travel plans [3]. The country of Albania does not have a unified national tourism portal which provides standardized information and booking capabilities and customized travel recommendations. The current promotional apps "Visit Tirana" [4], "Albania Tourist Guide" [5] fail to provide optimized routes and personalized itineraries and direct access to actual transportation services and do not have an e-ticketing system for public attractions.

The majority of visitors must rely on static maps and general platforms because of this gap which produces suboptimal routes and inconsistent travel experiences. The solution needs to use light-weight computational methods that operate within local boundaries and work within municipal or regional systems.

The research investigates three main questions about the trade-offs between solution quality and execution time when using exact and metaheuristic methods for day-trip itinerary planning. (Q1) How do exact and metaheuristic approaches trade off solution quality versus execution time for day-trip itineraries? Q2) How does performance scale with the instance size - the number of attractions (N)? Q3) Can lightweight metaheuristics be effectively implemented on resource-constrained mobile devices, enabling practical itinerary optimization without backend infrastructure?

We cast one-day tourist itinerary planning as a Travelling Salesperson Problem (TSP) and compare an exhaustive search Brute Force (BF) baseline with two metaheuristics—Genetic Algorithm (GA) and Simulated Annealing (SA) implemented in Python and evaluated on real inter-attraction distances for Tirana derived from OpenStreetMap. We report tour length (m), runtime (s), and success rate across multiple values of N, in Section 3. The success rate counts only runs whose tour length exactly matches the best-known solution.

This study makes three principal contributions: (i) a deployable evaluation framework for optimizing urban tourist routes using open map data in resource-constrained environments; (ii) realistic benchmarks that quantify the speed–accuracy trade-off on standard hardware; and (iii) an implementation pathway that enables cities with limited digital infrastructure to adopt the solution. The remainder of the paper is organized as follows: Section 2 reviews related work; Section 3 describes the the experimental setup, and the optimization methods, including BF, GA, SA, and the proposed Hybrid Greedy + SA. It also details the robustness experiments under noisy and incomplete distance data, the validation of straight-line distances against OpenRouteService walking routes, and the mobile execution experiment on mobile hardware. Section 4 presents the empirical results across different problem sizes, robustness scenarios, and routing models, as well as the comparative performance of the hybrid approach on clean and perturbed data. Section 5 discusses the main design trade-offs between Haversine and network-based routing, generalization to larger instances, and the implications for smart-tourism applications in low-infrastructure cities. Finally, the concluding section summarizes the key findings, highlights the practical relevance of lightweight metaheuristics for mobile itinerary planning.

2 Related work

The field of urban itinerary optimization exists in two main forms which include the Tourist Trip Design Problem (TTDP) that handles preferences and time and budget constraints and opening hours [6] or as adaptations of the Travelling Salesperson Problem (TSP) for tourism applications, which focus primarily on shortest paths. The main distinction between TTDP variants and TSP-style research emerges from their distinct methods for evaluating user satisfaction under various constraints because TTDP variants handle multiple restrictions yet TSP-style studies focus on efficient solutions with basic assumptions.

Beyond this distinction, recent work has examined several directions. Sylejmani et al. (2024) use Iterated Local Search to personalize tours under thematic (e.g. visiting a museum before a restaurant and cultural sites before natural sites), temporal, and financial constraints [6], while Zhang et al. (2008) incorporate perceptual dimensions by blending path optimization with visual experience, mining web data to approximate the "visibility" or scenic value of attractions [7]. Adamo et al. (2022) explore multimodality, combining walking and driving routes to create more realistic itineraries [8]. Tang et al. (2024 demonstrates how large language models (LLMs) can transform unstructured user requests into optimized travel plans [9]. Souffriau et al. (2008) further show that guided local search can efficiently solve orienteering-based TTDPs directly on mobile devices, highlighting the viability of lightweight metaheuristics for on-device itinerary generation [10]. While exact methods offer optimality guarantees, they are often limited by scalability. Androutsopoulos & Zografos (2008) address multimodal itinerary planning with strict sequencing and time window constraints using an exact dynamic programming approach based on problem decomposition [11]. In contrast, the integration of hybrid algorithms with metaheuristics has proven effective in addressing complex optimization problems. Mangini et al. (2021) use graph theory, Integer Linear Programming (ILP), and a multi-algorithm strategy to generate one-day tourist routes that minimize travel time while reflecting user preferences [12]. Several studies show that combining SA with A* can generate fast and efficient tours [13]; Li et al. (2022) apply a knowledge-based hybrid Ant Colony Optimization algorithm enhanced with bacterial foraging to address group-based tourist satisfaction under capacity and preference constraints, with a focus on efficient tourist route planning [14]; GAs achieve improved results with faster convergence in tourist route optimization, while K-means clustering followed by GA optimization supports the creation of personalized tourist sequences [15]. Context-aware TTDP applies fuzzy-logic-enhanced metaheuristics to model contextual constraints [16]; fuzzy systems further support uncertainty management in attraction planning, resource evaluation, and decision-making [17, 18].

To synthesize these trends, we provide a structured comparison, Table 1, showing how prior work varies across problem formulations, algorithmic strategies, constraints, and data scales. While prior studies often assume rich mobility datasets, multimodal transport layers, or server-side computational capabilities, they collectively establish that metaheuristics are well-suited for real-time itinerary planning.

The present study extends prior work by addressing a gap not sufficiently explored: itinerary optimization in resource-constrained environments with limited digital infrastructure. In contrast to models that depend on transport schedules, multimodal networks, or complex personalization layers, our approach uses only geometric distances from OpenStreetMap, ensuring a lightweight and reproducible formulation. We adopt a deployment-oriented perspective by evaluating three solvers (Brute Force, GA, SA) across multiple data regimes and by implementing a Hybrid Greedy + SA solver as a mobile-ready Progressive Web App. Computing optimized itineraries entirely on-device, without backend support and with near-instantaneous runtime, highlights the practicality and scalability of the proposed approach for cities with constrained computational resources, limited connectivity, or incomplete digital maps.

Table 1: Summary of representative tourist itinerary planning approaches

Ref	Problem / Model	Main Algorithm	Constraints	Scale / Data
[10]	TTDP modelled as orienteering problem (OP) for mobile guides	Guided local search metaheuristic	Time budget, POI scores, personalization on mobile	Real data, city of Ghent
[6]	Extended TTDP as multi-constraint team orienteering problem with time windows (MCTOPTW) with patterns.	Local search / Iterated local search (ILS)-based metaheuristics	Time, budget, multi-knapsack; POI category patterns	146 instances, city-scale test set
[8]	Multi-modal TTDP (road + pedestrian)	Ad-hoc ILS	Time windows, visit duration, multimodal car+walking	Up to 3 643 POIs, 7-day horizon
[9]	Open-domain urban itineraries (OUIP)	LLM + cluster-aware spatial optimization	Natural-language requests; dynamic POIs; citywalk	4 Chinese cities, 1200+ itineraries
[16]	Context-aware TTDP	MGA + fuzzy logic a-posteriori evaluation	Contextual constraints (e.g. accessibility, safety).	Case study, Granada (mobility-impaired)
[13]	TTDP with GIS-grounded multimodal routing	Hybrid SA + A* (metaheuristic + heuristic)	GIS/OSM paths, POI distances, real pedestrian network, personalization	Fez Medina, 15 POIs, 9400 paths
[14]	Tourism route planning (TRP)	Knowledge-based Hybrid Ant Colony + Bacterial Foraging	Tourist satisfaction model, clustering, capacity limits	1000 tourists; 300-1300 paths
[11]	Multi-criteria time-dependent itinerary planning with mandatory intermediate stops in a multimodal fixed-schedule network	Dynamic Programming after problem decomposition into elementary sub-problems	Ordered multimodal routing with strict time windows at origin, destination, and intermediate stops.	Test network with 900 nodes, 930 service links, 12,000 interchange links, 100 services;
[12]	One-day TTDP-style round trip with outward/return itineraries on an urban PoI graph	Multi-level heuristic combining ILP-based symmetric TSP with graph-based PoI exchange/add/delete procedures	Single-day itinerary planning with PoI priorities, visit durations, mode-dependent travel times (walk/transit), and interactive user customization.	Case study: Bari (Italy) PoI network, realistic travel-time matrix; cruise-tourist one-day scenario
[7]	Tourist route planning with scenic-visibility scoring	Web-based POI extraction + GIS routing + 3D visibility computation	Personalized POI selection; scenic-visibility optimization; road-network routing	Prototype system; Japan (multiple scenic sites; DEM 50m grid)
[15]	Tourism path recommendation from survey & social-media objectives (cluster + TSP)	GA-enhanced k-means for clustering + GA for optimal tour	Multi-objective (12 internal & external tourism criteria);	Survey (600 visitors) + TripAdvisor data for 6 POIs in Port Sudan, Red Sea State (Sudan)

3 Methods and materials

3.1 Experimental setup

All algorithms were implemented in Python and executed under the same computational environment to ensure a fair comparison. Experiments were conducted on a Windows 10 Pro (64-bit) system equipped with an Intel Core i7-7500U 2.70 GHz processor and 12 GB of RAM. All runs were performed in Python 3.12.2 using JupyterLab 4.1.2 as the development interface. Core libraries and package versions are listed in the project's requirements.txt, and the complete code and environment specifications are available in the project's GitHub repository [17].

Each method — including GA, SA, and the Hybrid Greedy + SA variant — was independently executed 30 times. To ensure reproducibility, each run used a deterministic seed. This repeated-run protocol guarantees statistical robustness, allowing the computation mean for all main performance metrics: tour distance, execution time, solution gap, and success rate. The reported values in all tables therefore represent the average performance across these 30 independent executions per scenario (clean, noisy, or incomplete). Brute Force was attempted in all cases, but when the 90-second limit was exceeded, its result was not included. In such cases, the best solution among GA and SA was adopted as the baseline.

Solution Gap quantifies the relative deviation of an algorithm's tour distance from the best-known solution. For each algorithm A, the Gap_mean was computed as:

$$Gap_{mean} = \frac{Distance_{mean} - Best\_known}{Best\_known} * 100$$

where *Distance_mean* is the average tour distance obtained by algorithm A over 30 runs, and *Best_known* represents the best-known tour length for the same scenario. When BF method successfully returned the optimal solution (for small N), its result was used as *Best_known*. For larger instances where BF exceeded the 90-second time limit, *Best_known* was set to the shortest tour obtained among GA, SA, or Hybrid Greedy + SA. Thus, the *Gap_mean* values allow a normalized comparison of solution quality across algorithms and scenarios, regardless of the absolute tour length.

To measure how often each algorithm reaches the best-known solution, a success rate metric was used. A run is considered successful if the route distance it produces is within a small tolerance ε of the *Best_known* distance. In this study, ε was set to 0.0 meters, meaning that only runs matching the optimal distance exactly are counted as successful. This strict criterion ensures fair and reproducible comparison between algorithms.

For GA, parameter values were selected based on a preliminary sensitivity analysis on the dataset (N = 31). Population size was varied across 30, 50, and 100 individuals while keeping the number of generations 50 and mutation rate 0.02 fixed. The results of this analysis, presented in Section 3.3.3 and Table 2, showed that a population of 50 provides a balanced compromise between runtime efficiency and search quality, with negligible performance differences for larger populations. Consequently, all subsequent GA experiments were performed using *population* = 50, *generations* = 50, and *mutation_rate* = 0.02 to ensure comparability across algorithms.

3.2 Study context and data

This study adopts Tirana, Albania, as a case study, a developing capital experiencing growing tourist flows but lacking adequate digital systems for tourism management. A selection of cultural sites and historical landmarks and recreational activities was made to create authentic travel plans. The geographic coordinates of these attractions were extracted from OpenStreetMap (OSM). Distances between every pair of locations were computed using the Haversine formula, which calculates great-circle distances from latitude and longitude. The values were merged into a complete N×N distance matrix that used meters to measure distances and served as the foundation for all optimization techniques. The tour cost was defined as the total distance of a closed itinerary visiting each attraction exactly once

3.3 Optimization algorithms

3.3.1 Brute force

The BF method was used as an exact baseline. It generates all possible permutations of attractions, starting and ending at the same location, computes the total distance of each tour, and returns the shortest one. The search tree grows to its complete depth of N−1 levels where N shows the total number of attractions. The time complexity is factorial, O(N!). The method becomes impractical for use with large instances because of this. In our implementation, we imposed a 90-second time limit; if exceeded, the solver was skipped and excluded from evaluation. When successful, BF serves as an exact optimal baseline for comparison with heuristic methods.

3.3.2 Genetic algorithm

The GA operates as a population-based metaheuristic which draws its inspiration from natural evolutionary processes. Each candidate solution (individual) was represented as a permutation of attractions. The algorithm followed this sequence of operations:

Population initialization - Generate an initial population of random tours. Each individual in the population represents one complete itinerary; a permutation of all tourist points, ensuring that every attraction appears exactly once within a tour. Population initialization is performed by randomly generating pop_size permutations.

Fitness evaluation - Compute the total distance of each tour, where shorter routes indicate higher fitness. In principle, an individual becomes infeasible if any pair of consecutive points in its route is not connected by a valid edge. In our formulation, such missing connections are assigned an infinite, or equivalently, prohibitively large cost. Consequently, these individuals obtain extremely poor fitness scores and are automatically removed during the selection phase, without requiring an explicit feasibility check.

Selection - Choose the fittest individuals (those with the shortest tours) to form the mating pool. Selection is applied to retain only the fittest half of the individuals; those with the lowest total distance. This eliminates poorly performing solutions, including those containing infeasible edges.

Crossover - Create new offspring by combining segments of parent tours while ensuring that each attraction is visited exactly once. We employ Order Crossover (OX), a widely used method for permutation-based optimization:

1. Two cut points (a, b) are chosen at random.
2. The segment between a and b is copied directly from parent p1 to the child.
3. The remaining positions are filled with the elements of parent p2 in order, skipping elements already present in the copied segment.

This operator preserves relative ordering from both parents and prevents duplication of nodes, producing a feasible offspring tour.

Mutation - Introduce diversity by occasionally swapping two positions in a tour, reducing the risk of premature convergence. In our implementation, mutation swaps two randomly chosen positions in the tour with a small probability (rate). This operator creates slight perturbations in routes, helping the search explore new regions of the solution space and avoid stagnation in local minima.

Replacement - Substitute the parent population with the newly generated offspring.

Termination - Repeat the process for a fixed number of generations while tracking the best-so-far distance.

3.3.3 Sensitivity of GA parameter setting

To verify the robustness of the selected GA parameters, a small sensitivity analysis was conducted on the clean dataset (N = 31). The population size was varied across three settings: 30, 50, and 100, while keeping the number of generations 50 and mutation rate 0.02, fixed. As shown in Table 2, the mean tour distance and gap exhibit only modest variations (Gap_mean ranging from 62.8% to 70.0%), while the success rate remains constant at 0%. Increasing the population size leads to slightly longer runtimes (from 0.08 s to 0.26 s) but does not substantially improve solution quality.

Table 2: Sensitivity of GA performance to population size (N = 31, clean data, 30 runs)

GA	generations=50, mutation_rate=0.02			
Pop_size	Dist_mean	T_mean	Succ	Gap_mean
30	10592.014	0.083	0	69.995
50	10143.115	0.141	0	62.791
100	10258.633	0.263	0	64.645

Therefore, a population size of 50 was selected for all subsequent experiments, as it provides a balanced trade-off between runtime efficiency and search quality, ensuring fair comparison with the SA and Hybrid algorithms.

3.3.4 Simulated annealing

SA was implemented as a single-solution metaheuristic with stochastic acceptance of worse solutions. The steps of the algorithm are as follows:

Random Initialization - The algorithm begins from a randomly generated tour obtained by shuffling all points, which serves as the initial feasible solution.

Distance Evaluation - The optimization requires repeated tour-length evaluations based on pairwise distances between locations. To obtain accurate geographic measurements, all distances are computed using the Haversine formula, which estimates great-circle distances on the Earth's surface. For a given permutation of points, the total distance is evaluated as:

$$D(\pi) = \sum_{i=0}^{n-1} d_{\pi_i,\pi(i+1) mod\, n}$$

The tour length D(π) is computed as the sum of the distances between every pair of consecutive locations in the tour. In this formulation, π represents a permutation of all points, and π_i denotes the point visited at position i in the tour. The term $d_{\pi_i,\pi(i+1) mod\, n}$ corresponds to the distance between the current point π_i and the next point in the sequence. The modulo operator ensures that when the index reaches the last point, the route closes by connecting back to the first point, forming a complete loop.

Neighborhood Exploration - During the search, the algorithm iteratively explores neighboring solutions using a 2-opt segment reversal operator. The operator works by selecting two random indices i and j along the current tour and reversing the entire subsequence between them. This creates a new tour in which a segment of the route is traversed in the opposite direction. The intuition behind 2-opt is that many inefficient tours contain crossing edges, and reversing the segment between two points often eliminates these crossings, leading to a shorter overall path. Because of its ability to systematically remove such geometric inefficiencies, 2-opt is widely adopted in local search and metaheuristic optimization methods to produce substantial improvements in route quality with minimal computational overhead.

Acceptance Mechanism - At each iteration, the candidate solution is evaluated by comparing its tour length to that of the current solution. If the new solution yields a shorter tour, that is if: $D_{new} < D_{current}$, then it is accepted deterministically. Otherwise, the new solution may still be accepted with a probability defined by the classical Metropolis criterion:

$$P_{(accept)} = e^{\frac{D_{current} - D_{new}}{T}}$$

This probabilistic acceptance allows the algorithm to escape local optima by admitting occasional uphill moves.

Temperature Schedule - The temperature was initialized at a high value (10,000), enabling broad exploration in early iterations. A cooling factor 0.995 is applied at each step.

$$T_{k+1} = 0.995 * T_k$$

As the temperature decreases, the probability of accepting inferior solutions diminishes, encouraging convergence toward an optimal or near-optimal tour.

Convergence Tracking - The best distance found so far is logged periodically throughout the search, producing a convergence history curve. This allows for a detailed comparison of SA's performance relative GA.

For the standalone SA implementation, the following parameters were used:

initial temperature T0=10000
cooling rate α=0.995
stopping condition T>1
neighborhood: 2-opt segment reversal
acceptance rule: Metropolis criterion
logging interval: 100 iterations

To account for stochastic variability, 30 independent runs were performed using run-level random seeds:

seed=4242+r with both *random* and *numpy.random* initialized per run to ensure full reproducibility.

3.3.5 Additional experiments: robustness under noisy and incomplete data

To further evaluate the robustness of the proposed metaheuristics (GA and SA), we conducted a set of supplementary experiments where the distance data were intentionally perturbed to simulate real-world imperfections.

These imperfections represent measurement noise (GPS inaccuracies), and incomplete connectivity (unavailable or missing links between points of interest).

(a) Perturbation model

Starting from the original distance matrix D, a perturbed version D′ was generated using the function:

$$D'_{ij} = D_{ij} * \left(1 + \epsilon_{i,j}\right), \qquad \epsilon_{i,j} \in [-\rho, +\rho]$$

where $\rho \in \{0.1, 0.2\}$ denotes the amplitude of the multiplicative noise (±10–20%). Additionally, a random fraction $r \in \{0.1, 0.2\}$ of the entries were replaced with +∞ indicating missing edges (no direct connection between those nodes). This setup generated five test scenarios:

- noisy10 (±10% noise)
- noisy20 (±20% noise)
- miss10 (10% missing edges)
- miss20 (20% missing edges)
- noisy15_miss15 (±15% noise, 15% missing edges combined)

Each perturbed matrix was stored for reproducibility and used as input for both GA and SA with identical hyperparameters.

(b) On-the-fly repair strategy

In the presence of missing edges, direct distance evaluation would normally fail. To overcome this, we introduced an on-the-fly repair algorithm, a simple yet effective mechanism that restores connectivity dynamically during route evaluation. For any pair of consecutive nodes (u, v) where$D'[u, v] = \infty$, the repair procedure searches for an intermediate node k that minimizes the detour cost:

$D_{repair}(u, v) = min(D'[u, k] + D'[k, v])$ where $k \neq u, v$

If such a node exists, the route temporarily diverts through k; otherwise, a large penalty value is assigned, so that the optimizer avoids this segment.

This mechanism ensures that the optimization process remains feasible and continuous even in degraded or incomplete graphs, emulating real-time adaptive behavior.

3.3.6 Hybrid Greedy + SA

Following the analysis of the standard GA and SA heuristics, we introduced a third variant — Hybrid Greedy + SA (Algorithm 3), to enhance anytime performance and robustness under incomplete or noisy distance data.

The hybrid algorithm begins with a Greedy initialization, which quickly constructs a valid tour by iteratively selecting the nearest unvisited node. This step provides a feasible and near-optimal starting solution. After obtaining this initial tour, the algorithm enters the Simulated Annealing refinement phase, where it applies 2-opt neighborhood moves to progressively improve the tour. At each iteration, a new candidate route is generated by reversing a random segment, and the change is accepted either if it improves the total distance or probabilistically if it is slightly worse, to avoid premature convergence. This two-stage design combines the speed and feasibility of Greedy construction with the exploratory power of Simulated Annealing, resulting in a fast yet robust method capable of maintaining good solution quality under noisy or incomplete data.

Because the Hybrid algorithm starts from a high-quality Greedy tour, it does not require the high initial temperature (T_0 = 10000) and slow cooling (α = 0.995) used in standalone SA to escape poor initial states. This allows the Hybrid configuration to allocate more computation to focused improvement rather than broad exploration, accelerating convergence without sacrificing robustness. The Hybrid Greedy + SA algorithm was executed with the following parameters:

- initial temperature T0=5000
- cooling rate α=0.996
- stopping temperature T>1
- neighborhood: 2-opt segment reversal
- acceptance rule: Metropolis criterion
- logging interval: 100 iterations
- 30 independent runs with different random seeds
- 2-hop repair and large penalty (1e12) for missing edges

For perturbed data, 10–20% noise and missing edges were introduced, and the Hybrid Greedy + SA used 2-hop repair with a large penalty (1e12), while keeping the same annealing parameters (T0=5000, α=0.996, 30 runs).

The experimental results in Section 4.5 directly assess its effectiveness under both clean and perturbed data conditions. The two-stage design—Greedy initialization followed by SA refinement—was specifically intended to accelerate convergence and enhance robustness when distance information is incomplete or noisy. The results presented in the section 4.5, validate this expectation: on clean data, the hybrid approach substantially reduces the optimality gap while maintaining competitive runtimes, and under noisy or incomplete distance matrices it preserves solution quality and feasibility more effectively than standalone GA or SA. Thus, the performance analysis confirms the practical benefits of the proposed hybrid strategy anticipated in the methodology, demonstrating its ability to deliver reliable and efficient itineraries across varying data scenarios.

3.4 Realistic distance validation using openrouteservice

To evaluate how geometric distance approximations differ from real-world travel distances within the urban environment of Tirana, we conducted an additional experiment that integrates route optimization with real street-network routing.

Experimental Setup

The analysis was performed using:

-Python 3.10+, OpenStreetMap (OSM) as the source of geographic coordinates for 31 tourist attractions in Tirana.

-The Hybrid Greedy + SA metaheuristic implemented in Python to obtain an optimized visiting order based on Haversine (straight-line) distances.

-OpenRouteService (ORS) API, which computes real walking distances using the actual street network derived from OSM.

The workflow consisted of three major steps:

-Build a full Haversine distance matrix for all 31 points using their OSM lat/long coordinates.

-Run the Hybrid Greedy + SA algorithm.

-Evaluate the optimized tour using OpenRouteService to compute true, network-based walking distances between consecutive attractions.

3.4.1 Visualization-based structural analysis of haversine and ORS routing models

In addition to the primary optimization framework based on Haversine great-circle distances, we conducted a complementary routing experiment aimed at assessing how the choice of distance metric influences the structure of the optimized tour. The same set of 31 tourist locations was evaluated using two alternative distance models: (i) geometric Haversine distances, which provide a fast, API-independent approximation of spatial proximity, and (ii) real walking distances obtained through the OpenRouteService (ORS) Matrix API. For the second model, the Hybrid Greedy + SA algorithm was executed directly on ORS-derived network distances to obtain a fully realistic, walkability-constrained shortest tour. This dual-routing setup enables a controlled methodological comparison between geometric and network-based optimization, ensuring that differences observed in the results can be attributed solely to the underlying distance model. The inclusion of both models strengthens the reliability of the experimental design and provides a foundation for understanding the impact of routing realism on itinerary planning.

3.5 Mobile execution and deployability evaluation

Since tourist route optimization is expected to run directly on users' smartphones during real-world use, testing solely on desktop hardware does not reflect practical performance. The mobile execution experiment was therefore included to measure responsiveness under realistic computational constraints.

To evaluate the real-world performance of the Hybrid Greedy + SA algorithm, we conducted an additional execution-time assessment on a mobile device. A lightweight web application (HTML/JavaScript/Leaflet) containing the full algorithm was deployed on GitHub Pages and installed on an iPhone as a Progressive Web App (PWA) using Safari's "Add to Home Screen" feature. When launched, the application runs in standalone mode and executes all computations locally on the device's JavaScript engine.

To guarantee cross-platform reproducibility, we used the same deterministic seed schedule in both desktop (Python) and mobile (JavaScript) implementations. Specifically, each run was initialized with seed = 2025 + r, where r is the run index. This ensures that the stochastic trajectory of SA remains consistent across environments, enabling a fair comparison of results. Tests were performed on Samsung Galaxy A10 running Android 11 using the GitHub Pages web-app. Execution time was measured using *performance.now()*, which provides high-resolution, device-level timing.

3.6 Output and visualization

All results were recorded at both run and summary levels. The experimental framework saved distance and runtime data along with success indicators in CSV format at the run level. The summary statistics displayed the average values and standard deviations for all recorded metrics. The analysis included two types of visualizations which showed distance and runtime distributions through boxplots and displayed GA and SA convergence patterns through median and interquartile range curves. All visual content was generated at 600 dpi resolution to ensure high image quality. In addition, two map visualization was developed to demonstrate how the generated itineraries would perform in the actual geography of Tirana.

4 Results and fiscussion

4.1 Performance on small instance (N=7)

Table 3 shows the results of BF and GA and SA when the number of attractions is seven. Reported metrics include mean and standard deviation of tour distance and runtime, success rate (percentage of runs achieving the best-known solution).

Table 3: Comparative results of BF, GA, and SA for N=7 attractions.

	Dist_mean	Dist_std	T_mean	T_std	Succ
BF	3886.4775	0	0.0069	0	100
GA	3886.4775	1.39e-12	0.0890	0.0220	100
SA	3886.6375	0.876156	0.0373	0.0180	96.66

Taken together, the results demonstrate that for small problem sizes (N=7), all three methods reach or approximate the optimal solution, but their runtime characteristics differ. BF is the fastest in absolute terms for this small instance. GA guarantees optimality but incurs higher runtime overhead due to its population-based operations. SA trades a small loss in accuracy for significantly faster and more consistent runtimes then GA. Figures 1 and 2 (N = 7) show that both GA and SA consistently attain the BF-optimal tour length: the distance distributions are essentially indistinguishable, with identical medians (orange lines) and nearly identical means (triangles). In contrast, the runtime boxplots reveal a clear separation: SA runs faster on average and exhibits lower variability, while GA shows higher median/mean times.

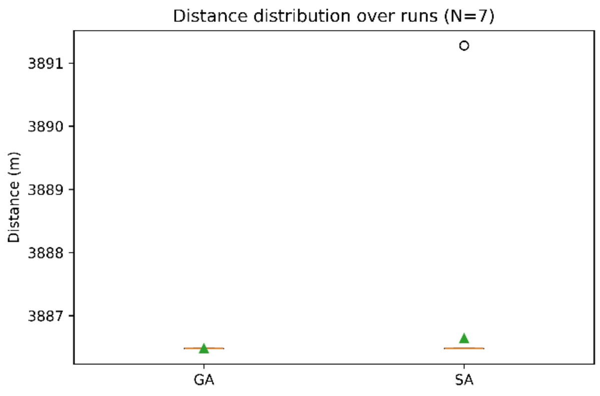


Figure 1: Performance evaluation for N = 7, showing the distance distribution across 30 independent runs of GA and SA. Means (triangles) and medians (orange lines).

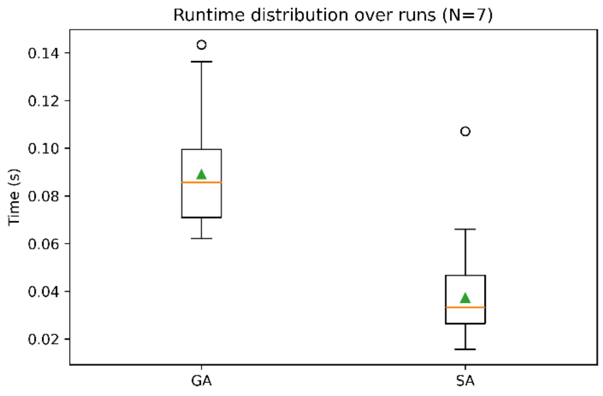


Figure 2: Performance evaluation for N = 7, showing the runtime distribution across 30 independent runs of GA and SA. Means (triangles) and medians (orange lines).

4.2 Results on the 10-attraction instance

Table 4 compares the performance of BF, GA, and SA for itineraries with ten attractions. Reported metrics include mean and standard deviation of tour distance and runtime, success rate (percentage of runs achieving the best-known solution).

Table 4: Comparative results of BF, GA, and SA for N=10 attractions.

	Dist_mean	Dist_std	T_mean	T_std	Succ
BF	4379.82	0	4.2005	0	100
GA	4385.80	7.070	0.1063	0.0177	43.3
SA	4383.69	5.231	0.0625	0.0108	43.3

For N=10, BF produced the exact optimum in 4.20 s, which remains tractable at this scale but already illustrates the rapid increase in computational cost. Both GA and SA found near-optimal tours, yet their success rates dropped to 43.3%. Between the heuristics, SA achieved slightly shorter tours on average and faster runtimes, offering a more favorable balance of accuracy and efficiency than GA.

To assess whether these differences are statistically significant, we complemented the descriptive results with bootstrap confidence intervals and Mann–Whitney U tests, as reported in Table 5.

Table 5: Statistical comparison of GA and SA using bootstrap CIs (95%) and Mann–Whitney U tests (30 runs).

	Alg.	Mean	95% Bootstrap CI	U	p-value
D	GA	4385.80	[4383.43, 4388.38]	536.5	1.953e-01
	SA	4383.69	[4381.99, 4385.69]		
T	GA	0.1202	[0.1001, 0.1126]	891.0	7.39e-11
	SA	0.0614	[0.0587, 0.0663]		

For tour distance (D), GA and SA have very similar mean values, and their 95% bootstrap confidence intervals overlap extensively. The Mann–Whitney test yields U = 536.5, p = 1.953e-01, indicating no statistically significant difference in tour length between the two algorithms at this scale.

For runtime (T), the difference is substantial. GA is nearly twice as slow as SA, and the confidence intervals do not overlap. The Mann–Whitney test gives U = 891.0, p = 7.39e-11, showing a highly significant advantage of SA in terms of execution time.

Overall, the results show that both algorithms achieve comparable tour lengths, but SA is far faster and more computationally efficient.

4.3 Large-scale performance (N=31)

Under the latency constraint (≤90 s), Brute Force cannot be executed for N = 31 and is therefore excluded from the comparison. Tables 6 and 7 summarize the empirical and statistical results obtained from 30 independent runs of GA and SA.

Table 6: Empirical performance of GA and SA for N = 31 attractions (30 runs). BF timed out and is omitted

	Dist_mean	Dist_std	T_mean	T_std	Succ
GA	10143.11	915.95	0.12	0.0138	0
SA	6782.44	461.25	0.06	0.0082	3.33

The statistical analysis in Table 6, confirms that SA strongly outperforms GA at N = 31 attractions spots. SA produces significantly shorter tours (mean = 6782.44 m) compared to GA (mean = 10143.11 m) and exhibits much lower variability (SD = 461.25 m vs 915.95 m). Runtime performance is similarly favorable to SA, which achieves 0.06 s on average nearly twice as fast as GA, and with lower dispersion across runs.

Table 7: Statistical comparison of GA and SA using bootstrap CIs (95%) and Mann–Whitney U tests (30 runs).

	Alg.	Mean	95% Bootstrap CI	U	p-value
D	GA	10143.11	[9815.20, 10449.87]	900	3.02e-11
	SA	6782.44	[6629.76, 6947.91]		
T	GA	0.1202	[0.1153, 0.1250]	900	3.02e-11
	SA	0.0614	[0.0584, 0.0642]		

Success rates are low for both algorithms at this problem size (SA: 3.3%, GA: 0%). However, only SA is able to reach the best-known tour at least once (1/30 runs).

The Mann–Whitney U tests for both distance and runtime yield U = 900 and $p \approx 3 \times 10^{-11}$, indicating highly statistically significant differences between GA and SA across all evaluated metrics.

Overall, SA offers the best accuracy–efficiency trade-off under real-time constraints, while GA becomes slower, less stable, and substantially less accurate as N increases.

While Tables 6 and 7 summarize the final performance outcomes of both algorithms, Figures 3 and 4 provide complementary insight by illustrating the optimization trajectory across iterations. These convergence plots show the median best-so-far tour length over 30 independent runs for N = 31, accompanied by the corresponding interquartile range (IQR) at each iteration.

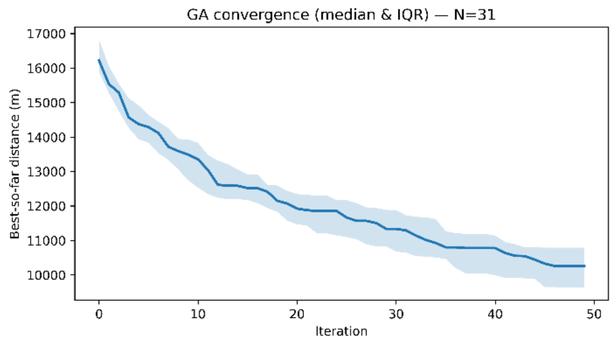


Figure 3: Convergence for N=31 over 30 runs of GA. Curve shows the median best-so-far tour length at each iteration; shaded band indicates the interquartile range (IQR)

A quantitative estimation of convergence speed was also derived from the median curves. For GA, the 95% of-final-value threshold (~10,500 m) is reached at approximately iteration 27–30. In contrast, SA reaches its 95% threshold (~7,140 m) at iteration 9–10. This indicates that SA converges roughly three times faster than GA, supporting the observed efficiency gap in the empirical results.

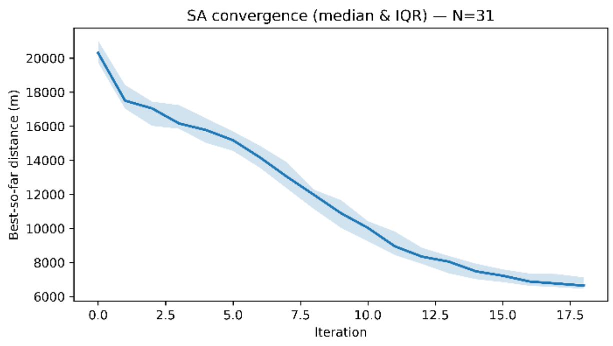


Figure 4: Convergence for N=31 over 30 runs of SA. Curve shows the median best-so-far tour length at each iteration; shaded band indicates the interquartile range (IQR)

4.4 Robustness of the algorithms under noisy and incomplete data

Tables 8-11 present the experimental results for both SA and GA under clean, noisy, and incomplete distance-matrix scenarios. The experiments were performed on a fixed instance consisting of 31 tourist attractions in Tirana.

The ΔDistance, ΔTime, and ΔGap indicators quantify the deviation from the clean baseline, while the Success_rate column reflects the algorithm's consistency in recovering near-optimal tours under uncertainty. Dist_mean denotes the mean tour length (in meters) across 30 runs, while ΔDistance (%) expresses the relative change in tour length compared to the clean baseline. T_mean (s) reports the average runtime, and ΔTime (%) quantifies the relative runtime increase under perturbation. Gap_mean (%) measures the deviation from the best-known solution, and ΔGap (percentage points (pp)) shows how this gap changes relative to the clean case. Success_rate (%) indicates the percentage of runs that exactly matched the best-known tour ($\varepsilon = 0$).

To provide a clearer algorithm-specific interpretation of robustness under uncertainty, the following subsections analyze in detail the behavior of SA and GA separately, with respect to solution quality, runtime stability, and convergence reliability under noisy and incomplete distance-matrix conditions.

4.4.1 SA robustness analysis

This subsection examines the robustness of SA algorithm under progressively degraded data conditions, including missing distances and random noise. The analysis focuses on how perturbations affect tour length, solution optimality gap, success rate, and computational time, in comparison with the clean baseline scenario. Tables 8 and 9 summarize the resulting performance metrics across all test cases.

For SA, both the solution gap and success rate remain almost unchanged across all perturbation types, confirming the strong resilience of the method.

Despite large relative increases in execution time (ΔTime% up to +2100%), the absolute runtimes remain below 1.5 seconds, making SA highly practical even when the data are incomplete or noisy.

Table 8: Comparative performance of SA algorithm under noisy and incomplete distance-matrix conditions, across all test scenarios.

Scenario	Dist_ mean	ΔDist.	Gap_ mean	ΔGap	Succ rate
clean	6782.44	0	8.854	0	3.3
miss10	7245.51	6.83	9.152	0.298	3.3
miss20	7388.59	8.94	6.254	-2.6	3.3
noisy10	6988.99	3.05	11.975	3.121	3.3
noisy15_ miss15	6778.24	-0.06	7.904	-0.95	3.3
noisy20	6559.38	-3.29	8.838	-0.01	3.3

Table 9: Runtime analysis of the SA algorithm under clean, noisy, and incomplete distance-matrix conditions, reporting mean execution time and its relative deviation from the clean scenario

Scenario	T_mean	ΔTime
clean	0.0653	0
miss10	0.8435	1192.07
miss20	1.439	2104.24
noisy10	0.2362	261.84
noisy15_miss15	1.0523	1511.93
noisy20	0.2178	233.59

The mean tour distance increases moderately under missing-edge scenarios (+6–9%), reflecting the expected effect of reduced connectivity, while under noisy conditions the distances fluctuate only slightly (±3%). Importantly, the solution gap remains within ±3 percentage points of the clean baseline (from 8.85% to a maximum of 11.98%), showing that SA can adapt efficiently to both random noise and incomplete information. The success rate is fully stable (3.3%) across all tests, indicating deterministic convergence toward near-optimal tours.

Overall, these findings demonstrate that SA degrades gracefully: its runtime scales with problem irregularity, but its solution quality and convergence behavior remain robust. This makes SA a reliable choice for real-world deployment, particularly in applications (e.g., mobile routing, tour recommendation) where the input data may be imperfect or partially missing.

4.4.2 GA robustness analysis

This subsection analyzes the robustness of GA under the same noisy and incomplete distance-matrix scenarios. The evaluation emphasizes variations in solution quality, optimality gap, feasibility preservation, and runtime overhead relative to the clean baseline. Tables 10 and 11 report the corresponding performance indicators.

Tables 10 and 11 demonstrate that, for GA, both the solution gap (ΔGap) and the runtime overhead (ΔTime) increase noticeably under missing and noisy distance conditions, indicating a high sensitivity to data degradation. Although GA consistently produces feasible tours across all scenarios, these tables clearly show that its solution quality varies substantially more than that of SA.

Table 10: Comparative performance of GA algorithm under noisy and incomplete distance-matrix conditions, across all test scenarios

Scenario	Dist_ mean	ΔDist	Gap_ mean	ΔGap	Succ rate
clean	10143.11	0	62.791	0	0
miss10	10186.23	0.43	53.454	-9.34	0
miss20	10568.79	4.2	51.988	-10.80	0
noisy10	10085.95	-0.56	61.594	-1.19	0
noisy15 miss15	10452.92	3.05	66.403	3.61	0
noisy20	10185.24	0.42	69.001	6.21	0

Table 11: Runtime analysis of the GA algorithm under clean, noisy, and incomplete distance-matrix conditions, reporting mean execution time and its relative deviation from the clean scenario

Scenario	T_mean	ΔTime
clean	0.1403	0
miss10	1.6976	1109.78
miss20	2.4688	1659.35
noisy10	0.3219	129.41
noisy15_miss15	1.7458	1144.17
noisy20	0.3576	154.86

Under missing-edge perturbations (miss10–miss20), the gap decreased slightly (≈ −9 to −11 pp) due to the on-the-fly repair mechanism that bypassed broken links, but this came at a significant computational cost, with runtime increases exceeding +1100–1600%. Similarly, when random noise was applied (±10–20%), GA exhibited relatively stable distances (ΔDistance < 1%) but still required additional iterations, leading to +130–150% longer runtimes. In the combined case (noisy15 + miss15), GA's mean gap reached ≈66%, showing that its population-based search is more sensitive to irregular graph structures compared with SA's single-solution adaptive exploration.

Across all experimental scenarios, the GA algorithm consistently produced complete and connected tours, confirming that the proposed on-the-fly link-repair mechanism effectively preserves route connectivity even when distance information is missing or corrupted. Overall, GA shows strong reliability in maintaining feasible tours under data degradation, but its efficiency is notably weaker: although it remains capable of constructing complete routes, its performance deteriorates much more rapidly than SA with respect to runtime and solution optimality.

Figure 5 complements Tables 8–11 by illustrating the anytime trade-off between solution quality (Gap %) and runtime (s) for both algorithms (SA and GA) under the same set of perturbation scenarios. Each point corresponds to the mean performance of a given scenario (clean, noisy, or incomplete), allowing a direct visual comparison of robustness and efficiency.

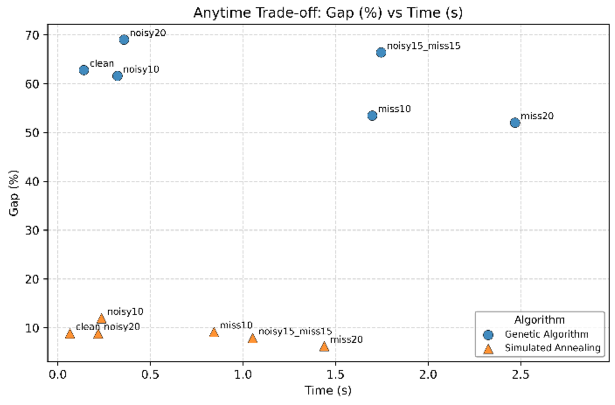


Figure 5: Anytime trade-off between solution quality (Gap %) and execution time (s) for GA and SA across clean, noisy, and incomplete data scenarios.

Each point corresponds to the mean performance of a given scenario (clean, noisy, or incomplete), allowing a direct visual comparison of robustness and efficiency. The plot clearly shows that SA (orange triangles) consistently achieves lower gaps with sub-second runtimes, while GA (blue circles) requires longer execution times and exhibits larger variability in solution quality, especially under missing-edge conditions. These visual trends confirm the tabular findings, that SA degrades smoothly and maintains stability under noisy and incomplete data, whereas GA's performance is more sensitive to data imperfections

4.5 Performance of the Hybrid Greedy + SA (Algorithm 3)

4.5.1 Performance on clean data

The Hybrid Greedy + SA algorithm was first evaluated on the clean dataset (no missing or noisy distances) to establish a baseline for performance comparison against the standalone GA and SA methods. The experiment contained 31 tourist attractions. As shown in Table 12, the hybrid approach achieved a significantly lower mean gap (≈ 1.86%) compared to SA (8.85%) and GA (62.79%), while maintaining a moderate runtime (≈ 0.25 s) and the highest success rate (10%). These results demonstrate that initializing SA with a Greedy heuristic drastically improves both convergence speed and solution quality, providing near-optimal tours at minimal computational cost.

Table 12: Baseline results for GA, SA, and Hybrid Greedy + SA on clean data.

Alg.	Dist_ mean	T_mean	Succ	Gap_ mean
Hybrid	6342.551	0.2499	10%	1.860
SA	6782.44	0.0653	3.3%	8.854
GA	10143.11	0.1403	0%	62.791

The results show that the Hybrid method consistently outperforms SA in terms of tour quality: its mean distance is substantially lower, and its confidence interval is considerably narrower, indicating both better performance and reduced variability. For execution time, SA is significantly faster, as expected from its simpler stochastic refinement process. The Hybrid method incurs a higher computational cost due to the deterministic greedy initialization and more intensive local search.

Table 13 provides a statistical comparison between SA and the Hybrid Greedy+SA algorithm using 95% confidence intervals and Mann–Whitney U tests over 30 independent runs.

Table 13: Statistical comparison of SA and Hybrid Greedy+SA using 95% CIs and Mann–Whitney U tests (30 runs).

	Alg.	Mean	95% CI	U	p-value
D	SA	6782.4	[6626.47, 6946.74]	824	2e-08
	Hybrid	6342.6	[6319.59, 6364.00]		
T	SA	0.067	[0.06363, 0.07070]	0	3.02e-11
	Hybrid	0.249	[0.24373, 0.25625]		

The Mann–Whitney U tests confirm that both distance and time differences between the two algorithms are statistically significant, with p-values far below the standard 0.05 threshold. This validates that the observed improvements in solution quality—and the corresponding increase in runtime—are not due to random variation but reflect systematic differences in algorithmic behavior.

Overall, the statistical evidence demonstrates that Hybrid Greedy+SA achieves the best accuracy among the compared methods, while SA offers superior speed, reflecting a clear trade-off between solution quality and runtime.

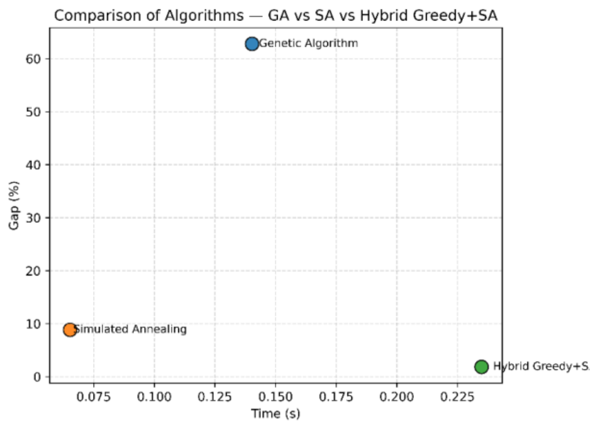


Figure 6: Comparison of GA, SA, and Hybrid Greedy + SA on clean data, showing the trade-off between runtime and optimality gap.

Figure 6, visualizes the clean baseline comparison of the three algorithms. The Hybrid Greedy+SA achieves a much smaller gap than both GA and SA while maintaining a sub-second runtime, confirming its strong potential for real-time applications in urban itinerary optimization.

4.5.2 Robustness under noisy and incomplete data

This section presents the comparative performance of the three algorithms: GA, SA, and the proposed Hybrid Greedy + SA, when evaluated under a combined perturbation scenario (noisy15_miss10). In this configuration, 15% of the pairwise distances were randomly perturbed with Gaussian noise, while 10% of the graph edges were removed to simulate missing or unreliable connectivity data. This setting reflects realistic conditions for low-infrastructure urban contexts, where digital maps or open data repositories often contain incomplete or imprecise geospatial information.

As shown in Table 14 and Figure 7, both GA and SA experience a degradation in performance when the data become noisy or incomplete, whereas the Hybrid Greedy + SA maintains lower gap values and stable success rates.

Table 14: Comparative performance of Genetic Algorithm (GA), Simulated Annealing (SA), and Hybrid Greedy + SA under the combined perturbation scenario (noisy15_miss10).

Scenario: noisy15_miss10				
Alg.	Dist_mean	T_ mean	Succ.	Gap_ mean
Hybrid	7303.99	0.942	3.3%	7.421
SA	7365.73	0.767	0%	8.329
GA	10732.95	1.407	0%	57.852

This demonstrates that initializing SA with a Greedy heuristic seed improves the anytime performance and robustness of the algorithm, even when link failures or measurement errors occur. While GA suffers from a substantial increase in gap (≈ 58%) and runtime, the hybrid method preserves feasibility and remains computationally efficient, maintaining sub-second runtimes (≈ 0.94 s) and a mean gap of 7.4%.

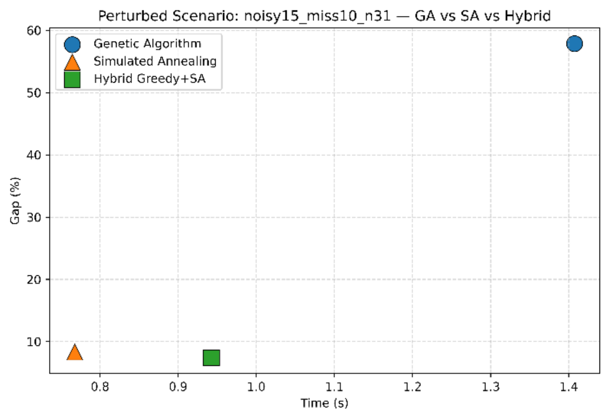


Figure 7: Performance comparison of GA, SA, and Hybrid Greedy + SA under the noisy15_miss10 scenario

Overall, the results confirm that the Hybrid Greedy + SA algorithm achieves a superior balance between robustness and efficiency, sustaining near-optimal solution quality even when data imperfections degrade the performance of traditional metaheuristics.

4.6 Comparison between straight-line and real route distances

To evaluate how the optimized itinerary behaves under real-world travel conditions, we compared the total straight-line distance produced by the Hybrid Greedy + SA algorithm with the actual routed distance obtained through the OpenRouteService (ORS) API. The optimized tour was first generated using Haversine distances computed from OpenStreetMap-based coordinates of 31 tourist attractions in Tirana. This straight-line model provides a simplified and computationally efficient geometric approximation commonly used in TSP and metaheuristic optimization research.

After the best tour was obtained, each consecutive pair of points was re-evaluated through ORS using the pedestrian routing profile. ORS calculates the true walking distance by following the real street network extracted from OpenStreetMap, thereby incorporating the constraints of urban geometry, pedestrian paths, intersections, and block structures.

The comparison revealed the following quantitative results:

- Total distance, Haversine: 6.193 km
- Total distance, ORS (real path): 10.165 km
- Deviation ratio (ORS / Haversine): 1.641

This shows that the true walkable distance across the full itinerary is approximately 64.1% longer than the geometric estimate. Such a deviation is expected in compact urban areas where direct point-to-point movement is constrained by building blocks, curved or discontinuous pedestrian paths, irregular street patterns, and one-way routing segments. The observed ratio of ≈1.64 reflects the natural divergence between idealized great-circle distances and actual walkable routes in dense city environments.

Importantly, although real-world distances differ, the relative spatial structure is preserved: closer points remain proportionally close, and distant points remain proportionally distant. This confirms that Haversine-based optimization provides a valid and computationally efficient foundation for comparing algorithmic performance. The post-evaluation using ORS reinforces the robustness of the optimized route and highlights the practical relevance of the proposed method.

4.6.1 Impact of routing method on the optimized itinerary

To further understand how the routing model affects not only the measured distance but also the structure of the optimized path, we performed a second experiment using the same set of 31 tourist locations. The Hybrid Greedy + SA algorithm was executed directly on ORS-derived real walking distances, rather than on Haversine distances. In this configuration, the algorithm produced an optimized tour of 9.355 km, which is shorter than the 10.165 km obtained when the Haversine-optimized itinerary was evaluated using ORS. This confirms that the true optimal tour under realistic routing constraints is indeed different from the one derived using geometric distances.

To illustrate these differences more clearly, Figure 8 and Figure 9 present the visualized itineraries generated by the two methods.

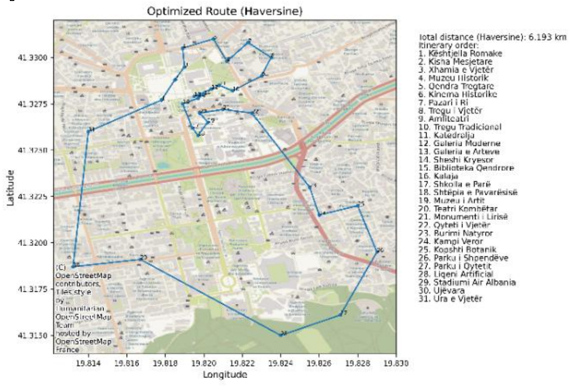


Figure 8: Optimized itinerary generated using the Hybrid Greedy + SA algorithm with Haversine great-circle distances.

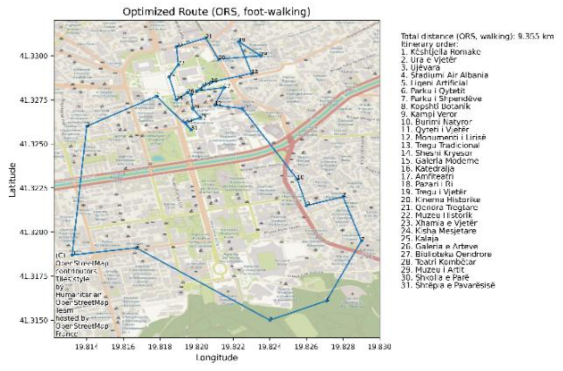


Figure 9: Optimized itinerary produced by applying the Hybrid Greedy + SA algorithm directly on real walking distances obtained through OpenRouteService (ORS).

The first map depicts the Haversine-based optimized tour, while the second shows the route obtained when the algorithm operates directly on real network distances. Although the total walking length differs substantially, due to pedestrian detours, street geometry, and one-way routing rules, the overall spatial layout of the itinerary remains similar. Clusters of nearby attractions, the direction of traversal across the city, and local segment orderings are largely preserved.

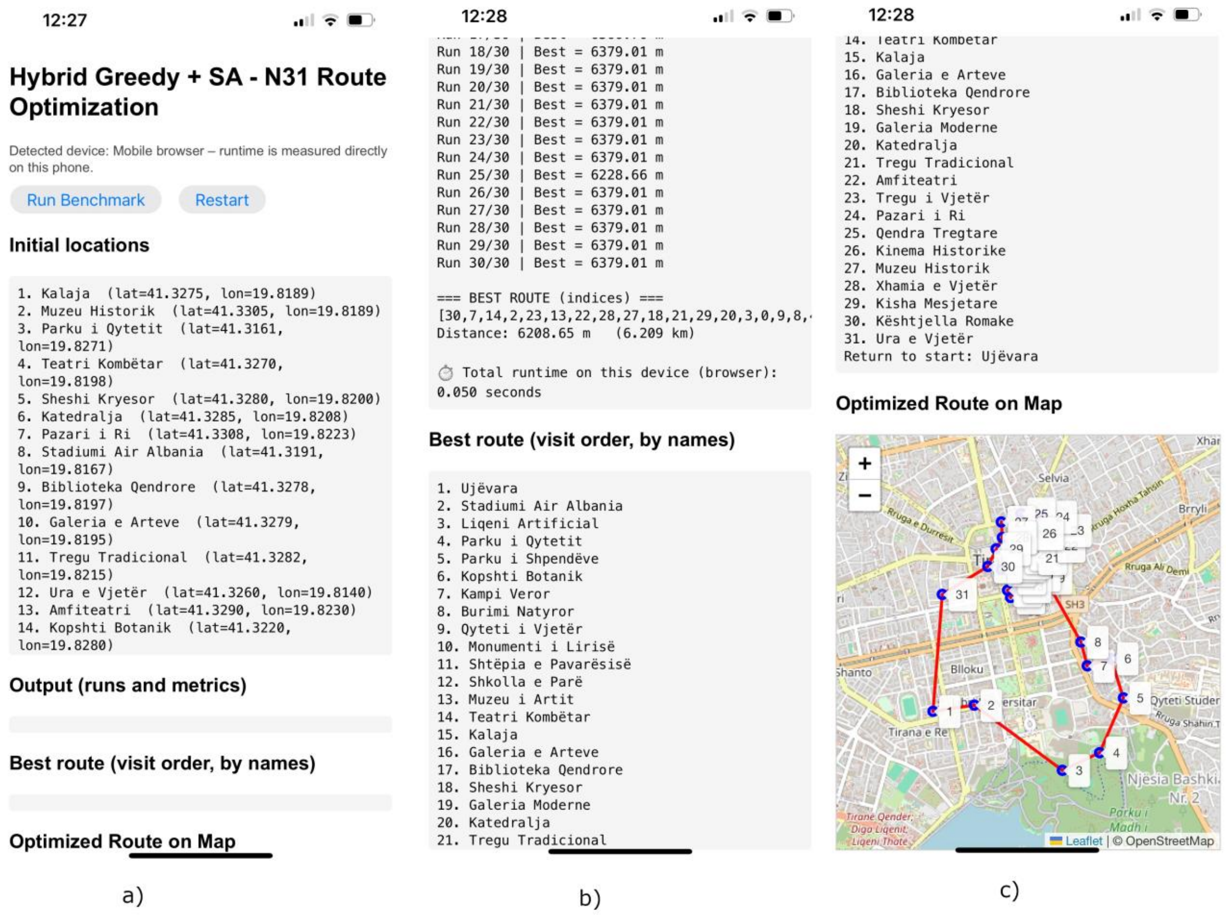
These visual comparisons demonstrate that the Haversine model successfully captures the geometric structure of the underlying optimization problem, while ORS primarily adjusts for real-world walkability constraints. Consequently, the maps strengthen the conclusion that Haversine-based optimization is suitable for fast, offline, and resource-efficient computation, whereas ORS is best used for post-validation and final distance refinement. By combining the optimization outcomes with their spatial context, the resulting visualizations offer an interpretable and actionable view of each tour, reinforcing the algorithm's suitability for applications in urban tourism planning, smart-city route recommendation, and pedestrian navigation support.

4.7 Mobile execution results

To assess performance on low-end hardware, the Hybrid Greedy + SA algorithm was executed on a Samsung Galaxy A10 running Android 11 (One UI 3.1). Using the same JavaScript implementation via the mobile browser, the full 31-point itinerary was optimized in approximately 0.24 seconds, confirming real-time capability even under constrained processing resources. The mobile run produced the same tour ordering as the desktop implementation, with only minor numerical differences in distance caused by standard floating-point variations between Python and JavaScript. These results demonstrate that the method remains practical and responsive on budget mobile phones, strengthening its suitability for deployment in real-world mobile tourism applications. Similarly, tests on an iPhone 13 (iOS 17) demonstrated robust performance, with an average execution time of 0.05 seconds over 30 runs. These results confirm that the complete optimization pipeline including initialization, search, and rendering can execute locally on mainstream mobile devices without requiring server interaction. Figure 10 presents the web-based interface running as a standalone Progressive Web App (PWA), added to the home screen, supporting local execution, route visualization over Tirana via Leaflet, and precise timing feedback. The application operates as a standalone Progressive Web App (PWA) after being added to the home screen.

5 Discussion

This work presents a reproducible, deployment-oriented evaluation of exact and metaheuristic solvers for urban tourist itineraries using open map data [15]. Although the experimental analysis is limited to Tirana, the methodology and algorithms are city-agnostic and can be directly applied to other urban contexts once attraction coordinates are available. The choice of Tirana reflects the study's focus on cities with limited digital infrastructure rather than dataset-specific characteristics. The findings support a simple operational rule: for cities with limited digital infrastructure, Simulated Annealing (SA) provides the best accuracy–efficiency trade-off under realistic latency constraints; Genetic Algorithm (GA) remains competitive at small scales but becomes less stable as the problem size (N) increases, while Brute Force (BF) serves primarily as an offline validator rather than a practical solver. The inclusion of the Hybrid Greedy + SA variant further strengthens these conclusions, showing that combining a deterministic initialization with stochastic refinement improves both convergence stability and resilience under uncertainty.



Hybrid Greedy + SA - N31 Route Optimization

Detected device: Mobile browser – runtime is measured directly on this phone.

Initial locations

```
1. Kalaja  (lat=41.3275, lon=19.8189)
2. Muzeu Historik  (lat=41.3305, lon=19.8189)
3. Parku i Qytetit  (lat=41.3161,
lon=19.8271)
4. Teatri Kombëtar  (lat=41.3270,
lon=19.8198)
5. Sheshi Kryesor  (lat=41.3280, lon=19.8200)
6. Katedralja  (lat=41.3285, lon=19.8208)
7. Pazari i Ri  (lat=41.3308, lon=19.8223)
8. Stadiumi Air Albania  (lat=41.3191,
lon=19.8167)
9. Biblioteka Qendrore  (lat=41.3278,
lon=19.8197)
10. Galeria e Arteve  (lat=41.3279,
lon=19.8195)
11. Tregu Tradicional  (lat=41.3282,
lon=19.8215)
12. Ura e Vjetër  (lat=41.3260, lon=19.8140)
13. Amfiteatri  (lat=41.3290, lon=19.8230)
14. Kopshti Botanik  (lat=41.3220,
lon=19.8280)
```

Output (runs and metrics)

Best route (visit order, by names)

Optimized Route on Map

```
Run 18/30 | Best = 6379.01 m
Run 19/30 | Best = 6379.01 m
Run 20/30 | Best = 6379.01 m
Run 21/30 | Best = 6379.01 m
Run 22/30 | Best = 6379.01 m
Run 23/30 | Best = 6379.01 m
Run 24/30 | Best = 6379.01 m
Run 25/30 | Best = 6228.66 m
Run 26/30 | Best = 6379.01 m
Run 27/30 | Best = 6379.01 m
Run 28/30 | Best = 6379.01 m
Run 29/30 | Best = 6379.01 m
Run 30/30 | Best = 6379.01 m

=== BEST ROUTE (indices) ===
[30,7,14,2,23,13,22,28,27,18,21,29,20,3,0,9,8,
Distance: 6208.65 m   (6.209 km)

⏱ Total runtime on this device (browser):
0.050 seconds
```

Best route (visit order, by names)

```
1. Ujëvara
2. Stadiumi Air Albania
3. Liqeni Artificial
4. Parku i Qytetit
5. Parku i Shpendëve
6. Kopshti Botanik
7. Kampi Veror
8. Burimi Natyror
9. Qyteti i Vjetër
10. Monumenti i Lirisë
11. Shtëpia e Pavarësisë
12. Shkolla e Parë
13. Muzeu i Artit
14. Teatri Kombëtar
15. Kalaja
16. Galeria e Arteve
17. Biblioteka Qendrore
18. Sheshi Kryesor
19. Galeria Moderne
20. Katedralja
21. Tregu Tradicional
```

```
14. Teatri Kombetar
15. Kalaja
16. Galeria e Arteve
17. Biblioteka Qendrore
18. Sheshi Kryesor
19. Galeria Moderne
20. Katedralja
21. Tregu Tradicional
22. Amfiteatri
23. Tregu i Vjetër
24. Pazari i Ri
25. Qendra Tregtare
26. Kinema Historike
27. Muzeu Historik
28. Xhamia e Vjetër
29. Kisha Mesjetare
30. Kështjella Romake
31. Ura e Vjetër
Return to start: Ujëvara
```

Optimized Route on Map

Figure 10: Execution of the Hybrid Greedy + SA algorithm directly on an iPhone 13 browser as a Progressive Web App. (a) Initial input with 31 attraction coordinates. (b) Runtime output and best tour ordering with stable distances across 30 runs. (c) Visualization of the optimized route over Tirana using Leaflet. The full optimization completes in under 0.05 seconds, confirming responsiveness on mobile devices

5.1 Design trade-offs: haversine vs. network routing

The present implementation computes pairwise distances using the Haversine formula, which provides a fast, lightweight, and fully reproducible measure of separation between geographic coordinates. This choice deliberately avoids reliance on external routing APIs and heavy graph-processing engines, making the method suitable for deployment in settings with limited connectivity or computational resources. Such characteristics are particularly relevant for urban environments like Tirana, where practical applications may need to operate offline, at low cost, or within platforms that cannot depend on persistent access to cloud-based routing services.

However, Haversine distances do not account for the structure of the street network, pedestrian pathways, one-way segments, or other real travel constraints. To quantify the impact of this simplification, our study conducted a secondary evaluation using OpenRouteService (ORS), which computes walking routes based on the actual OpenStreetMap road and pedestrian network. Applying ORS to the optimized Haversine-based tour showed that the actual routed distance is about 30.9% longer than the geometric estimate, a deviation expected in dense urban environments where movement is constrained by building blocks and irregular street layouts. Despite this difference, the experiment shows that the relative spatial structure of the problem is preserved: points that are close remain proportionally close, and the ranking of distances does not change substantially. Thus, the optimization landscape explored by the GA, SA, and Hybrid algorithms remains meaningful even when using Haversine. The Haversine model provides a stable and computationally efficient abstraction for algorithmic comparison, while the ORS-based evaluation confirms how geometric solutions translate into realistic walking distances.

5.2 Generalization and scalability

Although the experimental analysis focused primarily on instances with 31 locations, additional tests were conducted on a larger dataset containing 49 tourist points to assess scalability. The results confirmed the same performance trends: SA consistently achieved much shorter tours and lower optimality gaps than GA, while maintaining sub-second runtimes (≈ 0.08 s). In contrast, GA exhibited a substantial degradation in solution quality, with mean gaps exceeding 56% and increased variability. This indicates that the advantages of SA not only persist but become more pronounced as problem size grows, suggesting better scalability and robustness for larger urban itinerary planning scenarios. In addition, the Hybrid Greedy + SA algorithm maintained its performance advantages when the problem size increased. For the larger instance (N = 49), the hybrid method achieved a remarkably low mean optimality gap of only 2.64%, substantially outperforming both GA (56.31%) and SA (9.07%) under the same conditions. Although runtime increased to approximately 0.42 s, this growth remained moderate and well within real-time applicability thresholds. Notably, the hybrid approach preserved solution stability, as reflected by the very small variability across runs (Gap_std ≈ 1.17%), indicating that the Greedy initialization continues to provide high-quality starting tours even in larger search spaces, while the SA refinement efficiently exploits local structure. These results suggest that the hybrid strategy scales more gracefully than standalone heuristics, retaining both accuracy and robustness as the number of attractions grows.

5.3 Future integration with dynamic and real-time data

Building on the demonstrated robustness and anytime performance of the proposed algorithms, future extensions will focus on dynamic and real-time optimization. The current framework, which operates on a static distance matrix, can be enhanced with live contextual data such as real-time traffic flow, temporary event schedules, and user preference feedback.

For instance, the Hybrid Greedy + SA model could dynamically re-weight edges based on live travel times, temporary pedestrian restrictions, or the user's evolving interests (e.g., preference for cultural sites or shorter walking routes). Integrating such adaptive data streams would transform the current approach into a context-aware recommender system, capable of updating feasible itineraries on the fly.

This direction aligns with current trends in smart tourism and mobility-as-a-service, where systems must maintain robustness under uncertainty while delivering personalized and real-time route adjustments for end users.

6 Conclusion

In conclusion, this study contributes both methodological clarity and practical relevance. By framing one-day tourist itinerary generation as a TSP and benchmarking exact and metaheuristic solvers on real-world map data, it shows how even simple algorithms can inform the design of deployable systems for cities with limited digital infrastructure. The comparative analysis demonstrates that while BF provides an exact validation baseline, GA and especially SA deliver scalable, near-optimal solutions under realistic constraints. The extended experiments under noisy and incomplete distance data further confirmed SA's robustness, showing minimal degradation in solution quality despite input uncertainty. Moreover, the introduction of the Hybrid Greedy + SA variant significantly improved both convergence speed and accuracy, achieving the lowest mean gap and maintaining sub-second runtimes even under data perturbations, confirming that combining a deterministic greedy start with stochastic refinement enhances both reliability and efficiency.

The mobile execution results demonstrate that the proposed Hybrid Greedy + SA method is practical for deployment in real tourist applications. The ability to compute optimized routes directly on the smartphone without cloud computation enhances system robustness, enables offline functionality, and reduces dependency on external APIs. The sub-second performance observed on the iPhone indicates that the algorithm provides an acceptable user experience even on mobile hardware with limited computational resources. From a system-design perspective, the successful mobile execution demonstrates that lightweight metaheuristics can reliably support real-time decision-making in e-tourism scenarios, where users typically depend on smartphones while navigating urban environments. The results strengthen the external validity of the proposed method and confirm its suitability for integration into practical mobile routing tools. Furthermore, the demonstrated efficiency on resource-constrained hardware positions such techniques as strong candidates for next-generation smart-tourism systems, where applications may incorporate richer contextual information, preference-aware routing, and multi-day itinerary planning. In this way, the study effectively bridges algorithmic research with applied urban-tourism needs, offering actionable insights for both academia and practice.

References

[1] Wu, W., et al. *Digital Tourism and Smart Development: State-of-the-Art Review*. Sustainability, 2024. **16**, DOI: 10.3390/su162310382.

[2] Zeqiri, A., A. Ben Youssef, and T. Maherzi Zahar. *The Role of Digital Tourism Platforms in Advancing Sustainable Development Goals in the Industry 4.0 Era*. Sustainability, 2025. **17**, https://doi.org/10.3390/su17083482

[3] Instituti i, S. *Statistikat e Turizmit*. Industria, Tregtia dhe Shërbimet 2025-09-14; Available from: https://www.instat.gov.al/al/temat/industria-tregtia-dhe-sh%C3%ABrbimet/statistikat-e-turizmit/#tab2.

[4] Visit, T. *Visit Tirana*. Visit Tirana 2025-09-14; Available from: https://www.visit-tirana.com/.

[5] Albania Tourist, G. *Albania Tourist Guide*. Albania Tourist Guide 2025-09-14; Available from: https://albaniatouristguide.com/.

[6] Sylejmani, K., et al., *Solving the tourist trip planning problem with attraction patterns using meta-heuristic techniques.* Information Technology & Tourism, 2024. **26**(4): p. 633-678. https://doi.org/10.1007/s40558-024-00297-w

[7] Zhang, J., H. Kawasaki, and Y. Kawai. *A Tourist Route Search System Based on Web Information and the Visibility of Scenic Sights*. in *2008 Second International Symposium on Universal Communication*. 2008, retrieved https://www.cvg.ait.kyushu-u.ac.jp/papers/2007_2009/3-1/11-ISUC2008.pdf

[8] Adamo, T., Colizzi, L., Dimauro, G., Ghiani, G., & Guerriero, E. (2024). A multi-modal tourist trip planner integrating road and pedestrian networks. *Expert Systems with Applications, 237*(Part B), 121457. https://doi.org/10.1016/j.eswa.2023.121457

[9] Tang, Y., et al., *ITINERA: Integrating Spatial Optimization with Large Language Models for Open-domain Urban Itinerary Planning*, in *arXiv*. 2024. https://doi.org/10.18653/v1/2024.emnlp-industry.104

[10] Souffriau, W., Vansteenwegen, P., Vertommen, J., Berghe, G. V., & Oudheusden, D. V. (2008). A personalized tourist trip design algorithm for mobile tourist guides. *Applied Artificial Intelligence, 22*(10), 964–985. https://doi.org/10.1080/08839510802379626

[11] Androutsopoulos, K.N. and K.G. Zografos, *Solving the multi-criteria time-dependent routing and scheduling problem in a multimodal fixed scheduled network.* European Journal of Operational Research, 2009. 192(1): p. 18-28. https://doi.org/10.1016/j.ejor.2007.09.004

[12] Mangini, A. M., Roccotelli, M., & Rinaldi, A. (2021). A Novel Application Based on a Heuristic Approach for Planning Itineraries of One-Day Tourist. *Applied Sciences, 11*(19), 8989. https://doi.org/10.3390/app11198989

[13] Benchekroun, Y., et al. *Hybrid Framework: The Use of Metaheuristics When Creating Personalized Tourist Routes*. Digital, 2025. 5, https://doi.org/10.3390/digital5030036

[14] Li, S., Luo, T., Wang, L. *et al.* Tourism route optimization based on improved knowledge ant colony algorithm. *Complex Intell. Syst.* **8**, 3973–3988 (2022). https://doi.org/10.1007/s40747-021-00635-z

[15] Damos, M.A., et al. *Enhancing the K-Means Algorithm through a Genetic Algorithm Based on Survey and Social Media Tourism Objectives for Tourism Path Recommendations*. ISPRS International Journal of Geo-Information, 2024. **13**, https://doi.org/10.3390/ijgi13020040

[16] Pérez-Cañedo, B., Novoa-Hernández, P., Porras, C., Pelta, D. A., & Verdegay, J. L. (2024). Contextual analysis of solutions in a tourist trip design problem: A fuzzy logic-based approach. *Applied Soft Computing, 154,* 111351. https://doi.org/10.1016/j.asoc.2024.111351

[17] Chrysafiadi, K., Kontogianni, A., Virvou, M., & Alepis, E. (2025). Enhancing User Experience in Smart Tourism via Fuzzy Logic-Based Personalization. *Mathematics*, *13*(5), 846. https://doi.org/10.3390/math13050846

[18] Liu, X., & Li, Y. (2024). *Analysis of immersive virtual reality tourism resources combined with fuzzy comprehensive evaluation algorithm*. Informatica, 48, 189–210 DOI: https://doi.org/10.31449/inf.v48i21.6057 , retrieved from https://www.informatica.si/index.php/informatica/article/view/6057/3468

[19] Hyso, A., *tour-routing-tirana-bf-ga-sa*, in *GitHub repository*. 2025, GitHub. https://github.com/AlketaHyso/tour-routing-tirana-bf-ga-sa

[20] Hyso, A., 2025, GitHub.https://alketahyso.github.io/route_mobile_benchmark/