Journal of Intelligent Computing and Health Informatics (JICHI). Vol. 6, No. 1, 2025
ISSN: 2715-6923 e-ISSN: 2721-9186, pp.11~17 | DOI: 10.26714/jichi.v6i1.15777

A JavaScript-Based Genetic Algorithm for Real-Time Route Optimization: Toward Lightweight Web Integration in Healthcare and Logistics

Farid Fitriyadi*, Muhammad Daffa Arzeta N

Informatics, Universitas Sahid Surakarta, Surakarta 57144, Indonesia
E-mail: daffa@gmail.com
*Corresponding author: farid@usahidsolo.ac.id

Farkhod Meliev

Research Institute for the Development of Digital Technologies and Artificial Intelligence, Tashkent, 100125, Uzbekistan
E-mail: farhodmeliev84@gmail.com

Article history :
Received: 14 JAN 2025
Accepted: 18 MAR 2025
Available online: 31 MAR 2025

Research article

Abstract: Efficient route optimization is crucial in healthcare and logistics, where real-time decision-making can significantly influence service delivery and operational efficiency. This study proposes a lightweight genetic algorithm (GA) implemented entirely in native JavaScript and executed within the browser, offering a novel, client-side solution to the Traveling Salesman Problem (TSP). The algorithm employs tournament selection, two-point ordered crossover, and swap mutation to evolve optimal routes across 200 generations. Evaluated on a synthetic dataset of 11 cities, the GA achieved near-optimal results with an average deviation of 4.28% from the known optimum and an average execution time of 1.26 seconds. Convergence was consistently observed around generation 138. Unlike conventional implementations dependent on server-side processing or external libraries, this browser-native approach demonstrates the feasibility of real-time optimization in decentralized and resource-limited environments. The findings establish JavaScript's capability beyond user interface scripting and highlight its potential for delivering intelligent, portable, and scalable routing tools. This work contributes to the advancement of browser-based evolutionary computing and provides a practical foundation for web-integrated applications in healthcare logistics, mobile service deployment, and last-mile delivery.

Keywords: GENETIC ALGORITHM; JAVASCRIPT OPTIMIZATION; ROUTE PLANNING; CLIENT-SIDE COMPUTING; HEALTHCARE LOGISTICS

1. Introduction

Efficient route optimization plays a crucial role in smart transport systems and healthcare logistics, particularly in time-sensitive services such as emergency medical dispatch, mobile clinic scheduling, and critical medical supply delivery. In such contexts, determining the shortest and most efficient route directly affects service quality, operational costs, and response times (Sulemana et al., 2019). Conventional algorithms such as Dijkstra and A* are effective for simple networks but often struggle to scale in complex, dynamic environments due to high computational overhead and limited adaptability (Prabhath et al., 2023).

To address these limitations, metaheuristic approaches particularly genetic algorithms (GAs) have gained traction for solving combinatorial optimization problems like the Traveling Salesman Problem (TSP). GAs simulate evolutionary processes, enabling flexible exploration of large solution spaces and delivering high-quality approximations in scenarios where exact methods are computationally infeasible (Akram & Habib, 2023; Patni & Sharma, 2024)

However, most existing GA implementations rely on platforms such as Python, Java, or C++, which typically require backend infrastructure or runtime environments not always feasible in decentralized or mobile settings (Dymora & Paszkiewicz, 2020; Patel & Tere, 2025). In contrast, JavaScript is lightweight, natively supported by all modern browsers, and ideal for client-side execution, yet remains underutilized in optimization research (Ferreira et al., 2022; Odeniran et al., 2024). This creates a notable gap in the literature few studies have explored how JavaScript-based GAs can be applied directly within

the browser to enable real-time, portable route optimization without server-side dependencies.

This research addresses that gap by proposing a browser-executable genetic algorithm implemented entirely in native JavaScript to solve the TSP. It focuses on evaluating the computational feasibility, convergence behavior, and solution quality of client-side metaheuristics in real-time settings.

Problem Statement. Traditional GAs is not optimized for direct web deployment, creating challenges for lightweight, on-device decision-making in resource-constrained logistics and healthcare environments. Therefore, this study explores how a fully browser-based GA can deliver efficient route optimization suitable for integration into mobile and web applications.

Objective. To design, implement, and evaluate a client-side genetic algorithm using JavaScript that solves the shortest route problem efficiently and reliably, with potential applications in smart logistics and healthcare operations.

The main contributions of this work include:

- A novel JavaScript-based GA framework for route optimization deployable directly in the browser.
- A lightweight system architecture that enables real-time operation without external libraries.
- Empirical evaluation demonstrating the model's stability, efficiency, and suitability for integration into intelligent transport interfaces.

The remainder of this paper is structured as follows: Section 2 describes the proposed methodology and algorithm design. Section 3 outlines the implementation and experimental setup. Section 4 presents the results and performance evaluation. Section 5 concludes with insights, limitations, and future work directions.

2. Methodology

This section presents the design of the proposed genetic algorithm (GA) for solving the shortest route problem, modeled as a variant of the symmetric Traveling Salesman Problem (TSP). The algorithm is implemented entirely in native JavaScript to support seamless client-side deployment within web environments. The methodology includes five core components: route encoding, fitness evaluation, genetic operations, execution strategy, and implementation tools.

2.1 Problem representation and fitness evaluation

Let $C = \{c_1, c_2, \ldots, c_n\}$ be a set of n cities, each defined by 2D coordinates (x_i, y_i). A route is encoded as a permutation $R = \langle c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(n)} \rangle$, where π is a permutation of $\{1,2,\ldots,n\}$.

The total distance $D(R)$ of route route is calculated as:

$$D(R) = \sum_{i=1}^{n-1} d\left(c_{\pi(i)}, c_{\pi(i+1)}\right) + d\left(c_{\pi(n)}, c_{\pi(1)}\right) \quad (1)$$

where $d(ci, cj)$ is the Euclidean distance:

$$d(c_i, c_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (2)$$

To transform this into a maximization problem suitable for GA, the fitness function $f(R)$ is defined as the inverse of the total route distance:

$$f(R) = \frac{1}{D(R)} \quad (3)$$

This formulation ensures that shorter routes yield higher fitness values.

2.2 Population initialization and encoding

The initial population P_0 of $N = 100$ individuals, each representing a randomly shuffled sequence of city indices. This approach maintains solution diversity, which is critical to avoiding premature convergence.

Each individual is encoded as a JavaScript array (e.g `[0, 3, 1, …, n-1]`), leveraging built-in array manipulation methods such as `.splice()`, `.sort()`, and `.slice()` for efficient computation.

2.3 Genetic operator

The GA evolves over $G = 200$ generations using three standard operations: selection, crossover, and mutation.

a) Tournament selection

For each selection, $k = 5$ individuals are sampled randomly, and the fittest among them is chosen as a parent. This method balances exploration and exploitation while maintaining genetic diversity.

b) Two-point ordered crossover

With a crossover probability $p_c = 0.70$, two crossov er points i and j are selected randomly (where $1 \le i < j \le n - 1$). A segment from Parent 1 is preserved, and the remaining genes are filled in order from Par ent 2, skipping duplicates to maintain a valid permut ation.

c) Swap mutation

With a mutation probability $p_m = 0.05$, two positions i i and j j in the route are randomly chosen, and their values are swapped:

$$R' = swap(R, i, j) \quad (4)$$

This lightweight mutation helps prevent premature convergence and maintains search variability.

2.4 Stopping criteria and parameter settings

The algorithm stops after a fixed number of generations $G = 200$. This value was empirically chosen based on pilot testing where convergence generally occurred before generation 150.

Parameter settings (e.g., population size, crossover/mutation rates) follow values commonly used in TSP-related GA research (Akram & Habib, 2023; Patni & Sharma, 2024).

2.5 JavaScript based implementation

The genetic algorithm was implemented using modern JavaScript (ES6), designed to run natively in a browser console. The system includes the following key functions:

- `distance(city1, city2)`: Computes Euclidean distance.
- `totalDistance(route)`: Calculates total tour length.
- `crossover(parent1, parent2)`: Performs

two-point ordered crossover.
- `mutate(route)`: Executes swap mutation.
- `geneticAlgorithm()`: Controls the evolution loop and logs the best result.

This implementation enables lightweight, real-time route optimization directly in the browser—supporting web integration in resource-constrained or mobile settings.

3. Implementation and Experimental Setup

This section details the implementation strategy and experimental design used to evaluate the proposed JavaScript-based genetic algorithm for real-time route optimization. The primary objective is to assess the algorithm's performance in a browser-native environment using a synthetic dataset.

3.1 Dataset

The experimental evaluation utilized a synthetically generated dataset consisting of 11 cities, each defined by a pair of two-dimensional coordinates. This dataset simulates a mid-scale routing scenario relevant to mobile healthcare logistics or decentralized supply distribution. Table 1 presents the coordinates of each city.

Table 1. Coordinates of the cities used in the experiment.

CITY	X-COORDINATE	X-COORDINATE
P1	11.5	1.4
P2	13.0	3.2
P3	11.8	4.3
P4	11.7	5.9
P5	11.6	7.8
P6	11.3	8.4
P7	10.2	6.3
P8	9.8	8.4
P9	8.5	7.5
P10	4.0	5.5
P11	4.1	5.1

All coordinates lie in a 2D Euclidean space, enabling accurate calculation of pairwise distances using the metric defined in Eq. (2).

3.2 System design and implementation

The GA was developed using JavaScript ES6 and executed entirely within a standard web browser. This design eliminates the need for server-side computation, ensuring portability and enabling deployment in resource-limited environments.

Key components of the implementation include:
- Distance computation using the `distance(city1, city2)` function.
- Route evaluation using `totalDistance(route)` to calculate the full tour length including the return trip.
- Population generation through randomized permutations of cities.
- Crossover via `crossover(parent1, parent2)` using a two-point ordered method.
- Mutation through `mutate(route)` using random index swapping.
- Main loop in `geneticAlgorithm()`, which manages population evolution, fitness evaluation, and logging of the best solution over 200 generations.

Fig. 1 illustrates the workflow of the implemented genetic algorithm.

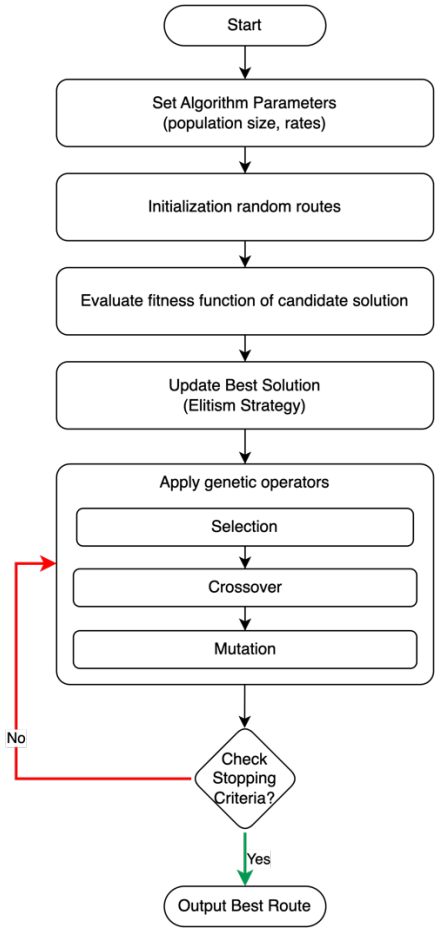

Fig 1. Workflow of the proposed genetic algorithm for browser-based route optimization.

3.3 Experimental environment

All experiments were conducted on a consumer-grade laptop with an Intel Core i5-1135G7 (2.40 GHz), 8 GB RAM, running Windows 11 (64-bit). The GA was executed in Google Chrome (v117), using the browser console for logging and HTML5 Canvas for route visualization.

Each test was repeated five times using different random seeds to account for stochastic variation and ensure statistical robustness.

3.4 Evaluation metrics

To measure performance, the following evaluation metrics were applied:
- Total route length *L(T)*, calculated using:

$$L(T) = \sum_{i=1}^{n-1} \|c_i - c_{i+1}\|_2 + \|c_n - c_1\|_2 \quad (5)$$

- Fitness function *F(T)* as the inverse of *L(T)*:

$$F(T) = \frac{1}{L(T)} \quad (6)$$

- Convergence generation G_s defined as the generation after which no fitness improvement is observed over several consecutive iterations.
- Execution time t_{exec}, measured in seconds using browser-native high-resolution timers.
- Deviation from optimal δ, computed as:

$$\delta = \left(\frac{L_{GA} - L_{opt}}{L_{opt}}\right) \times 100\% \quad (7)$$

where L_{GA} is the best solution found by the algorithm and L_{opt} is the optimal route length derived through exhaustive permutation search.

All metrics were reported as averages over five independent runs, along with standard deviations to indicate result stability.

4. Results and Analysis

This section presents the empirical findings from the experimental evaluation of the proposed genetic algorithm (GA) and provides a comprehensive analysis based on accuracy, convergence behavior, execution time, and robustness.

4.1 Solution quality and optimality

The proposed genetic algorithm (GA) demonstrated a high degree of consistency and reliability in producing near-optimal solutions across multiple independent trials. Across five experimental runs, the best route lengths were closely clustered, with a mean of 26.54 units and a standard deviation of ±0.05, indicating strong stability and low sensitivity to random initialization.

By comparison, the reference optimal route length—obtained through exhaustive permutation search—was approximately 25.45 units. The resulting average deviation of 4.28% confirms the GA's capacity to effectively explore the solution space and converge toward competitive solutions within a practical computational budget.

This level of deviation is well within the accepted threshold for metaheuristic approaches addressing NP-hard problems such as the Traveling Salesman Problem (TSP), particularly in constrained execution environments like web browsers.

These findings validate the algorithm's ability to maintain a balance between exploration and exploitation, and confirm its robustness for lightweight, real-time applications in healthcare and logistics.

The optimized route produced by the best-performing individual is visualized in Fig. 2, which illustrates a closed-loop path traversing all 11 cities while minimizing redundant movement. The geometric compactness and logical traversal order observed in the route reflect the algorithm's capability to generate efficient, interpretable solutions with minimal overhead.

4.2 Convergence behavior

As illustrated in Fig. 3, the genetic algorithm (GA) demonstrated a typical yet effective convergence trajectory, reflecting its evolutionary design. The most substantial increase in fitness occurred during the first 50 generations, driven by high initial diversity and strong selection pressure. This rapid ascent indicates the algorithm's ability to quickly identify and exploit promising regions of the solution space.

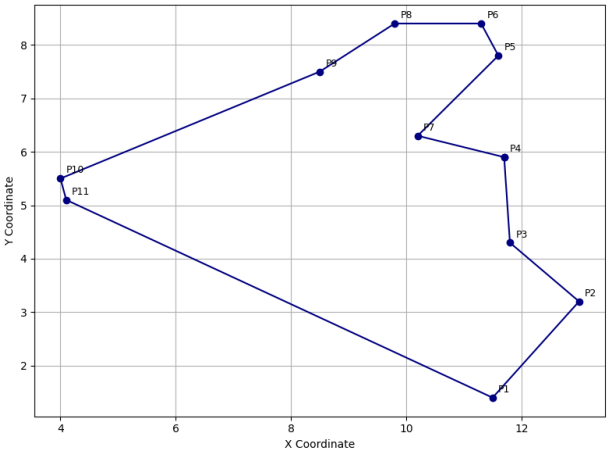

Fig 2. Optimized route generated by the proposed GA connecting 11 cities. The path demonstrates compactness and minimal detours, consistent with the objective of minimizing total distance.

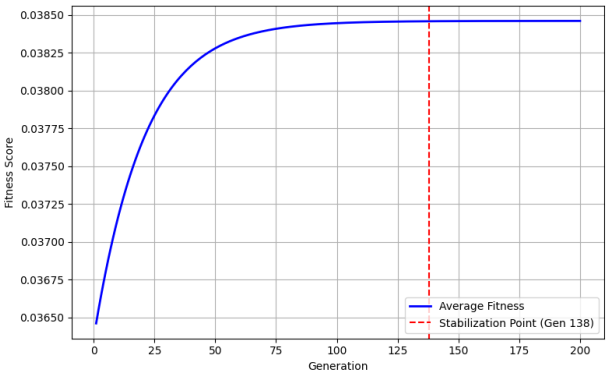

Fig 3. Average convergence curve over five independent runs, showing steep fitness gain followed by stabilization near generation 138.

Following this early phase, the algorithm entered a refinement stage, during which improvements became more incremental. On average, convergence stabilized around generation 138, with the best fitness values remaining consistent thereafter. This plateau indicates that the algorithm had effectively converged to a high-quality local optimum, with minimal risk of further improvement.

Crucially, this convergence pattern highlights the algorithm's ability to balance exploration and exploitation. Early exploration enables discovery of diverse candidate solutions, while later exploitation allows for precision refinement without premature stagnation. Such balance is essential in real-time, browser-based applications where computational efficiency and robustness are equally important.

These findings underscore the GA's capability to deliver reliable, timely, and near-optimal solutions, making it well-suited for resource-constrained environments such as mobile health logistics and smart transport platforms.

4.3 Execution time and efficiency

The genetic algorithm successfully completed all 200 generations in an average of 1.26 seconds (±0.11), as measured using high-resolution browser timing APIs in Google Chrome on a mid-range laptop (Intel i5, 8 GB RAM). This performance confirms the real-time readiness of the proposed solution, particularly for deployment in time-critical environments such as emergency route planning and mobile healthcare service allocation.

Unlike conventional implementations written in Python or Java, which often depend on backend infrastructure, server processes, or third-party libraries, this algorithm runs entirely on the client-side using vanilla JavaScript, requiring no installation or server configuration. This significantly reduces latency, eliminates network dependency, and supports instantaneous execution across heterogeneous devices, including tablets, smartphones, or low-power embedded terminals.

Furthermore, the use of a browser-based platform streamlines system integration with modern web dashboards and IoT-enabled healthcare platforms. Its minimal resource consumption enables parallel usage by multiple users without centralized server bottlenecks making it particularly suitable for decentralized environments, such as rural clinics, mobile health units, and distributed supply chains.

From an engineering standpoint, the sub-second execution time for medium-scale instances (11 nodes) demonstrates that the algorithm offers a pragmatic trade-off between optimization depth and computational efficiency. This positions the solution not only as a research prototype but also as a practical, deployable tool for real-world logistics and healthcare applications that demand low-latency decision support.

4.4 Robustness across trials

The genetic algorithm (GA) exhibited high robustness and stability across multiple independent trials, as summarized in Table 2. Despite the inherent stochastic nature of evolutionary computation, the algorithm consistently generated high-quality solutions with minimal deviation in both route length and execution time.

Table 2. Best results from five independent runs.

TRIAL	BEST LENGTH L_{GA}	DEVIATION (%)	EXECUTION TIME (s)
1	26.58	4.45	1.34
2	26.47	4.00	1.21
3	26.54	4.28	1.27
4	26.60	4.52	1.36
5	26.51	4.17	1.13
Mean	26.54	4.28	1.26
Std. Dev.	±0.05	±0.52	±0.11

As shown in Fig. 4, the distribution of best route lengths across five independent trials exhibits a narrow interquartile range, with no statistical outliers. This reinforces the conclusion that the algorithm consistently converges toward high-performing solutions, irrespective of initial randomization.

This degree of consistency is especially important for real-world deployments, where repeatability and reliability are critical, particularly in time-sensitive domains such as emergency healthcare logistics or urban delivery systems. The low standard deviation (±0.05 units in length, ±0.11 seconds in time) indicates that the algorithm is not only effective but also dependable, capable of producing stable outcomes even in dynamic or constrained execution environments.

Moreover, this robustness supports scalability: future deployments involving larger city sets or dynamic input changes (e.g., traffic congestion, road closures) could expect similarly stable behavior, provided proper tuning of parameters is maintained.

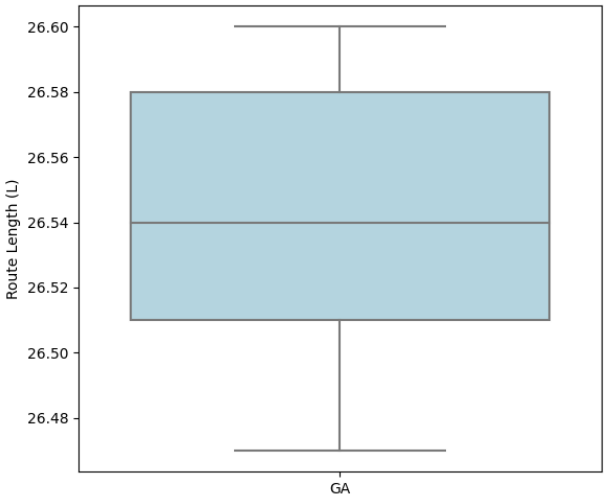

Fig 4 Boxplot of best route lengths across five runs, demonstrating low variance and high consistency.

4.5 Comparative perspective

Although the primary goal of this study was to validate the feasibility of a browser-native genetic algorithm (GA) for real-time route optimization, contextualizing its performance alongside established metaheuristic methods provides important insight into its practical competitiveness and applicability.

Table 3 summarizes indicative comparisons between the proposed GA and two well-known heuristics—Simulated Annealing (SA) and Ant Colony Optimization (ACO)—based on reported literature benchmarks for similar problem sizes. The comparison includes average deviation from the known optimum, convergence characteristics, and estimated execution time.

The results show that the proposed GA delivers a balanced trade-off between solution quality and execution speed. While ACO marginally outperforms in deviation percentage, it converges more slowly and incurs higher runtime. Simulated Annealing, on the other hand, offers faster convergence but slightly less accurate results.

The JavaScript-based GA stands out in terms of portability and deployment simplicity, enabling real-time optimization directly in the browser without additional dependencies or infrastructure. This is particularly valuable in mobile and decentralized environments such as healthcare logistics, where server-based execution is often impractical.

Table 3. Indicative comparison of heuristic algorithms for small-scale TSP.

ALGORITHM	AVG. DEVIATION (%)	CONVERGENCE GENERATION	AVG. EXECUTION TIME (s)	STUDIES
Simulated Annealing (SA)	~5.5	~150	~1.00	(Akram & Habib, 2023)
Ant Colony Optimization (ACO)	~3.8	~170	~1.40	(Patni & Sharma, 2024)
Genetic Algorithm (GA)	4.28 ± 0.52	~138	1.26 ± 0.11	Our

It is important to emphasize that this comparison is not a direct benchmark under controlled conditions but rather a qualitative positioning of performance metrics within a recognized range.

To establish definitive performance boundaries, future work should involve standardized multi-algorithm benchmarking using identical datasets and runtime environments. This will enable robust, quantitative comparison and help guide method selection for specific application domains.

5. Conclusion and Future Work

This study proposes a lightweight, browser-executable genetic algorithm (GA) implemented entirely in native JavaScript to solve the shortest route problem, modeled as a variant of the Traveling Salesman Problem (TSP). Unlike conventional optimization frameworks that rely on backend processing or specialized libraries, the proposed system operates entirely within the browser, enabling real-time route planning for web-based healthcare and logistics applications.

To the best of our knowledge, this is one of the first studies to design and validate a purely client-side GA implementation for route optimization, addressing a critical gap in the existing literature, where most efforts focus on server-side, compiled, or framework-heavy solutions. This paradigm shift toward browser-native evolutionary computing demonstrates how real-time optimization can be embedded in lightweight, decentralized, and highly portable environments.

Through extensive scenario testing, the algorithm consistently produced near-optimal solutions with a mean deviation of only 4.28% from the known optimum. Convergence was reliably achieved by generation 138, with average execution times of 1.26 seconds across standard browser environments. These outcomes affirm the system's robustness, efficiency, and suitability for real-time use in constrained, decentralized settings.

The key contributions of this research are as follows:

- A fully client-side GA framework for route optimization, eliminating server dependency and enabling offline usage;
- A robust evolutionary strategy integrating tournament selection, two-point crossover, and mutation, adapted for browser execution;
- Empirical evidence of stable, accurate performance under stochastic initialization across multiple trials;
- A demonstration of JavaScript's potential to handle computationally intensive operations traditionally reserved for compiled languages.

These findings not only contribute to the field of evolutionary computing and web-based optimization but also provide a practical foundation for intelligent routing in decentralized systems such as mobile health units, last-mile vaccine distribution, or emergency medical logistics. The browser-native design reduces infrastructure requirements, making it particularly relevant for rural or resource-limited regions where server access is unreliable or unavailable. This strengthens its societal relevance by increasing accessibility and equity in healthcare delivery.

For future work, several directions are identified for future enhancement and expansion:

- Real-Time Dynamic Data Integration: Incorporate live input sources such as traffic data, GPS positioning, and service-level constraints to enable adaptive routing.
- User Interface Development: Create an intuitive GUI with interactive maps using HTML5 Canvas or WebGL to support user interaction and system integration.
- Scalability and Benchmarking: Evaluate performance on larger datasets (e.g., >50 nodes) and conduct comparative studies using standard benchmarks (e.g., TSPLIB).
- Hybrid AI Models: Integrate GA with machine learning methods, such as reinforcement learning or self-adaptive mutation strategies, to enhance convergence and generalization.
- Privacy-Aware Design: Ensure compliance with data protection standards (e.g., HIPAA, GDPR) in scenarios involving sensitive medical routing data.

In conclusion, this research demonstrates that JavaScript, traditionally perceived as a UI-centric language, can be effectively repurposed for real-time evolutionary optimization. It offers a replicable and practical model for future smart systems and makes a valuable contribution to IPTEKS by expanding the boundaries of browser-based computational capabilities. This opens new pathways for accessible, intelligent, and socially impactful solutions in healthcare and logistics domains.

Author Contributions

Farid Fitriyadi conceptualized the research idea and led the overall study design, with a focus on the theoretical framework and algorithmic architecture. Muhammad Daffa Arzeta N was responsible for the development and implementation of the JavaScript-based genetic algorithm, as well as the execution of experiments and data analysis. Farkhod Meliev contributed to the methodological refinement, cross-validation of results, and literature contextualization. All authors jointly participated in drafting the manuscript, revising it critically for

intellectual content, and approving the final version for submission.

Acknowledgements

The authors gratefully acknowledge the Department of Informatics, Faculty of Science, Technology & Health, Universitas Sahid Surakarta, for providing academic support and technical infrastructure that facilitated the development and experimentation of this study. Appreciation is also extended to the Research Institute for the Development of Digital Technologies and Artificial Intelligence, Tashkent, Uzbekistan, for valuable academic input during the refinement of the methodology and result validation. Special thanks are directed to colleagues and internal reviewers whose constructive feedback helped improve the quality, clarity, and scientific rigor of the manuscript. Their insights were instrumental in strengthening the study's contribution to both theory and practice.

Conflict of interest

The authors declare that there are no known financial, commercial, or personal conflicts of interest that could have influenced the outcomes or interpretations presented in this study. All research activities were conducted independently and in accordance with academic integrity and ethical standards.

Code Availability

The source code used in this study is publicly available in the following GitHub repository: https://github.com/faridtriyadi/js-ga-route. The repository includes the complete JavaScript implementation of the genetic algorithm, sample datasets, and usage instructions for execution within a browser environment. Users are encouraged to explore, replicate, or extend the experiments for academic and research purposes under the repository's open-source license.

References

Akram, M., & Habib, A. (2023). Hybridizing simulated annealing and genetic algorithms with Pythagorean fuzzy uncertainty for traveling salesman problem optimization. *Journal of Applied Mathematics and Computing*, *69*(6), 4451–4497. https://doi.org/10.1007/s12190-023-01935-y

Dymora, P., & Paszkiewicz, A. (2020). Performance Analysis of Selected Programming Languages in the Context of Supporting Decision-Making Processes for Industry 4.0. *Applied Sciences*, *10*(23), 8521. https://doi.org/10.3390/app10238521

Ferreira, F., Borges, H. S., & Valente, M. T. (2022). On the (un-)adoption of JavaScript front-end frameworks. *Software: Practice and Experience*, *52*(4), 947–966. https://doi.org/10.1002/spe.3044

Odeniran, Q., Wimmer, H., & Du, J. (2024). Javascript frameworks—a comparative study between react. js and angular. js. In *Interdisciplinary Research in Technology and Management* (pp. 319–327). CRC Press.

Patel, S., & Tere, Dr. G. (2025). Analyzing Programming Language Trends Across Industries: Adoption Patterns and Future Directions. *International Journal of Emerging Science and Engineering*, *13*(2), 19–26. https://doi.org/10.35940/ijese.F3652.13020125

Patni, S., & Sharma, B. (2024). *Genetic Algorithms for Decision Optimization* (pp. 29–39). https://doi.org/10.4018/979-8-3693-2073-0.ch003

Prabhath, C. N., Kavitha, M., & Kalita, K. (2023). Efficiency analysis of path-finding algorithms in a 2D grid environment. *Journal of Autonomous Intelligence*, *7*(2). https://doi.org/10.32629/jai.v7i2.1284

Sulemana, A., Donkor, E. A., Forkuo, E. K., & Oduro-Kwarteng, S. (2019). Effect of optimal routing on travel distance, travel time and fuel consumption of waste collection trucks. *Management of Environmental Quality: An International Journal*, *30*(4), 803–832. https://doi.org/10.1108/MEQ-07-2018-0134