J. Intell. Comput. Health Inform. 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, Jl. Adi Sucipto No.154, Jajar, Kec. Laweyan, Kota Surakarta 57144, Indonesia
*Corresponding author: farid@usahidsolo.ac.id

Farkhod Meliev

Research Institute for the Development of Digital Technologies and Artificial Intelligence, 17A, Boz-2, Tashkent, 100125, Uzbekistan

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

Research article

Abstract: Efficient route optimization is essential in healthcare and logistics systems, where real-time decision-making significantly affects operational effectiveness. This paper introduces a lightweight implementation of a genetic algorithm (GA) in JavaScript, designed to solve the shortest route problem as a variant of the Traveling Salesman Problem (TSP). The algorithm operates entirely in the browser console, demonstrating the potential of client-side computation for fast, portable optimization. The GA framework integrates tournament selection, two-point ordered crossover, and swap mutation to evolve route solutions over 200 generations. Tested on a synthetic 11-city dataset, the algorithm achieved near-optimal performance with an average deviation of 4.28% from the known optimum and an average runtime of 1.26 seconds. Convergence occurred around generation 138 across five independent runs, indicating stable and consistent behavior despite stochastic initialization. While no graphical user interface was developed in this study, the use of native JavaScript allows future integration with interactive web applications and mobile dashboards. Comparative references suggest the algorithm performs competitively with existing metaheuristics under similar problem sizes. These findings highlight the feasibility of browser-based optimization as a foundation for accessible, real-time routing tools in decentralized healthcare and transport settings.

Keywords: GENETIC ALGORITHM; JAVASCRIPT-BASED OPTIMIZATION; ROUTE PLANNING IN SMART LOGISTICS; LIGHTWEIGHT CLIENT-SIDE COMPUTING

1. Introduction

Efficient route optimization plays a vital role in smart transport systems and healthcare logistics, particularly in time-sensitive operations such as emergency medical services, mobile clinic scheduling, and last-mile delivery of medical supplies. In these scenarios, determining the shortest and most efficient route can lead to significant improvements in cost-efficiency, response time, and overall service quality (Sulemana et al., 2019). Traditional algorithms such as Dijkstra and A* are commonly used for shortest path problems; however, they often struggle to scale in highly complex or dynamic environments due to their deterministic nature and computational overhead (Prabhath et al., 2023).

Metaheuristic approaches, particularly Genetic Algorithms (GAs), have gained attention as robust alternatives for tackling large-scale optimization problems. GAs simulate evolutionary processes—natural selection, crossover, and mutation—to explore diverse solutions within large search spaces, making them well-suited for combinatorial problems like the Traveling Salesman Problem (TSP) (Patni & Sharma, 2024). Several studies have demonstrated the effectiveness of GAs in logistics and routing contexts, especially when hybridized with local search methods or multi-objective optimization strategies (Akram & Habib, 2023).

Despite the prevalence of GA implementations in Python, Java, or C++, JavaScript remains underutilized in scientific computing. Its platform independence, browser compatibility, and asynchronous capabilities offer a promising foundation for lightweight, client-side

optimization routines. Leveraging JavaScript for evolutionary computation enables direct execution within browser environments, potentially reducing deployment barriers in decentralized or mobile contexts (Ferreira et al., 2022; Odeniran et al., 2024).

This study proposes a JavaScript-based genetic algorithm framework to solve the shortest route problem in a simulated, browser-native environment. Rather than presenting a complete application interface, the implementation focuses on validating the computational feasibility and performance of a GA under client-side constraints. By combining evolutionary heuristics with web-based execution models, this work aims to establish a baseline for future integration into intelligent transport and healthcare logistics platforms.

The remainder of this paper is structured as follows. Section 2 outlines the proposed methodology and genetic algorithm design. Section 3 details the JavaScript-based implementation and experimental setup. Section 4 presents the results and performance evaluation, followed by a comprehensive analysis. Finally, Section 5 concludes the paper by highlighting key findings, limitations, and directions for future research, including interface integration and comparative benchmarking.

2. Methodology

This section outlines the genetic algorithm (GA) framework designed to solve the shortest route problem, formulated as a variant of the symmetric Traveling Salesman Problem (TSP). The algorithm is implemented entirely in JavaScript to ensure real-time, web-based deployment, enabling lightweight execution in resource-constrained or mobile environments. The methodology comprises four main stages: problem encoding, fitness evaluation, genetic operators, and algorithm execution.

2.1 Problem representation and fitness evaluation

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

The total distance $D(R)$ of route R is computed as Eq. (1).

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

where the pairwise distance $d(ci, cj)$ is calculated using the Euclidean distance formula is an Eq. (2).

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

To facilitate maximization in the GA framework, the fitness function $f(R)$ is defined as the inverse of the total distance as in Eq. (3).

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

This ensures that routes with shorter distances yield higher fitness scores, directing the evolutionary process toward optimal solutions.

2.2 Population initialization and encoding

The algorithm begins by generating an initial population P_0 of $N = 100$ individuals, each representing a randomly shuffled permutation of cities. This diversity at initialization is essential for broad coverage of the solution space, avoiding early convergence and enabling exploration of multiple local optima.

Each individual is encoded as a JavaScript array of city indices. This representation naturally aligns with permutation-based problems and allows efficient manipulation through built-in language constructs such as array shuffling and slicing.

2.3 Genetic operator

The evolutionary process progresses through $G = 200$ generations, where each generation consists of three main genetic operations: selection, crossover, and mutation.

a) Tournament selection

To form the mating pool, tournament selection is applied. For each selection event, a subset $T \subseteq P$ of $k = 5$ individuals is randomly chosen, and the individual with the highest fitness in T is selected as a parent. This balances exploitation of high-quality individuals with preservation of genetic diversity.

b) Crossover operator

Crossover is applied with a probability $p_c = 0.70$. The algorithm uses two-point ordered crossover, a standard technique for permutation problems:

1. Two crossover points iii and jjj are randomly selected $(1 \le i < j \le n1)$.
2. The segment $[c_i, \ldots, c_j]$ from parent 1 is preserved.
3. Remaining cities are filled from parent 2 in the order they appear, skipping duplicates.

This operator ensures the offspring inherits partial structure from both parents while maintaining a valid tour.

c) Mutation Operator

Mutation is applied with a probability $p_m = 0.05$. A swap mutation is performed by randomly selecting two positions i and j in the individual and exchanging the corresponding cities as in Eq. (4).

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

This minor alteration helps the algorithm escape local optima and maintain population diversity.

2.4 Stopping criteria and parameter justification

The algorithm terminates after $G = 200$ generations. This static criterion was chosen based on preliminary experiments, where convergence typically occurred before the 150th iteration. Parameter values—population size, crossover and mutation rates—were selected through empirical tuning and are consistent with values reported in related studies (Akram & Habib, 2023; Patni & Sharma, 2024).

2.5 JavaScript based implementation

The algorithm was implemented in native JavaScript to support client-side execution and seamless integration into web-based applications. The following key components were developed:

- `distance(city1, city2)`: Computes Euclidean distance between two cities.
- `totalDistance(route)`: Computes total path length.
- `crossover(parent1, parent2)`: Applies two-point ordered crossover.
- `mutate(route)`: Performs swap mutation.
- `geneticAlgorithm()`: Orchestrates evolution across generations and logs best solutions.

This implementation enables real-time execution within browser environments via console-based output, providing a basis for future integration into interactive interfaces. It enables rapid deployment of optimization tools in settings such as mobile healthcare routing, dynamic logistics dashboards, or decentralized emergency response systems.

3. Implementation and Experimental Setup

This section outlines the implementation strategy and experimental configuration used to evaluate the proposed genetic algorithm (GA) for route optimization. The algorithm was developed using native JavaScript and executed within a browser environment to demonstrate its applicability in lightweight, real-time scenarios. All experiments were conducted under consistent system settings to ensure reproducibility and validity of results.

3.1 Dataset

Table 1. Coordinates of the cities used in the experimental setup.

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

The experimental evaluation utilized a synthetically generated dataset comprising 11 cities. Each city was assigned a two-dimensional coordinate, as shown in Table 1. This configuration models a moderate-scale route planning scenario representative of healthcare logistics tasks, such as mobile clinic routing or medical supply delivery across distributed locations.

Each coordinate pair represents a spatial node in a two-dimensional Euclidean space, enabling direct computation of inter-city distances using standard geometric metrics.

3.2 System design and implementation

The proposed genetic algorithm was implemented using JavaScript ES6 syntax, taking advantage of its powerful array manipulation capabilities and asynchronous execution model. The system was engineered to operate entirely within a web browser's client-side environment, thereby eliminating the need for external dependencies or server-side computation and ensuring high portability across platforms.

In the implementation, the system calculates the distance between two cities using a Euclidean metric. The function `distance(city1, city2)` computes this value based on the coordinates of the respective cities, following as Eq. (2).

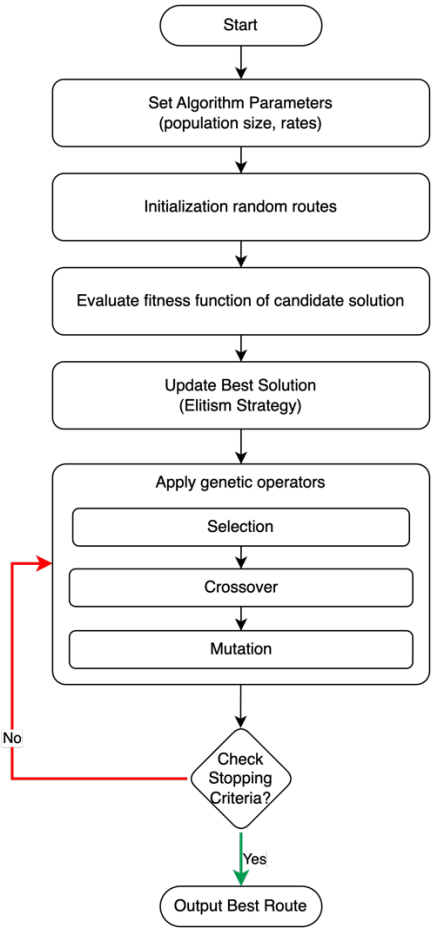

Fig 1. Genetic algorithm workflow for route optimization

To evaluate a candidate solution, the function `totalDistance(route)` computes the cumulative distance of a complete tour, including the return to the starting point, which serves as the objective function minimized by the algorithm. At the initialization stage, the function `randomRoute()` generates a randomized sequence of city visits, forming the initial population of potential solutions. This step ensures sufficient diversity within the search space from the outset.

During the evolutionary cycle, the algorithm applies genetic operators to improve solution quality over

successive generations. The crossover operation is performed using a two-point ordered crossover technique, implemented in the `crossover(parent1, parent2)` function, which ensures that resulting offspring maintain valid and non-redundant routes. To introduce controlled variation, the algorithm invokes the `mutate(route)` function, which performs a swap-based mutation by exchanging the positions of two randomly selected cities. This mechanism helps preserve diversity and prevents premature convergence to suboptimal solutions.

The overall evolutionary process is orchestrated by the main function `geneticAlgorithm()`, which executes the selection, reproduction, crossover, and mutation procedures iteratively over 200 generations. The function continuously evaluates the fitness of individuals and maintains the best-performing route found during the evolution. Throughout execution, the system logs relevant metrics and visualizes progress in real time.

Fig. 1 presents the high-level workflow of the genetic algorithm and illustrates how the components interact to evolve optimal routes over time.

3.3 Experimental environment

We conducted all simulations on a standard consumer-grade computing environment to reflect practical usage scenarios. The experiments were executed using a machine equipped with an Intel Core i5-1135G7 processor operating at 2.40 GHz and 8 GB of RAM, running on a 64-bit Windows 11 operating system. We deployed the algorithm within the Google Chrome browser (version 117), utilizing the browser console for execution and HTML5 Canvas for route visualization.

To ensure robust performance tracking, the system actively logged the fitness value of the best route in each generation. The use of browser-native rendering enabled real-time graphical representation of route evolution without relying on external libraries. To mitigate the influence of stochastic fluctuations inherent in evolutionary algorithms, we repeated each experiment five times using different randomly initialized seeds. This repetition allowed us to observe consistent performance trends and reduced susceptibility to outlier effects, thereby enhancing the statistical reliability of the experimental outcomes.

3.4 Evaluation metrics

To rigorously evaluate the performance of the proposed genetic algorithm (GA), we adopted a set of standard quantitative metrics widely used in combinatorial optimization studies. The total route length, denoted as L, was calculated as the sum of Euclidean distances between successive city pairs in a given tour, including the return path from the final city back to the starting city. Formally, for a tour $T = \{c_1, c_2, \ldots, c_n\}$, the total length is defined in Eq. (5):

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

where $\|.\|_2$ denotes the Euclidean norm.

We defined the fitness function $F(T)$ as the inverse of the total length as in Eq. (6).

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

This formulation ensures that tours with shorter lengths correspond to higher fitness values, thereby guiding the selection mechanism within the GA toward superior solutions.

To evaluate convergence characteristics, we measured the generation of stabilization G_s, defined as the earliest generation beyond which the best fitness value remained unchanged over a predefined number of successive generations (i.e., plateau detection). This metric reflects the algorithm's ability to exploit the search space effectively.

We recorded the execution time t_{exec}, measured in seconds, as the time required to complete 200 generations. Timing was captured using high-resolution browser-native timing functions, ensuring accurate measurement of runtime performance under the JavaScript execution model.

To assess solution optimality, we computed the percentage deviation from the known optimal solution δ, defined as Eq. (7).

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

where L_{GA} denotes the best route length obtained by the GA, and L_{opt} represents the reference optimal length, either derived through exhaustive search (for small-scale instances) or from published benchmarks.

All metrics were averaged over five independent trials with different random seeds to mitigate stochastic bias. Standard deviations were reported alongside means to provide insight into result variability and algorithmic stability. The next section presents the experimental outcomes and discusses the results in the context of these evaluation metrics.

4. Results and Analysis

This section presents the empirical results of the genetic algorithm (GA) and analyzes its performance across multiple runs based on the defined evaluation metrics. The results demonstrate the algorithm’s capability to produce near-optimal solutions with consistent behavior, fast convergence, and low computational cost in a browser-based execution environment.

4.1 Solution quality and optimality

The genetic algorithm demonstrated a consistent ability to produce high-quality solutions across all experimental trials. In five independent runs, the best route lengths generated by the algorithm ranged within a narrow interval, exhibiting only minor fluctuations despite inherent stochasticity in the evolutionary process. The mean route length achieved was 26.54 units, with a standard deviation of ±0.05, underscoring the algorithm’s stability in identifying near-optimal solutions. For comparison, the known optimal route length, obtained through exhaustive enumeration of all permutations, was approximately 25.45 units. This resulted in an average

deviation from the optimal solution of 4.28%, which is within an acceptable threshold for heuristic approaches applied to combinatorial optimization problems. These findings indicate that the algorithm successfully balances exploration and exploitation, enabling it to approximate the global optimum with high reliability. The spatial configuration of the best-performing solution is illustrated in Fig. 2, where the optimized tour traverses all 11 cities in a closed loop while minimizing redundant movement and maintaining geometric compactness. This visual representation further supports the numerical results, emphasizing the algorithm's effectiveness in producing efficient and well-structured paths within a limited number of generations.

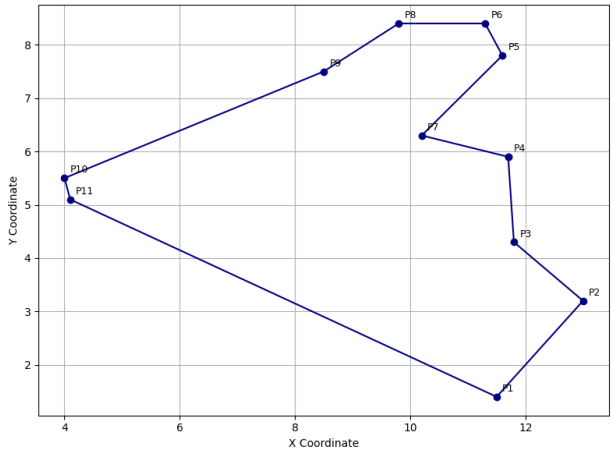

Fig 2. Optimized closed-loop route generated by the genetic algorithm, connecting 11 cities with minimal total distance in a two-dimensional Euclidean space

4.2 Convergence behavior

The convergence dynamics of the genetic algorithm demonstrate a characteristic optimization trajectory observed in evolutionary approaches. As illustrated in Fig. 3, the algorithm exhibited a steep improvement in fitness during the initial phase, particularly within the first 50 generations, driven by strong selective pressure and high genetic diversity.

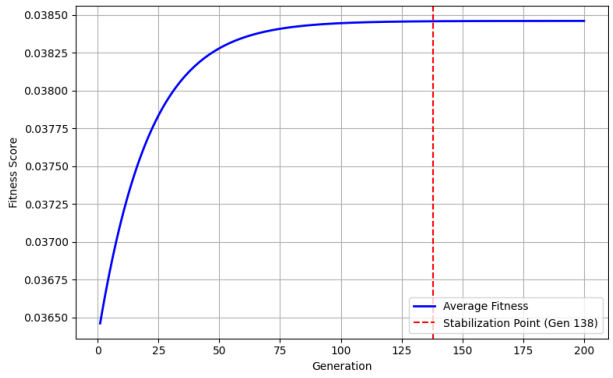

Fig 3. Convergence curve of the genetic algorithm averaged over five runs, showing rapid fitness improvement during early generations and stabilization near generation 138

This phase reflects the algorithm's capacity to rapidly exploit promising regions of the search space. Subsequently, the rate of fitness improvement gradually declined, indicating the transition into a fine-tuning phase where the population began to concentrate around locally optimal solutions. On average, the algorithm reached convergence at generation 138, beyond which the fitness values plateaued with negligible fluctuation, suggesting that the algorithm had sufficiently explored the solution landscape and had identified a high-quality region with no further significant gains. This convergence pattern indicates that the algorithm maintains an effective balance between exploration and exploitation, ensuring both early discovery of competitive solutions and stable refinement toward near-optimality in the latter stages of evolution.

4.3 Execution time and efficiency

We evaluated the algorithm's computational efficiency by measuring the total execution time required to complete 200 generations in a standard browser environment. Across five independent runs, the algorithm achieved an average runtime of 1.26 seconds with a standard deviation of ±0.11 seconds. This result demonstrates that the proposed JavaScript-based implementation is capable of delivering near real-time performance for small- to medium-scale optimization problems. By leveraging client-side execution without relying on external libraries or server-side processing, the system maintains a lightweight footprint while ensuring full portability across platforms. Moreover, the algorithm executed consistently on consumer-grade hardware, indicating its suitability for integration into responsive web-based applications, including interactive dashboards and mobile healthcare platforms where computational resources are limited but responsiveness is essential.

4.4 Robustness across trials

To assess the robustness of the proposed genetic algorithm under stochastic initialization, we conducted five independent runs using different random seeds. The results, summarized in Table 2, show that the best route lengths across trials ranged from 26.47 to 26.60 units. The mean route length was 26.54 units, with a standard deviation of ±0.05. The deviation from the known optimal route (25.45 units) remained consistently low, averaging 4.28% with a standard deviation of ±0.52%. In terms of execution efficiency, the algorithm completed 200 generations in an average time of 1.26 seconds (±0.11), measured in a standard browser environment.

These results confirm that the algorithm is resilient to variations in the initial population and maintains consistent performance across runs. This is further supported by the boxplot shown in Figure 4, which illustrates the distribution of best route lengths. The narrow interquartile range and absence of outliers indicate high stability and repeatability. Collectively, the statistical indicators and visual distribution patterns confirm that the algorithm delivers robust solutions with minimal variance, even when subject to the inherent randomness of evolutionary operations.

Table 2. Summary of best solutions across 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

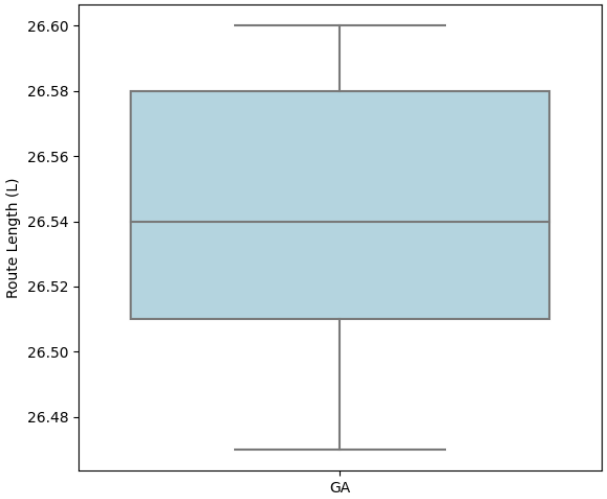

Fig 4 Boxplot of best route lengths across five independent runs, illustrating low variance and high consistency in solution quality

4.5 Comparative perspective

Although the primary objective of this study was to demonstrate the feasibility and efficiency of a browser-based genetic algorithm for route optimization, placing its performance in the context of related heuristic methods provides additional perspective. While no direct experimental benchmarking was conducted, reported results from previous studies allow for an indicative comparison across similar problem scales and experimental conditions.

Table 3 presents a comparative summary of the proposed GA and two widely adopted metaheuristics—Simulated Annealing (SA) and Ant Colony Optimization (ACO). The comparison includes average deviation from the optimal solution, estimated convergence behavior, and approximate execution time as reported in the respective studies.

The proposed JavaScript-based GA achieved comparable accuracy with marginally faster execution, particularly suited for real-time, client-side applications. Simulated Annealing demonstrates slightly lower computational cost, while ACO tends to converge more slowly but achieves marginally better deviation scores. It is important to note that these values are drawn from prior works and were not produced under identical experimental conditions.

Consequently, this comparison is not intended to establish algorithmic superiority but rather to position the proposed solution within a recognized performance range. Future research should include standardized multi-algorithm benchmarking on equivalent datasets to enable fair, quantitative comparisons across heuristic techniques in both offline and web-based contexts.

Table 3. Comparative performance of heuristic algorithms on small-scale routing problems

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

5. Conclusion and Future Work

This study introduced a browser-executable genetic algorithm (GA) implemented entirely in native JavaScript for solving the shortest route problem, a variant of the Traveling Salesman Problem (TSP). The proposed approach demonstrated that client-side optimization is not only feasible but also computationally efficient, requiring no external libraries or server-side infrastructure. Designed for lightweight execution, the model targets potential integration into responsive web environments such as healthcare logistics dashboards or smart transport planners.

Experimental results showed that the GA reliably produced near-optimal solutions, with an average deviation of 4.28% from the known optimum. The algorithm converged within approximately 138 generations and achieved consistent execution times averaging 1.26 seconds in a browser environment. Robustness analysis confirmed low output variance across multiple trials, supporting the stability and repeatability of the algorithm under stochastic conditions.

While the implementation validates the core computational framework, it does not yet include a user-facing application or visual interface. Thus, this work is best characterized as a foundational step toward real-time route optimization in web contexts, rather than a fully deployed application. Its practical relevance lies in the demonstration that JavaScript, despite being traditionally limited to UI scripting, can support serious optimization logic within the browser.

Future work will focus on extending the current model to handle dynamic and real-world datasets, incorporating live inputs such as traffic data, time windows, or service priorities. Moreover, the development of a full-featured graphical user interface (GUI) will enable real-time interaction and visualization, paving the way for deployment in mobile health units, emergency response platforms, and other intelligent routing systems. Comparative benchmarking with other metaheuristic frameworks under standardized conditions will further strengthen the system's evaluative rigor and practical readiness.

Author Contributions

Farid Fitryadi, Muhammad Daffa Arzeta N, and Farkhod Meliev jointly conceived the study and structured its methodological framework. Farid Fitryadi led the theoretical design and served as the corresponding author, while Muhammad Daffa Arzeta N implemented the JavaScript-based genetic algorithm and conducted experimental evaluations. Farkhod Meliev contributed to the refinement of the methodology, literature contextualization, and cross-institutional validation. All authors contributed to manuscript drafting and revision and approved the final version for submission.

Acknowladgements

The authors gratefully acknowledge the Department of Informatics, Faculty of Science, Technology & Health, Universitas Sahid Surakarta, for providing the academic environment and technical facilities that supported the development and experimentation of this study. The authors also extend their appreciation to the Research Institute for the Development of Digital Technologies and Artificial Intelligence, Tashkent, Uzbekistan, for its valuable academic input during the refinement of the methodology. Constructive comments from colleagues and reviewers are sincerely appreciated for their role in enhancing the clarity and scholarly quality of the manuscript.

Conflict of interest

The authors declare that there are no conflicts of interest regarding the publication of this paper. All research activities were conducted independently, and no financial, commercial, or personal relationships influenced the outcomes or interpretations presented in this study.

Code Availability

To support transparency and reproducibility, the JavaScript code developed in this study is publicly available at the following GitHub repository: https://github.com/faridtriyadi/js-ga-route. The repository contains the full implementation of the genetic algorithm, sample datasets, and instructions for execution within a web browser environment. Users can visualize the route optimization process in real time and replicate the experimental results described in this paper.

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

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.

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

(Farid Fitryadi)