Genetic Algorithm-Based Optimization of Two Transfer Points (2TP) Problem in Random Graph

Abstract

This paper addresses the Two Transfer Point (2TP) problem, an important optimization challenge in vehicle routing and ridesharing scenarios. The objective is to identify optimal meeting and drop-off points in a graph to minimize the total travel distance for two users with distinct origin-destination pairs. We propose a Fitness Caching Genetic Algorithm (GA)-based approach that utilizes binary encoding, fitness evaluation based on difference of total travel distance, and evolutionary operations including tournament selection, crossover, and mutation. The algorithm is further enhanced with a fitness caching mechanism to improve efficiency. We conduct two sets of comparative experiments with an exact algorithm: the first under fixed generation budgets, and the second under equal time constraints, where the GA is only allowed the same execution time τ as the exact method. Results on complete random graphs show that the GA yields near-optimal solutions for small instances and acceptable approximations for larger graphs, even under limited time. Additionally, we evaluate the GA's performance on a real-world road network derived from OpenStreetMap data, demonstrating its practical applicability. In this context, the GA highlights its potential for scalable and time-efficient deployment. Our findings emphasize the trade-off between runtime and solution quality and validate the GA as a robust heuristic for the 2TP problem across both synthetic and realistic graph structures.

CCS Concepts

• **Theory of computation** → **Shortest paths**; **Random search heuristics**.

Keywords

Two Transfer Point Problem, Vehicle routing, Ridesharing optimization, Genetic algorithm, Heuristic methods, Travel distance minimization

ACM Reference Format:

. 2025. Genetic Algorithm-Based Optimization of Two Transfer Points (2TP) Problem in Random Graph. In *Proceedings of The Genetic and Evolutionary Computation Conference 2025 (GECCO '25).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In this article, we start by defining the Two Transfer Point (2TP) problem, a routing optimization challenge in which two users with distinct start and destination points seek to minimize their combined travel distance by determining the optimal meeting and dropping points. We formalize the problem mathematically and provide an exact algorithm with quadratic time complexity to solve it as well as a genetic algorithm (GA) to find near-optimal solutions.

However, as the problem scales with larger graphs and more dynamic constraints, exact solutions become computationally prohibitive. To address this, we propose a Fitness Caching Genetic Algorithm (GA) that offers an efficient heuristic for solving the 2TP problem. To the knowledge of the authors, this is the first work to formally define and address the 2TP problem—minimizing the combined travel distance for exactly two users by optimizing both a meeting and a drop-off point—within a graph-based framework. This formulation distinguishes itself from general shared mobility problems by explicitly modeling transfer point optimization for two users, supported by both an exact algorithm and a GA. The GA is not only capable of producing near-optimal solutions, but is also empirically shown to scale significantly better than the exact approach on large graphs. Furthermore, the proposed dual-method strategy allows for a comprehensive evaluation: the exact algorithm serves as a benchmark for solution quality, while the GA demonstrates practical viability for large instances where exact computation is infeasible, or very slow. This comparative analysis between solution quality and computational efficiency, tailored specifically to the 2TP problem, highlights the novelty and relevance of our approach in the context of scalable shared mobility planning.

This problem is part of the broader context of Shared Mobility Problems (SMP) [9, 17], which include carpooling, cab sharing and buspooling [9, 17], and have become a major trend in many urban cities around the world [17]. Trip sharing, as a problem 2TP, is intrinsically linked to vehicle routing and allocation challenges [9, 19], often formulated as variants of Vehicle Routing Problems (VRP) [4, 17–19], especially in dynamic and stochastic contexts [2, 4, 9, 14, 15, 17, 19]. The sources point out that ride-sharing systems are similar to other on-demand transportation services such as cabs and dial-a-ride services [2, 17]. Research in this area has explored a variety of resolution methods [2, 17], ranging from exact algorithms [2, 17], such as mixed integer programming or branch and bound [2, 4, 17], effective for small instances [2, 17] but limited due to non-polynomial complexity and scalability issues for larger problems [2, 17, 19], to heuristics and metaheuristics [2, 15, 17]. Among the latter, genetic algorithms (GA) are frequently used for their ability to quickly find good, albeit approximate, solutions for large-scale problems and in dynamic environments [2, 4, 19]. Techniques such as **clustering** [2, 9, 19], which involves grouping requests or journeys according to their spatio-temporal compatibility (including on the basis of shortest paths) [2, 19], are also used to improve the efficiency and scalability of solutions [2, 9, 19]. The evaluation of these approaches is often carried out using real data from large cities [2, 4, 9, 14, 19], simulations [2, 9, 14], or by

GECCO '25, Málaga, Spain

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

comparing the results of heuristics with those of exact algorithms on small instances [2, 4].

The 2TP problem has diverse applications beyond transportation. For example, in collaborative robotics, coordinating meeting and transfer points for robots can optimize task efficiency and energy consumption. In supply chain management, it can streamline the transfer of goods between warehouses to reduce costs. Additionally, in disaster response, identifying optimal meeting and dropping points for rescue teams and supplies can save critical time.

The remainder of this paper provides a formal problem definition, introduces an exact algorithm alongside a Fitness Caching GA, assesses the GA's error relative to the exact solution, and evaluates the performance of both approaches both theoretically and empirically. The evaluation is conducted on randomly generated complete graphs as well as on small real-world road networks.

2 Two Transfer Point (2TP) Problem

Given a graph $G(V, E)$ and two paths $a - c$ and $b - d$ sharing a common segment $m - k$, the goal is to find the meeting node m and the drop-off node k, such that the pair of nodes m, k minimizes the sum of the parameters of the edges $e \in E$ of all the shortest paths SP included in this scenario. Figure 1 illustrates this problem graphically, and we can formalize this problem as in Definition 2.1.

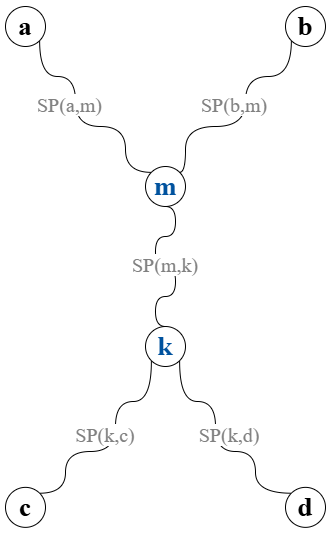


Figure 1: Two Transfer Point (2TP) problem in general graph.

Definition 2.1 (2TP Problem)**.** Given a graph $G(V, E)$, two paths $a - c$ and $b - d$ sharing a common segment $m - k$, with $a, b, c, d \in V$, $SP(u, v)$ the shortest path between $u, v \in V$ and $\tau(u, v)$ the sum of the parameters of the edges $e \in SP(u, v)$, is there exists a pair $m, k \in V$ such that:

$$(m, k) = \min_{m,k}(\tau(a, m) + \tau(m, k) + \tau(k, c) + \tau(b, m) + \tau(m, k) + \tau(k, d))$$

In a real-world context, the sum of the edge parameters will generally be the cumulative travel time for each edge or the total travel distance, in the remainder of this article, we will use the distance parameter.

The pseudo-code (Algorithm 1) gives an exact solution for solving the 2TP problem in a general graph with an approximate worst-case time complexity of $O(n^2)$ where $n = |V|$ (cf. Section 4.3). This algorithm begins by precomputing and caching all-pairs shortest path distances using Dijkstra's algorithm [5]. It then iterates over all possible pairs of meeting node m and dropping node k in the graph. For each pair, it retrieves the relevant shortest path distances from the cache to compute the total travel distance for both users. The algorithm keeps track of the minimum total distance, and at the end, it returns the optimal pair of nodes that minimizes this distance.

Algorithm 1 Exact Algorithm for the Two Transfer Point Problem with Precomputed Distances

Require: Graph G with $|V|$ nodes, sources nodes a, b, targets nodes c, d
Ensure: Optimal meeting and dropping points

```
Precompute and cache all-pairs shortest path distances using Dijkstra for all node pairs in G
Store the distances in a dictionary-like structure: dist[u][v] = shortest path distance from u to v
Initialize min_sum ← ∞
for each node m ∈ G do
    for each node k ∈ G, where k ≠ m do
        sum_m ← dist[a][m] + dist[b][m]
        sum_k ← dist[m][k]
        sum_e ← dist[k][c] + dist[k][d]
        sum ← sum_m + sum_k + sum_e
        if sum ⩽ min_sum then
            min_sum ← sum
        end if
    end for
end for
return min_sum
```

We hypothesize that the 2TP problem is 3SUM-hard, implying that there is no sub-quadratic algorithm for this problem. An algorithm is *sub-quadratic* if there exists an $\epsilon > 0$ such that the algorithm runs in time $O(n^{2-\epsilon})$. The 3SUM problem involves determining whether there exist three numbers within a given set that sum to zero. We can formalize this problem as in the Definition 2.2.

Definition 2.2 (3SUM Problem)**.** Given a set S of n integers, does there exist $x, y, z \in S$ such that $x + y + z = 0$?

The 3SUM problem can be solved using a $O(n^2)$ algorithm[1] and it is generally accepted that $O(n^2)$ is the worst case lower bound as expressed in Conjecture 2.1.

Conjecture 2.1 (3SUM conjecture)**.** *There is no sub-quadratic algorithm for the 3SUM problem [7].*

However, this conjecture has been refuted in the meantime by proving better lower bound [6, 11, 12]. While a formal reduction from 3SUM to 2TP, or vice versa, is not presented in this work, we hypothesize that the 2TP problem in general graphs is 3SUM-HARD.

[1] By first sorting the S set, then testing each 3-tuples

However, this complexity may not extend to road graphs. A rigorous investigation of these reductions is left for future work. This intuition arises from the understanding that road networks deviate from the structure of classical graphs, exhibiting distinct characteristics such as Highway Dimension [1]. Consequently, specialized techniques like Hierarchical Contraction [8] can be effectively applied to optimize their analysis and representation. In the following, our focus will be directed towards classical graphs, with particular emphasis on complete graphs, due to their high connectivity, which inherently results in higher complexity.

3 Fitness Caching Genetic Algorithm

GAs are a class of optimization techniques inspired by the principles of natural selection [10]. GAs operate by evolving a population of candidate solutions through selection, crossover, and mutation processes, guided by a fitness function. These algorithms are particularly effective for solving complex optimization problems where conventional methods may be computationally impractical or inefficient [16].

In the context of 2TP problem, we employ a Fitness Caching GA to optimize the transfer points, aiming to minimize the total travel distance for both users. As before, the 2TP problem is modeled using a complete random graph $G(V, E)$ with $|V| = N$, in our experiments the number of nodes N is chosen from $\{10, 25, 50, 100\}$. The pseudo-code (Algorithm 2) provides the details of the steps in the GA process. Two users are assigned random start and destination nodes, ensuring distinct locations. The algorithm iteratively evolves candidate solutions to identify the pair of nodes, m and $k \in V$, that results in the most efficient routing for the users. Solutions are encoded as binary strings, representing the indices of the meeting and dropping points, $m, k \in V$, details are given in the Section 3.1.

Fitness is computed as the negative total travel distance for both users, ensuring minimization by maximizing fitness values. The algorithm employs tournament selection for parent selection, bitwise crossover for offspring generation, and bit-flip mutation with a 10% probability to maintain diversity. The population evolves over a predefined number of generations T, with the best-performing individuals retained, this elitist strategy prevents the loss of promising candidates and accelerates convergence toward the (near) optimal solution.

It is true that the number of possible (m, k) pairs in this problem is limited to $|V|^2$, and for small values of $|V|$, enumerating all solutions may appear tractable. However, our GA implementation leverages a *fitness caching mechanism* that stores the evaluation results of previously seen individuals, thereby avoiding redundant computation and enabling the algorithm to focus resources on exploring new regions of the solution space. In contrast to brute-force enumeration—which exhaustively and indiscriminately evaluates all combinations—the GA efficiently guides the search using evolutionary pressure toward more promising regions of the search space. Moreover, enumeration lacks the adaptive sampling and probabilistic exploration mechanisms that allow GAs to generalize across different problem settings and scale more naturally to larger instances. As a result, even when the number of evaluations is on par with the number of total possible solutions, the GA often discovers better solutions faster and with fewer redundant checks.

Algorithm 2 Genetic Algorithm with Fitness Caching for the Two Transfer Point Problem

Require: Graph G with $|V|$ nodes, source nodes a, b, target nodes c, d, population size P, number of generations T
Ensure: Optimal meeting and dropping points

```
Initialize population P_0 with random binary-encoded solutions
fitness_cache ← ∅
best_fitness ← −∞
best_solution ← ∅
function EvaluateWithCache(individual)
    if individual ∈ fitness_cache then
        return fitness_cache[individual]
    else
        fitness ← ComputeFitness(individual)
        fitness_cache[individual] ← fitness
        return fitness
    end if
end function
for t ← 1 to T do
    for each individual in population do
        individual.fitness ← EvaluateWithCache(individual)
    end for
    Select parents using tournament selection
    Generate offspring using crossover and mutation
    for each child in offspring do
        child.fitness ← EvaluateWithCache(child)
    end for
    Merge offspring into the population
    Retain the top P individuals based on fitness
    if fitness of best individual in population > best_fitness then
        Update best_fitness and best_solution
    end if
end for
return best_solution with corresponding best_fitness
```

3.1 Encoding

Solutions in the genetic algorithm are encoded as fixed-length binary strings that represent the decision variables of the 2TP Problem, specifically the indices of the meeting and dropping points, $m, k \in V$. Each of these points is encoded using a fixed number of bits, typically one byte, resulting in a total individual length of 16 bits. This design constrains the maximum number of nodes to 256 (2^8), which is sufficient for our experiments; however, the encoding length can be increased to support larger networks.

The structure of the binary string is divided evenly: the first half encodes the meeting point and the second half encodes the dropping point. These binary segments are interpreted as unsigned integers and mapped to valid node indices using the modulo operation, ensuring all decoded values correspond to existing nodes in the graph regardless of the raw binary value. This encoding strategy allows efficient manipulation and crossover operations at the bit level. A simple example of this encoding scheme is illustrated in

Figure 2, where each individual represents a unique pair of terminal points in the network.

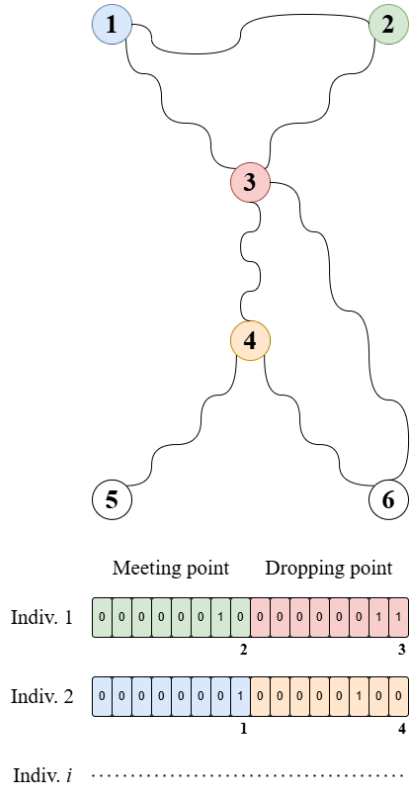


Figure 2: Two Transfer Point (2TP) individuals encoding.

3.2 Selection

The genetic algorithm employed in our experiments uses *tournament selection* to choose parents for crossover. This method involves randomly selecting a subset of individuals from the current population and choosing the one with the highest fitness among them to act as a parent. In the implementation, the tournament size k is defined as $\max(2, \lfloor \frac{P}{10} \rfloor)$, where P is the population size. This dynamic sizing ensures a reasonable balance between selection pressure and population diversity, adapting to small or large populations alike.

Tournament selection is particularly suited for problems like the 2TP problem, where the fitness landscape may contain numerous local optima and sharp discontinuities. By selecting the best individual from a random subset rather than directly based on proportional fitness, tournament selection reduces the risk of premature convergence caused by overly dominant individuals early in the search. This approach introduces a moderate degree of stochasticity, maintaining diversity in the gene pool while still favoring individuals with higher fitness. In contrast to more deterministic or proportional methods, tournament selection does not require fitness normalization and is less sensitive to the scale of the fitness values. This makes it robust in scenarios where raw fitness differences may be large or unevenly distributed across the population.

Alternative selection mechanisms such as roulette wheel selection and rank-based selection also exist, each with different trade-offs. Roulette wheel selection assigns selection probabilities directly based on relative fitness values, favoring individuals with high fitness but risking early loss of genetic diversity. Rank-based selection, on the other hand, assigns probabilities based on the ranking of individuals rather than their absolute fitness, providing smoother control over selection pressure. Although this implementation does not explicitly name elitism, an elitist strategy is effectively incorporated by preserving the top P individuals after each generation. Following the generation of a new offspring via crossover and mutation, the population is augmented and then truncated by sorting all individuals by fitness and retaining only the best-performing subset. This ensures that the fittest solutions persist across generations, guiding the evolutionary process steadily toward optimal or near-optimal regions of the search space.

3.3 Fitness

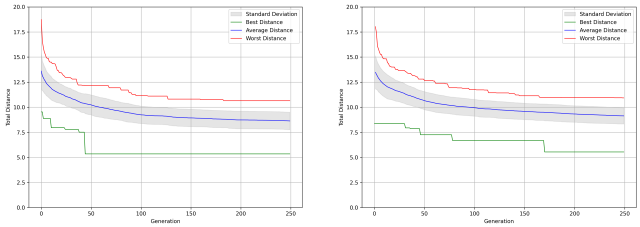
The fitness function evaluates the quality of solutions in the GA by calculating the total travel distance for a pair of meeting and dropping points. Invalid solutions, where the meeting and dropping points are identical, are penalized with $-\infty$. For valid solutions, the fitness is the negated sum of shortest path distances, using Dijkstra, for users traveling to the meeting point $(a - m, b - m)$, between meeting and dropping points $(m - k)$, and from the dropping point to their destinations $(k - c, k - d)$. This ensures the algorithm maximizes fitness while minimizing total travel distance.

4 Results

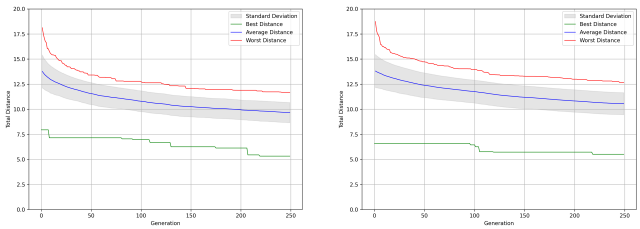
In this section, we present the results of our experiments evaluating the performance of the GA for solving the 2TP problem. We demonstrate that the population converges to a near-optimal solution over successive generations, highlighting the algorithm's effectiveness in minimizing the total travel distance for the users. The results also include a practical example using a real-world road network, illustrating the GA's applicability to real-world scenarios. Additionally, we compare the solutions obtained by the GA (Algorithm 2) with the solutions obtained by the exact algorithm (Algorithm 1) on graphs with $|V| \in \{10, 25, 50, 100\}$, where each edge weight is a floating-point distance δ randomly drawn from the interval $[1, 10]$. In both experiments we repeated 50 random experiments with a fixed seed to enable reproducibility, the code and results are available on a [GitHub repository]. The complete random graph structure ensures all possible routes and transfer points are represented, enabling a comprehensive evaluation of the algorithm while also reflecting the worst-case runtime scenario, as real-world road networks are typically not fully connected. In the first set of experiments of the errors, the GA operates with a fixed generation budget, while in the second, it is allocated the same time budget as the exact algorithm. Finally, we analyze the running time of both algorithms and provide insights into 10 instances of real world scenarios where the GA may be preferred over the exact solution, particularly in larger or more complex instances.

4.1 Convergence

In this section, we evaluate the performance of each GA instance using the 50 instance of complete random graphs $G(V, E)$, where $|V| = 100$. The GA instances are tested with population sizes $P \in \{5, 10, 20, 50\}$ and number of generations $T \in \{250, 500\}$. The corresponding results are presented in Figures 3, and 4. In the following results, it is important to note that the total distance difference between two users cannot be zero, as they are required to meet at a common intersection (node m) and subsequently separate at a different intersection (node k).



(a) GA with $P = 5$, $T = 250$ **(b) GA with $P = 10$, $T = 250$**

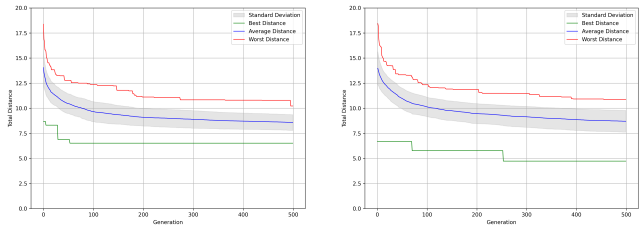


(c) GA with $P = 20$, $T = 250$ **(d) GA with $P = 50$, $T = 250$**

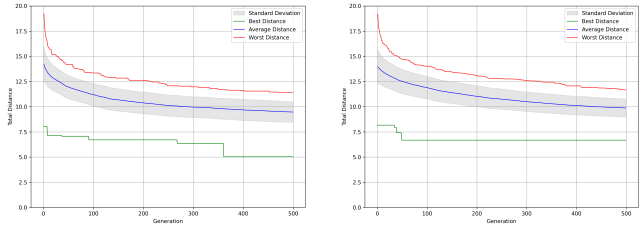
Figure 3: Evolution of the average fitness function over 50 instances of complete random graphs with $|V| = 100$ nodes for two GA configurations with $T = 250$.

Figure 3 presents a comparative analysis of various population sizes over the course of 250 generations. The results, averaged over 50 independent runs, indicate that smaller population sizes tend to converge more rapidly toward an optimal or near-optimal solution. Specifically, after 250 generations, the population with size $P = 5$ achieves an average distance difference of ≈ 7.75, whereas the population with size $P = 50$ attains an average distance difference of ≈ 11. Moreover, the optimal solution is identified as early as generation ≈ 45 for $P = 5$, while it remains undiscovered within the 250-generation limit for $P = 50$.

Figure 4 illustrates a comparative analysis of different population sizes over 500 generations. Consistent with the previous observations, and despite the doubled computational budget, smaller population sizes tend to converge more rapidly, on average (across 50 independent instances), toward optimal or near-optimal solutions. However, this advantage is counterbalanced by an increased risk of premature convergence, which can lead to sub-optimal outcomes. For instance, the best solution obtained by the population with size $P = 5$ (Figure 4a) remains stagnated at a best distance difference of 6.52. In contrast, the population with size $P = 10$ (Figure 4b) is able to further improve, reaching a solution with a distance difference of 4.73.



(a) GA with $P = 5$, $T = 500$ **(b) GA with $P = 10$, $T = 500$**



(c) GA with $P = 20$, $T = 500$ **(d) GA with $P = 50$, $T = 500$**

Figure 4: Evolution of the average fitness function over 50 instances of complete random graphs with $|V| = 100$ nodes for two GA configurations with $T = 500$.

The slower convergence occurs because larger populations provide greater genetic diversity, which increases the search space and delays premature convergence. While this delay can result in slower progression towards the optimal solution, it also reduces the likelihood of becoming trapped in suboptimal solutions, highlighting a trade-off between exploration and convergence speed.

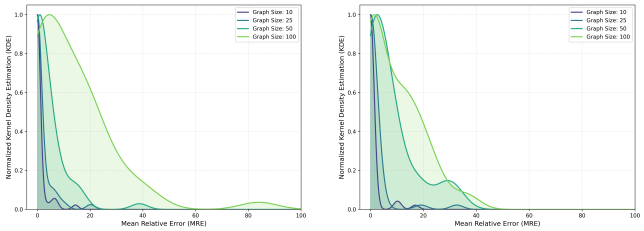
4.2 Mean Relative Error

In this section, we evaluate the solutions obtained by the GA (Algorithm 2) with different size of population to the solutions obtained by the exact algorithm (Algorithm 1) on the same 50 random instance of complete random graphs with $|V| \in \{10, 25, 50, 100\}$, first with a fixed budget of generation for the GA and then with an execution time budget equal to the one required by the exact algorithm.

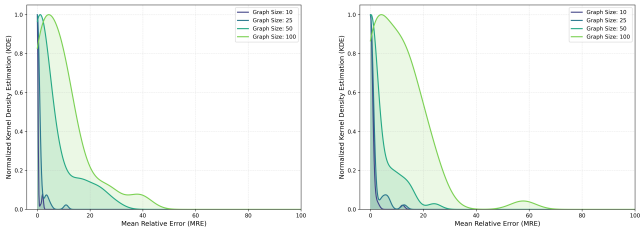
The Mean Relative Error (MRE) is calculated by first determining the relative error for each instance. This is defined as the percentage difference between the distance obtained by the GA and the exact solution, normalized by the exact solution's distance. The MRE is then computed as the arithmetic mean of these relative errors across all evaluated instances, providing a measure of the GA's average deviation from the optimal solution. In the following experiment, we used the *normalized Kernel Density Estimation (KDE)* to visualize the distribution of MRE values for different parameter configurations. KDE is a non-parametric method for estimating the probability density function of a random variable, which provides a smooth approximation of the underlying data distribution. By applying KDE to the MRE values and normalizing the resulting densities, we ensure consistent scaling across configurations, enabling clear visual comparisons. Curves are plotted over a common axis to reveal shifts in error distributions, where tighter, left-shifted

peaks indicate better performance and broader, right-shifted curves signify higher relative error.

4.2.1 MRE when fixed generation budget. In this experiment, each population size was given the same generation budget, namely $T = 500$, and was evaluated on the same 50 randomly generated instances of complete graphs.



(a) GA with $P = 5$, $T = 500$ **(b) GA with $P = 10$, $T = 500$**



(c) GA with $P = 20$, $T = 500$ **(d) GA with $P = 50$, $T = 500$**

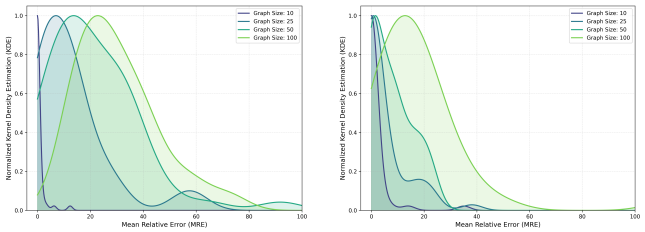
Figure 5: Normalized Kernel Density Estimation (KDE) of the Mean Relative Error (% MRE) for GA with fixed number of generations $T = 250$ and variable population size P over 50 instances of complete random graphs.

The first clear observation from Figure 5 is that, for a fixed population size P, the MRE tends to increase with the size of the graph, i.e., with the number of nodes and edges. This behavior is expected, as larger graphs imply a larger search space, making the optimization problem more challenging.

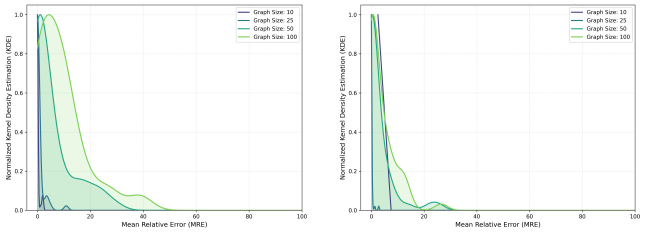
We also observe that for graph sizes $|V| \in \{10, 25, 100\}$, the population size $P = 20$ achieves the best average performance in terms of MRE, as shown in Figure 5c. In contrast, for graphs of size $|V| = 50$, the population size $P = 50$ yields the lowest average MRE, as illustrated in Figure 5d.

Given that the population size $P = 20$ achieves the lowest average Mean Relative Error (MRE), we proceed to evaluate the impact of the number of generations T on performance, using the same set of randomly generated graph instances.

Figure 6 demonstrates that, for a fixed population size and across all tested graph sizes, increasing the number of generations T systematically leads to a lower average MRE. This trend reflects the fact that a greater number of generations allows the algorithm more opportunity to converge toward the optimal solution. These findings are consistent with the behavior of the fitness function observed in Section 4.1. Furthermore, since the same set of random graph instances was used in both experiments, the results presented in Figure 5c and Figure 6c are identical.



(a) GA with $P = 20$, $T = 100$ **(b) GA with $P = 20$, $T = 250$**



(c) GA with $P = 20$, $T = 500$ **(d) GA with $P = 20$, $T = 1000$**

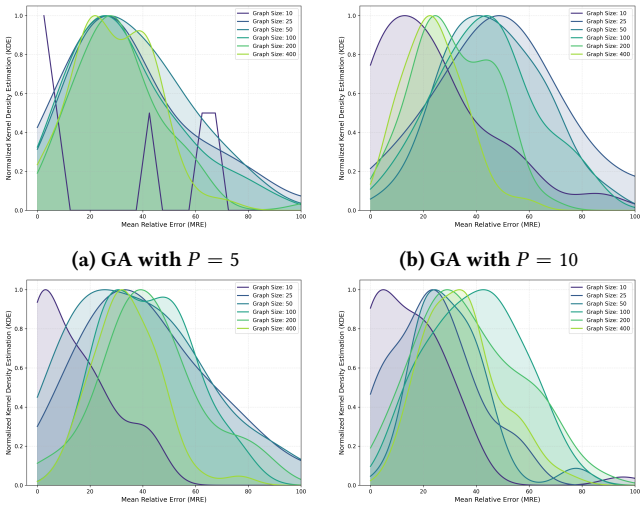
Figure 6: Normalized Kernel Density Estimation (KDE) of the Mean Relative Error (% MRE) for GA with fixed population size $P = 20$ and variable number of generations over 50 instances of complete random graphs.

4.2.2 MRE when τ generation budgets. Following the methodology of the previous experiment, we evaluate the performance of the GA (Algorithm 2) with varying population sizes by comparing its solutions to those obtained by the exact algorithm (Algorithm 1) across the same set of 50 randomly generated complete graphs, with vertex counts $|V| \in \{10, 25, 50, 100, 200, 400\}$. In contrast to the prior setup, the GA is now allotted a runtime τ equal to the execution time required by the exact algorithm for each instance. Consequently, for smaller graph sizes (where the exact algorithm requires less computational time), the GA is provided with proportionally less time, potentially limiting its ability to achieve convergence. Table 1 reports the runtime τ required by the exact algorithm (EA) for each tested size $|V|$ of the randomly generated complete graphs.

Table 1: Average running time of the the Exact Algorithm (EA) on the complete random graphs

Graph Size $\|V\|$	EA (τ)
10	0.001s
25	0.003s
50	0.020s
100	0.157s
200	1.239s
400	11.675s

It is important to note that in this experiment, when the allocated time τ was insufficient for the GA to complete a full evolutionary cycle, the first solution encountered was retained. This situation occurred, for instance, with graphs of size $|V| = 10$, and accounts for the irregular behavior observed in the curves shown in Figure 7a.



(a) GA with $P = 5$ **(b) GA with $P = 10$**

(c) GA with $P = 20$ **(d) GA with $P = 50$**

Figure 7: Normalized Kernel Density Estimation (KDE) of the Mean Relative Error (% MRE) for GA with T linked to τ and variable population size P over 50 instances of complete random graphs.

For small graph instances, the likelihood of identifying relatively good solutions in the initial iterations is higher. This may explain why all GA variants depicted in Figure 7 exhibit a low average MRE for graphs with $|V| = 10$.

Moreover, the GA configured with a population size of $P = 5$ exhibits the lowest variance in MRE across all graph sizes, see Figure 7a. This behavior may be attributed to the reduced computational overhead associated with smaller populations, allowing the algorithm to converge more quickly, regardless of the optimality of the solution. In contrast, the GA with a population size of $P = 50$ appears to perform relatively well under the imposed time constraint, see Figure 7d, which could be explained by the enhanced exploratory capability afforded by its larger population size.

Finally, for larger instances, specifically when $|V| = 400$ in the complete random graphs, the GA configured with a population size of $P = 10$, $P = 20$ demonstrates superior performance and appears to mitigate premature convergence, as illustrated in Figure 7b and 7c.

4.3 Real Road Network Case

In this experiment, instead of using completely random graphs, we evaluate the performance of the GA (Algorithm 2) with varying population sizes by comparing its solutions to those obtained by the exact algorithm (Algorithm 1) on 10 randomly generated instances derived from a single real-world road network. This network comprises $|V| = 2390$ vertices and $|E| = 5296$ edges. Unlike the previously tested random graphs, the real-world road network is not complete. The data were extracted from OpenStreetMap, and the encoding length was extended to 25 bits to accommodate the OpenStreetMap node identifiers.

The exact algorithm (Algorithm 1) systematically evaluates all possible pairs of meeting and dropping points in the graph. Its complexity can be expressed as:

$$O(|V|^2) + O(|E| + |V| \log |V|)$$

where $O(|E| + |V| \log |V|)$ is the time complexity of Dijkstra's algorithm for a graph with $|V|$ nodes and $|E|$ edges. Since Dijkstra's algorithm is only once at the beginning, the complexity of The exact algorithm (Algorithm 1) can therefore be simplified into:

$$O(|V|^2)$$

the overall complexity scales quadratically with $|V|$, making the exact algorithm computationally expensive as the graph size increases. The GA is therefore particularly advantageous for very large graph instances. While the exact algorithm provides guaranteed optimal solutions, its computational cost becomes prohibitive for large graphs due to its dependence on the quadratic scaling of node pairs.

4.3.1 MRE when fixed generation budget. Tables 2, 3, 4, and 5 present the execution times obtained on the small real-world road network for each tested population size $P \in \{5, 10, 20, 50\}$ and for each $T \in \{100, 250, 500, 1000\}$, along with the execution time τ of the exact algorithm, which remains constant across all tables.

Table 2: Average running time of the the Exact Algorithm (EA) and GA ($P = 5$) on a small real road network

$G = (\|V\|, \|E\|)$	EA (τ)	GA ($T = 100$)	GA ($T = 250$)	GA ($T = 500$)	GA ($T = 1000$)
2390, 5296	10.52s	0.92s	2.24s	4.34s	8.60s

Table 3: Average running time of the the Exact Algorithm (EA) and GA ($P = 10$) on a small real road network

$G = (\|V\|, \|E\|)$	EA (τ)	GA ($T = 100$)	GA ($T = 250$)	GA ($T = 500$)	GA ($T = 1000$)
2390, 5296	10.52s	0.95s	2.21s	4.41s	8.66s

Table 4: Average running time of the the Exact Algorithm (EA) and GA ($P = 20$) on a small real road network

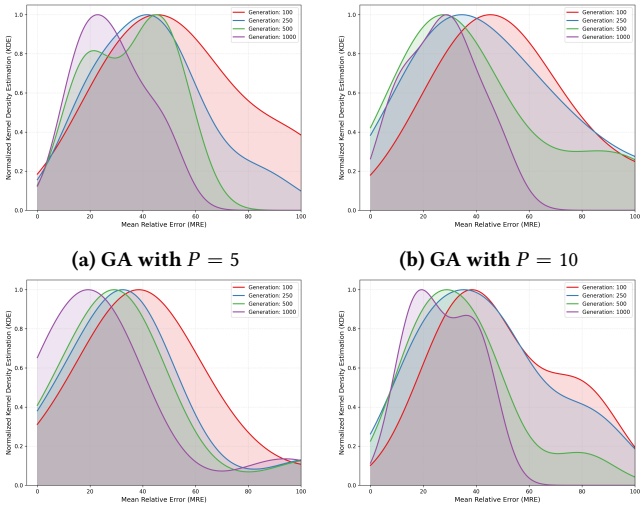
$G = (\|V\|, \|E\|)$	EA (τ)	GA ($T = 100$)	GA ($T = 250$)	GA ($T = 500$)	GA ($T = 1000$)
2390, 5296	10.52s	1.06s	2.40s	4.52s	8.89s

Table 5: Average running time of the the Exact Algorithm (EA) and GA ($P = 50$) on a small real road network

$G = (\|V\|, \|E\|)$	EA (τ)	GA ($T = 100$)	GA ($T = 250$)	GA ($T = 500$)	GA ($T = 1000$)
2390, 5296	10.52s	1.39s	2.79s	5.13s	9.39s

As anticipated based on previous results, the execution time of the GA increases with the population size for a fixed generation budget. Similarly, for a fixed population size, an increase in the generation budget leads to a longer execution time. Thus, the runtime of the GA is directly influenced by both the population size and

the number of generations. Notably, even under the most computationally intensive configuration tested—i.e., a population size of $P = 50$ and a generation budget of $T = 1000$ (see Table 5)—the GA's execution time remains lower than that of the exact algorithm on the small real-world road network.



(a) GA with $P = 5$ **(b) GA with $P = 10$**

(c) GA with $P = 20$ **(d) GA with $P = 50$**

Figure 8: Normalized Kernel Density Estimation (KDE) of the Mean Relative Error (% MRE) for GA with variable number of generations and population size P over 10 instances of a real world road network with $|V| = 2390$ and $|E| = 5296$.

Figure 8 shows that, across all tested population sizes, the GA benefits from an increased number of generations T in terms of MRE. Furthermore, among the population sizes considered, the configuration with $P = 20$, see Figure 8c, exhibits the lowest variance in MRE with respect to the number of generations T, indicating more consistent performance. Notably, when this configuration is allocated a generation budget of $T = 1000$, the MRE in terms of distance stabilizes around 19% in real world application.

Although the GA provides approximate solutions, it does so significantly faster than the exact algorithm, with its runtime primarily influenced by the P and T. This highlights an inherent trade-off between computational efficiency and solution accuracy. By appropriately tuning these parameters, the GA can effectively balance execution time and solution quality, making it a practical and scalable method for large-scale problems where exact solutions are computationally infeasible.

4.3.2 MRE when τ generation budgets. Similar to the experiments conducted on random graphs, and based on the best results obtained under fixed generation budgets, we evaluated the performance of the Genetic Algorithm (GA) (Algorithm 2) with varying population sizes by comparing its solutions to those produced by the exact algorithm (Algorithm 1), under the constraint that the GA was allotted a runtime τ equal to the execution time required by the exact algorithm for each instance.

As shown in Figure 9, the GA configured with a population size of $P = 20$ achieves the lowest average MRE, approximately 17%,

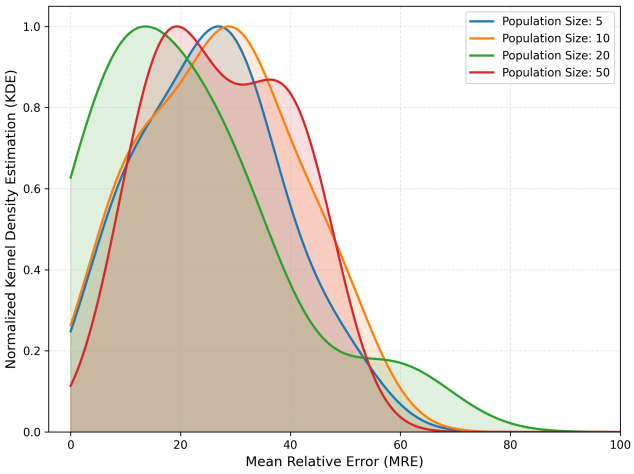


Figure 9: Normalized Kernel Density Estimation (KDE) of the Mean Relative Error (% MRE) for GA with T linked to τ and variable population size P over 10 instances of a real world road network with $|V| = 2390$ and $|E| = 5296$.

when executed under the same time budget as the exact algorithm. This configuration is followed in performance by the GA with $P = 50$.

5 Future Works

Future work could investigate further enhancements to the GA applied to the 2TP problem. One potential direction is the exploration of adaptive GAs, where parameters such as population size, mutation rate, and crossover probabilities adjust dynamically throughout the optimization process. Additionally, alternative selection mechanisms, such as rank-based or tournament selection with elitism, could be explored to improve convergence and prevent premature stagnation. Implementing an early stop criterion could also be beneficial, halting the algorithm when no significant improvement in fitness is observed over a number of generations, thus saving computational resources. Further, combining GAs with local search methods, such as hill-climbing or simulated annealing, could refine the solutions by intensifying the search around promising individuals. Another avenue for future research is to assess the performance of the GA on larger graphs, investigating its scalability and efficiency in solving 2TP problems in more complex scenarios. Finally, exploring multi-objective GAs to optimize additional criteria, such as minimizing travel time or balancing load between users, could provide more nuanced solutions for real-world applications.

Note that, in addition to the exact solution and GAs, other methods can be employed to solve the 2TP problem. Heuristic approaches, such as greedy algorithms, can be used to iteratively select candidate meeting and dropping points based on local optimizations, though they may not guarantee global optimality. In earlier research, a heuristic approach was proposed to optimize meeting points in ride-sharing scenarios within extensive multi-modal road networks [3], demonstrating its ability to find near-optimal solutions efficiently. Meta-heuristic methods, like simulated annealing or particle swarm optimization, offer alternative search strategies

that balance exploration and exploitation, potentially providing near-optimal solutions with reduced computational complexity. Furthermore, approximation algorithms could be applied to provide efficient solutions with bounded error in cases where exact solutions are too costly to compute. Further research could focus on extending the 2TP problem to include more than two users [13], as well as developing an exact algorithm that can compute solutions in subquadratic time.

6 Conclusion

This paper introduced a Fitness Caching Genetic Algorithm (GA) to address the Two Transfer Point (2TP) problem, a critical optimization challenge in vehicle routing and ride-sharing. The GA demonstrated its ability to find near-optimal solutions for smaller problem instances and acceptable solutions for larger instances, efficiently handling scenarios where the exact algorithm could become computationally intractable. The GA's iterative population-based approach balances exploration and exploitation, effectively navigating the solution space while ensuring scalability. These results highlight the GA's potential as a practical heuristic for solving the 2TP problem. However, a trade-off remains between runtime and solution quality, particularly when the GA operates under the same execution time as the exact algorithm. Further research is necessary to fully evaluate its applicability and effectiveness in other real-world scenarios.

References

[1] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. 2016. Highway Dimension and Provably Efficient Shortest Path Algorithms. J. ACM 63, 5, Article 41 (Dec. 2016), 26 pages. doi:10.1145/2985473

[2] Negin Alisoltani, Mahdi Zargayouna, Ludovic Leclercq, and Mostafa Ameli. 2020. Real-time autonomous taxi service: An agent-based simulation. Agents and Multi-Agent Systems: Technologies and Applications (2020), 199—-207. doi:10.1007/978-981-15-5764-4_18

[3] Julien Baudru and Hugues Bersini. 2024. Heuristic Optimal Meeting Point Algorithm for Car-Sharing in Large Multimodal Road Networks. In Proceedings of the 10th International Conference on Vehicle Technology and Intelligent Transport Systems. VEHITS. doi:10.5220/0000186800003702

[4] Yi Cao, Shanshan Wang, and Jiaqi Li. 2021. The Optimization Model of Ride-Sharing Route for Ride Hailing Considering Both System Optimization and User Fairness. Sustainability 13, 902 (2021), 902. doi:10.3390/su13020902

[5] E.W. Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs. Numer. Math. 1 (1959), 269–271. http://eudml.org/doc/131436

[6] Ari Freund. 2017. Improved Subquadratic 3SUM. Algorithmica 77 (2017), 440–458. https://api.semanticscholar.org/CorpusID:29179410

[7] Anka Gajentaan and Mark H. Overmars. 2012. On a class of O(n2) problems in computational geometry. Computational Geometry 45, 4 (2012), 140–152. doi:10.1016/j.comgeo.2011.11.006

[8] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In Experimental Algorithms (Lecture Notes in Computer Science, Vol. 5038), Catherine C. McGeoch (Ed.). Springer, Berlin, Heidelberg, 319–333. doi:10.1007/978-3-540-68552-4_24

[9] Fatemeh Golpayegani, Maxime Gueriau, Pierre-Antoine Laharotte, Saeedeh Ghanadbashi, Jiaying Guo, Jack Geraghty, and Shen Wang. 2022. Intelligent Shared Mobility Systems: A Survey on Whole System Design Requirements, Challenges and Future Direction. IEEE Access 10 (2022), 35302–35320. doi:10.1109/ACCESS.2022.3162848 hal:hal-03808868 Submitted 10 Oct 2022, Published 28 March 2022.

[10] John H. Holland. 1992. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press, Cambridge, MA, USA.

[11] Allan Grønlund Jørgensen and Seth Pettie. 2014. Threesomes, Degenerates, and Love Triangles. CoRR abs/1404.0799 (2014). arXiv:1404.0799 http://arxiv.org/abs/1404.0799

[12] Daniel M. Kane, Shachar Lovett, and Shay Moran. 2017. Near-optimal linear decision trees for k-SUM and related problems. arXiv:1705.01720 [cs.CG] https://arxiv.org/abs/1705.01720

[13] Kaijun Liu, Jianming Liu, and Jingwei Zhang. 2022. Advanced algorithms for optimal meeting points in road networks. IET Intelligent Transport Systems 17, 2 (2022), 221–231. doi:10.1049/itr2.12323

[14] Aishwarya Manjunath, Vaskar Raychoudhury, Snehanshu Saha, Saibal Kar, and Anusha Kamath. 2022. CARE-Share: A Cooperative and Adaptive Strategy for Distributed Taxi Ride Sharing. IEEE Transactions on Intelligent Transportation Systems 23, 7 (2022), 7028–7044. doi:10.1109/TITS.2021.3066439

[15] N. Mardešić, T. Erdelić, T. Carić, and M. Đurasević. 2024. Review of Stochastic Dynamic Vehicle Routing in the Evolving Urban Logistics Environment. Mathematics 12, 1 (2024), 28. doi:10.3390/math12010028

[16] Sakshi Patni and Bharti Sharma. 2024. Genetic Algorithms for Decision Optimization. In Intelligent Decision Making Through Bio-Inspired Optimization. IGI Global, 11. doi:10.4018/979-8-3693-2073-0.ch003

[17] Kien Hua Ting, Lai Soon Lee, Stefan Pickl, and Hsin-Vonn Seow. 2021. Shared Mobility Problems: A Systematic Review on Types, Variants, Characteristics, and Solution Approaches. Applied Sciences 11, 17 (2021), 7996. doi:10.3390/app11177996

[18] H. Zhang, H. Ge, J. Yang, and et al. 2022. Review of Vehicle Routing Problems: Models, Classification and Solving Algorithms. Archives of Computational Methods in Engineering 29 (2022), 195–221. doi:10.1007/s11831-021-09574-x

[19] M. Zhu, X.-Y. Liu, and X. Wang. 2018. An online ride-sharing path-planning strategy for public vehicle systems. IEEE Transactions on Intelligent Transportation Systems 20, 2 (2018), 616–627. doi:10.1109/TITS.2018.2821003