

Article

Optimizing Map Feature Collection Routing Using Graph Database Technology

Kaede Hasegawa *  and Antonis Bikakis * 

Department of Information Studies, University College London, London WC1E 6BT, UK;

* Correspondence: kaede.hasegawa.20@alumni.ucl.ac.uk, a.bikakis@ucl.ac.uk

Abstract

Updating road network information requires determining an optimal path that visits all roads requiring data collection. This problem, known as the Rural Postman Problem (RPP), is traditionally addressed using relational databases. However, graph databases may offer advantages by more naturally representing network structures, where nodes represent junctions and edges represent roads. This study explores the novel representation of road networks using graph databases and its unique application in optimizing RPP algorithms. We implemented three existing algorithms—(a) Nearest Neighbor, (b) “Biased” Monte Carlo, and (c) Genetic algorithm—using Cypher and compared them with (d) a novel Cypher-only algorithm designed to compute the optimal path. The results show that the Nearest Neighbor algorithm was the fastest and produced the shortest paths among all algorithms. The Cypher-only algorithm could also identify optimal paths but failed to scale beyond five required edges. These findings highlight the limitations of using Cypher alone for solving the RPP but suggest that Neo4j and Cypher hold promise for further exploration.

Keywords: routing problem; optimization; graph construction; graph database; metaheuristic; heuristic; road network

1. Introduction

Digital maps, particularly road networks, are crucial to the transportation and logistics sectors, with 81% of goods in the United Kingdom (UK) being transported by road in 2022 [1]. This highlights the need for efficient storage and updating of road network data to ensure accurate and up-to-date routing information, as well as effective routing algorithms that can identify optimal routes efficiently.

Road networks are typically represented as graphs, defined as $G = (V, A \cup E)$, where V is a set of vertices representing junctions connected by directed arcs (A) or undirected edges (E), which represent roads. Costs (e.g., distance and travel time) are assigned to arcs/edges, forming a weighted graph [2], and the optimal route is generally defined as the one with the lowest total cost, such as the shortest distance. This graph serves as the input for routing algorithms, which are designed to solve routing problems. Routing problems are generally classified into arc routing problems (ARPs) and node routing problems (NRPs), which traverse arcs (roads) and vertices (junctions), respectively [3]. This study focuses on ARPs, specifically the rural postman problem (RPP), first introduced by Orloff in 1974, which seeks the optimal route passing through a required set of roads [4]. The RPP has been widely studied in contexts such as meter reading, garbage collection, and mail delivery [2], but here, it is investigated in the context of map feature collection; companies such as



Received: 30 October 2025

Revised: 9 February 2026

Accepted: 20 February 2026

Published: 28 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

HERE Technologies deploy field staff to traverse roads requiring updates to collect data and update the digital maps.

This study proposes a novel method for representing road networks and solving routing problems, which makes use of graph database technology. Graph databases, a type of NoSQL database, store and manage data in a graph structure. They consist of nodes (equivalent to vertices or records in tables) connected by edges that represent the relationships between nodes. Edges are typically directed and can store attributes (e.g., distance) as properties, thereby enabling the construction of directed weighted graphs. In the context of road networks, nodes represent junctions and directed edges represent roads, with distances stored as edge properties [5].

Road network graphs are commonly stored in tabular format within relational databases, which remain the industry standard [6]. However, because relational databases store each road as a row or cell, they cannot naturally represent complex connectivity and impose higher computational costs when computing routes, which requires identification of the connections between certain rows or cells. Conversely, graph databases offer significant advantages for storing road networks because they directly represent the network structure. They also facilitate routing algorithms by enabling their more efficient execution compared with relational databases [7]. This is because graph databases examine only directly connected records, whereas relational databases must scan all records to satisfy the query conditions.

Despite these advantages, graph databases remain rarely used in academic research or industrial routing applications, likely because of the relatively recent development of graph database management systems (GDBMSs). Therefore, this study aims to develop and evaluate RPP optimization algorithms using a GDBMS (Neo4j) and its query language (Cypher).

2. Related Work

The mathematical formulation of the directed rural postman problem is given in [8] and is summarized as follows: Let $G = (V, A)$ be a connected directed graph, where the arc set A is partitioned into p_A required groups (r-groups) denoted by $A_1, A_2, \dots, A_{p_A}$. For each arc $(i, j) \in A$, the integer decision variable x_{ij} denotes the number of times arc (i, j) is traversed (the binary variable y_i indicates whether vertex $i \in V$ is visited), and c_{ij} denotes the traversal cost. The objective function (1) minimizes the total cost of all traversed arcs.

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \quad (1)$$

The model is subject to the following constraints. Constraints (2) ensure that at least one arc from each r-group is traversed, thereby covering all required groups. Constraints (3) enforce flow conservation by requiring that the total incoming flow equals the total outgoing flow for each vertex. Constraints (4) link arc usage to vertex visits by allowing arcs to leave a vertex, only if that vertex is marked as visited. Subtour elimination constraints (5) are imposed for connectivity such that, for each vertex i in the r-group HE_1 and each vertex i' of another r-group TA_q that belongs to any subset S of V not containing i , if both vertices are visited, then an arc leaving S must be used.

$$\sum_{(i,j) \in A_q} x_{ij} \geq 1, \forall q \in \{1, \dots, p_A\} \quad (2)$$

$$\sum_{(h,i) \in A} x_{hi} - \sum_{(i,j) \in A} x_{ij} = 0, \forall i \in V \quad (3)$$

$$x_{ij} - (|A| + 1) y_i \leq 0, \forall i \in V, (i, j) \in A \quad (4)$$

$$\sum_{(j,k) \in A: j \in S, k \notin S} x_{jk} - y_i - y_{i'} \geq -1, \quad \forall i \in HE_1, TA_q \ni i' \in S \subseteq V \setminus \{i\}, q \in \{2, \dots, p_A\}. \quad (5)$$

This RPP is an NP-hard problem, meaning that all nondeterministic polynomial-time (NP) problems can be reduced to it in a polynomial time. In simpler terms, the problem can be transformed into smaller sub-problems, but becomes exponentially harder to solve for global optimality as the problem size increases [9]. Consequently, several heuristic and approximation algorithms have been developed to address this computationally complex challenge.

The RPP is typically addressed using three types of algorithms:

- Exact Algorithms guarantee the optimal solution through exhaustive exploration of the solution space.
- Heuristic algorithms use practical techniques such as estimations or locally optimal choices (e.g., greedy algorithms) to find near-optimal solutions within reasonable time.
- Metaheuristics are higher-level strategies that guide heuristics to more effectively explore the solution space.

These approaches follow the exploration–exploitation trade-off [10], which balances the exploration of new solutions to improve optimality and the exploitation of known solutions to enhance the efficiency. Exact algorithms emphasize exploration, heuristics focus on exploitation, and metaheuristics balance the two. For example, an exact algorithm called integer linear programming solved the RPP to its optimality, except for two cases (out of 18), but took an average of 57.5 s and exhibited high variation in computation time. In comparison, a metaheuristic called the only solved three cases to optimality, but took a much shorter computation time (average of 44.9 s) with a smaller variation [8]. Furthermore, metaheuristics (e.g., memetic algorithms, genetic algorithms) typically produce results closer to the optimum than heuristics (e.g., the Monte Carlo method), albeit with a higher computational cost as the problem size increases [11].

Considering these trade-offs, this study focuses on heuristics and metaheuristics, which can provide scalable and efficient solutions for large road networks. As the literature review below suggests, the RPP is underinvestigated within the field of routing problems, and recent studies often add various constraints that make their proposed algorithms specific to those constraints. Therefore, this study aimed to investigate the potential of novel RPP algorithms that utilize graph database technology. Three classical and commonly used RPP heuristics and metaheuristics were chosen because their simplicity allows for easier fusion with graph database technology.

RPP heuristics and metaheuristics share a common foundation: they often reformulate the problem as a traveling salesman problem (TSP), which seeks the shortest route visiting each vertex exactly once before returning to the start [3]. This transformation swaps the roles of vertices and arcs, where vertices represent roads and arcs represent junctions, and introduces new arcs representing the shortest paths between roads. The resulting graph simplifies the problem and enables the use of well-studied TSP algorithms to solve the problem.

This transformation relies on the shortest-path algorithms, which identify the most efficient route between two points. A well-known example is Dijkstra’s algorithm, which performs an exhaustive search [12]. It maintains a priority queue of all vertices, with their tentative distances initialized to infinity, except for the starting vertex, which is set to zero. At each step, the algorithm selects the unexplored vertex with the smallest tentative distance, marks it as “visited,” and updates the distances to its adjacent vertices if shorter paths are found through the current vertex. This process is repeated until the target vertex is marked as visited. Another prominent algorithm is A*, which enhances Dijkstra’s method

by incorporating heuristics to increase search efficiency [12]. In addition to the distance already traveled, A* considers the estimated distance to the target, allowing it to prioritize vertices with the lowest total estimated cost and to reduce unnecessary exploration.

2.1. Exact Algorithms

The branch-and-bound method is an exact algorithm that systematically explores a decision tree by dividing the problem into subproblems and “pruning” those that cannot yield better solutions. This is done by comparing the upper bounds (best-known solutions) with the lower bounds (theoretical minimums) [13]. The branch-and-cut method enhances this approach by adding a cutting plane phase, which iteratively introduces additional constraints to remove infeasible fractional solutions. These phases are alternated with the branch-and-bound process to progressively refine the solution space.

In a study that applied the branch-and-cut method to the RPP, three primary types of inequalities were used. Connectivity inequality ensures sufficient graph connectivity such that each vertex subset has at least one outgoing edge. The parity inequality enforces even or odd constraints on the number of arcs and vertices in a solution. The K-C inequality maintains the structural properties necessary for the required connectivity. This method successfully solved nearly all instances to optimality and outperformed existing approaches, although it failed to find feasible solutions within a two-hour time limit for graphs with approximately 880 vertices and 4580 arcs [14].

2.2. Heuristics

One of the most well-known RPP heuristics is the method proposed by Frederickson [15]. As a first step, it constructs a new graph by connecting all required arcs using the minimum additional arcs, typically via the shortest spanning tree (SST). In the second step, the graph is transformed into an Eulerian graph, which enables a route that traverses each required arc exactly once and returns to the starting point of the route. This method has been the foundation for subsequent improvements, including the work of Christofides et al. [16], who introduced post-processing steps that replaced segments with shortcuts, improving Frederickson’s results by 3–5% [17].

Another heuristic is the Monte Carlo method-based algorithm [18], which simulates a vehicle navigating a route by randomly selecting roads based on their weighted probability, calculated using distance (i.e., shorter roads have a higher likelihood of being chosen). The generated path is then refined to remove redundancies. Repeating this simulation for a set number of iterations, the method selects the best path among those produced. Despite its simplicity, this algorithm achieved the optimal tour in 19 out of 26 instances within 1000 iterations, outperforming the heuristic of Christofides et al. [18].

Although the nearest neighbor algorithm has not been widely studied in the RPP context, one study applied it to find a path visiting a required set of vertices [19]. The method uses Dijkstra’s algorithm to identify the shortest path from the current vertex to all others, choosing the closest vertex as the next destination. This process repeats until all required vertices are visited. As a greedy algorithm, it offers a simple and efficient approach that is often used as a baseline or initial step for more advanced methods.

2.3. Metaheuristics

One of the most commonly used metaheuristics for the RPP is the genetic algorithm (GA), a nature-inspired technique that mimics the process of natural selection to evolve solutions over successive generations. The genetic algorithm outperformed the greedy algorithm and showed performance comparable to simulated annealing, although these comparisons were based on small graphs with fewer than 100 vertices and arcs [20]. In a subsequent study, the genetic algorithm was modified into the randomized population

genetic algorithm (RPGA), which omits the crossover operation and generates new populations from a single chromosome [11]. Compared to other approaches, including the Monte Carlo method and other genetic algorithm variants, the RPGA demonstrated superior performance on test graphs and a real-world street network of Porto, which contained 656 vertices and 876 arcs.

Another nature-inspired metaheuristic applied to the RPP is the ant colony algorithm, which simulates the behavior of ants to guide the search process towards optimality. Artificial “ants” explore the graph and deposit pheromone trails, with shorter paths accumulating more pheromones and thus being more likely to be followed by subsequent ants. The ant colony algorithm achieved a strong performance, solving 70% of the RPP instances to the best-known solution within seconds on small graphs with fewer than 50 vertices and 200 arcs [21].

Tabu Search is another metaheuristic used for the RPP. It employs a local search strategy that iteratively explores the solution space by moving from one solution to a neighboring one while avoiding cycles and local optima using a tabu list that records recently visited solutions or moves. When applied to 270 randomly generated mixed graphs, Tabu Search with a diversification phase achieved an average deviation from the optimal solution of no more than 0.53% [22].

2.4. Integration of Graph Databases

To date, only one study has used a graph database to solve a routing problem, specifically a multi-stop routing task with multimodal transportation [23]. The resulting paths were optimal and exhibited a slower increase in query time than the increase in the number of traversed nodes, highlighting the efficiency of graph databases. However, the traversal order was pre-specified; therefore, the approach involved sequential applications of shortest-path algorithms, which does not align with the RPP.

Although graph databases offer a wide range of built-in algorithms, routing research using them has primarily focused on shortest-path methods. For example, Neo4j, a leading GDBMS, includes a Graph Data Science (GDS) library with numerous predefined algorithms. Neo4j consistently outperforms relational databases (e.g., SQL) in shortest path queries, with execution times scaling more efficiently as the graph size and traversal depth increase [7]. Specifically, Dijkstra’s algorithm ran 30–35% faster in Neo4j than in PostgreSQL with pgRouting [6], and A* also performed better in Neo4j owing to its iterative nature [24].

The integration of graph databases with heuristics and metaheuristics is rare, even outside the routing domain. For instance, two studies on community detection in social networks incorporated a graph database into a genetic algorithm to generate initial populations and retrieve relevant data during fitness evaluations [25,26]. These implementations yielded improved community structures compared to conventional approaches, such as the label propagation algorithm or the use of relational databases. These examples underscore the effectiveness of graph databases over relational databases for optimization of highly connected data structures, although the methods are not directly applicable to routing because of contextual differences.

3. Materials and Methods

3.1. Data

The dataset was provided by HERE Europe B.V. Kennedyplein 222-226, 5611 ZT Eindhoven, Netherlands and it consists of real-world German road network data from their NavStreets product. The base map was created using data from multiple sources, including local government and commercial agencies. These data were then reviewed and digitized using aerial photos, differential GPS, and proprietary software to accurately position the geographical features. Quality was further enhanced through road tests conducted by field

staff, who collected new data, verified existing data, and applied additional proprietary methodologies to ensure an accurate representation of real-world conditions. The original data and a full description of it cannot be disclosed because of intellectual property rights.

For this study, we selected Area G5, which covers Berlin, because of its dense and well-connected road network. We used two shapefiles to construct the graph by utilizing the World Geodetic System 1984 coordinate reference system. The first shapefile, *Streets.shp*, contains road segments and associated attributes, where each record corresponds to a unique road. The second shapefile, *Zlevels.shp*, stores the relative vertical positions of roads at specific points, including junctions and curvature points.

Owing to the large size of the dataset, comprising 1,150,081 road records and 4,488,294 junction records, we pre-processed the data in R using the *sf* package. To simplify the experimental setup, we filtered the dataset to focus on specific administrative areas (e.g., cities), thereby generating smaller and more manageable road network graphs.

For *Streets.shp*, inaccessible roads were excluded. We computed the edge weights, which represent road distances, using the geometry column. We identified the road directions based on two columns indicating the start and end junction IDs. For bidirectional roads, we created duplicate records with swapped junction IDs. We then exported the processed data to a CSV file, which was used in Neo4j to define the edges.

In *Zlevels.shp*, we filtered out irrelevant points (i.e., non-junctions) and then joined the resulting data with the processed *Streets.shp* dataset to remove unused junctions. We removed duplicates and calculated the latitude and longitude of each junction from the geometry column. We then exported the data to a CSV file, which was used in Neo4j to define the nodes.

Using Neo4j's *LOAD CSV* function, we imported the two CSV files into an empty graph database. During import, some nodes and edges were found to be disconnected from the main graph, which may be due to filtering by administrative area or exclusion of inaccessible roads. To remove these disconnected components, we used the weakly connected components algorithm to identify the connected components (ignoring edge direction) and retained only the largest connected component. If more than 10% of the nodes or edges were removed during this step, a different administrative area was selected for the analysis. In this study, weakly connected components were used based on the assumption that real road network data should be strongly connected. However, it may be more feasible to use strongly connected components to ensure strong connectivity (i.e., there is a directed path between each pair of nodes within the set). The final structure of the graph database is shown in Figure 1, where the edges between the left node represent two-way roads and the edges between the right node represent one-way roads.

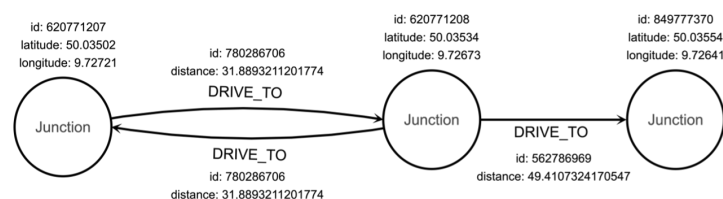


Figure 1. Structure of the road network graph database (used yEd live editor).

The format of the road and junction CSV files and their conversion into the Neo4j graph database are further explained in a markdown file available on the project's repository (see link below) to enable the replication of the graph database.

3.2. Algorithms

The code of all algorithms is available in this repository: https://github.com/Chicake/rpp_algorithms_using_gdbms.

3.2.1. A* Algorithm and Graph Transformation

Because all algorithms rely on shortest-path computations to locally optimize route searching, we selected the A* algorithm from the Neo4j GDS library for its use of heuristics that improve search efficiency. To estimate the heuristics, we used the Haversine formula [27], shown in (5).

$$h = 2R \sin^{-1} \left(\sqrt{\sin^2 \left(\frac{\phi_1 - \phi_2}{2} \right) + \cos \phi_1 \cdot \cos \phi_2 \cdot \sin^2 \left(\frac{\lambda_1 - \lambda_2}{2} \right)} \right) \quad (5)$$

Here, ϕ_1 and ϕ_2 are the latitudes, λ_1 and λ_2 are the longitudes of the start and target points, respectively, and R is the Earth's radius. This formula estimates the distance (h) between two points on a sphere, making it an ideal heuristic for routing. As Neo4j's A* algorithm operates on nautical miles, we converted all input distances from meters to miles.

While the RPP often transforms graphs to convert RPPs to TSPs, such transformations can discard important information, including road direction and the distinction between the end nodes of a required edge. To preserve this information, we developed a novel graph transformation method that maintains the required edges as edges and directly connects all end nodes of the required edges with new edges representing the shortest path between each pair of such nodes. Within this transformed graph, the required edges are treated as pairs of end nodes, and any resulting path alternates between traversing the required edges and the shortest paths connecting them.

3.2.2. Cypher-Only Algorithm (CA)

Although no prior RPP studies have used Cypher queries alone for optimization, we implemented this approach to assess Neo4j's native capabilities in solving the RPP.

To reduce the number of edge node combinations traversed during the query, we applied the graph transformation described in Section 3.2.1. This created shortest-path edges between the end nodes of the required edges, reducing the traversal size to the number of required edges + the number of shortest paths.

The algorithm begins by finding all paths composed of `SHORTEST_PATH` edges of a given length, starting from a specific node (`start_node_id`). The `all` and `any` functions are then used to verify that all required edge IDs (`required_edges`) are present in the retrieved paths. The matching paths are retained, and the `REDUCE` function is used to sum the weights of the edges, yielding the total path distance per path. The paths are then ordered by ascending distance, and the shortest path is returned. See Listing 1 for the implementation details.

To clarify, this algorithm performs an exhaustive search over condition-matching paths; therefore, the exploration–exploitation trade-off may not apply here. Nonetheless, it remains an algorithm because it systematically identifies feasible paths, and the search space is reduced by graph transformations and structured path constraints.

Listing 1: Cypher-Only Algorithm Query.

```
MATCH path = (a) -[r:SHORTEST_PATH*{num_edges*2}]-> ()
WHERE a.id = '{start_node_id}' AND all(relName IN {required_edges}
WHERE any(rel IN relationships(path) WHERE rel.id = relName))
WITH path, REDUCE(totalWeight = 0, r IN relationships(path)
| totalWeight + toFloat(r.totalCost)) AS pathWeight
ORDER BY pathWeight ASC
```

LIMIT 1

RETURN [node IN nodes(path) | node.id] AS nodeList,
pathWeight

3.2.3. Nearest Neighbor Algorithm (NN)

We selected the NN as a baseline to evaluate the performance of more advanced algorithms. The NN directly leverages the graph database by using the A* algorithm for immediate route decisions. This version is based on the NN described in [19], with modifications to treat the required edges as ordered pairs of end nodes; once one end node is visited, its corresponding counterpart must be visited next.

From a designated start node, it calculates the shortest paths to all other unvisited end nodes of the required edges using the A* algorithm. It then selects the closest node (and its corresponding required edge) and traverses to the other end node of the edge. Once both end nodes of the required edge are visited, they are removed from the search space. This process continues until all the required edges are traversed. The total distance is computed by adding the distances of all traversed edges. See Algorithm 1 for the implementation details.

Algorithm 1 Nearest Neighbor Algorithm Pseudocode

Require: $G = (V, E)$: Graph with nodes V and edges E
 $E'(V')$: List of unvisited required edges & end nodes
 $V_s \subseteq V$: Starting node

- 1: Initialize $w \leftarrow \emptyset, w_{\text{total}} \leftarrow 0$
- 2: **while** $E' \neq \emptyset$ **do**
- 3: **for all** $V'_n \in V'$ **do**
- 4: $w_n \leftarrow \text{A*ShortestPathAlgorithm}(V_s, V'_n)$
- 5: $w \leftarrow w \cup w_n$
- 6: **end for**
- 7: Find $\min(w)$ and its associated V'_n
- 8: Select $E'_n(V'_1, V'_2)$ where $V'_1 = V'_n$
- 9: $w_{\text{total}} \leftarrow w_{\text{total}} + (\min(w) + w_{E'_n})$
- 10: $V_s \leftarrow V'_2$
- 11: Remove E'_n from E'
- 12: **end while**
- 13: **return** w_{total}

3.2.4. Biased Monte Carlo Algorithm (MC)

The NN often falls into local optima owing to its tendency to select a locally optimal path. To address this limitation, we also experimented with the MC, which occasionally selects suboptimal choices to explore a broader solution space. We adopted this approach from [18], with a coding implementation based on [28]. It is important to note that this algorithm is called the "biased" Monte Carlo algorithm, as this is not a pure Monte Carlo method, but rather a biased stochastic heuristic that selects roads in a biased manner based on their distances. We introduced two major modifications to the original method: the insertion of the shortest paths between the required edges and the specification of a starting node. These changes rendered the three simplification routines from [18] inapplicable, as removing any edges could either break the computed shortest paths or prevent the traversal of all required edges from the starting node.

To avoid recalculating all the shortest paths between the required edge nodes in every generation, we stored the outputs of the A* algorithm in a pandas DataFrame. These shortest paths and required edges (one edge for one-directional roads and two edges for two-directional roads) are stored with their starting and ending nodes to correctly account

for the direction and traversal order of the edges, ensuring path validity. This significantly improved efficiency, as the MC selects one edge at a time based on the matching conditions, an operation more suited to a relational database than a graph database.

This algorithm operates similarly to the NN but differs in edge selection; edges are chosen randomly based on probability weights calculated from their total distance w (i.e., $1/w^a$, where the scaling factor $a = 1$). This biases the selection toward shorter paths while retaining randomness. To broaden the exploration, the algorithm generated 1000 different paths and selected the best among them, following the methodology in [18]. See Algorithm 2 for the implementation details.

Algorithm 2 Biased Monte Carlo Algorithm Pseudocode

Require: $G = (V, E)$: Graph with nodes V and edges E
 $E'(V')$: List of unvisited required edges & end nodes
 V_s : Starting node
 α : Alpha value for probability calculation
 g : Number of generations

- 1: Initialize $w_{\text{best}} \leftarrow 0$
- 2: Graph transformation
- 3: **for** $i \leftarrow 1$ to g **do**
- 4: Initialize $w_i \leftarrow 0$
- 5: **while** $E' \neq \emptyset$ **do**
- 6: **for all** $V'_n \in V'$ **do**
- 7: $w_n \leftarrow A * \text{ShortestPathAlgorithm}(V_s, V'_n)$
- 8: $p_n \leftarrow 1/w_n^\alpha$
- 9: $p \leftarrow p \cup p_n$
- 10: **end for**
- 11: $(V'_s, w_s) \leftarrow \text{RandomSelectionWithWeights}(V', p)$
- 12: Select $E'_n(V'_1, V'_2)$ where $V'_1 = V'_s$
- 13: $w_i \leftarrow w_i + (w_n + w_{E'_n})$
- 14: $V_s \leftarrow V'_2$
- 15: Remove E'_n from E'
- 16: **end while**
- 17: **if** $w_i < w_{\text{best}}$ **then**
- 18: $w_{\text{best}} \leftarrow w_i$
- 19: **end if**
- 20: **end for**
- 21: **return** w_{best}

3.2.5. Genetic Algorithm (GA)

We selected the GA as the representative metaheuristic because of its widespread use in RPP studies. This implementation adopts the GA from [20], with coding based on [29]. In this algorithm, a “gene” represents the ID of a required edge, a “chromosome” is a list of genes representing a traversal order, and a “population” consists of multiple chromosomes.

The process begins with the random generation of an initial population of 20 chromosomes, followed by an evolution phase that increases the variation and population size using crossover and mutation. For crossover (rate = 0.6), we used the partially matched crossover (PMX) method due to the importance of gene order, with the rate same as the referred study [20]. PMX creates two children by exchanging segments between parent chromosomes, where the crossover points are randomly selected. Mutations alter the gene order and can occur via two methods (both set at a rate of 0.1 instead of 0.03 and 0.04 in the study to compensate for the smaller population):

- Reciprocal exchange: Swaps two genes (e.g., $[1, 2, 3, 4, 5] \rightarrow [1, 4, 3, 2, 5]$)
- Inversion: Reverses a segment of the chromosome (e.g., $[1, 2, 3, 4, 5] \rightarrow [1, 4, 3, 2, 5]$)

The selection phase reduces the population back to its initial size using the roulette wheel method, which samples chromosomes probabilistically, based on fitness scores. Fitness is calculated in the same way as in the MC, using the total distance w (i.e., $1/w^a$, with $a = 1$). This simulates the “survival of the fittest,” where fitter chromosomes are more likely to be selected.

The evolution and selection processes alternated for 100 generations. The only significant difference between this GA and the GA presented in [20] is that the chromosome with the best fitness score is retained in every generation, rather than only in the final generation, as proposed in [20]. This ensures that the optimal path found in the explored search space is retained. See Algorithm 3 for the implementation details.

Algorithm 3 Genetic Algorithm Pseudocode

Require: $G = (V, E)$: GraphDB with nodes V and edges E

$E' \subseteq E$: List of unvisited required edges

$V_s \subseteq V$: Starting node

a : Alpha value for probability calculation

gen: Number of generations

x : Population size

r_c : Rate of cross-over

r_{m1} : Rate of reciprocal exchange

r_{m2} : Rate of inversion

```

1: Initialize  $\text{pop}_c \leftarrow \emptyset, w_{\text{best}} \leftarrow 0$ 
2: Graph transformation
3: for  $i \leftarrow 1$  to  $x$  do
4:    $\text{chrom} \leftarrow \text{RandomReorder}(E')$ 
5:    $\text{pop}_c \leftarrow \text{pop}_c \cup \{\text{chrom}\}$ 
6: end for
7: for  $i \leftarrow 1$  to gen do
8:    $\text{scores} \leftarrow \emptyset$ 
9:   for  $j \leftarrow 1$  to  $\lfloor \frac{x \cdot r_c}{2} \rfloor$  do
10:     $\text{parent}_a, \text{parent}_b \leftarrow \text{RandomlySelect}(\text{pop}_c, 2)$ 
11:     $\text{child}_a, \text{child}_b \leftarrow \text{crossover}(\text{parent}_a, \text{parent}_b)$ 
12:     $\text{pop}_c \leftarrow \text{pop}_c \cup \{\text{child}_a, \text{child}_b\}$ 
13:   end for
14:   for all  $\text{chrom} \in \text{pop}_c$  do
15:     if  $\text{random}(0, 1) < r_{m1}$  then
16:        $\text{chrom} \leftarrow \text{ReciprocalExchange}(\text{chrom})$ 
17:     else if  $\text{random}(0, 1) < r_{m2}$  then
18:        $\text{chrom} \leftarrow \text{Inversion}(\text{chrom})$ 
19:     end if
20:   end for
21:   for all  $\text{chrom} \in \text{pop}_c$  do
22:      $\text{score}_{\text{chrom}} \leftarrow \frac{1}{w_{\text{chrom}}^a}$ 
23:      $\text{scores} \leftarrow \text{scores} \cup \{\text{score}_{\text{chrom}}\}$ 
24:     if  $\text{score}_{\text{chrom}} > \text{score}_{\text{best}}$  then
25:        $w_{\text{best}} \leftarrow w_{\text{chrom}}$ 
26:        $\text{score}_{\text{best}} \leftarrow \text{score}_{\text{chrom}}$ 
27:     end if
28:   end for
29:   if  $i \neq \text{gen}$  then
30:      $\text{pop}_c \leftarrow \text{RouletteWheelMethod}(\text{pop}_c, x, \text{scores})$ 
31:   end if
32: end for
33: return  $w_{\text{best}}$ 

```

As with the MC, we pre-computed the shortest paths and stored them in a pandas DataFrame for an efficient lookup. When connecting one node of a required edge A to

another required edge B that is bidirectional, two shortest paths are possible because of the two distinct end nodes in edge B. We randomly selected one based on weighted probabilities, enabling variability, even when identical chromosomes were generated.

3.3. Experimental Setup

To simplify the testing environment, we selected minimal datasets. We converted three administrative areas with relatively small road networks into separate graph databases: one with 100 roads, one with 300 roads, and one with 501 roads. The actual number of edges and nodes varied owing to bidirectional roads and the filtering of disconnected components. The final details of each graph database are as follows:

- One hundred roads: 93 nodes and 187 edges, denoted as $G(V = 93, E = 187)$. No disconnected subgraphs were observed.
- Three hundred roads: 253 nodes and 550 edges, $G(V = 253, E = 552)$. Four nodes were removed because of disconnections.
- Five hundred one roads: 441 nodes and 950 edges, $G(V = 441, E = 950)$. Four nodes were removed.

For each graph, we randomly selected 3, 5, 10, and 30 required edges out of all edges, within the range used in previous studies (e.g., four required edges in [18]). We also randomly selected the starting node, ensuring that it was not an end node of any required edge. We did not predefine the target node because it was determined by the resulting optimal path. To capture the performance variability, we conducted 30 trials per configuration using different combinations of required edges and starting nodes and reported the mean of the results. The seed used for the random sampling-related code starts from 1 and increments by 1 per trial.

We employed two performance metrics that are commonly used in RPP studies. The first metric is the total distance (m), calculated as the sum of the distances/weights of all traversed edges. The second metric is the total computational time (s), which is the time taken from input to output, measured using Python's `time` library. We set a timeout threshold of 300 s; any trial exceeding this was recorded as null for both metrics. Optimality is another common metric used in RPP studies; however, it cannot be measured because the optimal path is unknown and no exact algorithm is used. Performance differences across methods are reported descriptively; no statistical tests or effect sizes were conducted, as it is not a common practice within this field of study. Therefore, comparisons should be interpreted as descriptive summaries of observed trends rather than as inferential claims about statistically significant differences.

We ran the experiments on a MacBook Pro 13-inch (2022) with an Apple M2 CPU, 16 GB RAM, and 494.38 GB of disk space, running the macOS Sonoma 14.3.1 (Apple, California, CA, USA). The database environment consisted of Neo4j 5.21.2 running under Java SE 17 and 21. Regarding the memory limit of Neo4j, JVM heap starts from 512MB with a maximum of 1 GB, 512MB RAM to keep graph data (page cache), and 716.80MiB of total transaction memory cap. The graph databases were hosted locally on Neo4j Desktop Version 1.5.9 using the GDS library Version 2.8.0. Python 3.9.13 was executed using Visual Studio Code Version 1.92.2. The query cache was cleared before each trial to ensure that all algorithms ran cold.

4. Results

4.1. Total Distance of the Path (m)

As shown in Tables 1–3, the distances produced by the different algorithms exhibit substantial variation across instances. The CA yielded the shortest paths among all algorithms in instances where it did not time out, namely those with three and five required

edges. However, only a small number of instances with five required edges produced feasible solutions. In these cases, two of the required edges shared a common vertex (e.g., selecting both directions of a bidirectional road), which effectively reduced the problem to one with only four distinct required edges, leading to completing computation in time. Except for these cases, the CA consistently timed out (i.e., took longer than 300 s to compute) when the number of required edges exceeded five. These instances were re-evaluated using an extended timeout of 5400 s, yet no feasible solutions were obtained. In the cases where the CA timed out (i.e., for instances with 10 and 30 required edges), the NN produced the shortest paths among all algorithms. This indicates that the CA performs best for instances with a small number of required edges, whereas the NN becomes more effective as the number of required edges increases.

Table 1. G(V = 93, E = 187): Descriptive Statistics of total distance (m) across algorithms per condition.

Algorithm	Req. E	n	Mean $\pm$ SD	Range	95% CI
CA	3	30	2213.74 $\pm$ 516.08	1370.25–3309.24	2021.03–2406.44
	5	2	2663.02 $\pm$ 308.83	2444.65–2881.40	-
	10	0	-	-	-
	30	0	-	-	-
NN	3	30	2250.99 $\pm$ 541.36	1370.25–3309.24	2048.85–2453.14
	5	30	3110.68 $\pm$ 706.42	1599.80–4826.75	2846.90–3374.46
	10	30	5024.88 $\pm$ 1148.52	2814.75–7148.16	4596.01–5453.75
	30	30	8952.71 $\pm$ 1201.44	6680.66–12,390.02	8504.08–9401.33
MC	3	30	2604.21 $\pm$ 756.60	1416.02–4689.91	2321.69–2886.73
	5	30	3774.38 $\pm$ 1077.70	2235.83–6125.18	3371.96–4176.80
	10	30	7094.04 $\pm$ 1365.84	4282.11–10,256.78	6584.03–7604.06
	30	30	14,887.64 $\pm$ 2463.84	11,032.62–19,748.79	13,967.62–15,807.65
GA	3	30	2298.92 $\pm$ 531.58	1416.02–3421.79	2100.42–2497.42
	5	30	2984.52 $\pm$ 642.79	1709.57–4538.72	2744.50–3224.54
	10	30	5563.44 $\pm$ 884.23	4068.16–7720.01	5233.26–5893.61
	30	30	19,113.14 $\pm$ 1841.94	15,134.72–22,643.43	18,425.34–19,800.93

Note. The best-performing algorithm per condition is bolded in the Mean $\pm$ SD column. Req. E is the number of required edges and n is the number of valid trials out of 30. "-" means all trials timed out. CA is Cypher-Only Algorithm, NN is Nearest Neighbor Algorithm, MC is Biased Monte Carlo Algorithm, and GA is Genetic Algorithm.

Table 2. $G(V = 253, E = 552)$: Descriptive statistics of total distance (m) across algorithms per condition.

Algorithm	Req. E	n	Mean $\pm$ SD	Range	95% CI
CA	3	30	3422.86 $\pm$ 736.26	1260.29–4696.26	3147.93–3697.78
	5	1	2915.86 $\pm$ –	–	–
	10	0	–	–	–
	30	0	–	–	–
NN	3	30	3451.55 $\pm$ 729.42	1260.29–4696.26	3179.18–3723.92
	5	30	4447.55 $\pm$ 906.39	2451.09–6391.21	4109.10–4786.00
	10	30	6110.33 $\pm$ 869.38	4638.85–8226.03	5785.70–6434.96
	30	30	11,587.19 $\pm$ 1390.53	8014.97–14,555.49	11,067.96–12,106.42
MC	3	30	4402.03 $\pm$ 1171.80	2098.71–6747.23	3964.47–4839.59
	5	30	6636.03 $\pm$ 1828.49	3969.01–9926.09	5953.26–7318.80
	10	30	10,246.56 $\pm$ 1918.12	6896.15–15,921.89	9530.32–10,962.80
	30	30	26,215.51 $\pm$ 4960.47	17,234.97–34,274.35	24,363.24–28,067.78
GA	3	30	3521.69 $\pm$ 751.58	1280.43–5104.56	3241.04–3802.33
	5	30	4576.01 $\pm$ 911.55	2328.60–6351.12	4235.63–4916.39
	10	30	7971.27 $\pm$ 984.14	6152.64–10,601.70	7603.78–8338.75
	30	29	29,566.52 $\pm$ 1916.95	25,252.58–33,455.95	28,850.71–30,282.32

Note. The best-performing algorithm per condition is bolded in the Mean $\pm$ SD column Req. E is the number of required edges and n is the number of valid trials out of 30. "–" means all trials timed out.

Table 3. $G(V = 441, E = 950)$: Descriptive Statistics of Total Distance (m) across Algorithms Per Condition.

Algorithm	Req. E	n	Mean $\pm$ SD	Range	95% CI
CA	3	30	3154.39 $\pm$ 1217.85	1020.50–5556.93	2699.64–3609.14
	5	1	4010.02 $\pm$ –	–	–
	10	0	–	–	–
	30	0	–	–	–
NN	3	30	3324.58 $\pm$ 1335.56	1024.69–6186.11	2825.88–3823.29
	5	30	6313.25 $\pm$ 1509.39	3349.82–9793.28	5749.63–6876.86
	10	30	9491.38 $\pm$ 1594.72	5568.81–12,017.78	8895.90–10,086.86
	30	30	19,484.31 $\pm$ 2413.75	15,076.79–24,546.58	18,583.00–20,385.61
MC	3	30	3688.85 $\pm$ 1558.71	1046.64–7984.13	3106.82–4270.88
	5	30	7556.27 $\pm$ 2107.02	4288.69–12,426.79	6769.49–8343.04
	10	30	12,814.18 $\pm$ 2332.50	9050.13–16,912.15	11,943.21–13,685.15
	30	30	36,492.06 $\pm$ 4893.48	28,168.13–46,921.46	34,664.80–38,319.31
GA	3	30	3222.33 $\pm$ 1202.16	1046.64–5556.93	2773.43–3671.22
	5	30	5842.48 $\pm$ 1507.87	3064.14–9637.49	5279.43–6405.53
	10	30	10,136.86 $\pm$ 1624.94	6402.97–13,324.36	9530.09–10,743.62
	30	30	37,222.25 $\pm$ 3251.43	31,058.21–46,089.47	36,008.14–38,436.35

Note. The best-performing algorithm per condition is bolded in the Mean $\pm$ SD column Req. E is the number of required edges and n is the number of valid trials out of 30. "–" means all trials timed out.

Conversely, the MC generally produced the longest paths and exhibited the greatest variabilities across instances. The only exception occurred in instances with 30 required edges, where the GA yielded the longest paths. Moreover, the GA demonstrated competitive performance in a limited number of cases: it was the second-best performing algorithm for instances with five required edges in the graph $G(V = 93, E = 187)$ and for instances with three

and five required edges in $G(V = 441, E = 950)$. In all other instances, the GA ranked as the third-best performing algorithm.

These results are somewhat counterintuitive when considered in light of the exploration–exploitation trade-off. Specifically, the NN is the most exploitation-oriented algorithm, whereas the GA is the most exploration-oriented. Therefore, one might expect the GA to perform better as the problem complexity increases; however, the empirical results suggest otherwise.

Finally, as expected, the total distance generally increased with the increase of both the number of required edges and the graph size.

4.2. Total Computational Time (s)

As shown in Tables 4–6, the NN algorithm significantly outperformed all other methods in terms of computational time for all experimental instances. The CA was the second fastest for instances with three required edges; however, it timed out for most instances with more than five required edges. Although a small number of five-required-edge instances were solved before timeout of 300 s (with the same reason explained in Section 4.1), these instances required the longest computational time among all algorithms. All other instances with five or more required edges timed out (i.e., took longer than 300 s to compute). They were re-evaluated using an extended timeout of 5400 s, yet no solution was yielded.

In contrast, the MC and GA required substantially more computation time. The GA was consistently the slowest method, with one instance in 30 required edges timing out. These results are consistent with the exploration–exploitation trade-off, as the NN is the most exploitation-oriented algorithm, whereas the GA is the most exploration-oriented and therefore incurs a higher computational cost.

Overall, the computational time increased with the increase of both the number of required edges and the size of the graph. However, increases in the number of required edges had a substantially greater impact on the runtime than increases in the graph size, indicating that the number of required edges is the primary driver of computational complexity in this problem.

Table 4. G(V = 93, E = 187): Descriptive statistics of total time (s) across algorithms per condition.

Algorithm	Req. E	n	Mean $\pm$ SD	Range	95% CI
CA	3	30	0.64 $\pm$ 0.19	0.29–1.10	0.57–0.71
	5	2	149.52 $\pm$ 10.35	142.20–156.84	-
	10	0	-	-	-
	30	0	-	-	-
NN	3	30	0.08 $\pm$ 0.01	0.06–0.12	0.08–0.09
	5	30	0.13 $\pm$ 0.01	0.11–0.17	0.12–0.13
	10	30	0.28 $\pm$ 0.16	0.17–0.81	0.22–0.33
	30	30	0.88 $\pm$ 0.22	0.64–1.75	0.80–0.96
MC	3	30	6.44 $\pm$ 0.46	5.41–8.13	6.26–6.61
	5	30	10.82 $\pm$ 1.19	8.69–14.41	10.38–11.26
	10	30	23.73 $\pm$ 5.29	18.60–46.78	21.76–25.70
	30	30	69.21 $\pm$ 14.07	57.74–130.94	63.96–74.47
GA	3	30	13.93 $\pm$ 5.21	10.48–40.98	11.99–15.88
	5	30	20.97 $\pm$ 2.20	16.52–26.56	20.15–21.79
	10	30	45.26 $\pm$ 4.41	38.22–54.32	43.62–46.91
	30	30	155.01 $\pm$ 10.07	141.44–177.67	151.25–158.77

Note. The best-performing algorithm per condition is bolded in the Mean $\pm$ SD column Req. E is the number of required edges and n is the number of valid trials out of 30. “-” means all trials timed out.

Table 5. G(V = 253, E = 552): Descriptive statistics of total time (s) across algorithms per condition.

Algorithm	Req. E	n	Mean $\pm$ SD	Range	95% CI
CA	3	30	0.71 $\pm$ 0.25	0.29–1.47	0.61–0.80
	5	1	166.16 $\pm$ -	-	-
	10	0	-	-	-
	30	0	-	-	-
NN	3	30	0.08 $\pm$ 0.02	0.05–0.18	0.07–0.09
	5	30	0.16 $\pm$ 0.08	0.12–0.54	0.14–0.19
	10	30	0.31 $\pm$ 0.20	0.21–1.32	0.24–0.39
	30	30	1.76 $\pm$ 0.42	1.18–3.24	1.60–1.92
MC	3	30	7.48 $\pm$ 3.42	6.17–22.36	6.20–8.75
	5	30	11.37 $\pm$ 2.11	8.96–19.20	10.58–12.16
	10	30	24.06 $\pm$ 3.87	19.96–37.48	22.61–25.50
	30	30	84.31 $\pm$ 10.11	72.35–112.49	80.53–88.09
GA	3	30	14.13 $\pm$ 3.84	10.33–27.62	12.69–15.57
	5	30	23.37 $\pm$ 3.62	17.89–34.78	22.02–24.72
	10	30	47.81 $\pm$ 5.31	40.47–60.60	45.82–49.79
	30	29	179.15 $\pm$ 13.84	153.98–211.84	173.98–184.32

Note. The best-performing algorithm per condition is bolded in the Mean $\pm$ SD column Req. E is the number of required edges and n is the number of valid trials out of 30. “-” means all trials timed out.

Table 6. $G(V = 441, E = 950)$: Descriptive statistics of total time (s) across algorithms per condition.

Algorithm	Req. E	n	Mean $\pm$ SD	Range	95% CI
CA	3	30	0.93 ± 0.39	0.40–2.28	0.78–1.07
	5	1	$147.14 \pm -$	-	-
	10	0	-	-	-
	30	0	-	-	-
NN	3	30	0.15 ± 0.14	0.08–0.85	0.10–0.21
	5	30	0.26 ± 0.42	0.12–1.89	0.11–0.42
	10	30	0.29 ± 0.04	0.24–0.45	0.27–0.30
	30	30	2.76 ± 2.05	1.43–11.02	1.99–3.52
MC	3	30	7.49 ± 1.46	5.95–11.40	6.94–8.03
	5	30	10.98 ± 1.74	8.69–17.44	10.33–11.63
	10	30	21.72 ± 0.81	19.22–23.49	21.42–22.02
	30	30	89.13 ± 12.32	76.43–128.53	84.53–93.73
GA	3	30	14.71 ± 3.66	10.69–28.05	13.35–16.08
	5	30	21.27 ± 2.86	17.74–33.79	20.20–22.34
	10	30	44.49 ± 2.25	39.97–48.90	43.65–45.33
	30	30	198.49 ± 20.22	163.13–235.93	190.94–206.04

Note. The best-performing algorithm per condition is bolded in the Mean $\pm$ SD column. Req. E is the number of required edges and n is the number of valid trials out of 30. “-” means all trials timed out.

4.3. Extended Evaluation of the Genetic Algorithm

As shown in the preceding sections, the GA often exhibits the poorest performance, particularly as the number of required edges increases. To investigate the underlying causes of this behavior, an ablation study and parameter sensitivity analysis were conducted using the $G(V = 93, E = 187)$ graph with 30 required edges, for which the GA yielded the worst performance.

In the ablation study, crossover and mutation operators (inversion and/or reciprocal exchange) were systematically removed to evaluate the impact of individual operators and their combinations on GA performance. Each configuration, hereafter referred to as a setting, was evaluated over five trials with random seeds set from 1 to 5, ensuring identical starting nodes and required edge sets across all settings.

The results of the ablation study are shown in Figure 2. Although the setting without inversion achieved the smallest mean total distance, the performance of the full GA model was within the error bars of the setting without inversion. This indicates that the full GA already attains performance comparable to the best-performing ablated variant and that removing any single operator would not yield a meaningful improvement.

In the parameter sensitivity analysis, all parameters were incrementally tested to identify which parameters had the greatest impact on GA performance. The following variations of parameters were tested across three trials with the seed set from 1 to 3:

- Mutation rates = [0.0, 0.1, 0.25, 0.5, 0.75, 1.0];
- Crossover rates = [0.0, 0.1, 0.25, 0.5, 0.75, 1.0];
- Population sizes = [20, 50, 100, 150, 200];
- Generations = [10, 50, 100, 150, 200].

According to the results of the analysis presented in Figure 3, the mutation rate and number of generations were already close to their optimal values, whereas the crossover rate and population size were not. Using the same parameter settings as in the main experimental section, the NN achieved a mean distance of 8017.51 m with a runtime of 0.79 s, whereas the MC yielded a mean distance of 14,275.15 m with a runtime of 71.98 s. None of the GA parameter

combinations evaluated in the sensitivity analysis achieved comparable performance. The GA was subsequently re-tested using the same random seed, and the best-performing values were identified for each parameter (i.e., mutation rate = 0.25, crossover rate = 1.0, population size = 200, and generations = 50). Despite this tuning, the GA yielded a mean distance of 17,005.48 m with a runtime of 853.01 s, which still failed to outperform the NN and MC.

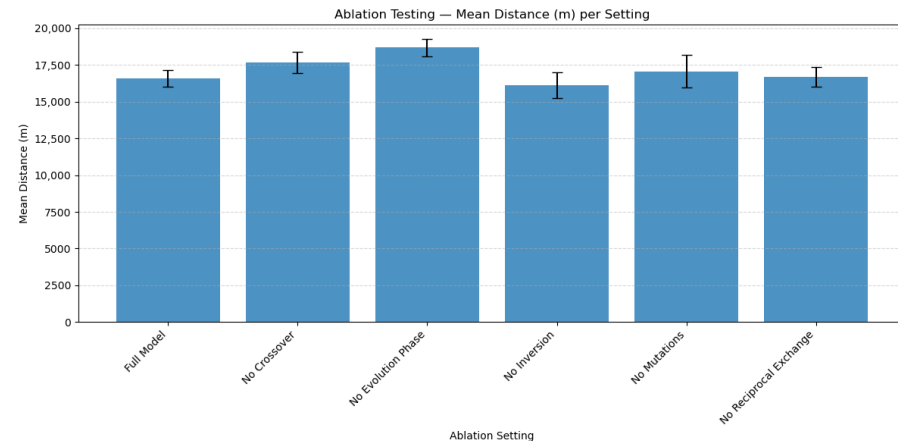


Figure 2. Ablation study: mean distance (m) of genetic algorithm per setting with sd error bars.

Although the population size analysis indicates a steady reduction in the mean distance as the population size increases, larger populations also incur substantially higher computational costs and runtimes, making this approach impractical. There are instances in which the GA outperformed the MC and NN (e.g., cases with five required edges in the graph $G(V = 93, E = 187)$), suggesting that parameter tuning may be beneficial for smaller or less complex instances. However, parameter selection alone does not explain the GA's poor performance when the number of required edges is large, indicating that other structural limitations dominate in more complex settings.

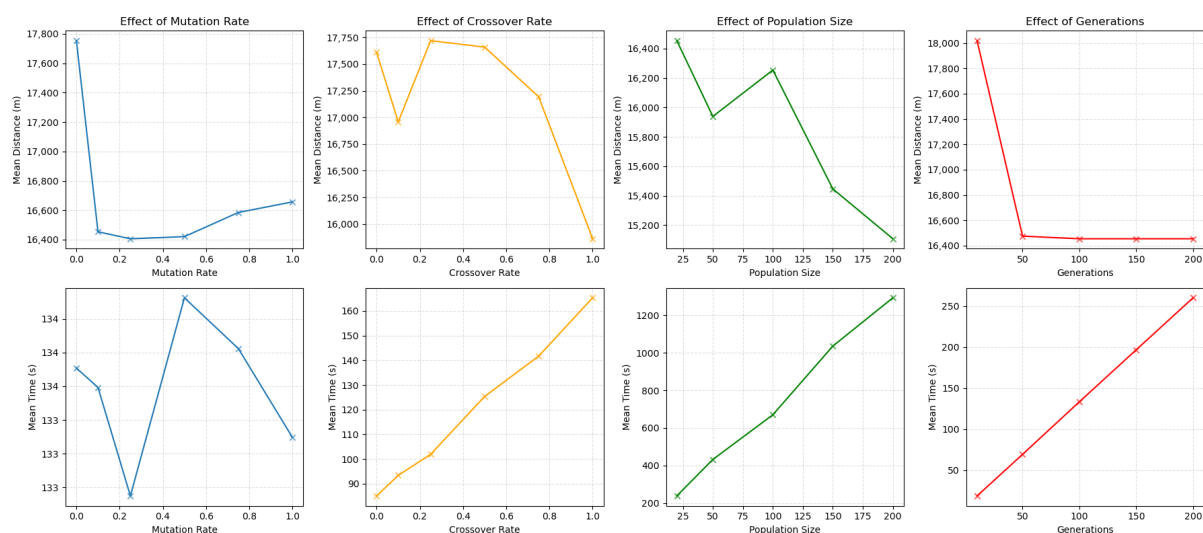


Figure 3. Parameter sensitivity analysis: mean distance (top) and mean total time (bottom) of genetic algorithm per parameter variation.

5. Discussion

This study developed novel heuristics and metaheuristics for the rural postman problem (RPP) by integrating the A* shortest-path algorithm within Neo4j and proposing a novel graph transformation method that preserves the required edges as edges rather than

converting them into nodes. In addition, an exhaustive search approach was implemented using Cypher queries alone to evaluate the inherent capability of graph database technology in solving the RPP. Overall, the Cypher algorithm (CA), for instances with a small number of required edges, and the nearest neighbor (NN) algorithm, for instances with a larger number of required edges, were the best-performing algorithms in terms of path optimality. The NN also demonstrated superior computational efficiency, completing all but one instance faster than the other algorithms. In contrast, the biased Monte Carlo (MC) algorithm and genetic algorithm (GA) underperformed in terms of both solution quality and runtime; the MC consistently produced the longest paths, whereas the GA required the greatest computational time. Consequently, the NN emerges as the most effective algorithm when both optimality and efficiency are considered.

The Cypher algorithm further demonstrated strong performance by consistently producing the shortest paths among all algorithms in instances where it did not time out. However, additional experiments revealed that the CA generally fails to produce feasible solutions when more than three required edges are required, even when the timeout limit is extended to 5400 s. The computational complexity of the Cypher-based exhaustive search, as shown in (6), can be decomposed into three main components. First, Cypher enumerates all possible paths of length L by selecting from d available edges, resulting in d^L . Second, each candidate path is checked to determine whether it contains all k required edges, which incurs a cost proportional to kL . Finally, all feasible paths are sorted by total distance, contributing to the logarithmic term.

$$d^L (kL + \log(d^L)) \quad (6)$$

Consequently, the search space of the Cypher-based RPP query grows exponentially with the number of required edges. Because Cypher's pattern-matching engine relies primarily on brute-force path enumeration and is not designed to handle exponential search spaces efficiently, the query becomes infeasible once the number of required edges exceeds a small threshold value. Therefore, improving performance necessitates the incorporation of more efficient search strategies within the Cypher framework, such as Neo4j's built-in graph algorithms. For example, `apoc.path.expandConfig` can partially mitigate the exponential term (d^L) by efficiently enumerating paths of required lengths and relationship types from a given start node. Consequently, Cypher alone has inherent limitations in solving the RPP for larger or more complex instances. Nonetheless, its strong performance in small-scale cases and the effectiveness of Cypher-integrated approaches highlight the potential of Neo4j and Cypher for RPP applications within appropriate constraints.

The proposed graph transformation method precludes direct comparison with existing studies, because no prior study has implemented a Cypher-based approach to the RPP. Consequently, direct benchmarking of the CA against the literature is not possible. Nevertheless, the general performance trends are comparable to the exploration–exploitation trade-off [10]. The NN represents an exploitation-extreme strategy, the GA an exploration-extreme strategy, and the CA an exhaustive search beyond conventional exploration. As expected, the NN was the fastest algorithm, the GA was the slowest, and the CA yielded the most optimal solutions when it did not exceed the time limit. However, it is counterintuitive that the exploitation-heavy NN consistently outperformed the more exploration-oriented MC and GA in terms of solution optimality. Given that the MC is derived from the NN with added randomization to increase exploration, this result is particularly unexpected. Moreover, the GA occasionally produced the least optimal solutions despite evaluating twice as many candidate paths as the MC (2000 paths for the GA versus 1000 for the MC). These observations suggest that both the MC and GA were unusually ineffective and inefficient for the considered RPP instances.

The ablation study and parameter sensitivity analysis of the GA further showed that removing evolutionary parameters (crossover and/or mutation) degraded performance relative to the full model, and parameter tuning alone could not outperform the MC and NN. These results indicate that these parameters are not the primary cause of the GA's underperformance. Although increasing the number of generations generally improved solution quality, the experimental results demonstrated that randomly sampling many paths could not outperform a single locally optimal path produced by the NN. This suggests that the GA's underperformance is not primarily due to parameter selection but rather to excessive randomization in its path construction process, which leads to inefficient and overly exploratory search behavior.

The MC exhibited high variance in solution quality, reflecting its broad exploration of the solution space. However, because its sampling process is unguided and independent across iterations, convergence toward high-quality solutions largely depends on chance. One potential improvement is to bias future sampling on the basis of previously successful solutions. For example, ant colony optimization employs a similar stochastic sampling mechanism as the MC but incorporates pheromone updates to reinforce high-quality paths and guide future exploration [21].

In contrast, the GA exhibited relatively low variance in solution quality, comparable to that of the NN, but consistently produced longer paths. This indicates restricted exploration, which converges prematurely to suboptimal regions of the search space, possibly owing to an ineffective initial population. To address this issue, the initial population can be locally optimized by incorporating other optimization methods, known as hybrid approaches. For instance, ref. [30] combined simulated annealing and the greedy randomized adaptive search procedure (GRASP) with the GA to generate high-quality initial populations, achieving optimal solutions in 25 out of 26 cases and outperforming MC.

For both the MC and GA, the quality of the initial solution plays a critical role in determining the final performance. Therefore, initializing these algorithms with locally optimal solutions (e.g., NN-generated paths) may substantially improve the results. In addition, because the paths were generated randomly without reference to previously explored solutions, duplicate paths may have been evaluated multiple times. Preventing the generation of previously explored paths represents another promising direction for improving both the efficiency and solution quality. Overall, these suggest that the underperformance of MC and GA arises primarily from algorithmic inefficiency rather than from the use of graph databases or Cypher, highlighting the need for further investigation into their application to the RPP.

In addition, it is important to note that this study is predicated on the claim that graph databases are better suited than relational databases for road network storage and routing problems. A key advantage of the graph database technology leveraged by all the proposed algorithms is the use of efficient predefined shortest-path algorithms in Neo4j. According to [7], Neo4j computed shortest paths in less than half the time required by MySQL across all instances for the graph $G(V = 354, E = 926)$, which is the same as one of the graphs used in this study ($G(V = 354, E = 926)$). Because shortest-path computations are performed repeatedly for combinations of the required edges, this performance advantage can lead to substantial reductions in the overall computational time. Moreover, this performance benefit persisted as the graph size increased, indicating good scalability and generalizability to larger and more realistic road network graphs.

Although these observations support the use of graph databases for routing problems, a direct empirical comparison between graph and relational database implementations would more rigorously validate this claim and quantify the true performance gains. Additionally, comparisons with other non-relational databases, such as MongoDB, which

has previously been applied to the traveling salesperson problem [31], could further assess the broader applicability of non-relational database technologies for the RPP. For these types of extensive comparative studies, unlike the comparison in this study which is purely descriptive, a statistical evaluation should also be conducted to assess the robustness and generalizability of the observed differences.

Furthermore, future work should apply graph databases and query languages to other RPP algorithms. Exact methods, such as branch-and-cut, can leverage graph databases to optimize initial solution space generation, mitigating their high computational costs. Similarly, graph convolutional networks (GCNs), which outperform many metaheuristics while requiring less computational time [32], could benefit from graph database integration, for example, via Neo4j's GDS library for node embeddings. Real-world applicability should also be explored, as the road network graph in this study was simplified for algorithm development. This includes using larger road network data, incorporating practical features (e.g., road availability, traffic volume, road width) as edge properties to refine edge selection and enabling customized weights [5].

Author Contributions: Conceptualization, K.H. and A.B.; methodology, analysis, and investigation, K.H.; writing—original draft preparation, K.H.; writing—review and editing, K.H. and A.B.; visualization, K.H.; supervision, A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by HERE Europe B.V. Kennedyplein 222-226, 5611 ZT Eindhoven, Netherlands, and the payment was managed through the Consumer Data Research Centre (CDRC). This is part of their Master's Dissertation Scheme (<https://www.cdrc.ac.uk/education-and-training/masters-dissertation-scheme/>, accessed on 21 February 2024) organized by the CDRC.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The dataset, NavStreets, was provided by HERE Europe B.V. Kennedyplein 222-226, 5611 ZT Eindhoven, Netherlands, however, this is publicly unavailable due to intellectual property rights.

Acknowledgments: We are grateful to HERE Europe B.V. Kennedyplein 222-226, 5611 ZT Eindhoven, Netherlands, for providing the dataset and digital mapping expertise. In particular, we sincerely thank Ákos Magdó for his valuable feedback and insightful contributions to this study. During the preparation of this manuscript, the authors used generative AI tools (i.e., ChatGPT versions 4o and 5, Paperpal) for proofreading and aiding in understanding the relevant materials. The authors have thoroughly reviewed and edited all generated content and takes full responsibility for the final version of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Transport Statistics Great, B. *Transport Statistics Great Britain: 2022 Freight*; GOV.UK: Great Britain, **2023**.
2. Eiselt, H.A.; Gendreau, M.; Laporte, G. Arc Routing Problems, Part II: The Rural Postman Problem. *Operations research* **1995**, *43*, 399–414.
3. Corberán, Á.; Laporte, G. *Arc Routing: Problems, Methods, and Applications*; Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, **2015**.
4. Orloff, C.S. A fundamental problem in vehicle routing. *Networks* **1974**, *4*, 35–64.
5. Bachechi, C.; Po, L. Road network graph representation for traffic analysis and routing. In *European Conference on Advances in Databases and Information Systems*; Springer: Cham, Switzerland **2022**; pp 75–89.
6. Miler, M.; Medak, D.; Odobašić, D. The shortest path algorithm performance comparison in graph and relational database on a transportation network. *Promet-Traffic Transp.* **2014**, *26*, 75–82.
7. Patras, V.; Laskas, P.; Koritsoglou, K.; Fudos, I.; Karvounis, E. A comparative evaluation of RDBMS and GDBMS for shortest path operations on pedestrian navigation data. In *2021 6th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECSM)*, Preveza, Greece, 24–26 September **2021**; pp 1–5.

8. Drexel, M. On the generalized directed rural postman problem. In *Journal of the Operational Research Society* **65**, no. 8 **2014**, 1143–1154.
9. Lenstra, J.K.; Kan, A.R. On general routing problems. *Networks* **1976**, *6*, 273–280.
10. Xu, J.; Zhang, J. Exploration-exploitation tradeoffs in metaheuristics: Survey and analysis. In *Proceedings of the 33rd Chinese control conference*, Nanjing, China, 28–30 July **2014**; pp 8633–8638.
11. Ferreira, J.A.S.; do Rosario Pereira, M. A Genetic Algorithm for the Undirected Rural Postman Problem. **2010**. Available online: <https://hdl.handle.net/10216/19839> (accessed on 15 August 2024).
12. Goldberg, A.V.; Harrelson, C. Computing the shortest path: A search meets graph theory. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, Vancouver, BC, Canada, 23–25 January **2005**; pp 156–165.
13. Christofides, N.; Campos, V.; Corberán, A.; Mota, E. An algorithm for the rural postman problem. In *Imperial College Report* **1981**, *81*, 81.
14. Avila, T.; Corberán, Á.; Plana, I.; Sanchis, J.M. A new branch-and-cut algorithm for the generalized directed rural postman problem. *Transportation Science* **2016**, *50*, 750–761.
15. Frederickson, G.N. Approximation algorithms for some postman problems. In *Journal of the AC* **1979**, *26*, 538–554.
16. Christofides, N.; Campos, V.; Corberán, A.; Mota, E. An algorithm for the rural postman problem on a directed graph. In *Netflow at Pisa*; Springer: Berlin/Heidelberg, Germany, **1986**; pp. 155–166.
17. Muyldermans, L.; Beullens, P.; Catrysse, D.; Van Oudheusden, D. Exploring variants of 2-opt and 3-opt for the general routing problem. *Operations research* **2005**, *53*, 982–995.
18. De Córdoba, P.F.; Raffi, L.G.; Sanchis, J.M. A heuristic algorithm based on Monte Carlo methods for the rural postman problem. *Computers & operations research* **1998**, *25*, 1097–1106.
19. Adnan, S.; Abood, E.W.; Abdulmuhsin, W. The multi-point delivery problem: Shortest Path Algorithm for Real Roads Network using Dijkstra. *Journal of Physics: Conference Series* **2020**, *1530*, 012040.
20. Kang, M.-J.; Han, C.-G. Solving the rural postman problem using a genetic algorithm with a graph transformation. In *Proceedings of the 1998 ACM symposium on Applied Computing*, Georgia, GA, USA, 7 February 1998 - 1 March **1998**; pp 356–360.
21. Pérez-Delgado, M.L. A solution to the rural postman problem based on artificial ant colonies. In *Conference of the Spanish Association for Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, **2007**; pp 220–228.
22. Corberán, A.; Martí, R.; Romero, A. Heuristics for the mixed rural postman problem. *Computers & operations research* **2000**, *27*, 183–203.
23. Park, S.; Cheng, T. Framework for constructing multimodal transport networks and routing using a graph database: A case study in London. *Transactions in GIS* **2023**, *27*, 1391–1417.
24. Chen, J.; Song, Q.; Zhao, C.; Li, Z. Graph database and relational database performance comparison on a transportation network. In *International Conference on Advances in Computing and Data Sciences*; Springer: Singapore **2020**; pp 407–418.
25. Ariadi, A.; Shi, T.; Ma, H.; Da Silva, A.S.; Hartmann, S. A graph database supported GA-based approach to social network analysis. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, Orland, USA, 5–7 December **2021**; pp 1–8.
26. Trueman, J.; Ma, H.; Chen, A.; Hartmann, S. Multi-objective Social Network Detection—A Graph Database Supported NSGA-II Based Approach. In *Australasian Database Conference*; Springer: Cham, Switzerland **2022**; pp 21–33.
27. Chopde, N.R.; Nichat, M. Landmark based shortest path detection by using A* and Haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering* **2013**, *1*, 298–302.
28. zachdj. *rpp-algorithms/MonteCarlo.py at Master*; GitHub: USA, **2017**.
29. zachdj. *rpp-algorithms/GeneticAlgorithm.py at Master*; GitHub: USA, **2017**.
30. de la Peña, M.G.B. Heuristics and metaheuristics approaches used to solve the rural postman problem: A comparative case study. In *Proceedings of the Fourth International ICSC Symposium on Engineering of Intelligent Systems (EIS 2004)*, Madeira, Portugal, 29 February–2 March **2004**.
31. Lattu, D.; Ahirkar, V.; Pisupati, K.S. Shopping Planner, **2016**. Available online: https://sailakshmipisupati.github.io/assets/docs/Shopping_Planner_Final_Paper.pdf (accessed on 15 August 2024).
32. Li, H.; Li, G. Learning to solve capacitated arc routing problems by policy gradient. In *IEEE Congress on Evolutionary Computation*, Wellington, New Zealand, 10–13 June **2019**; pp 1291–1298.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.