Vehicle Routing Problem using Meta-Heuristics: A Collaboration with Fermanagh Community Transport

Kai Quan Lian

Project advisor: Dr Gareth A. Tribello

December 17, 2025

School of Mathematics and Physics

Queen's University Belfast

Abstract

The Vehicle Routing Problem (VRP) is a central challenge in combinatorial optimization, focusing on the assignment of optimal routes for multiple vehicles under a variety of constraints. This paper is in collaboration with Fermanagh Community Transport (FCT), an organization delivering essential mobility services across rural Fermanagh in Northern Ireland. Their testimonial, highlighting the impact of our work on their operations, follows immediately after. Three distinct methods for solving the VRP, that is Linear Programming (LP), Variable Neighbourhood Search (VNS), and Simulated Annealing (SA) are explored in depth, with detailed methodology and self-developed programming algorithms benchmarked against leading VRP solver tools. While LP guarantees mathematically exact solutions, it quickly becomes computationally prohibitive for larger instances. In contrast, VNS and SA demonstrate robust performance by balancing solution quality with practical runtime demands. In addition to algorithmic development, this study examines whether high-demand users or 'frequent flyers' have a significant impact on FCT's overall service provision. Numerical simulations on real-world data highlight the substantial potential of meta-heuristics to optimize route planning, lower operational costs, and expand service availability for logistics and transportation services.

Practical Implications of VRP Optimization: A Testimonial from Jason Donaghy, CEO of Fermanagh Community Transport

This is an incredible piece of work. Kai's examination and simulation of FCT's passenger data through the meta-heuristics approach has been a powerful exercise in testing our internal and experiential led intuitive approach to scheduling. The degree of apparent correlation between the meta-heuristic driven output and the petal-like strategy we currently deploy has been incredibly reassuring. It tells us, as an organisation that is a relatively small charitable company committed to providing access to basic and essential services in deeply rural Fermanagh for predominantly elderly and disabled people, that we are not that far off the mark in terms of route optimisation. To have our scheduling approach tested and scrutinised in this way helps build confidence and demonstrates that we are doing our very best to maximise opportunities for travel whilst keeping costs lean.

The research from the perspective of the 'frequent flyers' versus 'every member is served equally' shows considerable correlation. This reflects the areas of highest transport poverty in the rural reaches of the county, which are illustrated by the distribution and location of our registered members/passengers. The parameters are the same in terms of vehicles, capacity, and time in both scenarios. What would be interesting is to explore further the merging of the demand from the frequent flyers and the potential for more and greater non-frequent flyers, and what this would mean for vehicle numbers, capacity, and time. The analysis with a maximum 90-minute time factor would bring even greater proximity to reality in terms of maximum journey time for any passenger. Kai's research also demonstrates that the Vehicle Routing Problem is a problem for which it is challenging to find a truly optimum answer. The theoretical is complicated by the reality of geography, topography, and demography, as well as the continued limitations of the reliability of Google reporting data.

Contents

1 Introduction

The Vehicle Routing Problem (VRP) is a foundational challenge in logistics and transportation sectors, centred on the task of determining optimal routes for multiple vehicles to visit a set of locations. Our initial encounter with the VRP took place during the first semester of the final year of BSc (Hons) Mathematics undergraduate program at Queen's University Belfast, specifically in the SOR 3012 module on Stochastic Processes and Risk. This paper extends earlier work in partnership with Fermanagh Community Transport (FCT) [23], a provider of essential transport services for individuals in rural and socially isolated regions across County Fermanagh in Northern Ireland. Fermanagh is predominantly rural, with scattered communities that necessitate dependable mobility solutions. FCT primarily serves older adults, people with disabilities, and those with limited access to conventional transport options. Among its key offerings are the Dial-A-Lift service and the Home to Hospital transport scheme, both of which help vulnerable populations maintain their independence and reach critical destinations such as healthcare facilities. A fleet of minibuses and volunteer-driven cars underpins FCT's operations, but the rural expanse and spread-out service areas present significant challenges for route planning. A cost-effective approach to vehicle routing is therefore central to FCT's mission and financial sustainability. In my previous semester project [16], I proposed an initial, high-level approach tailored to FCT's operational requirements, highlighting the promise of specialized VRP solutions in real-world settings.

In the present study undertaken as part of the AMA 3011 Applied Mathematics Project, we delve more deeply into the VRP with a major emphasis on meta-heuristic methods. These methods have shown significant promise in handling complex and large-scale optimization problems, making them particularly relevant for real-world scenarios such as public transportation in rural communities. Building on our previous collaboration with Fermanagh Community Transport (FCT), we incorporate real operational data gathered from FCT's historical customer records. By applying meta-heuristic approaches to these datasets, we frame our approach as a tool for exploration and comparison, allowing FCT to assess the effectiveness of these methods as part of a broader strategy to enhance their operational efficiency. In particular, we investigate whether customer demand patterns or the batching of trips is having a marked effect on FCT's ability to deliver services to members. We compare an optimised journey that incorporates every request that has been made for the service over the last 11 months, with an optimised journey that services every member who has requested equally. By comparing these two journeys, we can thus determine whether so-called frequent flyers are having a marked effect on FCT's ability to serve all members. Through more efficient vehicle assignments and scheduling, FCT stands to reduce operational costs while improving overall service quality and customer satisfaction, which are key metrics in enabling rural populations, including older adults and individuals with mobility challenges, to maintain vital connections within and beyond County Fermanagh.

The paper is structured as follows. In Section 2, I conduct a concise literature review, outlining relevant background, established approaches, various branches, and the many complexities involved in the VRP. Section 3 delves into the specific constraints and extensive methodologies of the three meta-heuristic approaches I have implemented in this project, also detailing how these methods can be encoded in computer programs. Section 4 then offers a comparison and benchmarking of these three approaches, including a demonstration of their performance on real-world operational data provided by FCT. This comparison also assesses how the optimal routes returned from our algorithms stack up against FCT's existing routing strategies. Section 5 discusses the limitations of our methodologies, highlighting model constraints, parameter sensitivity, and practical challenges in real-world implementation. Finally, in Appendix 7, the complete programming scripts used throughout this research are presented, along with the FCT datasets and detailed output results of these scripts.

2 Literature Review

The Vehicle Routing Problem (VRP) is one of the most important, and studied, combinatorial optimization problems [22]. It involves determining the most optimal delivery or collection routes from a central depot to a set of geographically dispersed customers while adhering to constraints such as vehicle capacity, route length, time windows, and precedence relations [15]. The VRP is NP-hard because it includes the Travelling Salesman Problem (TSP) as a special case when there is only one vehicle with infinite capacity, and in practice, the VRP is considerably more difficult to solve than a TSP of the same size [15]. TSP itself is a well-known NP-complete combinatorial optimization problem whose purpose is to find the shortest tour covering each city once and only once [28]. The VRP is a fundamental problem in logistics and operational research and carries great economic value as it influences operational costs and service efficiency directly. Real-life requirements often include time window restrictions, travel costs, and capacities, multi-dimensional capacity constraints, order/vehicle compatibility constraints, orders with multiple pickup, delivery, and service locations, different start and end locations for vehicles, and route restrictions [8]. A large number of real-world applications have widely shown that the use of computerized solution methods for the solution of VRP, both at the planning and the operational levels, yields substantial savings in global transportation costs [22].

With the effort of more than six decades, a large number of VRP variants with real-world attributes have been studied [17]. Vehicle Routing Problems with Time Windows (VRPTWs) specify that customers must be visited within predefined time intervals [2]. The Capacitated Vehicle Routing Problem (CVRP) is the most studied version of the VRP [22], focusing on

respecting a strict vehicle capacity limit when serving customers. In the General Vehicle Routing Problem (GVRP) [8], a transportation request is specified by a non-empty set of pickup, delivery, and/or service locations which must be visited in a particular sequence by the same vehicle, possibly within given time windows, and yield a certain revenue when served. The Vehicle Routing Problem with Simultaneous Delivery and Pickup (VRPSDP) [24] extends the classical VRP by requiring vehicles to handle both deliveries and pickups during the same route, ensuring that items can be dropped off and collected in one integrated schedule. The Vehicle Routing Problem with AND/OR Precedence Constraints [20] addresses an additional restriction where each node's AND-type predecessors must be visited first, and at least one OR-type predecessor must precede it as well. Meanwhile, the Multi Trip Vehicle Routing Problem (MTVRP) allows a vehicle to perform several routes within a given time period, and the Vehicle Routing Problem with Backhauls (VRPB) permits picking up goods to bring back to the depot once deliveries are made. Consequently, the Multi Trip Vehicle Routing Problem with Backhauls (MTVRPB) [25] combines these characteristics, i.e., vehicles may perform multiple trips and collect goods in each trip. Another important extension is the Capacitated Vehicle Routing Problem with Two-Dimensional Loading Constraints (2L-CVRP) [26], in which a set of two-dimensional, rectangular, weighted items must be packed into vehicles without violating spatial feasibility or capacity. These and other constraints can also be implemented simultaneously, introducing additional layers of complexity when modelling real-world conditions.

A key consideration when formulating the VRP is defining the objectives to be optimized. In many cases, the goal is a single objective, commonly minimizing total routing costs. This can be achieved by assigning zero or prohibitively high costs to certain edges to exclude or discourage them. Besides, the objective function can incorporate profits or fixed costs to reflect the expense of using particular types or numbers of vehicles. Another objective is to minimise the total idle time of vehicles, thereby incentivising vehicles to remain actively engaged in profitable work [1]. Additional factors such as driver wages, route durations, and waiting times can be included when time scheduling and service quality play an important role, for example in humanitarian applications or in industries where customer satisfaction is crucial [22]. Beyond single-objective formulations, objectives in VRP can also be arranged hierarchically, thus requiring algorithms to find efficient trade-offs rather than a single global optimum. In recent years, there has been a growing emphasis on environmental considerations and corresponding real-world cost functions in vehicle routing. The Green Vehicle Routing Problem (GVRP) [4] expands the standard VRP framework by explicitly incorporating environmental externalities into route planning. This increased focus on eco-friendly transportation reflects the broader intensification of interest in sustainable logistics, and it is increasingly crucial for organizations striving to meet green targets without compromising service quality.

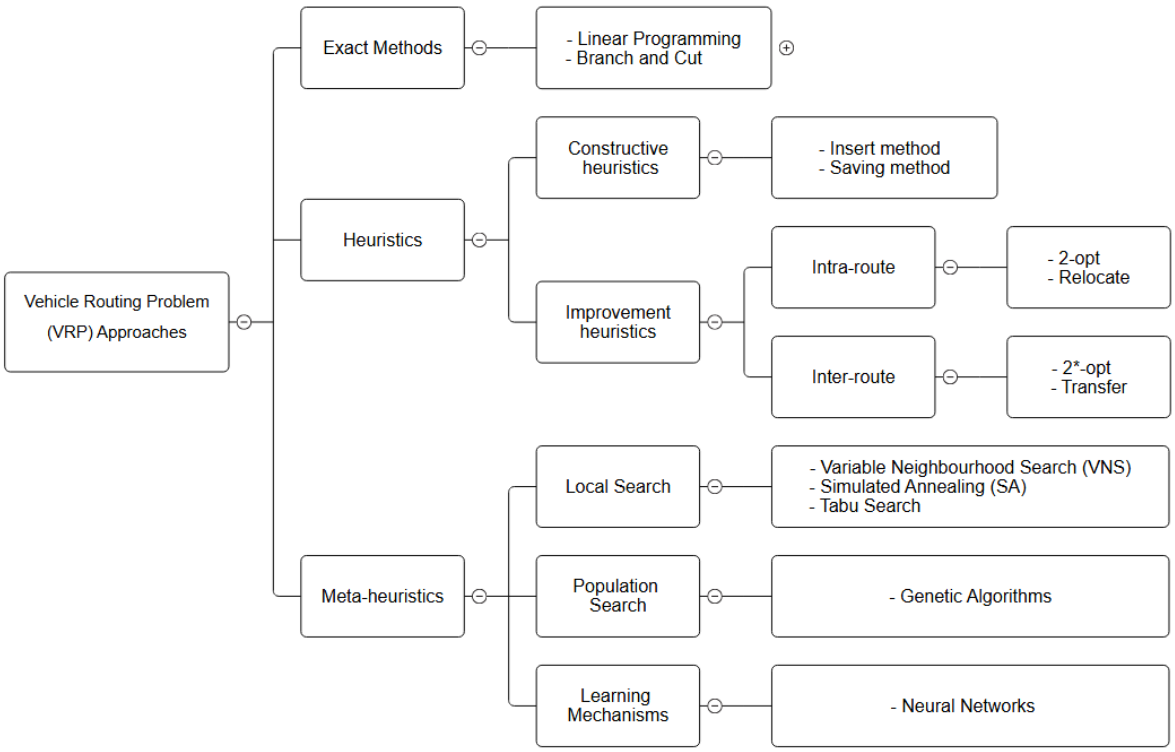

Figure 1: Methods of Vehicle Routing Problem

Figure 1 illustrates the diverse range of approaches for solving the VRP. Among these, exact methods offer a mathematically rigorous approach with guaranteed optimality. However, they often struggle with the computational complexity of large-scale, real-world VRP instances. As problem size and constraints grow, exact methods become increasingly impractical due to their exponential computational requirements. In contrast, heuristic approaches provide high-quality solutions within a reasonable time frame, making them more suitable for real-world applications. Heuristics are designed to find near-optimal solutions efficiently by using problem-specific rules and approximations rather than exhaustive enumeration. Building on this concept, meta-heuristic methods offer a more advanced framework by incorporating adaptive strategies that enhance heuristic performance. Over the past decades, significant research has been devoted to refining and expanding heuristic and meta-heuristic techniques to improve solution quality and computational efficiency [17].

Linear Programming (LP) is a widely used exact method for solving the VRP, where the problem is formulated using mathematical inequalities that define constraints, decision variables, and an objective function. The goal is to find a globally optimal solution that satisfies all constraints while minimizing or maximizing the objective function. This is typically achieved by solving a system of linear equations using optimization solvers in programming languages like Python. The solution space is explored systematically to ensure that the obtained result

is truly optimal. One of the advanced exact methods for solving VRP is the Branch and Cut algorithm [27]. In this approach, the solution space is recursively divided (branching) into smaller sub-problems, while additional constraints (cuts) are introduced to eliminate infeasible regions and accelerate convergence. This method enhances computational efficiency by strategically pruning the search tree, thus avoiding unnecessary calculations. This paper will cover the mathematical theory of LP as an exact method for VRP, with detailed documentation and demonstrations available in [16]. However, as the problem size increases, the computational complexity grows exponentially, making exact methods impractical for large-scale instances. In such cases, heuristic methods must be employed to provide high-quality solutions within a reasonable time frame.

Heuristic methods aim to find near-optimal solutions efficiently by employing problem-specific strategies rather than exhaustively searching for a global optimum. Heuristics are generally divided into two main categories: constructive heuristics and improvement heuristics. Constructive heuristics build a feasible solution from scratch. The Savings Method starts with individual routes for each customer and iteratively merges routes that yield the highest cost savings while maintaining feasibility [14]. Similarly, the Insert Method follows a greedy approach by selecting and inserting the next node that results in the minimal cost increase at each iteration. On the other hand, improvement heuristics refine an existing solution by modifying routes to enhance efficiency. These are further categorized into intra-route and inter-route methods. Intra-route heuristics operate within a single route, such as Relocate, which moves a node to a different position within the same route, and 2-opt, which reconnects two edges within the same route to reduce travel distance. Inter-route heuristics modify multiple routes, such as 2-opt* selects two routes and swaps edges between them, while Transfer moves a node from one route to another, aiming for an overall cost reduction.

Meta-heuristics are high-level problem solving strategies that enhance heuristic methods by guiding the search process toward improved solutions. These approaches are particularly useful for solving complex combinatorial optimization problems such as the VRP, where exact methods become computationally impractical. Unlike simple heuristics, meta-heuristics incorporate mechanisms for exploration and exploitation, allowing them to navigate large solution spaces efficiently while avoiding local optima. Given the increasing number of VRP meta-heuristics published in recent years, as well as their growing complexity, providing a comprehensive review of all related literature is challenging [14]. However, these methods can be broadly classified into local search, population search, and learning mechanisms. Learning-based Mechanisms, such as Neural Networks, have gained attention for their ability to approximate solutions to routing problems. These networks process problem instances such as node locations and distances to generate feasible routing sequences. Reinforcement learning techniques can further refine these

solutions through iterative trial and error [13]. Additionally, hybrid approaches integrate neural networks with traditional meta-heuristics or optimization algorithms to enhance performance. For instance, neural networks can improve the initialization phase of an optimization algorithm or assist in guiding search strategies toward promising regions of the solution space. Another prominent category of meta-heuristics is Population-based Search methods, with Genetic Algorithms (GA) being one of the most widely studied approaches [11]. GA operate by evolving a population of candidate solutions through a process of selection, crossover, and mutation. Parent solutions are recombined to create offspring, which replace weaker individuals in the population if they offer improvements. A mutation mechanism is applied to maintain diversity and prevent premature convergence. By balancing exploration and exploitation, GA efficiently navigate large search spaces, making them particularly well-suited for solving complex VRP instances. These meta-heuristic strategies provide robust and adaptable solutions to VRP, outperforming traditional heuristics in many real-world applications.

Local search techniques are fundamental in meta-heuristic optimization for VRP, as they iteratively explore the solution space by moving from a current solution to a better one within its neighbourhood [14]. One prominent local search method is Variable Neighbourhood Search (VNS). Unlike traditional local search algorithms that follow a fixed trajectory through the solution space, VNS systematically explores increasingly larger and diverse neighbourhoods. It moves to a new solution only if an improvement is found, allowing the algorithm to retain favourable characteristics of the current solution while efficiently navigating the search space [18]. This adaptive exploration enables VNS to escape local optima and discover high-quality solutions. Several variants of VNS have been developed to tackle specific VRP challenges. For example, a Two-Level VNS algorithm has been proposed for large instances of the MT-VRPB. This approach integrates sequential Variable Neighbourhood Descent (VND) and a multi-layer local search, balancing intensification and diversification throughout the search process [25]. Another significant advancement in local search is the Very Large Scale Neighbourhood (VLSN) Search. VLSN methods focus on exploring neighbourhoods that are exponentially large relative to the problem size. By utilizing compounding independent moves (CIM) like 2-opts, swaps, and insertions, VLSN efficiently handles complex routing problems with additional constraints [5]. Another well-known local search technique is Tabu Search, which functions as a meta-heuristic framework built on top of a heuristic approach. It improves the search process by forbidding or penalizing certain moves i.e., tabu moves, to prevent cycling and escape local optima [7]. Beyond simple move restrictions, Tabu Search incorporates additional mechanisms such as diversification, which penalizes frequently encountered attributes to explore new areas of the search space, and intensification, which performs a deeper search around promising solutions [15]. A widely used implementation, TABUROUTE, enhances traditional Tabu Search by allowing infeasible solutions through penalty terms in the objective function, thereby reducing

the likelihood of being trapped in poor local optima. Additionally, it integrates the GENI insertion heuristic to generate improved routes while introducing periodic perturbations, further enhancing solution quality [6]. Tabu Search has also been adapted to dynamic and real-world scenarios. For instance, a hyper-heuristic algorithm based on Tabu Search has been designed to optimize vehicle routing under time-dependent traffic conditions, balancing distribution costs and customer satisfaction by efficiently selecting search operators through a high-level heuristic strategy [24]. Simulated Annealing (SA) is a highly effective meta-heuristic inspired by statistical mechanics, particularly the annealing process in solids. This connection provides a structured framework for optimizing complex combinatorial problems by gradually refining solutions [12]. Unlike traditional local search methods, SA probabilistically accepts worse solutions, allowing it to escape local optima and explore a broader solution space before converging to an improved solution. The implementation of SA follows the Metropolis algorithm, which relies on several key components. These include a clear representation of the system's configuration, a mechanism for generating random moves or perturbations, an objective function that accounts for trade-offs, and an annealing schedule that determines how the system transitions from exploration to exploitation over time [12]. The cooling schedule is particularly important, as it controls the rate at which the system settles into an optimal or near-optimal solution. Various adaptations of SA have been developed to enhance its performance in VRP. One approach involves dynamically adjusting the temperature by periodically increasing and decreasing it, which helps balance the effort spent on different routes and prevents premature convergence [26]. This modification allows the search process to explore a diverse range of solutions while maintaining efficiency in large-scale problem instances. In this paper, we will provide a detailed discussion of Variable Neighborhood Search (VNS) and Simulated Annealing (SA) meta-heuristics, covering their methodologies, along with a demonstration of their application to VRP, illustrating how these meta-heuristic techniques effectively navigate complex solution spaces and enhance routing efficiency.

In this study, we explore three distinct approaches to solving the VRP, i.e., Linear Programming (LP), Variable Neighbourhood Search (VNS), and Simulated Annealing. These methods were selected for their well-established effectiveness and their complexity, which is suitable for undergraduate-level research. Linear Programming provides a mathematically rigorous approach, offering exact solutions under well-defined constraints. Its straightforward formulation allows for a clear representation of the optimization problem. However, due to its computational limitations with large-scale problems, heuristic and meta-heuristic approaches are necessary to find high-quality solutions more efficiently. VNS was chosen for its adaptability and systematic exploration of different neighbourhoods, allowing it to escape local optima and find improved solutions efficiently. Simulated Annealing on the other hand, introduces controlled randomness to explore diverse solutions while gradually refining the search towards optimal or near-optimal

results. Its probabilistic acceptance of worse solutions helps to prevent premature convergence, making it particularly effective in handling the non-linear and combinatorial complexity of VRP. By implementing and comparing these three methods, this study aims to assess their relative performance, computational efficiency, and suitability for solving real-world VRP instances, including the operational data provided by FCT.

3 Methodology and Theory

This section establishes our analytical framework for vehicle routing optimization. In this study, two distinct approaches are utilized for solving the Vehicle Routing Problem (VRP): Linear Programming Model and Meta-Heuristics. We begin with Section 3.2: Linear Programming Model, presenting an exact mathematical formulation of the Capacitated Vehicle Routing Problem with Time Window, which provides theoretical optimality guarantees but becomes computationally prohibitive for large-scale instances. The formulation of the linear programming model is adapted from our paper [16], which also provides a comprehensive guide on solving a real world instance of the VRP explicitly using Python. In 3.3 and 3.4, we transition to meta-heuristic approaches, i.e., Variable Neighbourhood Search (VNS) and Simulated Annealing (SA) algorithms. These methods combine constructive heuristics for initial solution generation with improvement heuristics for neighbourhood optimization, employing strategic perturbation mechanisms to escape local optima. We implement a hybrid strategy incorporating both intra-route and inter-route neighbourhood structures.

3.1 Assumptions and Constraints

The two methods, linear programming and meta-heuristics are based on specific assumptions and constraints that influence the formulation and solution process. Both methods are formulated based on the following assumptions. An instance of the described VRP model is illustrated in Figure 2.

- There are n nodes in the system, denoted $\{i : i = 0, 1, 2, \ldots, n-1\}$.
- There are n_k serviceable vehicles in the system, denoted $\{k : k = 0, 1, 2, \ldots, n_k - 1\}$.
- The travel times between node i and node j are written in a matrix t_{ij}.
- All vehicles depart from the same node, referred to as the departure node.
- All vehicles end their trip at the same node, designated as the terminating node.
- The customers' demand d_i at every node must be satisfied.
- Each customer node must be visited exactly once by a vehicle.

- The departure and terminating nodes are often called the depots, both have zero demand.
- The maximum number of vehicles available for service is n_k. Note that the optimal solutions may not require the utilization of all available vehicles.
- Each vehicle has the same fixed capacity Q, meaning the total demand of the customers visited by a single vehicle in a trip must not exceed Q.
- The total routing time of each vehicle in a single trip must not exceed the temporal constraint T, ensuring all customers reach their destination on time.
- For the linear programming model, one extra constraint is imposed, which is the time window constraint. The arrival time of the vehicles must satisfy all customers' specified time windows $[e_i, l_i]$, where e_i and l_i are the earliest and latest permissible arrival time at node i respectively. Vehicles are allowed to arrive earlier than the specified time window, with the waiting time accounted for in the total route time.
- The total costs incurred is directly proportional to the travel time for all vehicles. Hence the objective of the model is to minimize the total travel time across all vehicles.

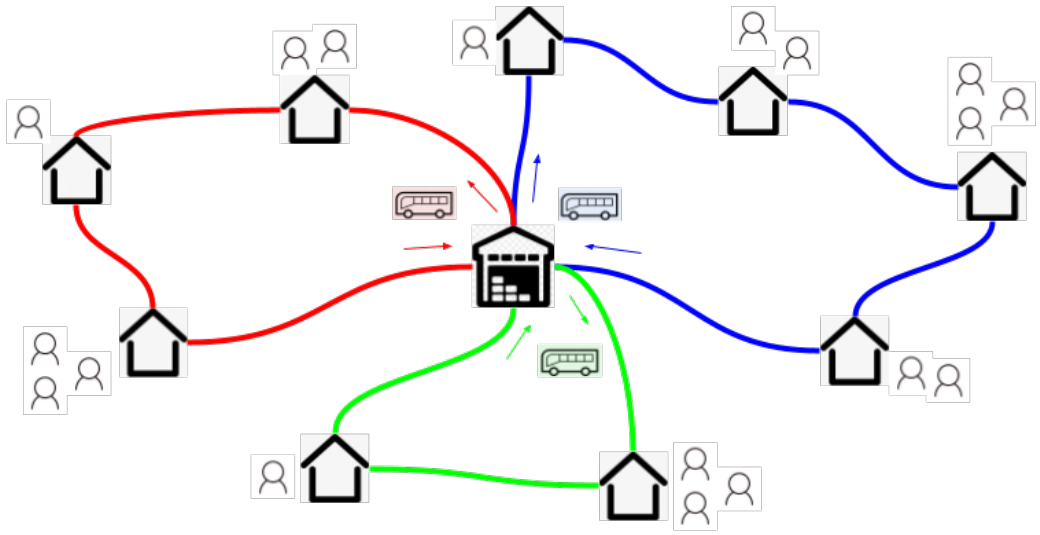

Figure 2: A vehicle routing instance with $n = 9$ nodes, $n_k = 3$ vehicles, and varying demands at each node.

However, in this paper, we apply different settings for these two approaches due to specific methodological limitations. The key differences and similarities between the linear programming model and the meta-heuristics approach are summarized in Table 1.

Table 1: Comparison of constraints and assumptions between two approaches.

Aspect	**Linear Programming**	**Meta-Heuristics**
Time Window Constraint	Imposed	Not imposed
Departure Node	Node $i = 0$	Node $i = 0$
Terminating Node	Node $i = n - 1$, can be set independently	Node $i = 0$, set to be identical for both depots

3.2 Linear Programming

The linear programming model is designed to minimize the total travel time of all vehicles while ensuring that various operational constraints are met. This model utilizes a combination of binary and continuous decision variables to represent vehicle routes, node visits, and arrival times. The objective is to find the routing plan with the least total travel time, that satisfies capacity, time windows, temporal constraint and other service requirements. The decision variables implemented in the model are as follows:

- $x_{ijk} \in \{0, 1\}$: A binary decision variable that equals 1 if vehicle k travels from node i to node j, and 0 otherwise.
- $z_{ik} \in \{0, 1\}$: A binary decision variable that equals 1 if vehicle k visits node i, and 0 otherwise.
- $u_k \in \{0, 1\}$: A binary decision variable that equals 1 if vehicle k is used in the routing plan, and 0 otherwise.
- $a_{ik} \in \mathbb{R}^+$: A continuous variable that represents the arrival time of vehicle k at node i.
- $max\_a_k \in \mathbb{R}^+$: A continuous variable that represents the maximum arrival time for each vehicle k, i.e., the arrival time at the final drop station for vehicle k.

The objective function and constraints of the model, following the assumptions made in Section 3.1, are formulated below.

- **Objective Function:** The model aims to minimize the total travel time across all vehicles.
$$\min z = \sum_{k=0}^{n_k-1} \text{max\_a}_k$$
- **Initial and Final Conditions:** Every vehicle departs from the central hub at time 0 and visits the final drop node. The variable u_k in the subsequent equations ensures that the constraints apply only to the vehicles that are utilised.
$$a_{0k} = 0, \quad z_{0k} = u_k, \quad z_{n-1,k} = u_k \quad \forall k \in vehicles$$

- **End Node Constraint:** The arrival time at the final drop node must be greater than or equal to the arrival times at all customer nodes for each vehicle, ensuring that all vehicles conclude their routes at the drop station. The arbitrarily large constant, M enables the termination of the following inequality if vehicle k does not visit node i, by making the corresponding side of the inequality below to be arbitrarily small.

$$a_{n-1,k} \geq a_{ik} - M(1 - z_{ik}) \quad \forall i \in \{1, 2, \ldots, n-2\}, \forall k \in vehicles$$

- **Visit Constraint:** Each customer node is visited exactly once by one vehicle.

$$\sum_{k=0}^{n_k-1} z_{ik} = 1 \quad \forall i \in \{1, 2, \ldots, n-2\}$$

- **Capacity Constraint:** The total demand served by any vehicle in a trip must not exceed its capacity. Similar to Eq (3.2), the variable u_k ensures that the constraint applies only to the vehicles that are utilised.

$$\sum_{i \in nodes} d_i z_{ik} \leq Q u_k \quad \forall k \in vehicles$$

- **Time Flow and Sequence Logic:** The following inequality ensures that if vehicle k travels from node i to node j, the arrival time at node j is at least the arrival time at node i plus the travel time between the two nodes. Similar to Eq (3.2), the arbitrary large constant M makes the inequality ineffective if vehicle k does not travel from node i to node j.

$$a_{jk} + M(1 - x_{ijk}) \geq a_{ik} + t_{ij} - M(2 - z_{ik} - z_{jk}) \quad \forall i, j \in nodes, \forall k \in vehicles$$

  Also, the following constraint ensures that vehicles can only travel between nodes if they visit both nodes:

$$x_{ijk} + x_{jik} \geq 1 + M(z_{ik} + z_{jk} - 2) \quad \forall i, j \in nodes, \forall k \in vehicles$$

- **Time Window Constraint:** The arrival time at each customer node must lie within its specified time window. Again, the variable z_{ik} ensures that the constraint applies only to vehicle k which visits node i.

$$e_i z_{ik} \leq a_{ik} \leq l_i z_{ik} \quad \forall i \in nodes, \forall k \in vehicles$$

- **Temporal Constraint:** The maximum arrival time (i.e., the total routing time) for each vehicle k must not exceed T.

$$T \geq a_{n-1,k} \quad \forall k \in vehicles$$

- **Objective Variable:** The maximum arrival time (i.e., the total routing time) for each vehicle k is equal to the arrival time at the final drop node.

$$max\_a_k = a_{n-1,k} \quad \forall k \in vehicles$$

This linear programming formulation ensures constraint satisfaction, providing feasible and cost-effective solutions for the VRP.

3.3 Variable Neighbourhood Search (VNS)

Variable Neighbourhood Search (VNS) is an iterative meta-heuristic designed to explore multiple neighbourhood structures for improving a feasible initial solution within a given computation time δ. Starting from a randomized constructive heuristic, VNS applies sequential neighbourhood refinements, i.e., node transfers, inter-route 2-opt*, and intra-route 2-opt which aimed at reducing total travel time without violating capacity or time constraints. To avoid getting stuck in local optima, the method diversifies solutions by shuffling the set of nodes before the start of each major iteration, while retaining the best solutions found. This multi-start framework continues until δ is reached, providing a robust balance between intensification (via neighbourhood search) and diversification (via initial randomization). The complete Python script for the VNS algorithm can be found in Appendix A. The VNS framework consists of several key phases, each contributing to solution improvement through specific local search operators. These phases are outlined below:

1. **Initial Solution:** The algorithm initiates with a randomized constructive heuristic, i.e., Insert Method [17]. This heuristic combines cluster-first, route-second principles with greedy insertion. Nodes are shuffled to ensure solution diversity, then sequentially assigned to n_k vehicle routes through a two-phase insertion process. For each node v_i, the algorithm evaluates all possible insertions across vehicle routes R_k between depots, with depot bookends (0) automatically added before feasibility verification and cost calculation. Each candidate insertion must satisfy:

$$\sum_{v \in R'_k} d_v \leq Q \quad \text{and} \quad \tau([0] \oplus R'_k \oplus [0]) \leq T \tag{1}$$

where R'_k denotes the modified route segment between depots, $\oplus$ represents sequence concatenation, and $\tau()$ calculates total duration of the vehicle route. The insertion with minimal duration increase in cost:

$$\Delta\tau_{ij} = \tau'(R'_k) - \tau'(R_k) \quad \text{, where} \quad \tau'(R_k) = \tau([0] \oplus R_k \oplus [0]) \tag{2}$$

between original and modified depot-augmented routes is retained. When no time-feasible insertion exists, the system relaxes the temporal constraint temporarily while maintaining

strict capacity adherence. The implementation features rigorous infeasibility detection, immediately terminating if any node cannot be placed without violating Q in the depot-to-depot route context. The worst time complexity of this constructive method is $O(n^3)$ [17].

2. **Node Transfer:** The node transfer phase implements an inter-route neighbourhood search operation, i.e., by transferring node v_i from route R_k to position j in route R_m. For each vehicle pair (k, m) where $k \neq m$, the algorithm evaluates all $v_i \in R_k$ through complete enumeration, generating trial solutions:

$$R'_k = R_k \setminus \{v_i\} \quad \text{and} \quad R'_m = (R_m[1:j], v_i, R_m[j+1:]) \tag{3}$$

Feasibility requires simultaneous satisfaction of:

$$\begin{cases} \sum_{v \in R'_m} d_v \leq Q \\ \tau'(R'_k) \leq T \quad \text{and} \quad \tau'(R'_m) \leq T \end{cases} \tag{4}$$

The cost reduction function:

$$\Delta C(k, m, i, j) = (\tau'(R_k) + \tau'(R_m)) - (\tau'(R'_k) + \tau'(R'_m)) \tag{5}$$

guides the greedy selection, with the implementation retaining only transfers where $\max(\Delta C) > 0$ across all (k, m, i, j) combinations. The time complexity of finding the best node transfer solution is $O(n^2)$ [17].

3. **Inter-route 2-Opt*:** This phase implements a generalized 2-opt operation between vehicle pairs to optimize cross-route connectivity. For each unique pair of routes (R_k, R_m), the algorithm examines all possible edge exchanges between route segments $R_k[i:i+1]$ and $R_m[j:j+1]$, where i and j denote cutting positions in their respective routes. The operator generates modified routes:

$$\begin{aligned} R'_k &= [v_1^k, ..., v_i^k] \oplus [v_{j+1}^m, ..., v_{|R_m|}^m] \\ R'_m &= [v_1^m, ..., v_j^m] \oplus [v_{i+1}^k, ..., v_{|R_k|}^k] \end{aligned} \tag{6}$$

Each exchange is evaluated through depot-bookended feasibility checks:

$$\begin{cases} \sum_{v \in R'_k} d_v \leq Q \quad \text{and} \quad \sum_{v \in R'_m} d_v \leq Q \\ \tau'(R'_k) \leq T \quad \text{and} \quad \tau'(R'_m) \leq T \end{cases} \tag{7}$$

The cost reduction potential:

$$\Delta C(k, m, i, j) = (\tau'(R_k) + \tau'(R_m)) - (\tau'(R'_k) + \tau'(R'_m)) \tag{8}$$

computes total travel time through the depot-inclusive route. The implementation performs exhaustive search across all (k, m, i, j) combinations, retaining only the exchange

with maximum $\Delta C > 0$ that satisfies all constraints. This cross-route perturbation mechanism enables escape from local optima while maintaining solution feasibility through simultaneous constraint validation of both modified routes. The time complexity of finding the best reconnection is $O(n^2)$ [17].

4. **Intra-route 2-Opt:** This phase applies the classical 2-opt heuristic independently to each vehicle route $R_k = [v_1, ..., v_m]$ through an iterative edge reversal process. For each route, the algorithm examines all possible segment reversals between positions i and j where $1 \leq i < j \leq m$, generating modified routes:

$$R'_k = [v_1, ..., v_{i-1}] \oplus \overleftarrow{[v_i, ..., v_j]} \oplus [v_{j+1}, ..., v_m] \tag{9}$$

where $\overleftarrow{[.]}$ indicates reversed sequence order. Each reversal is evaluated using the depot-bookended travel time:

$$\Delta\tau_{ij} = \tau'(R_k) - \tau'(R'_k) \tag{10}$$

The implementation employs a best-improvement strategy, greedily applying the reversal with maximum $\Delta\tau_{ij} > 0$ during each iteration until local optimality is reached, i.e., all (k, i, j) combinations are explored. Feasibility is inherently maintained as the optimization preserves both the route's node composition (ensuring $\sum_{v \in R_k} d_v \leq Q$ remains unchanged) and temporal validity through monotonically non-increasing travel times, i.e., $\tau'(R'_k) \leq \tau'(R_k) \leq T$. The time complexity of this improvement heuristic is the same as Phase 3, i.e., $O(n^2)$ [17].

5. **VNS Meta-heuristic Framework:** The algorithm implements an iterated variable neighbourhood descent (VND) strategy within a multi-start framework, governed by the computation time limit δ. Let $\mathcal{N} = \{N_1(\text{transfer}), N_2(\text{2-opt*}), N_3(\text{2-opt})\}$ denote the neighbourhood search structures stated in Phases 2, 3 and 4 respectively. At each major iteration t, the process:

$$\begin{aligned}
&1.\ \text{Generate initial solution } S_0 \quad (\text{Phase 1})\\
&2.\ \text{While } \Delta S > 0 \text{ and } \delta \text{ not exhausted:}\\
&\quad a.\ S' \leftarrow \arg\max_{S \in N_m(S_0)} \Delta C(S_0, S) \quad (\text{Phases 2 - 4})\\
&\quad b.\ \text{If } C(S') < C(S_0):\\
&\qquad S_0 \leftarrow S' \quad (\text{Move to improved solution})\\
&\quad c.\ \text{Else:}\\
&\qquad \text{Shuffle nodes in } S_0 \quad (\text{Diversification})
\end{aligned} \tag{11}$$

where $C(S) = \sum_{k=1}^{n_k} \tau'(R_k)$ represents the total solution cost. The implementation features three key mechanisms: *Neighbourhood Sequencing*: Strictly follows $N_1 \rightarrow N_2 \rightarrow N_3$ ordering, only progressing to subsequent neighbourhoods when no improvement is found

in the current. *Solution Perturbation*: Introduces controlled randomization through node shuffling before the start of each major iteration. *Elitist Archiving*: Maintains $S_{best} = \min\{C(S_0^{(t)})\}_{t=1}^{T}$ across all iterations. The main function executes this VND algorithm iteratively within a time-constrained multi-start framework. Final solution quality is guaranteed to be non-decreasing through the preservation chain $Sbest^{(k+1)} = \min(S_{best}^{(k)}, S_{final}^{(k)})$, ensuring monotonic improvement across restarts.

3.4 Simulated Annealing (SA)

The Simulated Annealing (SA) algorithm implements a stochastic optimization process combining three neighbourhood move operators within a temperature-controlled acceptance regime. The method explores solution space through controlled randomization while gradually intensifying the search towards optimal regions, governed by a geometric cooling schedule. The complete Python script for the SA algorithm can be found in Appendix B. The SA phases are described below:

1. **Initial Solution:** The algorithm employs a randomized construction process with automatic restarts to generate feasible solutions. Nodes are shuffled and sequentially assigned to vehicles. For each node, vehicle indices are randomly permuted and insertion positions are tested in random order within each route. The first valid insertion satisfying both vehicle capacity and temporal constraints (Eq 1) is accepted. Unlike VNS's greedy insertion (Eq 2) that selects the position with minimal increase in duration, this randomized approach prioritizes solution diversity over immediate cost optimization, accepting the first feasible insertion found during random position within vehicles sampling to accelerate initial solution generation while maintaining feasibility. If any node cannot be placed without violating vehicle capacity, the entire construction process restarts with a new node permutation, repeating up to a preset maximum of max_r restarts.

2. **Random Neighbourhood Moves:** The algorithm employs three neighbourhood operators $\mathcal{M} = \{\mathcal{M}_1(\text{transfer}), \mathcal{M}_2(\text{2-opt*}), \mathcal{M}_3(\text{2-opt})\}$ corresponding to the VNS neighbourhood search mechanisms Phase 2, 3 and 4 respectively. At each temperature step, one operator is selected with equal probability, and a single neighbourhood perturbation is performed to generate a candidate solution S', i.e.,

$$S' \leftarrow \mathcal{M}_i(S) \quad \text{where} \quad i \sim U\{1, 2, 3\} \tag{12}$$

A single move consists one application of a randomly chosen operator:

- For $\mathcal{M}_1$: Transfers one random node between vehicle pairs
- For $\mathcal{M}_2$: Performs one cross-route edge exchange

- For $\mathcal{M}_3$: Applies one intra-route segment reversal

Contrasting with VNS's exhaustive neighbourhood exploration (which evaluates all possible moves per operator), SA's approach generates one candidate solution per iteration through random operator selection and single move evaluation. This enables broader solution space exploration with $O(1)$ computational overhead per iteration versus VNS's $O(n^2)$ complexity, at the cost of potentially slower local improvement rates.

3. **Metropolis Acceptance Criterion:** The algorithm employs a probabilistic acceptance mechanism governed by the current temperature T and cost difference:

$$\Delta C = C(S') - C(S) \quad \text{where} \quad C(S) = \sum_{k=1}^{n_k} \tau'(R_k) \tag{13}$$

All candidate solutions undergo evaluation through:

$$P_{\text{accept}} = \begin{cases} 1 & \text{if } \Delta C \leq 0 \quad \text{(improved solution)} \\ \exp\left(-\frac{\Delta C}{T}\right) & \text{otherwise} \end{cases} \tag{14}$$

This dual acceptance strategy ensures automatic acceptance of all cost-reducing moves ($\Delta C < 0$), driving solution improvement, while allowing controlled exploration of worse solutions ($\Delta C \geq 0$), through acceptance probabilities decreasing exponentially as ΔC increases or T decreases.

The temperature follows a geometric cooling schedule where T decreases by a fixed ratio α after each iteration. This gradual reduction balances early-stage exploration (allowing acceptance of worse solutions) with late-stage exploitation (converging to local optima). The cooling rate α governs the exploration-to-exploitation transition speed, with values near 1 preserving theoretical convergence guarantees while enabling extensive solution space sampling during the initial phases of the annealing process.

4. **SA Meta-heuristic Framework:** The algorithm implements the following process:

1. Generate initial solution S_0 (Phase 1)
2. While $T > T_{end}$ and $t < t_{max}$:
   a. $S' \leftarrow \mathcal{M}_i(S)$ $(i \sim U\{1,2,3\})$ (Phase 2)
   b. Compute $\Delta C = C(S') - C(S)$
   c. If $\Delta C < 0$ or $U(0,1) < \exp(-\Delta C/T)$:
      $S \leftarrow S'$ (Phase 3)
   d. $S^* \leftarrow \min(S^*, S)$ (Retain best solution)
   e. $T \leftarrow \alpha T$ (Update temperature)
   f. $t \leftarrow t + 1$
3. Return S^*

(15)

This SA framework combines hybrid neighbourhood operators (inter and intra-route moves) enabling diverse exploration with geometric cooling that balances exploration-exploitation through temperature dependent acceptance probabilities. Operators which obey the set of constraints maintain feasibility while the cooling factor α, ensures asymptotic convergence, complemented by elite preservation that guarantees non-decreasing solution quality throughout the annealing process.

4 Results

This section evaluates the computational performance of the proposed vehicle routing solution. The Linear Programming (LP) approach exhibits exponential time complexity, limiting its practicality for larger problems. To address this, we compare two meta-heuristic methods, i.e., Variable Neighbourhood Search (VNS) and Simulated Annealing (SA) against the hybrid Google OR-Tools VRP solver. Finally, we apply VNS to real-world operational data from Fermanagh Community Transport (FCT). By clustering historical service records, we generate optimized routing solutions that align with FCT's operational constraints. A comparison of different clustering configurations highlights trade-offs in travel time, service efficiency, and route structure, demonstrating the potential of meta-heuristics for real-world transport planning.

4.1 Linear Programming Benchmark

In this subsection, we evaluate the time efficiency of the Linear Programming (LP) algorithm we proposed in Section 3.2. The parameters were chosen with relaxed constraints to ensure solution feasibility while maintaining real-world problem characteristics:

- **Vehicles**, n_k: Fleet size of 4 serviceable vehicles

- **Capacity**, Q: Maximum vehicle capacity of 16 units
- **Temporal Constraint**, T: Maximum routing time of 90 minutes
- **Earliest window**, e_i: Random integer between [0,45] minutes for customer nodes
- **Latest window**, l_i: Computed as $e_i + 15$ minutes for customer nodes
- **Demand**, d_i: Random integer demand between [1,3] units for customer nodes
- **Travel time**, t_{ij}: Random integer between [5,15] minutes for $i \neq j$

The arbitrary large M parameter was set to $M = 10^4$ to ensure constraint logic validity. Node configurations were tested across $n \in \{5, 6, \ldots, 13\}$ with 10 simulations per node count. Figure 3 shows the box-plot of computation time distribution across different node counts in logarithmic scale to accommodate the wide range of solution times observed during benchmarking.

Table 2 presents the computational performance statistics across node counts, showing median computation times (seconds), 5th/95th percentiles, and feasible solution counts. The data reveals exponential time complexity growth with respect to node count. Median solution times increase from 0.627 seconds for 5 nodes to 266.35 seconds (4.44 minutes) for 13 nodes, with solution times first exceeding one minute at $n = 11$ (median 62.09 seconds). This growth pattern follows the characteristic $O(c^n)$ complexity of exact integer linear programming approaches. Outliers are present across the range of node count as seen in Figure 3. For $n = 13$, the 95th percentile reaches 5,533 seconds (92.2 minutes), while one extreme outlier required 9,424.64 seconds (157.1 minutes $\approx$ 2.6 hours), i.e., exceeding the median time by 35 times.

While producing optimal solutions, the algorithm becomes computationally prohibitive for $n \geq 12$ nodes. This demonstrates that the approach is impractical for real-world logistics applications where typical vehicle routing problems involve vast datasets. The results strongly suggest the need for heuristic or meta-heuristic approaches that sacrifice optimality guarantees to achieve polynomial-time complexity and operational feasibility.

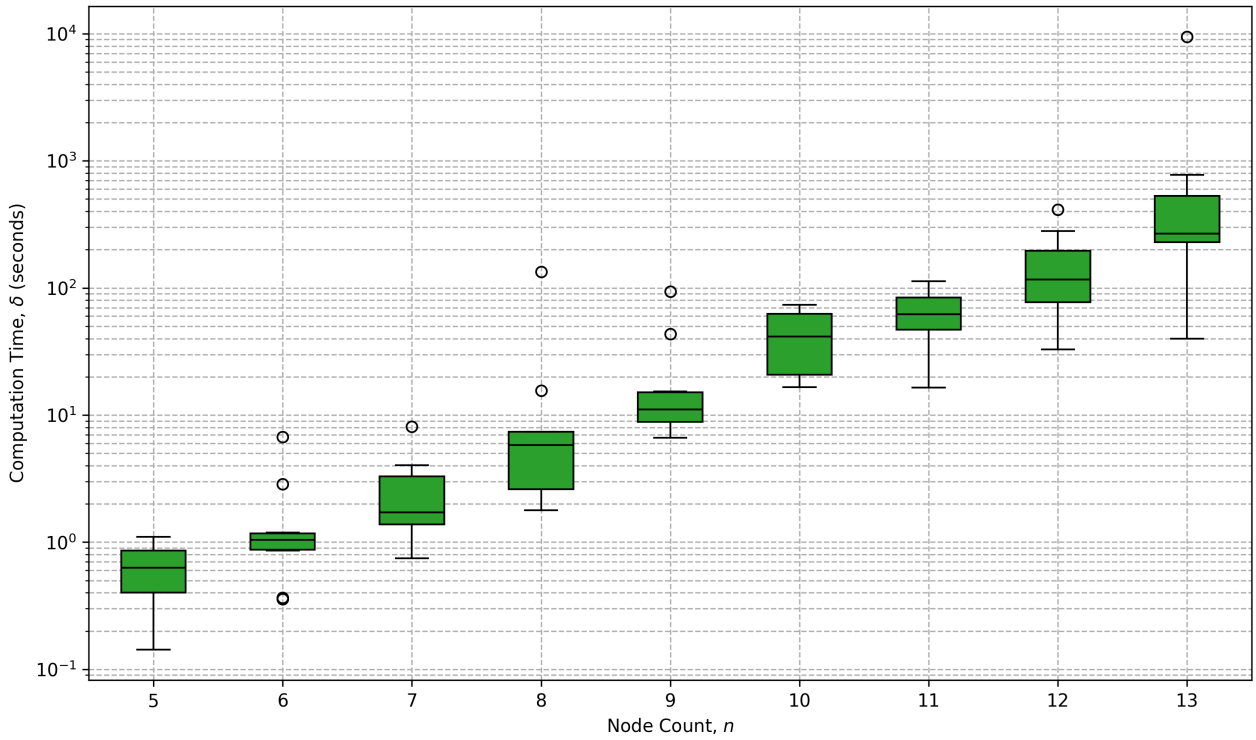

Figure 3: Computation Time Plot for LP with 10 Simulations per node

Table 2: Computation Time Statistics for LP with 10 Simulations per node

Node Count	Median (s)	5th %ile (s)	95th %ile (s)	Feasible	Infeasible
5	0.627	0.229	1.022	10	0
6	1.039	0.361	4.987	10	0
7	1.719	0.746	6.284	10	0
8	5.823	1.831	80.669	10	0
9	11.094	7.085	70.896	10	0
10	41.326	17.701	71.180	10	0
11	62.092	21.492	105.152	10	0
12	116.528	37.656	352.726	10	0
13	266.354	42.520	5533.471	10	0

4.2 Meta-Heuristics Benchmark

The experimental setup evaluates three Meta-Heuristic VRP algorithms: Google OR-Tools' VRP solver, 3.3 Variable Neighbourhood Search (VNS), and 3.4 Simulated Annealing (SA) implementation, across five problem instances of increasing complexity. The Google OR-Tools VRP solver [10] serves as our performance benchmark, employing a hybrid approach that combines constraint programming with meta-heuristics. Its configuration uses a first-solution strategy of *PATH_CHEAPEST_ARC* followed by *GUIDED_LOCAL_SEARCH* intensification,

with search duration limited by the same δ parameter as our VNS algorithm. This sophisticated combination of exact and heuristic methods provides a strong baseline for comparing our pure meta-heuristic approaches.

Parameters were scaled proportionally with problem size to maintain realistic vehicle routing problem constraints while ensuring feasible solution space, as detailed in Table 3. The travel time matrix contains random integers between 5 and 15 minutes for node-to-node traversal, with customer demands randomly generated from 1 to 3 units at each node, excluding the depot. All experiments used a fixed random seed (42) for reproducible instance generation.

Table 3: Parameter Configuration by Problem Scale

Instance	**Nodes**	**Vehicles**	**Capacity**	**Temporal**	**Comp. Time**	**SA Param.**
i	n	n_k	Q	T (min)	δ (sec)	(T_{end}, α)
1	25	6	8	60	4	(0.01, 0.999)
2	50	8	12	60	10	(0.05, 0.999)
3	100	12	18	90	15	(0.10, 0.999)
4	200	25	18	90	20	(0.10, 0.999)
5	500	35	35	120	30	(0.01, 0.999)

Box plots in Figures 4–8 present the distribution of total costs across 10 trials for each algorithm and instance pair, showing median values (central lines) and interquartile ranges (box boundaries). The SA parameters were carefully calibrated to match the fixed time limits δ of Google OR-Tools and VNS, with the exception of Instance 5 where extended cooling is necessary for the increased solution complexity.

Google OR-Tools consistently achieves the best solutions (i.e., lowest optimal cost), as expected from its hybrid of exact and heuristic approach, shows almost no variance across the 10 trials. By contrast, while the SA algorithm often attains a lower median cost than VNS as seen in Figures 4–7, it exhibits substantially higher variance. A potential improvement to reduce SA's variability is to restart the annealing process from multiple distinct initial solutions (as done in VNS) and return the overall best result.

Table 4 presents the median solution costs and percentage differences relative to Google OR-Tools. For instance, in the 25-node instance, Google OR-Tools achieves a median cost of 182, while VNS and SA return 187.5 (+3.0%) and 185.5 (+1.9%) respectively. As the problem size increases, the gap between Google OR-Tools and the meta-heuristic approaches remains relatively consistent, ranging between 2.7% and 5.6%.

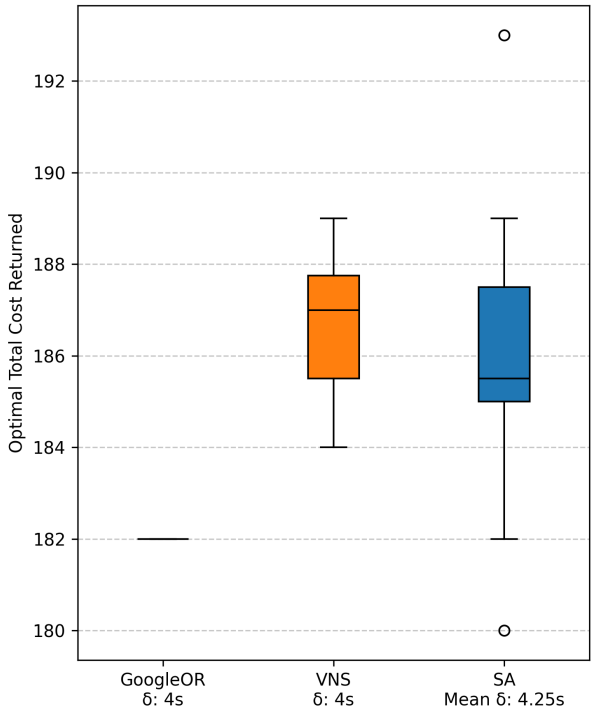

Figure 4: 25 Nodes and 6 Vehicles

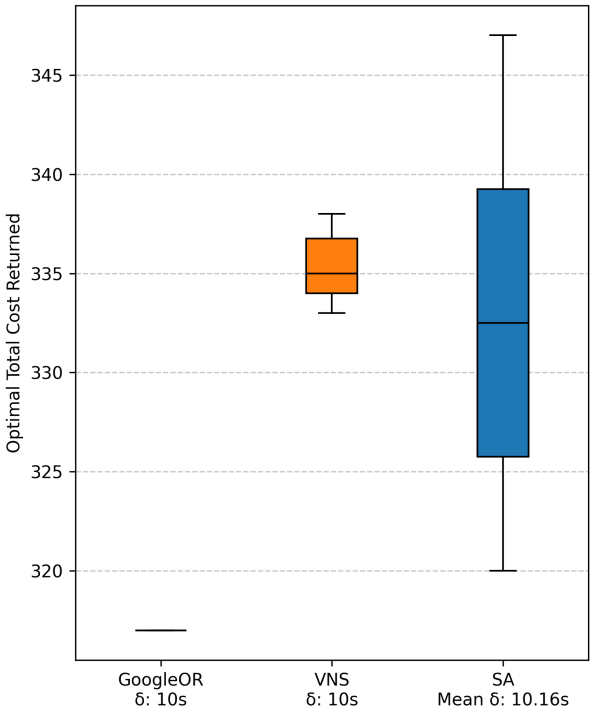

Figure 5: 50 Nodes and 8 Vehicles

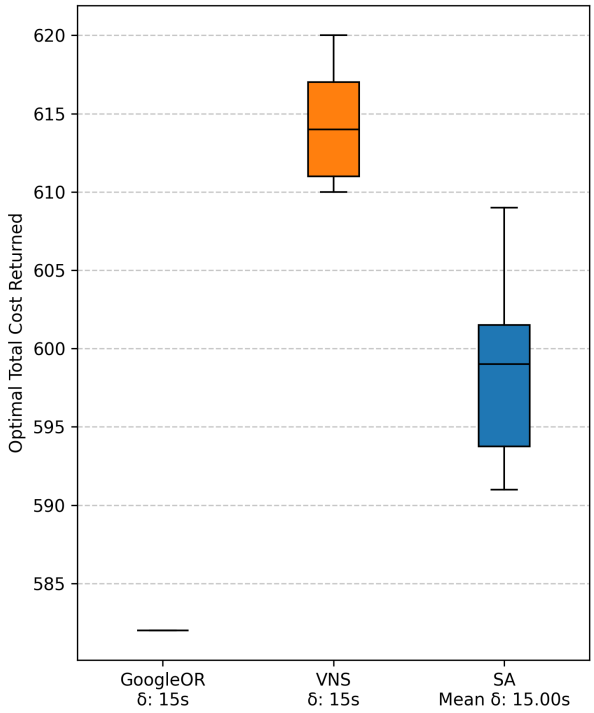

Figure 6: 100 Nodes and 12 Vehicles

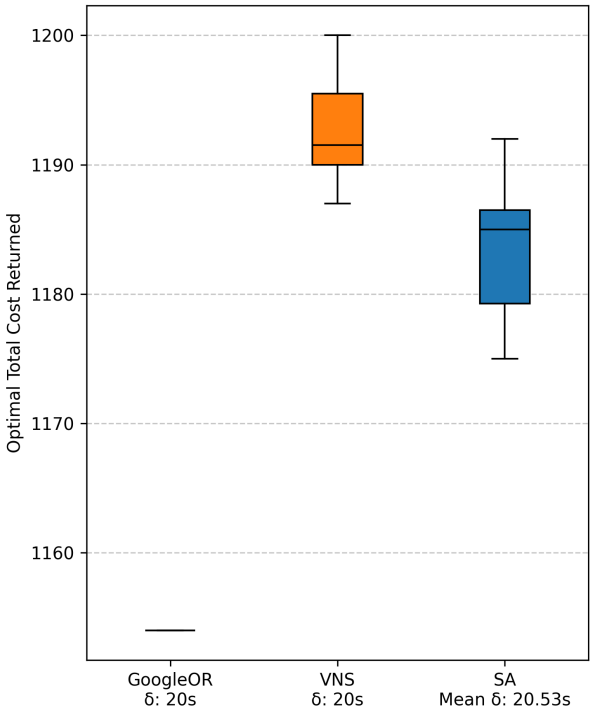

Figure 7: 200 Nodes and 25 Vehicles

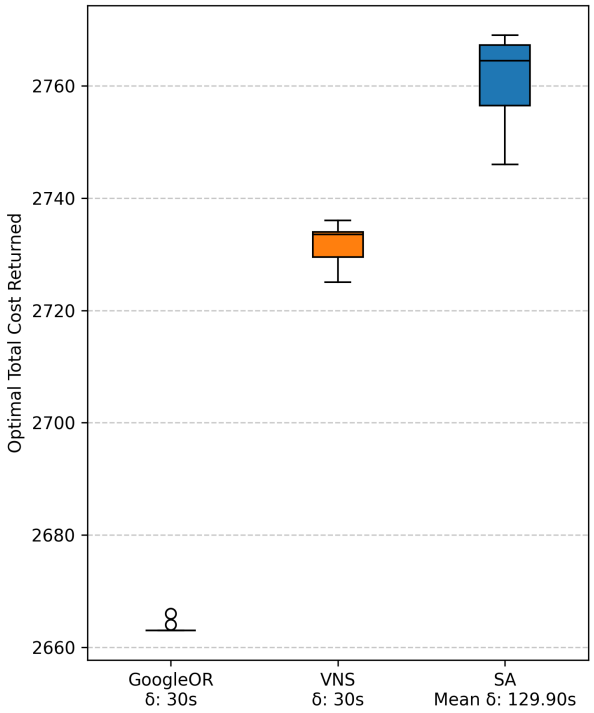

Figure 8: 500 Nodes and 35 Vehicles

Table 4: Median costs and relative differences of VRP methods.

Instance	**Nodes**	**Google OR**	**VNS**	**SA**
1	25	182	187.5 (3.0%)	185.5 (1.9%)
2	50	317	335 (5.6%)	332.5 (4.9%)
3	100	582	614 (5.5%)	599 (2.9%)
4	200	1154	1192 (3.3%)	1185 (2.7%)
5	500	2663	2734 (2.7%)	2764.5 (3.8%)

For the largest instance (Figure 8, 500 nodes), the SA algorithm requires extended cooling by lowering the ending temperature ($T_{\text{end}} = 0.01$) to remain competitive, which significantly increases its runtime. This indicates that SA encounters a bottleneck when handling larger datasets, requiring additional computational time to explore a broader solution space effectively. It is important to emphasize that all parameters, including cooling rates, neighbourhood structures, and time limits δ, were selected heuristically for these experiments. Adjusting these parameters could significantly influence solution quality and convergence behaviour. Additionally, the instance parameters were chosen arbitrarily, meaning the benchmark results may vary for different datasets. Nevertheless, our proposed pure meta-heuristics remain competitive with Google OR-Tools in terms of solution quality, particularly as problem size increases.

4.3 Demonstration using FCT Operational Data

In this subsection, we demonstrate the application of our Variable Neighbourhood Search (VNS) algorithm on the operational dataset from Fermanagh Community Transport (FCT) [23]. The dataset undergoes a structured transformation pipeline, progressing through multiple stages from raw data processing to the visualisation of optimal routing solutions. The process is visualized in Figure 9, which provides an overview of the key steps involved in this demonstration.

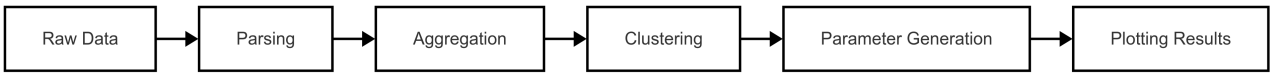

Figure 9: Sequential data processing pipeline for VNS algorithm demonstration.

4.3.1 Data Preparation Pipeline

This real-world dataset, spanning service records from April 2024 to March 2025, contains 475 postcode entries across Fermanagh. The transformation pipeline from raw records to VRP parameters follows four sequential stages:

- **Raw Data:** Initial processing begins with the unmodified service records containing historical trip data, requiring handling of various formatting inconsistencies.
- **Data Parsing:** Automated validation procedures standardize postcodes to official 'BTX XXX' formats and convert all temporal data to consistent 'MM/DD/YYYY' formatting. Numerical fields undergo range checking and outlier detection.
- **Postcodes Aggregation:** Duplicate postcode entries merge through summation of trip volumes and passenger counts, while preserving the most recent service date. This aggregation reduces the number of postcodes from 475 to 339 unique entries. The complete aggregated spreadsheet of the dataset can be found in Appendix 8.
- **Geographic Clustering:** Spatial grouping employs Scikit-learn's Density-based Clustering (DBSCAN) [21] with haversine distance metrics implementation and Postcodes.io API [19] for converting postcodes into geographical coordinates. The DBSCAN groups postcode entries within a fixed radial threshold ϵ using haversine spherical distance calculations, with a minimum cluster size parameter c, i.e., all clusters with less than c postcode entries are omitted. The resulting output for $\epsilon = 0.005$ and $c = 1$ is available in Appendix 9 and plotted on a geographical map, as shown in Figure 10.

The processed output defines our VRP constraints: cluster centroids as nodal locations and aggregated trip counts as demand weights. This structured approach ensures systematic inputs faithfully represent FCT's operational reality while maintaining computational tractability.

Figure 10: Service clusters generated through DBSCAN spatial clustering with $\epsilon = 0.005 \approx$ 500 meters and minimum cluster size of $c = 1$. Coloured markers represent 219 clusters with all postcode entries retained due to the singleton tolerance parameter.

4.3.2 Operational Assumptions and Parameter Generation

The VRP formulation requires three key parameters derived from the clustered nodes established in Section 4.3.1: travel duration matrix t_{ij}, vehicle capacity Q, and maximum route duration T. The nodal configuration comprises 219 service clusters as displayed in Figure 10 plus one central depot node at Enniskillen (BT74 6JA), yielding $n = 220$ total nodes. Vehicle fleet size $n_k = 6$ mirrors FCT's daily operational capacity, while demand weights d_i correspond to each cluster's aggregated passenger counts. The remaining operational parameters derive from these foundations:

- **Travel Duration Matrix:** Google's Distance Matrix API [9] calculates real-time traffic durations between nodal points using historical mobility patterns. One postcode entry is selected from each cluster to serve as input, with departure time fixed to Monday 10$^{\text{th}}$ February 2025 at 08:00 AM to reflect morning service patterns. The implementation constructs a lower triangular matrix through batched API requests, assumes symmetric

travel durations i.e., $t_{ij} = t_{ji}$, and anchors routes at the Fairgreen Shopping Centre, Enniskillen BT74 6JA as the departure and terminating depot. This generates a 220×220 symmetric travel time matrix.

- **Vehicle Capacity:** Calculated through demand balancing with operational buffer:

$$Q = \left\lceil \frac{\text{Total Passengers}}{n_k} \right\rceil + \alpha = \left\lceil \frac{24,992}{6} \right\rceil + 234 = 4,400$$

where α accommodates rural route variability and ensures feasibility of solution.

- **Temporal Constraints:** Maximum route duration is fixed at $T = 360$ minutes (6 hours) for each vehicle.

4.3.3 Visualized Solutions and Analysis

We demonstrate the application of our Variable Neighbourhood Search (VNS) algorithm, utilizing the clustered nodes and parameters outlined in Sections 4.3.1 and 4.3.2 respectively. The computation time parameter δ of the VNS is intentionally set with a generous tolerance to explore a broader solution space, enhancing the accuracy of the optimal routes. The map in Figure 11 illustrates the routes for six vehicles, each colour-coded for clarity. The depot at Enniskillen (BT74 6JA) serves as the central hub, with routes extending across the service area. The full algorithm output demonstrated in this section, detailing the optimal routes for the vehicle fleet, is documented in Appendix C. This demonstration underscores the effectiveness of meta-heuristic algorithm in handling complex routing challenges within a real world transport network.

In Figure 11, Vehicle 6 (brown) has the longest travel time of 293.00 minutes. In contrast, Vehicle 1 (red) completes its route in 146.00 minutes, likely due to varying demand levels within its assigned clusters. Plotting the results geographically offers an immediate visual cue to potential inefficiencies or bottlenecks particularly in areas where routes converge and intersect, thereby providing a practical foundation for improvement in future route scheduling. Next, we raise the minimum cluster size to $c = 2$, thereby eliminating all clusters composed of a single postcode entry. The VNS algorithm's optimal solution under these settings is shown in Figure 12, resulting in a reduction of total clusters from 220 to 39 and a significant decrease in overall travel cost from 1197 to 450 minutes. Moreover, this configuration exhibits sparser route crossings and clearer delineations between service areas.

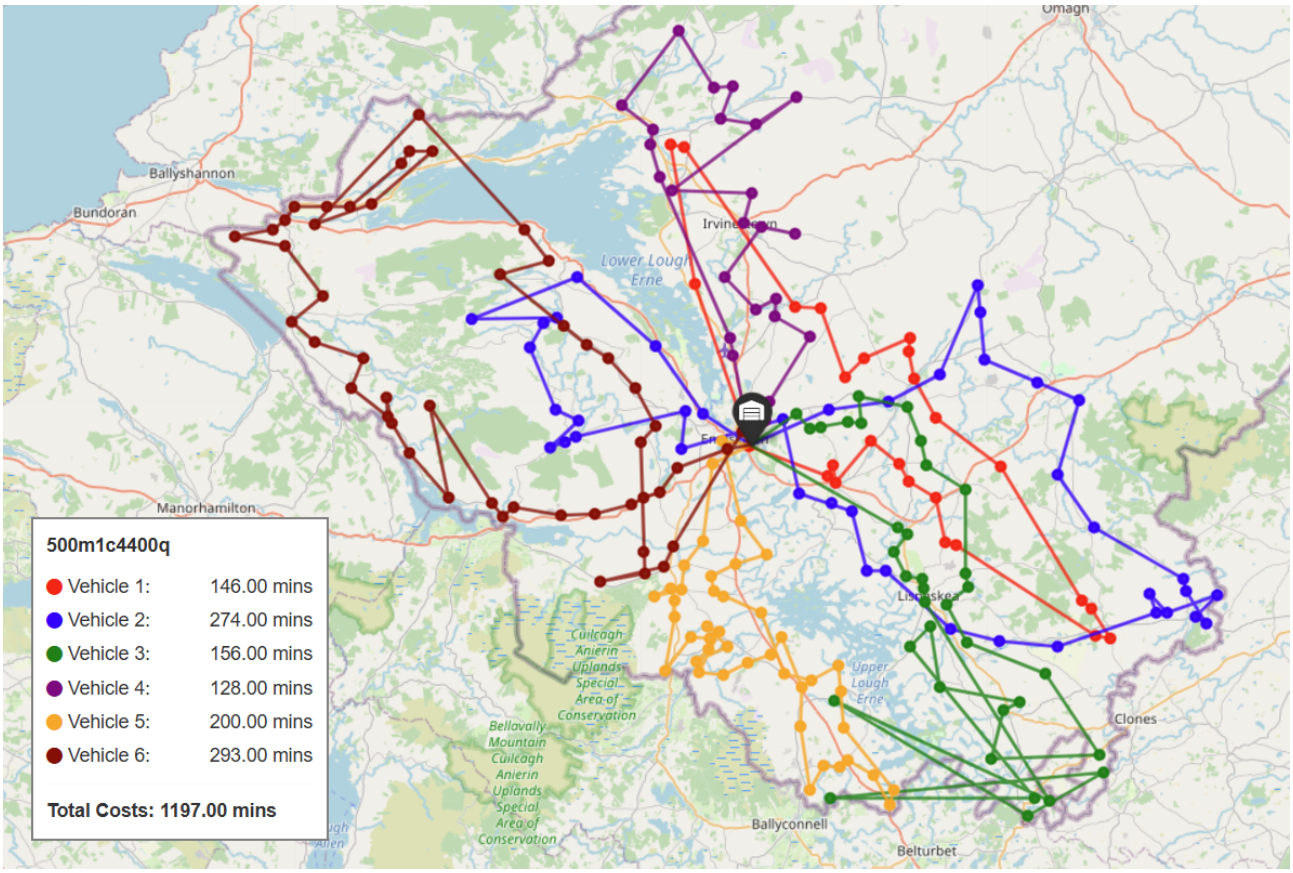

Figure 11: Optimal routes for 219 serviceable clusters in Fermanagh, with a departure and terminating depot located in Enniskillen. Parameters: $n_k = 6$ (vehicles), $Q = 4400$ (capacity), $T = 360$ minutes (route time) and demand weights d_i correspond to each cluster's aggregated passenger counts.

In a subsequent analysis, instead of using the aggregated passenger counts as demand weights for each cluster, we now assume that each postcode entry carries an equal weight of 1. As a result, the demand weight for each cluster corresponds to the number of postcode entries within it. This adjustment is illustrated in Figure 14. For this change of assumption, the vehicle capacity has been appropriately scaled down, changing from a total passenger count of 24,992 to the total number of unique postcode entries, which is 339. While the total costs remain nearly unchanged, a more distinct delineation between service areas emerges compared to Figure 12, with reduced route overlapping and a more structured clustering of service zones. This routing strategy shows a strong resemblance to the petal-like routing approach employed by FCT, as illustrated in Figure 13.

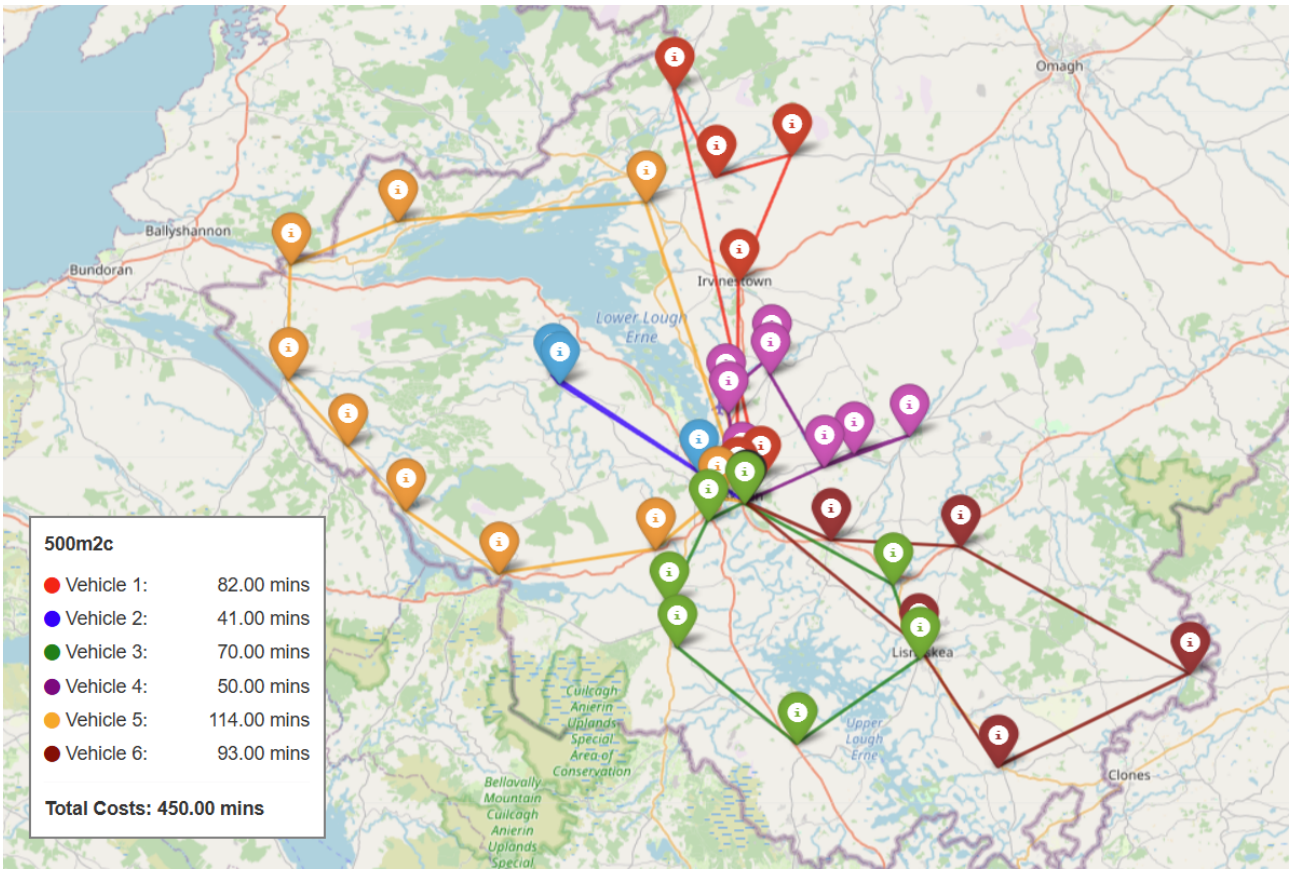

Figure 12: Optimal routes for 38 serviceable clusters in Fermanagh, with minimum cluster size $c = 2$. Parameters: $n_k = 6$ (vehicles), $Q = 2000$ (capacity), $T = 180$ minutes (route time).

The comparison between Figure 11 and 14 provides valuable insight into how frequent flyers (i.e., clusters with higher passenger counts) influence routing decisions compared to an approach where all customers carry equal weight. Under the aggregated demand model, clusters with heavier demand are prioritized in the solution, potentially leading to more route overlap in high-demand regions. In contrast, assigning a uniform weight to each postcode entry spreads the service more evenly, resulting in clearer delineations between service zones. Nevertheless, some inconsistencies are evident in the results. For example, the yellow route crossing Lower Lough Erne and the zigzag pattern observed in the green route in the far south suggest potential inaccuracies in the solution. These anomalies are likely a result of Google API travel time errors, particularly given the remoteness and sparse road network of the rural area in Fermanagh. Such discrepancies highlight the challenges of relying on external mapping services for precise travel time estimations in less accessible regions.

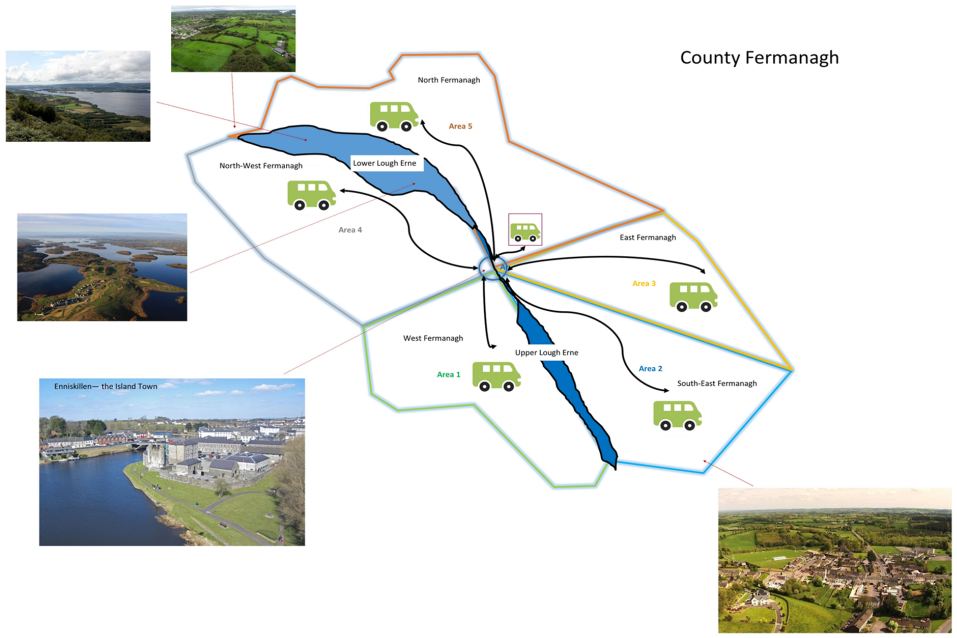

Figure 13: Petal-like Routing Strategy of FCT

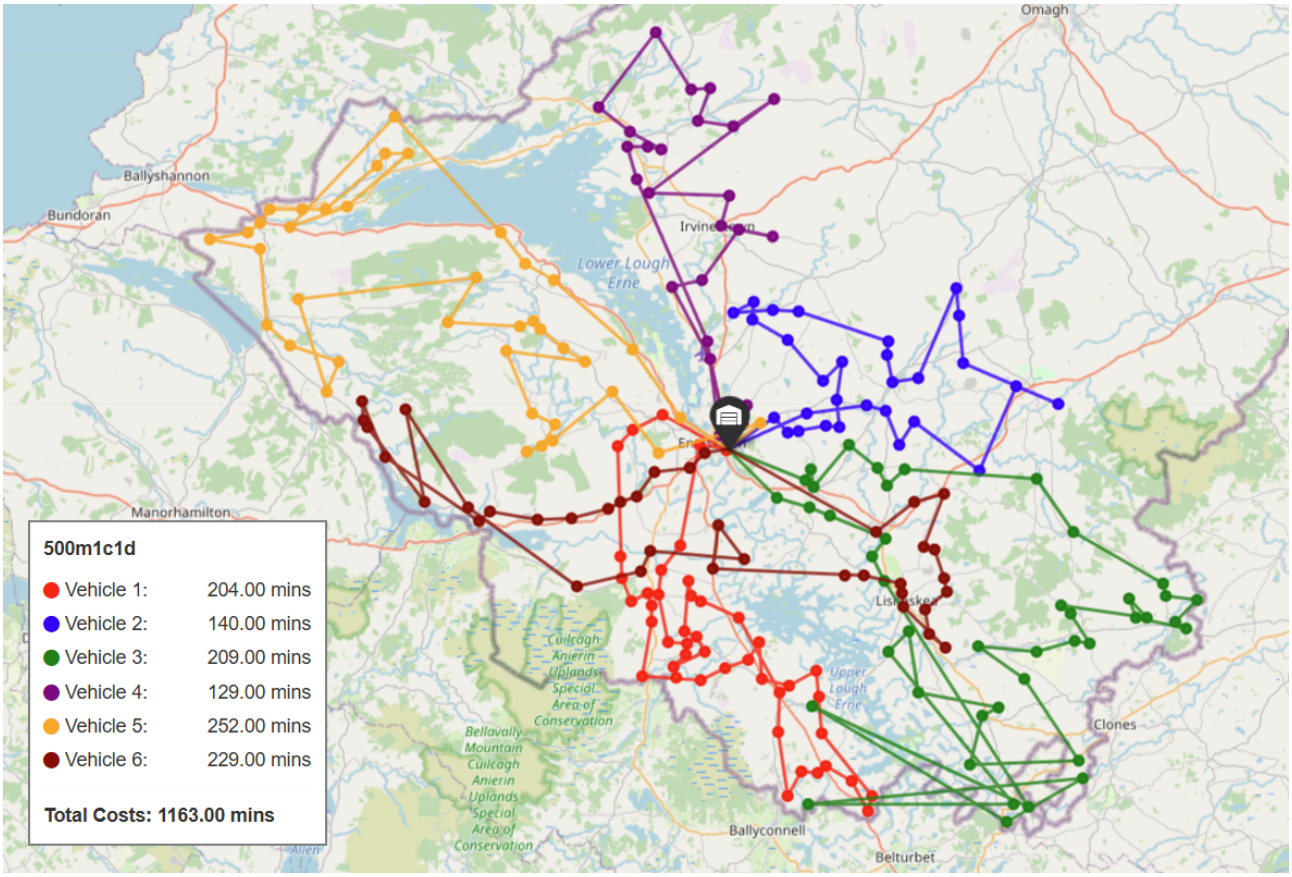

Figure 14: Optimal routes for 219 serviceable clusters in Fermanagh, with demand weights d_i equal to the number of postcode entries in each cluster. Parameters: $n_k = 6$ (vehicles), $Q = 60$ (capacity), $T = 360$ minutes (route time).

5 Limitations

One key limitation of this study lies in its assumption that all costs scale directly with vehicle travel time, overlooking factors such as driver wages, toll charges, environmental fees, and maintenance costs, which can fluctuate independently of distance or duration. Furthermore, the Linear Programming (LP) model treats waiting time the same as travel time, even though waiting typically incurs lower expenses. Beyond these cost-related considerations, the meta-heuristic methods require the departure and terminating nodes to be the same location, an oversimplification that does not reflect many real-world cases where routes may start and end at different depots or hubs. Additionally, neither approach accounts for negative demands i.e., passengers disembarking mid-route, a common scenario in public transport services. Although the LP model includes time-window constraints, the Variable Neighbourhood Search (VNS) and Simulated Annealing (SA) algorithms omit them, thus not capturing critical scheduling restrictions like pickup or delivery deadlines. Both LP and meta-heuristic formulations also assume a uniform fleet, whereas real operations might require multiple vehicle types, for example, wheelchair accessible vans, which have diverse capacities and may incur additional operational costs. In practical contexts, customer locations and demands are often not constant and may change abruptly. Sudden cancellations or late requests can invalidate pre-planned routes, forcing on-the-fly reassignments. Furthermore, the reliance on Google's Distance Matrix API for travel-time data presupposes that future traffic conditions are reliably predictable. In reality, accidents, adverse weather, or sudden congestion can render these estimates inaccurate. Consequently, while our models provide a structured means of route optimization, they do not yet address the dynamic and often unpredictable nature of real-world transportation.

While LP guarantees mathematically optimal solutions, its computational time grow exponentially, making it impractical for medium to large scale VRPs. Even advanced techniques like Branch and Cut struggle with real-world instances containing hundreds or thousands of nodes. In contrast, meta-heuristic methods are often able to explore the whole solution space for smaller instances and thus yield the same optimal results as LP, while achieving this in a fraction of the time required by exact methods. This efficiency makes meta-heuristics far more practical for large-scale routing problems. Both VNS and SA heavily depend on fine-tuned parameter choices, including neighbourhood exploration depth, cooling rate α and the starting/ ending temperatures. Poorly tuned parameters can either lead to suboptimal solutions or excessive computation time. Additionally, stopping criteria, such as the computation time limit δ in VNS or the maximum iterations in SA, directly impact solution quality. A longer runtime typically improves results but also increases computational costs, creating a trade-off between efficiency and accuracy. Finding the right meta-heuristic parameters is problem-specific and can be time-consuming, as no universal guidelines exist for optimizing settings across different VRP instances. Parameter tuning often requires extensive experimentation or adaptive

learning techniques, further complicating the implementation of meta-heuristics in practical applications.

Beyond computational limitations, real-world deployment of these models presents additional challenges. The Google Distance Matrix API, while useful for estimating travel times, is not cost-effective for large-scale applications. For example, generating a 219 by 219 travel time matrix as used in this study incurs an estimated cost of 120 USD, making frequent usage impractical for organizations with budget constraints. Moreover, Google's integrated VRP solver tools already offer highly optimized complementary routing solutions, often outperforming custom implementations, which raises questions about the practicality of developing independent VRP solvers when off-the-shelf solutions exist. Integration with existing operational systems is another major challenge. Even if a meta-heuristic or LP-based solution performs well in simulations, deploying it within an organization's scheduling and dispatch framework requires extensive adaptation. For instance, many transport services operate on legacy software that may not support direct integration with advanced routing algorithms. Transitioning to a new system would demand retraining staff, modifying internal workflows, and ensuring compatibility with real-time scheduling updates. Public transport services, for example, often rely on dynamic scheduling where last-minute cancellations or changes must be accommodated, which is something that a stand-alone optimization model may struggle to handle effectively. While LP, VNS, and SA provide effective approaches to solving the VRP, their real-world applicability is constrained by computational complexity, parameter sensitivity, and integration challenges. Future improvements should focus on adaptive tuning, hybrid methods, and real-time data integration to enhance scalability and practicality in dynamic transport systems.

6 Conclusions

This paper has demonstrated how advanced meta-heuristic algorithms can substantially contribute to solving real-world Vehicle Routing Problems (VRP), especially within resource-constrained rural community services like Fermanagh Community Transport (FCT). By exploring Linear Programming (LP), Variable Neighbourhood Search (VNS), and Simulated Annealing (SA), this study not only evaluated computational efficiencies and algorithmic robustness but also underlined the critical balance between solution optimality and practical runtime demands. The collaboration with FCT has been particularly valuable, offering operational insights that highlight the alignment and validation of theoretical models with intuitive routing strategies. The comparative analysis between solutions prioritizing high-demand passengers versus equal service distribution was especially insightful. This analysis has reassured FCT about the effectiveness and proximity to optimality of their current intuitive routing approach.

Moreover, the methodologies developed in this dissertation serve as comparative tools, enabling FCT to benchmark their intuitive strategies against algorithmically generated solutions, enhancing their confidence in their existing operational practices.

Having addressed the limitations encountered in this research, including oversimplification of parameters, methodology limitations and external dependencies on mapping services, future research can enhance and extend these initial findings through several promising directions. Applying these methodologies to operational datasets from diverse community or commercial transport providers beyond FCT could offer valuable comparative insights, especially when addressing the distinctive challenges between rural and urban routing environments. Further, analysing the effects of seasonality or variable daily traffic patterns on route efficiency, vehicle utilization, and operational costs can yield critical insights for dynamic, real-time planning scenarios. Also, Dynamic Vehicle Routing Problems (DVRP) can be explored, where customer demands and travel times evolve during route execution. Methodological innovation also presents numerous opportunities for future exploration. Introducing novel neighbourhood structures or customized search operators that exploit the specific characteristics of VRP could significantly enhance solution efficiency and quality. Besides, developing adaptive or self-tuning meta-heuristic algorithms that dynamically adjust parameters based on ongoing search progress or historical performance trends would considerably improve algorithmic robustness and operational flexibility. Additionally, advancing theoretical and computational analyses will enhance the foundational understanding of these methodologies. Performing deeper complexity analyses of the proposed heuristics and LP formulations to define their theoretical optimality bounds and complexity limitations clearly would strengthen theoretical justifications. Moreover, scalability studies examining computational resource requirements like memory and CPU or GPU utilization with increasingly larger datasets would be invaluable for assessing operational scalability and facilitating effective deployment in practice.

Overall, this research offers both theoretical and practical contributions, presenting an actionable framework that could assist logistics and transport services like FCT in refining their operations by reducing operational costs, improving service quality, and increasing accessibility for vulnerable populations. This applied mathematical exploration was initially motivated by the ambition to solve complex routing problems purely mathematically, this research has ultimately highlighted the profound complexity and uncertainty inherent in real-world scenarios. Parameters in practical contexts are often dynamic, subjective, and influenced by countless unpredictable factors, thus the pursuit of a singular, definitive solution is fundamentally unattainable. The ultimate goal in optimization may not lie solely in achieving the perfect solution, but rather in consistently approaching closer approximations that resonate more deeply with real-world complexities. Thus, this applied mathematical exploration not only contributes

to combinatorial optimization literature but also profoundly influences real-world community mobility solutions, reflecting a thoughtful balance between theoretical ideals and practical realities.

7 Acknowledgements

The computer scripts developed in this project were written using Cursor [3], an Integrated Development Environment (IDE) that incorporates generative AI tools to assist with code generation and debugging. This tool was used to streamline routine coding tasks and explore alternative approaches during development. All outputs were carefully reviewed, validated, and adapted to ensure accuracy and alignment with the project objectives.

References

[1] Çam, Ö. and Sezen, H. (2020) 'Linear Programming Formulation for Vehicle Routing Problem Which Is Minimized Idle Time', *Decision Making: Applications in Management and Engineering*. Available at: https://doi.org/10.31181/dmame2003132h (Accessed: 10 February 2025).

[2] Cordeau, J-F., et al. (2001) 'A unified tabu search heuristic for vehicle routing problems with time windows', *The Journal of the Operational Research Society*, 52(8), pp. 928–936. Available at: http://www.jstor.org/stable/822953 (Accessed: 6 February 2025).

[3] Cursor: 'The AI-Powered Code Editor'. Available at: https://www.cursor.com/ (Accessed: 14 February 2025).

[4] Derbel, H., Jarboui, B., and Siarry, P. (2020) *Green Transportation and New Advances in Vehicle Routing Problems*. Springer International Publishing. Available at: https://doi.org/10.1007/978-3-030-45312-1 (Accessed: 19 February 2025).

[5] Ergun, Ö., Orlin, J.B., and Steele-Feldman, A. (2006) 'Creating very large scale neighborhoods out of smaller ones by compounding moves', *Journal of Heuristics*, 12, pp. 115–140. Available at: https://doi.org/10.1007/s10732-006-5561-5 (Accessed: 10 February 2025).

[6] Gendreau, M., Hertz, A., and Laporte, G. (1994) 'A tabu search heuristic for the vehicle routing problem', *Management Science*, 40, pp. 1276–1290.

[7] Glover, F. (1986) 'Future paths for integer programming and links to artificial intelligence', *Computers & Operations Research*, 13(5), pp. 533–549. Available at: https://doi.org/10.1016/0305-0548(86)90048-1 (Accessed: 10 February 2025).

[8] Goel, A. and Gruhn, V. (2008) 'A General Vehicle Routing Problem', *European Journal of Operational Research*, 191(3), pp. 650–660. Available at: https://doi.org/10.1016/j.ejor.2006.12.065 (Accessed: 7 February 2025).

[9] Google Maps Platform: Distance Matrix API. Available at: https://developers.google.com/maps/documentation/distance-matrix (Accessed: 6 February 2025).

[10] Google OR-Tools: Optimizing routing and scheduling problems. Available at: https://developers.google.com/optimization (Accessed: 6 February 2025).

[11] Holland, J.H. (1992) 'Genetic algorithms', *Scientific American*, 267(1), pp. 66–73. Available at: http://www.jstor.org/stable/24939139 (Accessed: 10 February 2025).

[12] Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983) 'Optimization by simulated annealing', *Science*, 220(4598), pp. 671–680. Available at: https://www.science.org/doi/abs/10.1126/science.220.4598.671 (Accessed: 10 February 2025).

[13] Kovács, L. and Jlidi, A. (2024) 'Neural Networks for Vehicle Routing Problem'. Available at: https://arxiv.org/abs/2409.11290 (Accessed: 10 February 2025).

[14] Laporte, G. (2009) 'Fifty years of vehicle routing', *Transportation Science*, 43(4), pp. 408–416. Available at: http://www.jstor.org/stable/25769465 (Accessed: 10 February 2025).

[15] Laporte, G. (2007) 'What you should know about the vehicle routing problem', *Naval Research Logistics (NRL)*, 54(8), pp. 811–819. doi:10.1002/nav.20261.

[16] Lian, K.Q. and Tribello, G.A. (2024) 'An elementary approach to the Vehicle Routing Problem via Python and Google API', American Journal of Operations Research, 14(06), pp. 169–190. Available at: https://doi.org/10.4236/ajor.2024.146009 (Accessed: 6 February 2025).

[17] Liu, F., Lu, C., Gui, L., Zhang, Q., Tong, X., and Yuan, M. (2023) *Heuristics for Vehicle Routing Problem: A Survey and Recent Advances*. Available at: https://doi.org/10.48550/arXiv.2303.04147 (Accessed: 6 February 2025).

[18] Mladenović, N. and Hansen, P. (1997) 'Variable neighborhood search', *Computers and Operations Research*, 24(11), pp. 1097–1100. Available at: https://doi.org/10.1016/S0305-0548(97)00031-2 (Accessed: 6 February 2025).

[19] Postcodes.io 'Postcodes.io API Documentation'. Available at: https://postcodes.io/docs (Accessed: 17 February 2025).

[20] Roohnavazfar, M., Pasandideh, S.H.R., and Tadei, R. (2022) 'A hybrid algorithm for the Vehicle Routing Problem with AND/OR Precedence Constraints and time windows', *Computers & Operations Research*, 143, 105766. Available at: https://doi.org/10.1016/j.cor.2022.105766 (Accessed: 18 February 2025).

[21] Scikit-learn 'DBSCAN — A Density-Based Clustering Algorithm'. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html (Accessed: 17 February 2025).

[22] Toth, P. and Vigo, D. (eds) (2014) *Vehicle Routing*. Philadelphia, PA: Society for Industrial and Applied Mathematics. Available at: https://epubs.siam.org/doi/abs/10.1137/1.9781611973594 (Accessed: 7 February 2025).

[23] Transport: Fermanagh Community Transport: Northern Ireland. Available at: https://www.fermanaghcommunitytransport.com/ (Accessed: 6 February 2025).

[24] Wang, Z., Liu, J., and Zhang, J. (2023) 'Hyper-heuristic algorithm for traffic flow-based vehicle routing problem with simultaneous delivery and pickup', *Journal of Computational Design and Engineering*, 10(6), pp. 2271–2287. Available at: https://doi.org/10.1093/jcde/qwad097 (Accessed: 7 February 2025).

[25] Wassan, N., Nagy, G., and Salhi, S. (2017) *The Multiple Trip Vehicle Routing Problem with Backhauls: Formulation and a Two-Level Variable Neighbourhood Search*. Available at: https://doi.org/10.1016/j.cor.2015.12.017 (Accessed: 6 February 2025).

[26] Wei, L., Zhang, Z., Zhang, D., and Leung, S.C.H. (2018) 'A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints', *European Journal of Operational Research*, 265(3), pp. 843–859. Available at: https://doi.org/10.1016/j.ejor.2017.08.035 (Accessed: 7 February 2025).

[27] Zhang, X., Chen, L., Gendreau, M., and Langevin, A. (2022) 'A branch-and-cut algorithm for the vehicle routing problem with two-dimensional loading constraints', *European Journal of Operational Research*, 302(1), pp. 259–269. Available at: https://doi.org/10.1016/j.ejor.2021.12.050 (Accessed: 10 February 2025).

[28] Zhong, Y., Lin, J., Wang, L., and Zhang, H. (2018) 'Discrete comprehensive learning particle swarm optimization algorithm with Metropolis acceptance criterion for traveling salesman problem', *Swarm and Evolutionary Computation*, 42, pp. 77–88. Available at: https://doi.org/10.1016/j.swevo.2018.02.017 (Accessed: 7 February 2025).

Appendices

The VRP meta-heuristic algorithm scripts described in Sections 3.3 and 3.4 are written from scratch in Python and documented in this section. Additionally, the dataset used in Section 4.3 and the corresponding algorithm output results are also presented here. The complete implementation, including all scripts, datasets, and result visualizations, is available in the public GitHub repository: *https://github.com/bebarebears/VRP-Metaheuristics-via-Python.git*.

A Variable Neighbourhood Search (VNS) Script

The input parameters n, n_k, Q and T are defined in line 5 to 8, while the computation time δ is specified in line 10. The demands d_i and travel time matrix t_{ij} in line 13 and 14 are randomly generated using seed 42.

```python
import time
import random
import numpy as np

n = 20
n_k = 4
Q = 12
T = 90

max_time = 5

random.seed(42)
demands = [0] + [random.randint(1, 3) for _ in range(1, n)]
travel_time = [[0 if i == j else random.randint(5, 15) for j in range(n)] for i in range(n)]

def calculate_route_time(route, travel_time):

    time = 0
    for i in range(len(route) - 1):
        time += travel_time[route[i]][route[i + 1]]
    return time


def is_feasible(route, vehicle_capacity, demands, travel_time, T):

    total_demand = sum(demands[node] for node in route if node != 0 and node != n-1)
    total_time = calculate_route_time(route, travel_time)
    return total_demand <= vehicle_capacity and total_time <= T

def build_full_routes(vehicle_routes):

    full_routes = []
    for route in vehicle_routes:
        full_routes.append([0] + route + [0])
    return full_routes

def initial_clustering(n, travel_time, n_k, demands, Q, T):

    nodes = list(range(1, n))
    np.random.shuffle(nodes)
    routes = [[] for _ in range(n_k)]
    route_demands = [0] * n_k

    for node in nodes:
        best_cost = float('inf')
        best_route = None
        best_position = None
        feasible_insertion_found = False


        for k in range(n_k):
            if route_demands[k] + demands[node] > Q:
                continue

            for pos in range(len(routes[k]) + 1):

                hypothetical_route = routes[k][:pos] + [node] + routes[k][pos:]

                full_route = [0] + hypothetical_route + [0]
                cost = calculate_route_time(full_route, travel_time)

                if cost <= T:
                    if cost < best_cost:
                        best_cost = cost
                        best_route = k
                        best_position = pos
                        feasible_insertion_found = True

        if not feasible_insertion_found:
            for k in range(n_k):
                if route_demands[k] + demands[node] > Q:
                    continue

                for pos in range(len(routes[k]) + 1):

                    hypothetical_route = routes[k][:pos] + [node] + routes[k][pos:]

                    full_route = [0] + hypothetical_route + [0]
                    cost = calculate_route_time(full_route, travel_time)
```

```
                    if cost < best_cost:
                        best_cost = cost
                        best_route = k
                        best_position = pos

        if best_route is None:
            return  No feasible solution found!

        routes[best_route].insert(best_position, node)
        route_demands[best_route] += demands[node]

    return routes

def transfer_node(vehicle_routes, demands, travel_time, vehicle_capacity):

    best_vehicle_routes = vehicle_routes
    best_cost = float('inf')

    for k1 in range(len(vehicle_routes)):
        for k2 in range(len(vehicle_routes)):
            if k1 == k2 or not vehicle_routes[k1]:
                continue

            for node in vehicle_routes[k1]:

                new_route_k1 = [n for n in vehicle_routes[k1] if n != node]
                new_route_k2 = vehicle_routes[k2] + [node]

                full_route_k1 = [0] + new_route_k1 + [0]
                full_route_k2 = [0] + new_route_k2 + [0]

                if is_feasible(full_route_k1, vehicle_capacity, demands, travel_time, T) and is_feasible(
                     full_route_k2, vehicle_capacity, demands, travel_time, T):

                    current_cost = calculate_route_time(full_route_k1, travel_time) + calculate_route_time(
                         full_route_k2, travel_time)
                    if current_cost < best_cost:
                        best_vehicle_routes[k1] = new_route_k1
                        best_vehicle_routes[k2] = new_route_k2
                        best_cost = current_cost

    return best_vehicle_routes

def two_opt_between_vehicles(vehicle_routes, travel_time, demands, Q):

    for k1 in range(len(vehicle_routes)):
        for k2 in range(k1 + 1, len(vehicle_routes)):
            best_cost_reduction = 0
            best_reconnection = None

            for i in range(len(vehicle_routes[k1]) - 1):
                for j in range(len(vehicle_routes[k2]) - 1):

                    route1 = vehicle_routes[k1]
                    route2 = vehicle_routes[k2]

                    new_route1 = route1[:i + 1] + route2[j + 1:]
                    new_route2 = route2[:j + 1] + route1[i + 1:]

                    if is_feasible(new_route1, Q, demands, travel_time, T) and is_feasible(new_route2, Q, demands,
                         travel_time, T):

                        old_cost = calculate_route_time([0] + route1 + [0], travel_time) + \
                                   calculate_route_time([0] + route2 + [0], travel_time)
                        new_cost = calculate_route_time([0] + new_route1 + [0], travel_time) + \
                                   calculate_route_time([0] + new_route2 + [0], travel_time)

                        cost_reduction = old_cost - new_cost

                        if cost_reduction > best_cost_reduction:
                            best_cost_reduction = cost_reduction
                            best_reconnection = (new_route1, new_route2)

            if best_reconnection:
                vehicle_routes[k1], vehicle_routes[k2] = best_reconnection

    return vehicle_routes

```

```
def two_opt(route, travel_time):

    if isinstance(route[0], list):
        optimized_routes = []
        for sub_route in route:
            full_route = [0] + sub_route + [0]
            optimized_sub_route = two_opt(full_route, travel_time)[1:-1]
            optimized_routes.append(optimized_sub_route)
        return optimized_routes

    best_route = route
    best_cost = calculate_route_time(route, travel_time)
    improved = True

    while improved:
        improved = False
        for i in range(1, len(route) - 2):
            for j in range(i + 1, len(route) - 1):
                new_route = best_route[:i] + best_route[i:j + 1][::-1] + best_route[j + 1:]
                new_cost = calculate_route_time(new_route, travel_time)
                if new_cost < best_cost:
                    best_route = new_route
                    best_cost = new_cost
                    improved = True

    return best_route

def vns_with_2opt(demands, travel_time, n_k, Q, T):

    initial_routes = initial_clustering(len(travel_time), travel_time, n_k, demands, Q, T)

    if isinstance(initial_routes, str):
        return [], float('inf'), 0

    best_routes = initial_routes
    best_cost = float('inf')
    improvement = True

    vns_iterations = 0
    while improvement:
        vns_iterations += 1
        improvement = False

        current_routes = transfer_node(best_routes, demands, travel_time, Q)

        current_routes = two_opt_between_vehicles(current_routes, travel_time, demands, Q)

        current_routes = two_opt(current_routes, travel_time)

        full_routes = build_full_routes(current_routes)
        total_cost = sum(calculate_route_time(route, travel_time) for route in full_routes)

        if total_cost < best_cost:
            best_routes = current_routes
            best_cost = total_cost
            improvement = True

    final_routes = build_full_routes(best_routes)
    return final_routes, best_cost, vns_iterations

def heuristic_vrp_vns(demands, travel_time, n_k, Q, T, max_time):

    best_overall_routes = None
    best_overall_cost = float('inf')
    start_time = time.time()
    initial_solution_count = 0
    total_vns_iterations = 0

    while time.time() - start_time < max_time:

        optimized_routes, total_time, vns_iterations = vns_with_2opt(demands, travel_time, n_k, Q, T)
        initial_solution_count += 1
        total_vns_iterations += vns_iterations


        feasible = all(
            is_feasible(route, Q, demands, travel_time, T) for route in optimized_routes
        )

```

```
235         if feasible and total_time < best_overall_cost:
236             best_overall_routes = optimized_routes
237             best_overall_cost = total_time
238 
239     if best_overall_routes is None:
240         return  No feasible solution found! , None, initial_solution_count, total_vns_iterations
241     return best_overall_routes, best_overall_cost, initial_solution_count, total_vns_iterations
242 
243 optimized_routes, total_time, initial_count, vns_iterations = heuristic_vrp_vns(demands, travel_time, n_k, Q, T,
        max_time)
244 
245 if isinstance(optimized_routes, str):
246     print(optimized_routes)
247 else:
248     for k, route in enumerate(optimized_routes):
249         full_route = [0] + route + [0]
250         route_time = calculate_route_time(full_route, travel_time)
251         print(f Vehicle {k + 1}: {route} Route time: {route_time} )
252     print(f Total Travel Time: {total_time} )
253 print(f Initial solutions generated: {initial_count} )
254 print(f Total VNS iterations: {vns_iterations} )
```

Listing 1: Variable Neighbourhood Search (VNS) Script

B Simulated Annealing (SA) Script

The input parameters n, n_k, Q and T are specified in line 6 to 9, while the SA parameters are defined in line 15 to 18. The demands d_i and travel time matrix t_{ij} in line 12 and 13 are randomly generated using seed 42.

```
1  import time
2  import math
3  import random
4  import numpy as np
5 
6  n = 20
7  n_k = 4
8  Q = 12
9  T = 90
10 
11 random.seed(42)
12 demands = [0] + [random.randint(1, 3) for _ in range(1, n)]
13 travel_time = [ [0 if i == j else random.randint(5, 15) for j in range(n)] for i in range(n) ]
14 
15 max_iterations = 10000
16 start_temp = 100.0
17 end_temp = 0.01
18 alpha = 0.99
19 
20 def calculate_route_time(route, travel_time):
21     total = 0.0
22     for i in range(len(route) - 1):
23         total += travel_time[route[i]][route[i + 1]]
24     return total
25 
26 def is_feasible(route, vehicle_capacity, demands, travel_time, T):
27     total_demand = sum(demands[node] for node in route if node != 0)
28     total_time = calculate_route_time(route, travel_time)
29     return (total_demand <= vehicle_capacity) and (total_time <= T)
30 
31 def total_solution_time(vehicle_routes, travel_time):
32     return sum(calculate_route_time(route, travel_time) for route in vehicle_routes)
33 
34 def initial_random_feasible_solution(n, n_k, Q, T, demands, travel_time, max_restarts=10000):
35     start_time = time.time()
36     nodes = list(range(1, n))
37     iterations = 0
38 
39     for iterations in range(max_restarts):
40         np.random.shuffle(nodes)
41         routes = [[0, 0] for _ in range(n_k)]
42         success = True
43         for node in nodes:
```

```
            inserted = False
            route_indices = list(range(n_k))
            np.random.shuffle(route_indices)
            for k in route_indices:
                for pos in range(1, len(routes[k])):
                    if node in routes[k]:
                        continue
                    candidate_route = routes[k][:pos] + [node] + routes[k][pos:]
                    if is_feasible(candidate_route, Q, demands, travel_time, T):
                        routes[k] = candidate_route
                        inserted = True
                        break
                if inserted:
                    break
            if not inserted:
                success = False
                break
        if success:
            return routes, iterations+1, time.time()-start_time
    return None, max_restarts, time.time()-start_time

def simulated_annealing_vrp_combined(demands, travel_time, n_k, Q, T,
                                     max_iterations,
                                     start_temp,
                                     end_temp,
                                     alpha):

    n = len(demands)

    init_start = time.time()
    sa_start = 0
    total_transfer_moves = 0
    total_inter_2opt = 0
    total_intra_2opt = 0

    current_solution, init_iters, init_time = initial_random_feasible_solution(
        n, n_k, Q, T, demands, travel_time, max_restarts
    )
    if current_solution is None:
        return  No feasible initial solution found! , None, (init_iters, 0, 0, 0, init_time, 0)

    current_cost = total_solution_time(current_solution, travel_time)
    best_solution = [route[:] for route in current_solution]
    best_cost = current_cost

    temp = start_temp
    iteration = 0

    sa_start = time.time()

    while temp > end_temp and iteration < max_iterations:

        move_type = np.random.choice(['transfer', 'inter', 'intra'])

        if move_type == 'transfer':
            total_transfer_moves += 1
            move_accepted = False
            snapshot_solution = [r[:] for r in current_solution]
            vehicle_indices = list(range(n_k))
            np.random.shuffle(vehicle_indices)

            for k1 in vehicle_indices:
                route1_nodes = snapshot_solution[k1][1:-1]
                np.random.shuffle(route1_nodes)
                for node in route1_nodes:
                    random_k2_list = list(range(n_k))
                    np.random.shuffle(random_k2_list)
                    for k2 in random_k2_list:
                        if k2 == k1:
                            continue
                        for pos in range(1, len(snapshot_solution[k2])):
                            if node in snapshot_solution[k2]:
                                continue
                            new_solution = [r[:] for r in snapshot_solution]
                            if node in new_solution[k1]:
                                new_solution[k1].remove(node)
                            new_solution[k2].insert(pos, node)
                            if (is_feasible(new_solution[k1], Q, demands, travel_time, T) and
                                is_feasible(new_solution[k2], Q, demands, travel_time, T)):
```

```
                                new_cost = total_solution_time(new_solution, travel_time)
                                delta = new_cost - current_cost
                                accept_prob = math.exp(-delta/temp) if delta >= 0 else 1.0
                                if np.random.random() < accept_prob:
                                    current_solution, current_cost = new_solution, new_cost
                                    if current_cost < best_cost:
                                        best_solution, best_cost = [r[:] for r in current_solution], current_cost
                                    move_accepted = True
                                    break
                            if move_accepted:
                                break
                        if move_accepted:
                            break
                    if move_accepted:
                        break
                if move_accepted:
                    break
        elif move_type == 'inter':
            total_inter_2opt += 1
            k1, k2 = np.random.choice(range(n_k), size=2, replace=False)
            route1 = current_solution[k1]
            route2 = current_solution[k2]

            if len(route1) >= 3 and len(route2) >= 3:
                i = np.random.randint(1, len(route1)-1)
                j = np.random.randint(1, len(route2)-1)
                new_route1 = route1[:i+1] + route2[j+1:]
                new_route2 = route2[:j+1] + route1[i+1:]

                if (is_feasible(new_route1, Q, demands, travel_time, T) and
                    is_feasible(new_route2, Q, demands, travel_time, T)):
                    new_solution = [r[:] for r in current_solution]
                    new_solution[k1], new_solution[k2] = new_route1, new_route2
                    new_cost = total_solution_time(new_solution, travel_time)
                    delta = new_cost - current_cost
                    accept_prob = math.exp(-delta/temp) if delta >= 0 else 1.0
                    if np.random.random() < accept_prob:
                        current_solution, current_cost = new_solution, new_cost
                        if current_cost < best_cost:
                            best_solution, best_cost = [r[:] for r in current_solution], current_cost
        else:
            total_intra_2opt += 1
            k = np.random.randint(0, n_k)
            route = current_solution[k]

            if len(route) >= 4:
                i = np.random.randint(1, len(route)-2)
                j = np.random.randint(i+1, len(route)-1)

                new_route = route[:i] + route[i:j+1][::-1] + route[j+1:]

                if is_feasible(new_route, Q, demands, travel_time, T):
                    new_solution = [r[:] for r in current_solution]
                    new_solution[k] = new_route
                    new_cost = total_solution_time(new_solution, travel_time)
                    delta = new_cost - current_cost

                    accept_prob = math.exp(-delta/temp) if delta >= 0 else 1.0
                    if np.random.random() < accept_prob:
                        current_solution, current_cost = new_solution, new_cost
                        if current_cost < best_cost:
                            best_solution, best_cost = [r[:] for r in current_solution], current_cost

        temp *= alpha
        iteration += 1

    sa_time = time.time() - sa_start
    stats = (
        init_iters,
        total_transfer_moves,
        total_inter_2opt,
        total_intra_2opt,
        init_time,
        sa_time
    )
    return best_solution, best_cost, stats

def validate_solution(solution, demands, Q, travel_time, T):

```

```
    for i, route in enumerate(solution):
        total_demand = sum(demands[node] for node in route if node != 0)
        total_time = calculate_route_time(route, travel_time)
        if total_demand > Q:
            print(f        Vehicle {i+1} capacity violation: {total_demand} > {Q} )
        if total_time > T:
            print(f        Vehicle {i+1} time violation: {total_time:.2f} > {T} )

def solve_vrp_combined():
    best_solution, best_cost, stats = simulated_annealing_vrp_combined(
        demands, travel_time, n_k, Q, T,
        max_iterations, start_temp, end_temp, alpha
    )

    if isinstance(best_solution, str):
        print(best_solution)
    else:
        print( Best Combined SA Solution: )
        for i, route in enumerate(best_solution):
            route_time = calculate_route_time(route, travel_time)
            print(f   Vehicle {i+1}: {route}  Time: {route_time:.2f} )
        print(f Total Travel Time: {best_cost:.2f} )

        validate_solution(best_solution, demands, Q, travel_time, T)
        print(f   Total Iterations: {stats[1]+stats[2]+stats[3]} )
        print(f   SA Runtime: {stats[5]:.2f}s )

if __name__ ==  __main__ :
    solve_vrp_combined()
```

Listing 2: Simulated Annealing (SA) Script

C Demonstration Output Result

The optimal results returned by the meta-heuristic scripts using the parameters described in Section 4.3 are presented here.

Table 5: Optimal Routes in Figure 11, i.e., $\epsilon = 0.005$, $c = 1$.

```
Vehicle 1: [0, 171, 71, 189, 177, 32, 169, 73, 197, 117, 12, 31, 54, 215, 166, 146, 156, 192, 204, 96, 118, 77,
170, 7, 11, 167, 29, 0]  Route time: 146.00
Vehicle 2: [0, 149, 23, 165, 109, 59, 38, 173, 9, 82, 190, 69, 134, 132, 19, 62, 176, 98, 140, 95, 121, 154, 57,
5, 124, 129, 76, 179, 126, 27, 110, 133, 143, 1, 120, 216, 175, 21, 211, 125, 201, 30, 0]  Route time: 274.00
Vehicle 3: [0, 153, 151, 47, 147, 150, 49, 212, 188, 178, 58, 108, 135, 159, 196, 180, 160, 25, 55, 152, 93, 157,
66, 72, 104, 131, 142, 181, 218, 8, 36, 161, 193, 80, 217, 41, 0]  Route time: 156.00
Vehicle 4: [0, 88, 84, 119, 107, 155, 198, 206, 86, 123, 3, 207, 78, 136, 191, 13, 16, 97, 44, 205, 79, 35, 137,
144, 43, 92, 0]  Route time: 128.00
Vehicle 5: [0, 56, 24, 105, 116, 14, 85, 112, 15, 26, 172, 115, 50, 45, 208, 183, 158, 2, 103, 163, 199, 99, 33, 127,
94, 130, 18, 164, 28, 141, 6, 122, 162, 17, 174, 100, 91, 74, 39, 61, 63, 219, 195, 34, 0]  Route time: 200.00
Vehicle 6: [0, 60, 20, 68, 52, 89, 184, 111, 128, 83, 185, 48, 102, 4, 22, 210, 213, 106, 148, 138, 10, 182, 194,
51, 37, 75,113, 202, 70, 186, 46, 40, 209, 42, 67, 81, 114, 101, 90, 214, 168, 87, 139, 64, 187, 203, 65, 53, 145,
200, 0]  Route time: 293.00
Total Travel Time: 1197.00
```

Table 6: Optimal Routes in Figure 12, i.e., $\epsilon = 0.005$, $c = 2$.

```
Vehicle 1: [0, 28, 19, 5, 34, 1, 26, 0]  Route time: 82.00
Vehicle 2: [0, 8, 31, 10, 0]  Route time: 41.00
Vehicle 3: [0, 24, 7, 33, 11, 3, 16, 9, 0]  Route time: 70.00
Vehicle 4: [0, 29, 18, 35, 37, 32, 36, 4, 21, 0]  Route time: 50.00
Vehicle 5: [0, 12, 22, 27, 30, 17, 20, 15, 38, 13, 0]  Route time: 114.00
Vehicle 6: [0, 2, 25, 6, 23, 14, 0]  Route time: 93.00
Total Travel Time: 450.00
```

Table 7: Optimal Routes in Figure 14, i.e., $\epsilon = 0.005$, $c = 1$ and demand weights equal to the number of postcode entries in each cluster.

```
Vehicle 1: [0, 29, 27, 128, 111, 184, 89, 14, 85, 112, 15, 26, 2, 103, 163, 199, 99, 100, 174, 17, 162, 122,
6, 141, 28, 164, 18, 130, 94, 127, 33, 91, 74, 39, 61, 183, 158, 208, 45, 50, 115, 172, 116, 105, 24, 34, 0]
Route time: 204.00
Vehicle 2: [0, 149, 23, 212, 188, 31, 54, 173, 9, 38, 59, 109, 165, 12, 117, 197, 32, 177, 198, 155, 107, 119,
169, 73, 49, 150, 147, 47, 151, 153, 0]  Route time: 140.00
Vehicle 3: [0, 76, 129, 124, 217, 80, 193, 218, 181, 142, 131, 104, 72, 66, 157, 93, 152, 55, 25, 160, 180, 121,
95, 156, 146, 166, 215, 176, 98, 140, 132, 19, 62, 134, 69, 190, 82, 178, 118, 77, 170, 7, 11, 167, 0]  Route
time: 209.00
Vehicle 4: [0, 60, 144, 137, 35, 71, 189, 79, 205, 44, 97, 16, 13, 191, 136, 78, 207, 3, 123, 86, 206, 171,
43, 92, 84, 88, 0]  Route time: 129.00
Vehicle 5: [0, 30, 201, 125, 22, 210, 213, 106, 148, 138, 10, 182, 194, 51, 37, 75, 113, 202, 70, 46, 40, 209,
42, 186, 4, 211, 175, 21, 102, 48, 185, 216, 120, 1, 133, 143, 110, 83, 126, 179, 0]  Route time: 252.00
Vehicle 6: [0, 200, 56, 145, 53, 65, 203, 187, 64, 139, 87, 101, 81, 67, 114, 90, 214, 168, 52, 68, 20, 219,
195, 63, 5, 57, 161, 36, 8, 196, 154, 159, 135, 108, 192, 204, 58, 96, 41, 0]  Route time: 229.00
Total Travel Time: 1163.00
```

D FCT Operational Dataset

The aggregated postcode dataset, presented in Table 8, covers the operational period of Fermanagh Community Transport from April 2024 to March 2025. It includes 339 unique postcode entries, along with their last usage date and the number of trips recorded. Additionally, the clustered postcode dataset, generated using the Haversine spherical distance with $\epsilon = 0.005 \approx$ 500 meters and a minimum cluster size of $c = 1$, is shown in Table 9. These datasets are utilized for the VRP demonstration in Section 4.3.1.

Table 8: Aggregated Postcodes Dataset

Postcode	Last Used	Trips
BT74 5GZ	05/10/2024	45
BT74 8BW	01/30/2025	39
BT92 4ED	01/16/2025	13
BT94 1DW	02/26/2025	53
BT93 6FA	04/04/2024	1
BT94 5JU	09/26/2024	4
BT92 9HJ	01/27/2025	2
BT94 5FF	11/12/2024	2
BT92 0LU	02/27/2025	298
BT75 0NB	05/24/2024	1
BT93 2BN	12/13/2024	19
BT94 5BX	01/21/2025	10
BT94 3LF	03/04/2025	411
BT93 0AJ	01/31/2025	87
BT92 1BE	12/17/2024	30
BT92 3DD	08/29/2024	29
BT94 1JJ	02/26/2025	250
BT93 0FB	05/31/2024	10
BT92 9HT	08/16/2024	6
BT92 9QD	01/31/2025	30
BT92 7QY	03/04/2025	211
BT92 2DZ	01/22/2025	19
BT93 6GA	03/03/2025	1198
BT93 6BR	09/06/2024	4
BT94 5BT	10/31/2024	8
BT94 3NB	01/13/2025	17
BT94 1ET	02/26/2025	45
BT92 2DH	01/31/2025	181
BT92 8FT	12/20/2024	4
BT92 4ES	01/17/2025	53
BT74 5PB	02/04/2025	43
BT92 9BY	03/04/2025	72
BT74 6HP	12/10/2024	15
BT74 4RB	03/03/2025	167
BT94 3PH	03/04/2025	536
BT94 3LY	01/15/2025	21
BT94 2FW	02/26/2025	152
BT92 9LH	02/04/2025	255
BT74 7LX	09/06/2024	2
BT93 1RF	01/10/2025	31
BT93 1RT	03/03/2025	155
BT94 5ER	01/27/2025	41
BT93 1TF	03/03/2025	185
BT92 0GT	01/22/2025	35
BT93 3GG	01/13/2025	31
BT75 0RG	02/27/2025	80
BT94 1FS	02/26/2025	73
BT92 3ER	03/04/2025	118
BT93 4DJ	09/05/2024	1
BT94 4PB	01/27/2025	94
BT94 1FL	07/05/2024	28
BT93 4AR	01/15/2025	18
BT92 7JR	02/03/2025	34
BT94 2FG	01/31/2025	150
BT93 1FY	01/28/2025	34
BT92 4BW	01/31/2025	219
BT93 4ET	02/10/2025	98
BT94 3DD	01/27/2025	59
BT93 6JE	02/26/2025	101
BT94 3DR	12/18/2024	78
BT92 4FQ	03/04/2025	213
BT94 4RY	01/21/2025	20
BT93 3EN	04/03/2024	1
BT92 1EL	01/15/2025	52
BT74 9BL	04/29/2024	2
BT94 4BY	01/30/2025	2
BT92 0JB	12/13/2024	43
BT92 8AG	11/25/2024	18
BT74 5ND	03/03/2025	224
BT94 1FU	01/13/2025	8
BT94 5ES	07/09/2024	6
BT92 0PH	02/21/2025	58
BT94 4EZ	01/29/2025	26
BT93 1TR	02/14/2025	72
BT94 3BD	02/04/2025	52
BT93 0EN	03/04/2025	136
BT74 7BG	10/18/2024	13
BT74 6DW	12/04/2024	39
BT92 9LU	04/02/2024	2
BT92 3DS	02/04/2025	43
BT92 7QU	03/03/2025	50
BT92 3BY	09/26/2024	8
BT93 5GL	03/03/2025	247
BT74 9FH	11/28/2024	8
BT92 0JS	02/03/2025	51
BT93 5FH	02/03/2025	71
BT92 9GE	03/04/2025	240
BT92 9PU	01/22/2025	22
BT92 1GL	02/21/2025	129
BT74 6JF	03/04/2025	88
BT92 8FL	11/25/2024	2
BT94 4QX	01/23/2025	39
BT93 1JJ	11/22/2024	4
BT92 7JW	10/04/2024	4
BT74 7PW	10/16/2024	16
BT93 3BL	02/26/2025	64
BT92 9AB	01/21/2025	12
BT93 3AY	03/04/2025	154
BT93 1TU	03/04/2025	844
BT92 8GW	03/03/2025	308
BT94 3AG	02/03/2025	30
BT92 3EP	02/27/2025	170
BT93 3FU	12/17/2024	25
BT74 4LS	03/04/2025	155
BT94 4SJ	02/03/2025	54
BT74 6BT	11/29/2024	47
BT92 0HG	02/06/2025	4
BT74 7LW	01/16/2025	31
BT92 7DG	02/27/2025	315
BT94 1SS	01/21/2025	88
BT93 1QX	09/27/2024	18
BT92 0QE	03/04/2025	271
BT94 1HQ	02/04/2025	110
BT92 8DQ	03/04/2025	90
BT93 5EW	01/16/2025	35
BT75 0NT	10/24/2024	8
BT74 8FB	02/21/2025	49
BT74 4AP	01/31/2025	51
BT94 1FJ	03/04/2025	302
BT94 3HQ	02/06/2025	33
BT92 8EU	07/29/2024	6
BT92 1FR	12/04/2024	69
BT94 1HU	03/04/2025	245
BT94 5BQ	05/29/2024	2
BT93 4ES	11/21/2024	17
BT93 0DD	03/03/2025	114
BT93 5DA	12/20/2024	22
BT74 4EN	02/06/2025	3
BT92 1AA	12/20/2024	9
BT93 5DZ	02/03/2025	30
BT92 0HD	11/21/2024	59
BT92 3BP	01/23/2025	28
BT74 6NX	03/04/2025	146
BT92 8DY	02/27/2025	22
BT94 1DD	02/03/2025	92
BT92 9DN	12/16/2024	4
BT74 7LU	01/23/2025	102
BT93 3FA	03/04/2025	234
BT92 7RA	03/04/2025	49
BT94 4AQ	08/01/2024	2
BT94 4EY	06/27/2024	6
BT93 1HW	03/04/2025	277
BT92 7EE	03/03/2025	208
BT92 9JT	02/10/2025	35
BT93 6JA	12/09/2024	4
BT92 9ND	10/07/2024	2
BT93 5EL	11/29/2024	28
BT93 6HD	10/25/2024	90
BT93 5EH	01/15/2025	56
BT92 4DD	03/04/2025	299
BT92 9EQ	05/02/2024	2
BT92 3DZ	03/04/2025	274
BT74 9EH	01/16/2025	4
BT93 3FT	01/30/2025	30
BT94 2EA	07/30/2024	2
BT92 0HS	01/20/2025	14
BT94 3GX	01/17/2025	14
BT74 5BB	07/29/2024	3
BT74 6HN	11/11/2024	4
BT74 9EF	01/31/2025	105
BT92 3DE	02/26/2025	60
BT93 3DF	03/04/2025	119
BT93 5ER	01/30/2025	2
BT92 4FG	02/27/2025	30
BT92 3EE	01/27/2025	15
BT94 4RZ	01/21/2025	19
BT74 6AP	01/30/2025	20
BT94 3JU	02/07/2025	83
BT94 3LU	07/04/2024	10
BT92 7DD	11/12/2024	18
BT94 5FU	01/08/2025	49
BT92 8GT	11/22/2024	4
BT74 4SA	10/02/2024	13
BT93 1UJ	01/27/2025	6
BT93 4AB	11/19/2024	20
BT94 4FW	10/10/2024	22
BT93 1FN	03/04/2025	194
BT93 3EP	09/26/2024	4
BT94 4QA	02/04/2025	3
BT94 2BE	08/14/2024	1
BT74 5AZ	09/26/2024	2
BT93 0EF	11/29/2024	3
BT92 5DZ	03/03/2025	103
BT92 9NA	11/21/2024	31
BT94 1DS	12/17/2024	6
BT93 3GL	02/04/2025	32
BT94 1AE	02/06/2025	27
BT94 5FG	03/03/2025	44
BT93 7DB	01/31/2025	122
BT93 4AP	01/21/2025	20
BT74 6HA	04/23/2024	1
BT74 5PH	12/12/2024	22
BT92 9QR	06/19/2024	6
BT74 5DN	01/31/2025	24
BT94 5EA	02/04/2025	32
BT74 6GF	08/09/2024	6
BT92 9EZ	09/26/2024	14
BT92 6DX	01/16/2025	41
BT92 7PT	01/30/2025	59
BT74 5BF	01/28/2025	81
BT92 7JU	03/04/2025	222
BT93 4EN	01/29/2025	48
BT92 5GY	03/04/2025	298
BT92 8DD	03/04/2025	308
BT93 0DN	02/21/2025	69
BT93 1RW	01/29/2025	35
BT93 6JX	01/22/2025	76
BT93 2DR	02/03/2025	2
BT93 5GG	01/31/2025	48
BT92 7DS	10/31/2024	3
BT92 9BR	03/04/2025	486
BT74 6EW	02/03/2025	36
BT92 9PD	01/30/2025	21
BT92 8DP	01/14/2025	16
BT74 5BH	01/31/2025	60
BT93 5EY	02/04/2025	3
BT93 1UL	01/23/2025	5
BT94 2JT	01/30/2025	58
BT74 5DE	02/21/2025	90
BT92 7AW	09/05/2024	2
BT93 6DZ	03/03/2025	234
BT93 3FJ	01/16/2025	10
BT92 2DY	12/17/2024	48
BT94 5FN	01/31/2025	117
BT93 2AA	01/30/2025	8
BT74 4ES	01/29/2025	26
BT94 5BN	03/03/2025	193
BT94 3DE	03/04/2025	155
BT92 8LA	03/04/2025	331
BT94 3EL	05/14/2024	1
BT92 5FF	12/13/2024	42
BT94 2AQ	04/24/2024	2
BT93 0BU	06/06/2024	19
BT92 7AU	02/21/2025	122
BT92 0AX	01/06/2025	35
BT94 2JB	10/31/2024	2
BT92 4EX	12/17/2024	9
BT92 5GU	01/07/2025	2
BT92 6ED	12/18/2024	11
BT93 5EX	01/17/2025	12
BT74 7HQ	05/10/2024	19
BT92 0LB	12/11/2024	11
BT74 5NU	01/31/2025	11
BT92 9NX	01/29/2025	79
BT92 4DJ	03/04/2025	69
BT92 9QT	11/07/2024	22
BT94 3FJ	02/27/2025	95
BT92 7DX	02/06/2025	142
BT92 8EW	03/04/2025	57
BT94 5EE	10/24/2024	12
BT93 5DJ	11/22/2024	11
BT94 3EH	11/07/2024	40
BT94 5EP	03/04/2025	164
BT92 0GD	11/12/2024	4
BT94 3HS	03/04/2025	225
BT94 4EW	12/16/2024	2
BT93 0FQ	11/29/2024	14
BT94 5DF	03/04/2025	233
BT93 4AE	03/03/2025	209
BT74 4DA	03/04/2025	36
BT74 7NR	01/10/2025	6
BT94 3NP	12/11/2024	16
BT94 1HE	12/04/2024	9
BT94 1RL	04/16/2024	2
BT92 4GU	03/04/2025	144
BT75 0RY	07/03/2024	1
BT92 9FZ	10/10/2024	4
BT93 6GB	01/22/2025	43
BT94 1EQ	03/04/2025	26
BT92 7FN	06/11/2024	17
BT94 2JL	11/13/2024	94
BT94 4QS	01/29/2025	98
BT74 6HR	05/07/2024	4
BT74 4GU	10/25/2024	74
BT92 6AA	12/06/2024	1
BT94 1DX	12/18/2024	2
BT74 5JE	01/31/2025	35
BT92 9NT	08/21/2024	31
BT93 5FW	01/31/2025	2
BT94 4EE	03/04/2025	297
BT93 2BY	02/03/2025	78
BT92 4BP	12/04/2024	3
BT94 2AW	05/08/2024	4
BT92 0HU	01/27/2025	6
BT92 1BJ	03/04/2025	130
BT93 2AZ	02/27/2025	64
BT74 8EA	01/28/2025	46
BT93 4BA	02/27/2025	75
BT94 3BH	10/22/2024	2
BT74 9DW	02/03/2025	2
BT94 4TH	03/04/2025	93
BT93 1NU	01/16/2025	152
BT92 7AA	03/04/2025	299
BT93 0JH	01/16/2025	24
BT94 4DQ	01/22/2025	22
BT93 1SN	01/31/2025	17
BT92 0PF	03/03/2025	202
BT93 3EU	12/18/2024	4
BT93 3GQ	01/10/2025	4
BT74 7NX	11/08/2024	2
BT92 2BF	01/22/2025	44
BT94 2FN	03/04/2025	54
BT93 0DT	02/21/2025	32
BT92 5FD	12/18/2024	19
BT94 3FR	11/12/2024	26
BT94 2GN	11/22/2024	33
BT94 2DU	12/11/2024	9
BT92 9DW	12/16/2024	8
BT74 7LE	01/28/2025	4
BT93 7EJ	01/31/2025	144
BT93 3DH	12/11/2024	15
BT74 9FJ	03/04/2025	384
BT94 4HJ	01/23/2025	4
BT93 8BX	01/21/2025	22
BT74 7NA	11/13/2024	4
BT94 2GA	01/17/2025	2
BT94 1BQ	03/04/2025	367
BT94 5AS	11/12/2024	11
BT92 4BE	11/15/2024	57
BT93 4FL	06/10/2024	2
BT93 6LL	08/07/2024	10
BT93 6DF	11/19/2024	9
BT74 6FR	08/28/2024	8
BT94 3JP	09/17/2024	4
BT93 2DY	05/24/2024	3
BT74 7EX	12/10/2024	18
BT93 5BT	01/31/2025	25
BT93 5FJ	03/04/2025	322
BT92 7DZ	03/04/2025	332
BT93 6HE	01/22/2025	23
BT93 0ED	11/21/2024	4
BT94 4RB	03/04/2025	396
BT94 3NZ	03/04/2025	54
BT92 8GZ	12/13/2024	2
BT93 4AD	04/24/2024	2
BT94 2AL	03/04/2025	247
BT92 8AD	03/04/2025	47
BT92 3EU	02/03/2025	73
BT92 2AQ	11/20/2024	24

Table 9: Clustered Postcodes Dataset with $\epsilon = 0.005$ and $c = 1$

Cluster	Postcode	Trips
0	BT74 8BW	39
1	BT92 4ED	13
2	BT94 1DW	1004
-	BT94 1JJ	-
-	BT94 1ET	-
-	BT94 1FS	-
-	BT94 1FL	-
-	BT94 1FU	-
-	BT94 1HQ	-
-	BT94 1FJ	-
-	BT94 1DD	-
-	BT94 1DS	-
-	BT94 1HE	-
-	BT94 1EQ	-
-	BT94 1DX	-
3	BT93 6FA	1
4	BT94 5JU	4
5	BT92 9HJ	2
6	BT94 5FF	352
-	BT94 5BT	-
-	BT94 5ER	-
-	BT94 5ES	-
-	BT94 5BQ	-
-	BT94 5FU	-
-	BT94 5DF	-
-	BT94 5AS	-
7	BT92 0LU	367
-	BT92 0HD	-
-	BT92 0GD	-
-	BT92 0HU	-
8	BT75 0NB	1
9	BT93 2BN	19
10	BT94 5BX	10
11	BT94 3LF	754
-	BT94 3LY	-
-	BT94 3HQ	-
-	BT94 3LU	-
-	BT94 3HS	-
-	BT94 3NZ	-
12	BT93 0AJ	358
-	BT93 0EN	-
-	BT93 0DD	-
-	BT93 0EF	-
-	BT93 0FQ	-
-	BT93 0ED	-
13	BT92 1BE	30
14	BT92 3DD	29
15	BT93 0FB	10
16	BT92 9HT	6
17	BT92 9QD	30
18	BT92 7QY	578
-	BT92 7JR	-
-	BT92 7DG	-
-	BT92 7DD	-
19	BT92 2DZ	67
-	BT92 2DY	-
20	BT93 6GA	1278
-	BT93 6JA	-
-	BT93 6JX	-
21	BT93 6BR	4
22	BT94 3NB	17
23	BT92 2DH	181
24	BT92 8FT	4
25	BT92 4ES	53
26	BT74 5PB	43
27	BT92 9BY	72
28	BT74 6HP	171
-	BT74 7BG	-
-	BT74 6JF	-
-	BT74 6BT	-
-	BT74 6HN	-
-	BT74 6HR	-
29	BT74 4RB	202
-	BT74 5JE	-
30	BT94 3PH	536
31	BT94 2FW	152
32	BT92 9LH	550
-	BT92 9GE	-
-	BT92 9PU	-
-	BT92 9AB	-
-	BT92 9PD	-
33	BT74 7LX	88
-	BT74 7PW	-
-	BT74 7LW	-

Cluster	Postcode	Trips
-	BT74 7HQ	-
-	BT74 7NX	-
-	BT74 7EX	-
34	BT93 1RF	471
-	BT93 1RT	-
-	BT93 1TF	-
-	BT93 1TR	-
-	BT93 1UJ	-
-	BT93 1UL	-
-	BT93 1SN	-
35	BT92 0GT	78
-	BT92 0JB	-
36	BT93 3GG	379
-	BT93 3BL	-
-	BT93 3FA	-
-	BT93 3EP	-
-	BT93 3GL	-
-	BT93 3FJ	-
-	BT93 3GQ	-
37	BT75 0RG	80
38	BT92 3ER	118
39	BT93 4DJ	1
40	BT94 4PB	194
-	BT94 4RY	-
-	BT94 4QX	-
-	BT94 4RZ	-
-	BT94 4FW	-
41	BT93 4AR	38
-	BT93 4AP	-
42	BT94 2FG	204
-	BT94 2FN	-
43	BT93 1FY	228
-	BT93 1FN	-
44	BT92 4BW	219
45	BT93 4ET	394
-	BT93 4ES	-
-	BT93 4AB	-
-	BT93 4EN	-
-	BT93 4AE	-
-	BT93 4AD	-
46	BT94 3DD	59
47	BT93 6JE	101
48	BT94 3DR	94
-	BT94 3NP	-
49	BT92 4FQ	213
50	BT93 3EN	1
51	BT92 1EL	52
52	BT74 9BL	6
-	BT74 9EH	-
53	BT94 4BY	2
54	BT92 8AG	532
-	BT92 8FL	-
-	BT92 8DQ	-
-	BT92 8EU	-
-	BT92 8GT	-
-	BT92 8DD	-
-	BT92 8EW	-
-	BT92 8AD	-
55	BT74 5ND	347
-	BT74 7LU	-
-	BT74 5NU	-
-	BT74 7NR	-
-	BT74 7NA	-
56	BT92 0PH	58
57	BT94 4EZ	331
-	BT94 4EY	-
-	BT94 4EW	-
-	BT94 4EE	-
58	BT94 3BD	52
59	BT74 6DW	59
-	BT74 6AP	-
60	BT92 3DS	43
61	BT92 7QU	50
62	BT92 3BY	8
63	BT93 5GL	247
64	BT74 9FH	8
65	BT92 0JS	51
66	BT93 5FH	71
67	BT92 1GL	129
68	BT92 7JW	4
69	BT93 3AY	154
70	BT93 1TU	844
71	BT92 8GW	308
72	BT94 3AG	30

Cluster	Postcode	Trips
73	BT92 3EP	170
74	BT93 3FU	25
75	BT74 4LS	155
76	BT94 4SJ	54
77	BT94 1SS	88
78	BT93 1QX	18
79	BT92 0QE	271
80	BT93 5EW	35
81	BT75 0NT	8
82	BT74 8FB	49
83	BT74 4AP	51
84	BT92 1FR	69
85	BT94 1HU	245
86	BT93 5DA	359
-	BT93 5EY	-
-	BT93 5EX	-
-	BT93 5FJ	-
87	BT74 4EN	53
-	BT74 6GF	-
-	BT74 4DA	-
-	BT74 6FR	-
88	BT92 1AA	9
89	BT93 5DZ	30
90	BT92 3BP	28
91	BT74 6NX	183
-	BT74 6HA	-
-	BT74 6EW	-
92	BT92 8DY	22
93	BT92 9DN	4
94	BT92 7RA	49
95	BT94 4AQ	2
96	BT93 1HW	277
97	BT92 7EE	208
98	BT92 9JT	35
99	BT92 9ND	2
100	BT93 5EL	84
-	BT93 5EH	-
101	BT93 6HD	324
-	BT93 6DZ	-
102	BT92 4DD	299
103	BT92 9EQ	2
104	BT92 3DZ	274
105	BT93 3FT	30
106	BT94 2EA	35
-	BT94 2GN	-
107	BT92 0HS	14
108	BT94 3GX	14
109	BT74 5BB	3
110	BT74 9EF	105
111	BT92 3DE	60
112	BT93 3DF	119
113	BT93 5ER	2
114	BT92 4FG	30
115	BT92 3EE	88
-	BT92 3EU	-
116	BT94 3JU	83
117	BT94 4QA	3
118	BT94 2BE	1
119	BT74 5AZ	2
120	BT92 5DZ	103
121	BT92 9NA	31
122	BT94 1AE	27
123	BT94 5FG	44
124	BT93 7DB	122
125	BT74 5PH	22
126	BT92 9QR	6
127	BT74 5DN	24
128	BT94 5EA	32
129	BT92 9EZ	14
130	BT92 6DX	41
131	BT92 7PT	59
132	BT74 5BF	81
133	BT92 7JU	222
134	BT92 5GY	298
135	BT93 0DN	120
-	BT93 0BU	-
-	BT93 0DT	-
136	BT93 1RW	35
137	BT93 2DR	2
138	BT93 5GG	48
139	BT92 7DS	3
140	BT92 9BR	486
141	BT92 8DP	16
142	BT74 5BH	60

Cluster	Postcode	Trips
143	BT94 2JT	305
-	BT94 2AL	-
144	BT74 5DE	90
145	BT92 7AW	2
146	BT94 5FN	117
147	BT93 2AA	8
148	BT74 4ES	28
-	BT94 3BH	-
149	BT94 5BN	193
150	BT94 3DE	155
151	BT92 8LA	331
152	BT94 3EL	1
153	BT92 5FF	42
154	BT94 2AQ	8
-	BT94 2JB	-
-	BT94 2AW	-
155	BT92 7AU	122
156	BT92 0AX	35
157	BT92 4EX	9
158	BT92 5GU	2
159	BT92 6ED	11
160	BT92 0LB	11
161	BT92 9NX	79
162	BT92 4DJ	69
163	BT92 9QT	22
164	BT94 3FJ	95
165	BT92 7DX	142
166	BT94 5EE	12
167	BT93 5DJ	11
168	BT94 3EH	40
169	BT94 5EP	164
170	BT94 1RL	2
171	BT92 4GU	144
172	BT75 0RY	1
173	BT92 9FZ	4
174	BT93 6GB	43
175	BT92 7FN	17
176	BT94 2JL	94
177	BT94 4QS	98
178	BT74 4GU	74
179	BT92 6AA	1
180	BT92 9NT	31
181	BT93 2BY	142
-	BT93 2AZ	-
182	BT92 4BP	3
183	BT92 1BJ	130
184	BT74 8EA	46
185	BT93 4BA	75
186	BT74 9DW	2
187	BT94 4TH	93
188	BT93 1NU	152
189	BT92 7AA	299
190	BT93 0JH	24
191	BT94 4DQ	22
192	BT92 0PF	202
193	BT93 3EU	4
194	BT92 2BF	44
195	BT92 5FD	19
196	BT94 3FR	26
197	BT94 2DU	9
198	BT92 9DW	8
199	BT74 7LE	4
200	BT93 7EJ	144
201	BT93 3DH	15
202	BT74 9FJ	384
203	BT94 4HJ	4
204	BT93 8BX	22
205	BT94 2GA	2
206	BT94 1BQ	367
207	BT92 4BE	57
208	BT93 4FL	2
209	BT93 6LL	10
210	BT93 6DF	9
211	BT94 3JP	4
212	BT93 2DY	3
213	BT93 5BT	25
214	BT92 7DZ	332
215	BT93 6HE	23
216	BT94 4RB	396
217	BT92 8GZ	2
218	BT92 2AQ	24