A Genetic Algorithm for the Multiple-Product Uncapacitated p-Hub Location and Arc Problem

by

D. (Demi) de Best

to obtain the degree of Master of Science
at the Erasmus University Rotterdam.

Erasmus University Rotterdam
Erasmus School of Economics
Master Econometrics and Management Science
Specialization Analytics and Operations Research In Logistics

Student number:	508454		
Project duration:	February 3, 2023	–	August 31, 2023
Thesis committee:	dr. R. Spliet		Supervisor
	B.T.C. van Rossum		Second assessor

Acknowledgments

I am immensely grateful to everyone who has made completing my master's thesis possible. The support and contributions have been invaluable.

First, I thank the Road Network Planning & Engineering team at FedEx Express for the opportunity to conduct this research. I am especially grateful for the guidance of M. Faber. You have broadened my understanding of the challenges in the design of road networks.

I wish to express my gratitude to dr. R. Spliet, for the guidance and support throughout this research. Your expertise provided crucial insights and motivated me to delve deeper into the problem. Thank you for trusting my abilities, as it was a driving force that kept me on track. Further, I thank B.T.C. van Rossum for thoroughly reviewing my thesis and preparing my defense.

Lastly, I would like to thank my friends and family for their support throughout my academic journey. Every moment has been meaningful, from the hours spent studying together to the following celebrations. In particular, I would like to express my heartfelt gratitude to Ruben. Your unwavering encouragement and positivity have been immeasurable.

Abstract

For FedEx Express, it is necessary to offer competitive prices and reliable services to attract customers. Achieving this depends on the efficiency of their operational processes. One of these processes is the movement of parcels between different depots through hubs. By strategically positioning these hubs, this linehaul process can be optimized.

This research aims to optimize the location of a new hub from a limited number of possibilities. To determine this location, we focus on minimizing linehaul costs while respecting the service time conditions. A vital cost component is the transportation by trucks. The advantages of routing via hubs should be reflected when determining these costs. Compared to the traditional modeling of economies of scale, we propose an alternative method by introducing fixed and variable arc costs to capture the complexity of the network of FedEx Express. We extend the literature on hub location problems and propose a genetic algorithm to tackle this $\mathcal{NP}$-hard problem. The genetic representation reflects which new hub location is chosen and whether we allow transportation between two locations.

We test our approach on seven instances using four experiments. The algorithm consistently selects one hub location for smaller instances across various runs. However, the algorithm encounters difficulties when applied to an extensive network with closely located alternatives. It selects different neighboring locations while the obtained linehaul costs are similar. Even though the algorithm identifies an area with high potential, it fails to find the exact optimal location for these instances. To improve the algorithm's performance, future research is necessary. Finally, we emphasize that the magnitude of the fixed arc costs relative to the variable arc costs influences the network optimization. When using this alternative modeling approach, we recommend making an informed decision on these cost components by analyzing the existing network.

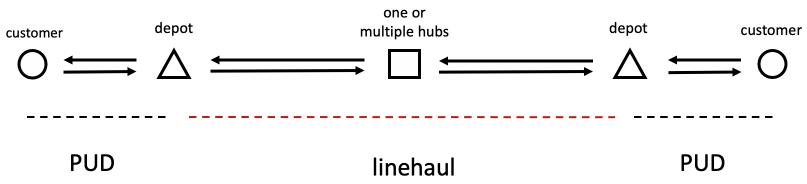
Contents

1 Introduction

FedEx Express handles time-guaranteed door-to-door shipments. In Europe, it operates an extensive road network comprising sorting centers. The shipment of packages through their network is divided into two processes, as seen in Figure 1. First, packages are collected and delivered to and from customers through a depot, known as the pickup and delivery (PUD) process. The transportation of packages between different depots through hubs is called the linehaul process. These processes are separated by fixed cut-off times.

Within a linehaul network, packages are classified based on their product type, origin, and destination. We refer to a collection of packages with the same characteristics as a *commodity*. Each commodity is transported via one or more hubs to consolidate package flows and reduce transportation costs. The specific path taken by a commodity from its origin to destination is defined as its *route*. The number of hubs encountered during transit is referred to as the number of *hub touches*.

Figure 1: Representation of package transport.



There are several benefits to optimizing hub locations for FedEx Express. The company can offer faster and more reliable deliveries. Increasing these service levels makes FedEx Express more competitive. Moreover, it leads to a reduction in overall costs. To be precise, optimizing hub locations will enhance the consolidation of packages.

This research aims to determine the optimal new hub location from a finite set of possibilities. To achieve this, we minimize the linehaul costs consisting of three components. The first component is a fixed cost incurred upon establishing and maintaining a hub. The second component is associated with transferring commodities between trucks and storing them temporarily at a hub. Finally, the third component is the cost of transportation via trucks. When determining this last component, it is crucial to take into account the benefits of routing commodities through hubs.

In classical Hub Location Problems, these economies of scale are often modeled by applying a constant discount factor to the cost of transportation between hubs. As a result, all hubs are directly connected, and the number of hub touches is restricted to at most two. However, such a modeling approach may fail to capture the complexity of the FedEx Express network by being overly restrictive.

This study proposes an alternative approach to model the economies of scale. As we focus on the strategic decision of establishing a new hub by constructing routes for commodities, the time scheduling of these routes is beyond the scope of this research. Therefore, the precise consolidation of commodities is unknown, and the number of required trucks cannot be ascertained. Consequently,

the transportation costs are estimated through linearization. We incorporate a *variable cost* per unit of commodity transported and include a *fixed cost* for transportation between two locations to avoid underestimating linehaul costs.

Our approach provides flexibility as it allows commodities to be routed via more than two hubs, which can lower the linehaul costs. Moreover, hub locations do not have to be directly connected, removing the need for transport between all hub combinations. Additionally, the allocation of depots to hubs is optimized within the model. This flexibility provides advantages over the traditional practice of assigning depots to a single or all hubs.

Furthermore, optimizing hub locations in the FedEx Express network involves additional constraints. These include a service time condition and considering multiple product types, with specific hubs having different processing capabilities. We introduce this problem as the Multiple-Product Uncapacitated p-Hub Location and Arc Problem (MP-UpHLAP). This paper establishes the $\mathcal{NP}$-hardness of the MP-UpHLAP. To address this complex problem, we propose a genetic algorithm (GA). The genetic information determines the selection of the new hub location and the permissibility of transportation between different locations.

Without the presence of developed alternative algorithms, we cannot verify the GA's ability to identify the optimal location. Therefore, we investigate whether the GA consistently converges towards a single hub location. When the GA is unable to achieve this, it suggests that the algorithm is unsuccessful in identifying the optimal hub location. We consider three real-world networks and four experimental instances. We observe that randomness within the GA and the number of possible locations do not directly impact the obtained linehaul costs in a significant way. However, when presented with extensive networks and closely situated hubs, the GA faces challenges in achieving convergence towards a singular location. To be precise, the algorithm selects different neighboring locations across various runs. We conclude that the GA fails to find the exact optimal location for these instances, even though it identifies an area with high potential. Further research is necessary to improve the performance of the GA and its analysis.

Finally, we recognize that specific instance parameters related to the alternative modeling approach can affect the optimal solution and, therefore, the outcome of GA. FedEx Express must make an informed decision regarding the values of these parameters. By increasing the maximum number of hub touches, the GA may lower the linehaul costs by further consolidating commodities. There is, however, a trade-off between the allowed number of hub touches and the computation time. Additionally, the magnitude of fixed arc costs drives the design of the network as it impacts the locations between which transportation is allowed. When these costs increase, depots are connected to fewer hubs directly.

The remainder of this paper has the following structure. The MP-UpHLAP is formally introduced in Section 2, and Section 3 discusses the relevant literature. We elaborate on our solution approach in Section 4, and Section 5 presents the experimental results. Finally, Section 6 remarks on the findings, limitations, and recommendations.

2 Problem Description

The objective is to locate a new hub in the current network to minimize the linehaul costs. To achieve this, we consider a finite set of possible locations. This section formalizes the problem by introducing the mathematical notation (see Table 1).

2.1 Mathematical Representation

Let $G = (V, A)$ be a connected graph representing the linehaul network, where V is the set of nodes. We partition V into three disjoint subsets: D represents the depots, H_e represents the existing hubs, and H_p represents the potential new hubs. We define H as the set of all hubs, i.e., $H = H_e \cup H_p$. The set A contains directed arcs between the nodes, such that the arcs $(i, j), (j, i) \in A$ if and only if $i \in H$, $j \in D \cup H_e$ and $i \neq j$. It is worth noting that the graph is incomplete since there are no direct connections between depots and between potential hubs.

Let K be the set of commodities that must be transported through the graph, and let v_k be the volume of commodity $k \in K$. We define $o(k)$ and $d(k)$ as the origin and destination depot of commodity $k \in K$, respectively. Further, P represents the different product types, such that $p(k) \in P$ corresponds to the product type of commodity $k \in K$.

Table 1: Description of sets and parameters.

	Description
V	Set of nodes
D	Set of depots
H	Set of hubs
H_e	Set of existing hubs
H_p	Set of potential hubs
A	Set of arcs
K	Set of commodities
P	Set of products
v_k	Volume of commodity k
$t_{i,j}$	Travel time of arc (i, j)
p_v	Processing time of node v
s_k	Maximum service time of commodity k
ζ	Maximum number of hub touches
b_h^p	Binary parameter that is equal to 1 if product p can be routed via hub h, and 0 otherwise
f_h	Fixed cost of hub h
κ_v^p	Variable cost of node v for product p
$\delta_{i,j}$	Fixed cost of arc (i, j)
$\tau_{i,j}$	Variable cost of arc (i, j)

For each commodity, we seek to find a path in the graph that connects its origin and destination depot. Each arc $(i, j) \in A$ has a travel time $t_{i,j}$, and every node $v \in V$ has a processing time p_v. We define the service time of a path as the sum of the travel times and processing times of the arcs and nodes in the path. A feasible path, or route, for commodity $k \in K$ has at most ζ hub touches, and the service time does not exceed the maximum service time s_k. Additionally, $b_h^{p(k)} = 1$ must be satisfied for every hub on the route. Here, b_h^p indicates whether product type $p \in P$ can be routed via hub $h \in H$. Moreover, only one new hub $h \in H_p$ is established. We assume every commodity has a route in G for $\zeta \geq 2$, independent of the location of the new hub.

The objective is to find routes for all commodities that minimize the linehaul costs consisting of three components. Firstly, a fixed cost f_h is incurred when hub $h \in H$ is established. The second component involves a variable cost κ_v^p that is charged for every volume unit of product type $p \in P$ routed through a node $v \in V$. Finally, the third component comprises transportation costs. These costs consist of a fixed cost $\delta_{i,j}$ that is charged when the arc $(i, j) \in A$ is included in any of the routes, in addition to a variable cost $\tau_{i,j}$ that is incurred for each unit volume routed via the arc.

3 Literature Review

This chapter provides an in-depth review of the current research on the problem and positions our work within the existing literature.

3.1 Hub Location Problem

The problem of finding the optimal location of hubs within a network is commonly referred to as the Hub Location Problem (HLP). In their paper, Alumur et al. (2021) provide a comprehensive overview of classical HLPs and their problem formulations. These HLPs route demand flows through the network and aim to minimize the routes' total cost (distance, time) or the maximum cost (distance, time) of a single route. The typical assumptions made in HLPs are:

- ❑ Demand flows between origin-destination nodes can be directed through hubs.
- ❑ There are benefits associated with routing flows through hubs, i.e., economies of scale.
- ❑ The distances between all nodes satisfy the triangular inequality.
- ❑ The objective depends on the hub locations and the routing of the demand flows.

Our problem satisfies these conditions, and the focus is on literature that considers a general network topology[1] as this coincides with our research.

The Uncapacitated Hub Location Problem (UHLP) (O'Kelly, 1992) assumes that hubs have no capacity limit and a fixed cost is associated with opening a hub. One of the model's decisions is to determine the number of hubs to be established. If this number is given as input, the problem is known as the Uncapacitated p-Hub Location Problem (UpHLP) (O'kelly, 1987). In our research, we aim to place only one new hub in the network. However, the solution approaches for UHLPs can often easily be adjusted to fit pUHLPs.

HLP's are generally divided into two categories: single allocation and multiple allocation. In the former, all incoming and outgoing volumes of a non-hub can only be routed via a single hub, whereas in the latter, the volume can be split up and come in or go out through all hubs. Campbell (1994) presented the first linear programming formulations for both allocation types.

In our research, the hubs to which a depot is allocated are optimized within the model. It is related to the Uncapacitated r-Allocation Hub Location Problem, presented by Mokhtar et al. (2019), where each non-hub is allocated to exactly r hubs. This assumption, however, is too restrictive for our problem as the number r may vary per depot and is unknown in advance.

The modeling of economies of scale in HLPs is strongly related to the optimal number of hub touches. In most research, a constant discount factor is used to reflect the benefits of routing flows through hubs (Alumur et al., 2021). This factor is multiplied by the cost (distance, time) of arcs

[1] Some studies assume a specific network structure, such as hub-and-spoke or tree, which is then utilized in their model (Alumur et al., 2021).

between hubs, or *hub arcs*. This approach, in combination with the triangle inequality, results in an optimal solution where flows have at most two hub touches, and all hubs are fully connected (Alumur et al., 2021). To reduce the complexity of the problem, the formulations and heuristics are conditioned on this result.

Given the cost structure considered in our paper, hubs may not be directly connected. Additionally, having more than two hub touches may be optimal. Therefore, evaluating the literature on alternative modeling approaches for economies of scale is necessary. Alumur et al. (2021) provide an up-to-date overview on the approaches that have been explored. It is worth noting that for these approaches, the literature is generally limited to exact algorithms.

Several studies explore using non-linear or piece-wise linear cost functions to determine transportation costs per arc (O'Kelly and Bryan, 1998; Horner and O'Kelly, 2001). Other studies have incorporated discount factors only when the flow through the arc exceeds a certain threshold (Podnar et al., 2002). Also, Brimberg et al. (2020) relax the triangular inequality condition, which can lead to an optimal solution with more than two hub touches.

Furthermore, Kimms (2006) models the economies of scale using fixed vehicle costs. The objective depends on the number of vehicles necessary to route flows via arcs. These models focus less on the strategic decision of establishing hubs and are referred to as Network Decision Problems (NDPs). Their solution may be sensitive to fluctuations in daily demand as this can change the number of necessary vehicles. Other studies have extended this approach by allowing for multiple vehicle types to be selected per arc in the network (Van Essen, 2009; Serper and Alumur, 2016).

To prevent connecting all hubs directly, another possibility is to select a subset of hub arcs by including fixed hub arc costs. This is a variant of the UHLP, known as the Incomplete Hub Network Problem (IHNP) (Alumur et al., 2009; de Sá et al., 2018) or Hub Arc Location Problem (HALP) (Nickel et al., 2001). In contrast to our problem, fixed costs on arcs between hubs and non-hubs are not considered. These costs are deemed irrelevant in classical single allocation problems since every non-hub is connected to only one hub. Finally, the multiple allocation UHLP with fixed arc costs is introduced by Yoon and Current (2008) and O'Kelly et al. (2015). While not considering all our problem's features, these papers are the only HLP studies that align with our approach. This leads us to conclude that our problem considers a relatively new angle on economies of scale.

Many more extensions of classical HLPs have been explored in the literature. One current area of research involves incorporating uncertainty in both demand and cost. Contreras et al. (2011b) introduce the Stochastic Hub Location Problem (SHLP), and Correia et al. (2018) extend their work to include multiple periods and hub capacities. Recently, Ghaffarinasab (2022) proposed a formulation for the SHLP with demand following a Bernoulli distribution.

To our knowledge, no research has incorporated all the features considered in this paper. We extend the classical UpHLP to account for fixed arc costs and multiple product types. We formally introduce our problem as the Multiple-Product Uncapacitated p-Hub Location and Arc Problem (MP-UpHLAP). Before discussing solution approaches, it is important to determine the complexity of the problem. In Appendix A, we present proof demonstrating the $\mathcal{NP}$-hardness of the MP-

UpHLAP with $|H| \geq 3$. This result holds even when fixed arc costs are excluded, and the maximum number of hub touches is restricted to at most two.

3.2 Solution Approaches

Several methods are proposed to solve HLPs, with an overview presented in Farahani et al. (2013) and Contreras and O'Kelly (2019). Exact algorithms such as Branch-and-Bound (Tanash et al., 2017), Branch-and-Cut (Labbé et al., 2005), and Benders Decomposition (de Sá et al., 2018) have been applied to HLPs. Notably, Contreras et al. (2011a) achieve promising outcomes by applying Benders decomposition to solve the UHLP, solving instances up to 200 nodes to optimality.

Our goal is to solve large-scale problems with up to 400 nodes. To the extent of our knowledge, this falls within the highest tier of HLP sizes. Additionally, we consider four distinct product types contributing to the problem's complexity. Consequently, we focus on metaheuristics to tackle this problem. These algorithms generally exploit that HLPs consist of two components: hub selection and allocation of non-hubs to hubs. We explore approaches for single and multiple allocation HLPs, although the former has received more extensive research attention.

Various types of iterative local search algorithms have been applied to solve HLPs. Algorithms such as Generalized Variable Neighborhood Search (GVNS), Tabu Search (TS), and Generalized Randomized Adaptive Search Procedure (GRASP) use a combination of randomization and local search to improve a candidate solution iteratively. To do so, neighborhood structures are specified that are problem-specific. The efficacy of these methods was initially demonstrated by Klincewicz (1992). He uses TS and GRASP to solve the single allocation pUHLP, utilizing a single neighborhood to switch between hub configurations. Non-hubs are directly allocated to the closest open hub. Furthermore, Ghaffarinasab et al. (2017) address the multiple allocation UHLP using TS and solve instances with up to 100 nodes using the same neighborhood structure. Commodities are directly routed through the least-cost hub(s) in their approach.

The single allocation UHLP has been further studied by Abyazi-Sani and Ghanbari (2016) and Guan et al. (2018), who not only switch between hub configurations but also included neighborhoods that reallocated non-hubs and changed the number of hubs. While Abyazi-Sani and Ghanbari (2016) propose a TS using a candidate list for selecting hubs, Guan et al. (2018) utilize a learning-based probabilistic TS. Here, the probability of selecting a hub depends on how frequently that option has been chosen earlier in the algorithm. This method outperforms Abyazi-Sani and Ghanbari (2016)'s TS in computation time and attempts to solve instances with up to 900 nodes.

Various alternative metaheuristics have been applied to HLPs that aim to explore a large search space and find high-quality solutions without relying heavily on the problem structure. For instance, Simulated Annealing (SA) is used to solve the single allocation UpHLP with up to 200 nodes, as studied by Ernst and Krishnamoorthy (1996). Their approach uses similar neighborhoods as iterative local search algorithms, but transitions are randomly selected using an acceptance probability.

In addition, Meyer et al. (2009) proposes an Ant Colony Optimization (ACO) algorithm for the single allocation UpHLP. This method uses a colony to optimize the hub selection and the allocation of non-hubs to hubs. Instances with up to 200 nodes are considered.

A Genetic Algorithm (GA) for the single allocation UHLP was first proposed by Abdinnour-Helm (1998) and since then has been extensively studied (Cunha and Silva, 2007; Van Essen, 2009). The algorithm determines which hubs to open and directly allocates non-hubs to the closest open hub. Topcuoglu et al. (2005) and Kratica et al. (2007) extend this method by allocating non-hubs to hubs part of the individuals in the GA to explore a larger solution space. Additionally, Marić et al. (2013) include two local search heuristics to improve every individual in the GA. They solve instances with up to 900 nodes within a reasonable time and claim to obtain a high-quality solution.

The multiple allocation Capacitated HLP is solved using a GA by Lin et al. (2012). The individuals reflect the hub configuration and the routes taken by commodities. Capacity violations are incorporated into the fitness function by including a penalty term. This study considers instances with 40 nodes.

In conclusion, it has been shown that SAs and local search algorithms such as GVNS, TB, and GRASP can generate high-quality solutions for HLPs. However, it should be noted that the effectiveness of the specified neighborhoods is contingent upon the assumption of single or multiple allocation. Moreover, due to the relatively small improvement per iteration, these methods may not be suitable for solving large-scale instances of our problem. Other metaheuristics are more appropriate for incorporating the various aspects of the problem since their approach is more general. Although research has been conducted on ACOs, such studies have primarily focused on small to medium instance sizes. Consequently, a GA appears to be the most appropriate method for resolving the MP-UpHLAP.

4 Methodology

This section provides a detailed description of the solution approach. First, we discuss the general aspects of GAs. Following that, we elaborate on the different elements comprising an individual in our approach. Then, we present an overview of the implemented genetic algorithm. The following subsections delve into the genetic operators and heuristics, further elaborating on their functionalities.

4.1 Framework

Genetic algorithms are optimization techniques that imitate the process of natural selection of biological evolution to find a (sub)optimal solution to a given problem. It is a search algorithm where a set of potential solutions is iteratively improved. While a more comprehensive explanation of GAs can be found in Mitchell (1998) and Katoch et al. (2021), we will briefly cover the key characteristics.

An individual in a genetic algorithm represents a candidate solution to the problem or can be decoded into one. The individual is specified by a chromosome, a string of genetic information representing the individual's features. Each chromosome comprises genes, units of genetic information that encode specific parts of the solution.

A group of individuals is called a population. Initially, a population of random individuals is generated, or an initialization method can be used to create an initial population. The genetic algorithm evaluates each individual using a fitness function that measures the individual's performance in solving the problem. The fitness function depends on the nature of the problem being solved.

In each iteration or generation, the aim is to improve the overall fitness of the following population by applying genetic operators. The *selection* procedure is used to choose individuals for reproduction based on their fitness values. Typically, fitter individuals have a higher probability of being selected for reproduction. Offspring is created by combining the chromosomes of two individuals using a *crossover* method. Lastly, *mutation* is applied to offspring to keep diversity within the population, i.e., explore new regions of the search space.

The new population consists of the offspring created by the reproduction process. This process is repeated for multiple generations until a stopping criterion is met. The stopping criterion may be a fixed number of generations or a fixed number of generations in which no improvement was found. The algorithm keeps track of the best individual found, which is the fittest individual.

4.2 Individual

In the proposed representation, an individual is characterized by a chromosome. Additionally, each individual also stores a feasible solution based on the chromosome. To the best of our knowledge, this particular representation has not been utilized in previous studies addressing hub location problems. However, it has similarities to approaches employed in the field of telecommunications

(Flores et al., 2003; Kulturel-Konak and Konak, 2008), particularly for network optimization. In telecommunication applications, such a representation often generates spanning trees. The following paragraphs provide an in-depth discussion of the components.

4.2.1 Chromosome

It is necessary to assign each arc in the graph an index based on the following rules:

(i) Let $a_1 = |H_e| * (|H_e| - 1 + 2 * |D|)$ and $a_2 = a_1 + 2 * |D \cup H_e|$.

(ii) For arcs connecting existing hubs with depots and other existing hubs, unique indices are assigned ranging from 0 to a_1.

(iii) For every potential hub $h \in H_p$, the outgoing and incoming arcs are assigned distinct indices ranging from a_1 to a_2.

(iv) For every node $i \in D \cup H_e$, all arcs (i, h) with $h \in H_p$ share the same index.

(v) For every node $i \in D \cup H_e$, all arcs (h, i) with $h \in H_p$ share the same index.

The genetic code consists of two segments. The first segment represents a potential hub $h \in H_p$. This hub, along with the existing hubs, is considered open. The second segment consists of a_2 binary genes representing the arc indices. An arc is active if the corresponding gene of its index is set to 1, provided that it does not originate from or terminate at a closed hub.

Figure 2: Visualization of an illustrative graph. Together, the left subfigures display all arcs with their assigned indices. The right subfigure highlights the active arcs corresponding to the presented chromosome.

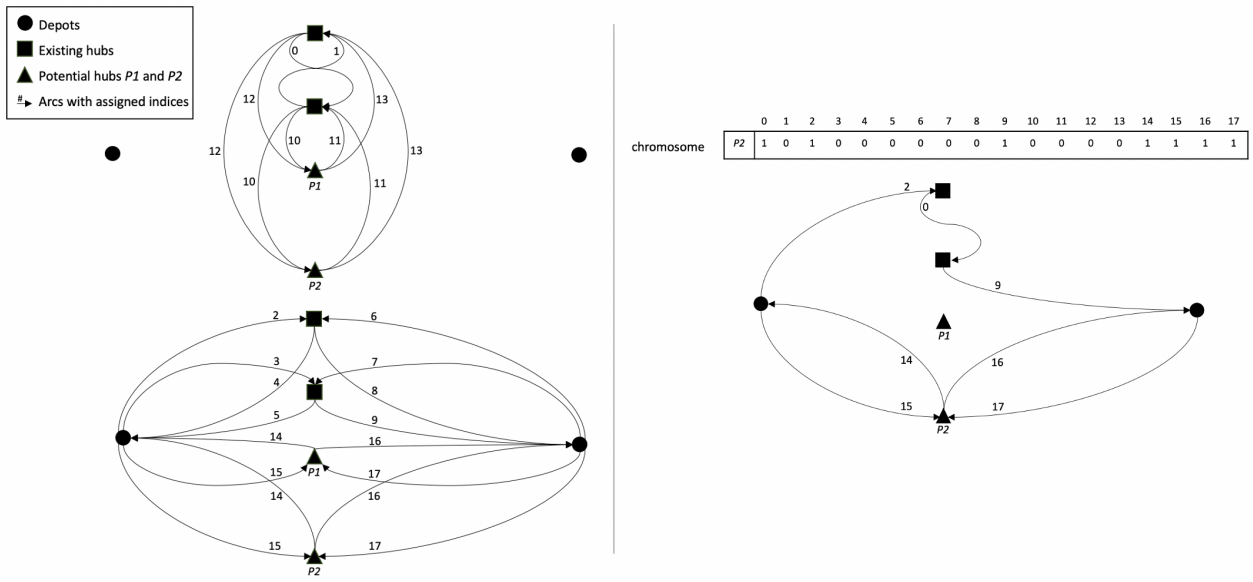


Figure 2 presents an example graph illustrating a scenario where $|D| = |H_e| = |H_p| = 2$. Consequently, the values a_1 and a_2 are 10 and 18, respectively. While the left subfigures present a feasible index assignment, the right subfigure highlights the active arcs of a given chromosome.

4.2.2 Feasible Solution

A feasible solution, denoted by $\mathcal{R}$, can be represented by a set that includes a route for each commodity. It is important to note that although the chromosome contains information about active arcs and open hubs, the genes themselves do not directly determine the routes for commodities. Consequently, the chromosome must be decoded into a feasible solution to evaluate an individual.

To guarantee the existence of a route for each commodity based on the subset of active arcs, we exclusively consider chromosomes that meet this condition. This requirement necessitates the utilization of specialized genetic operators.

The genetic operators are designed to generate new chromosomes and simultaneously decode them into feasible solutions. To be precise, the operators construct routes of sufficient quality for each commodity. While these routes may not be optimal given the sets of active arcs, this approach reduces the computation time. The feasible solution is stored along with the chromosome, and this process is discussed in more detail in Section 4.5 through Section 4.8.

4.2.3 Fitness

As mentioned in Section 2.1, the objective is to minimize the linehaul costs by opening a new hub. To achieve this, each individual is assigned a fitness value that reflects the associated costs. While the cost of active arcs and open hubs can be determined from the corresponding chromosome, the remaining costs are dependent on the routes taken by commodities. Since genetic operators decode chromosomes into feasible solutions, which are then stored, the algorithm can approximate these costs. The fitness function denoted as $\mathcal{F}(\varrho, \mathcal{R})$, takes an individual ϱ with the corresponding feasible solution $\mathcal{R}$ as input and comprises three components. The cost of a route in $\mathcal{R}$ is calculated as the sum of the variable costs, including the costs related to nodes and arcs, multiplied by the volume of the transported commodity. Additionally, the chromosome of ϱ indicates the status of hubs and arcs. Therefore, the fitness is computed by summing the costs of all routes and adding the fixed cost of active arcs and open hubs.

4.3 Genetic Algorithm

This section presents the main steps in the genetic algorithm, and the pseudo code is given by Algorithm 1. The algorithm begins with the initialization of generation zero, as discussed in Section 4.5. The population size, denoted as p_{size}, determines the number of individuals created. In the subsequent step, a local search is performed on the top fraction l_p of the individuals. Details of this local search algorithm can be found in Section 4.9. Following the local search, the individuals with the lowest fitness are marked as elite, as explained in Section 4.10. The corresponding elite set has size $e_p \times p_{size}$. Then, the algorithm continues by creating the first generation.

Algorithm 1 Genetic Algorithm

Input: population size (p_{size}), mutant proportion (m_p), elite proportion (e_p), selection proportion (s_p), local search proportion (l_p), max. generations (G_{max}), time limit (T_{max}). stagnation limit (S_{max}).
Output: individual (ϱ).

Step 0.1: initialization
$j \longleftarrow 0$ and initialize $time = stagnation = 0$.
Generate initial population Ω_j of size p_{size}. ▷ Section 4.5

Step 0.2: local search
Perform local search on the best $(l_p \times p_{size})$ individuals in Ω_j. ▷ Section 4.9

Step 0.3: elite set
$\Upsilon_j \longleftarrow \Omega_j$.
Determine the elite set Σ_j of size $(e_p \times p_{size})$ from Υ_j. ▷ Section 4.10

Step 1: selection
$j \longleftarrow j + 1$
Determine the selection set Φ_j of size $(s_p \times p_{size})$ from Υ_{j-1}. ▷ Section 4.6

Step 2: crossover
Generate $((1 - m_p - e_p) \times p_{size})$ new individuals Ω_j by applying crossover on Φ_j . ▷ Section 4.7

Step 3: mutation
Generate $(m_p \times p_{size})$ mutants Π_j using Σ_{j-1}. ▷ Section 4.8

Step 4: local search
$\Upsilon_j \longleftarrow \Pi_j \cup \Omega_j$.
Perform local search on the best $(l_p \times p_{size})$ individuals in Υ_j. ▷ Section 4.9

Step 5: elite set
$\Upsilon_j \longleftarrow \Upsilon_j \cup \Sigma_{j-1}$.
Determine the elite set Σ_j of size $(e_p \times p_{size})$ from Υ_j. ▷ Section 4.10

Step 6: stopping criteria
Update *time* and *stagnation*.
If j equals G_{max} then go to Step 7.
If *time* exceeds T_{max} then go to Step 7.
If *stagnation* exceeds S_{max} then go to Step 7.
Otherwise, go to Step 1.

Step 7: best individual
If $l_p = 0$ then perform local search on the best individual ϱ in Σ_j
Return best individual ϱ from Σ_j.

In each generation $j \geq 1$, the population Υ_j consists of three types of individuals: mutants, offspring, and elite individuals from the previous generation. These are represented by the sets Π_j, Ω_j, and Σ_{j-1}, respectively. The fractions of these individual types are as follows: mutants comprise a fraction m_p of the population, elite individuals make up a fraction e_p of the population, and the remaining fraction $(1 - m_p - e_p)$ is allocated to offspring. The selection process is explained in Section 4.6 in where individuals from the previous generation are chosen for crossover, resulting in a set Φ_j containing $s_p \times p_{size}$ individuals. The crossover operator, described in Section 4.7, generates offspring. Further details regarding the mutation process can be found in Section 4.8. Finally, another round of local search is performed on the top fraction l_p of the offspring and mutants, and the elite set is updated, marking the end of the generation.

The algorithm includes three stopping criteria to determine whether a new generation should be generated. These criteria include a maximum number of generations (G_{max}) and a time limit (T_{max}). Further, there is a limit (S_{max}) on the number of sequential iterations where the chromo-

some of the best individual remains identical. In such cases, the algorithm terminates and returns the individual with the lowest fitness. It is important to note that when the local search proportion l_p is set to zero, the algorithm first performs local search on this particular individual to improve its fitness. Otherwise, if none of the stopping criteria are met, a new generation is created.

As mentioned, the algorithm exploits specialized methods to create individuals. These operators require finding the lowest-cost route in the graph for a commodity. The following subsection focuses on an exact approach to solving this problem. Afterward, the steps of the GA are explained in detail.

4.4 Exact Route Algorithm

This section presents an exact algorithm designed to determine the lowest-cost route for a given commodity based on various input parameters. The algorithm will be utilized by multiple genetic operators in the proposed GA.

As previously mentioned in Section 2.1, we define a route as a path from the origin to the destination depot that satisfies multiple restrictions. Specifically, for a commodity $k \in K$, a path $\mathcal{P}^*$ from $o(k)$ to $d(k)$ must meet the following set of feasibility criteria:

(i) The number of hub touches in $\mathcal{P}^*$ does not exceed the maximum ζ.

(ii) The service time of $\mathcal{P}^*$ does not exceed the maximum service time of k.

(iii) Each hub in $\mathcal{P}^*$ can process product type $p(k)$.

(iv) Each hub can only be visited once in $\mathcal{P}^*$.

(v) Only depots $o(k)$ and $d(k)$ are present in $\mathcal{P}^*$.

(vi) Closed hubs can not be in $\mathcal{P}^*$.

This problem closely resembles the well-known Constrained Shortest Path Problem (CSP). It is noteworthy that the CSP is proven to be $\mathcal{NP}$-hard (Hartmanis, 1982) even for a single recourse. Hence, to address this computational complexity, we adapt the pulse algorithm by Lozano and Medaglia (2013), which has been shown to reduce the computation time compared to complete enumeration successfully.

Its general idea revolves around sending pulses through the graph, starting from the origin node and progressing toward the destination node. The algorithm follows a depth-first search strategy, and as the pulses traverse the network, the visited nodes form a path. Throughout the algorithm, the cost and resources of these paths are tracked. When a pulse reaches the destination node, it signifies the completion of a path. The pulse algorithm employs various pruning strategies to improve efficiency and limit the search space. These pruning strategies aim to discard suboptimal paths early on.

Algorithm 2 provides an overview of the adapted pulse algorithm. It takes as input a commodity $k \in K$, an open hub $h \in H_p$, and the maximum number of hub touches ζ. Moreover, the input includes temporary arc costs Δ, where $\widetilde{\delta}_{i,j} \in \Delta$ represents the temporary cost of arc $(i,j) \in A$. The algorithm begins with a path from the origin depot, and the cost and service time are initialized accordingly (see lines 4-5). Then, the pulse function is called, which returns the optimal path[1] and its corresponding cost.

Algorithm 2 Exact Route Algorithm

1: **Input:** commodity (k), new hub (h), max. hub touches (ζ), temporary costs (Δ).
2: **Output:** optimal path ($\mathcal{P}_b$).

3: Set $n = o(k)$.
4: Set $\mathcal{P} = \{n\}$.
5: Set $\mu = p_n$.
6: Set $\varsigma = \kappa_n^{p(k)} \times v_k$.
7: $\mathcal{P}_b$, $\varsigma_b \longleftarrow$ Pulse Function (k, n, h, μ, ς, $\mathcal{P}$, M, $-$, ζ, Δ).

8: **Return** $\mathcal{P}_b$.

Algorithm 3 Pulse Function

1: **Input:** commodity (k), node (n), new hub (h), service time (μ), cost (ς), path ($\mathcal{P}$), best cost (ς_b), best path ($\mathcal{P}_b$), max. hub touches (ζ), temporary costs (Δ).
2: **Output:** best path ($\mathcal{P}_b$), best cost (ς_b).

3: **if** *feasible* == *True* **then** ▷ Section 4.4.1
4: **if** *not bounded* == *True* **then** ▷ Section 4.4.2
5: **if** *not dominated* == *True* **then** ▷ Section 4.4.3
6: **if** n = d(k) **then**
7: Set $\mathcal{P}_b = \mathcal{P}$.
8: Set $\varsigma_b = \varsigma$.
9: **else**
10: **for** every arc $(i,j) \in A$ with $i = n$ **do**
11: Set $\mathcal{P}_j = \mathcal{P} \cup j$.
12: Set $\varsigma_j = \varsigma + \tau_{i,j} \times v_k + \kappa_j^{p(k)} \times v_k + \widetilde{\delta}_{i,j}$.
13: Set $\mu_j = \mu + t_{i,j} + p_j$.
14: $\mathcal{P}'$, $\varsigma' \longleftarrow$ Pulse Function (k, j, h, μ_j, ς_j, $\mathcal{P}_j$, ς_b, $\mathcal{P}_b$, ζ, Δ).

15: **if** $\varsigma' < \varsigma_b$ **then**
16: Set $\mathcal{P}_b = \mathcal{P}'$.
17: Set $\varsigma_b = \varsigma'$.

18: **Return** $\mathcal{P}_b$, ς_b.

Algorithm 3 shows the pulse function that considers a path $\mathcal{P}$ to a node n. Three pruning strategies are applied, which are further explained in Sections 4.4.1 to 4.4.3. If the path is not pruned, the algorithm checks whether it is complete. If not, it explores extending the current path to all nodes with an incoming arc from n. The cost and service time of these extended paths are computed accordingly (see lines 12-13), incorporating the temporary costs. The pulse function is recursively called, and the algorithm keeps track of the lowest-cost path to the destination.

It is worth noting that the optimal path is never pruned, making it an exact method. The performance of the algorithm is heavily dependent on the pruning strategies. Therefore, it is crucial

[1] The optimal path corresponds to the lowest-cost route since the algorithm prunes infeasible paths.

to adapt these strategies according to the specifications of the optimization problem. In the worst-case scenario, no paths are pruned, and the algorithm results in complete enumeration.

4.4.1 Feasibility Pruning

As mentioned earlier, a path needs to satisfy multiple criteria. The first pruning strategy exploits this by pruning paths that violate one or multiple restrictions. The algorithm performs a series of checks for each path $\mathcal{P}$ to a node n. When n is a depot, it verifies whether n corresponds to the destination depot. Additionally, the algorithm examines whether the service time is below the maximum service time. If either one of the checks fails, the pulse is pruned.

On the other hand, when n is a hub, it can only be visited once, and it needs to be an open hub, i.e., $n \in H_e \cup h$. Further, the algorithm examines whether n can handle commodity k and checks if the maximum number of hub touches constraint is satisfied. Finally, it verifies whether the minimum service time to the destination is below the maximum service time. That is, $\mu + t_{n,d(k)} + p_{d(k)} \leq s_k$ since the quickest path from n to the destination is by connecting them directly. If any of these checks are not passed, the path is pruned.

By employing this pruning strategy, the algorithm effectively narrows the search space by discarding infeasible paths at an early stage. Particularly, exploiting the upper bound on the number of hub touches improves the algorithm's efficiency.

4.4.2 Bound Pruning

Due to the depth-first search strategy, the algorithm can quickly find an upper bound, denoted as ς_b, on the cost. By comparing the cost of a path $\mathcal{P}$ with ς_b, the path can be pruned when this bound is exceeded. The justification for this pruning strategy lies in the non-negativity of the cost parameters. As a result, there exists no path that includes $\mathcal{P}$ that can improve the upper bound. Initially, ς_b is assigned an arbitrarily large value M, and it is updated throughout the algorithm when lower-cost paths to the destination node are discovered. Consequently, the algorithm prioritizes exploring paths with lower costs, enhancing overall computational performance.

4.4.3 Dominance Pruning

The pulse algorithm incorporates a pruning strategy based on dominance checking that keeps track of labels for all nodes encountered during the search. A path $\mathcal{P}$ to node n is dominated if there is a labeled path to node n with a lower cost, lower service time, and an equal or fewer number of hub touches. Introducing a new label for all non-dominated paths can make the number of labels per node relatively high. Even in small graphs, this results in a significant number of dominance checks. Therefore, we limit the number of labels per node to three. The algorithm keeps track of the lowest cost path, the quickest path, and an additional randomly updated path. If a path is dominated, the path is pruned from further consideration. Otherwise, the algorithm updates the labels accordingly.

4.5 Initialization

Traditional initialization methods generate individuals by randomly setting each gene in its chromosome to zero or one with a known probability. For this specific problem, this may result in chromosomes that cannot be decoded into a feasible solution. To address this challenge, a heuristic is proposed that ensures the existence of such a solution. The heuristic, outlined in Algorithm 4, outputs an individual and is repeatedly called until the initial population has the desired size.

Algorithm 4 Initialization Heuristic

Input: max. hub touches (ζ).
Output: individual (ϱ).

Draw a random hub h from H_p.

for every arc $(i,j) \in A$ **do**
  Draw a random number r between 0 and 1.
  Set $\widetilde{\delta}_{i,j} = \delta_{i,j} \times r$.

Put $\widetilde{\delta}_{i,j}$ in Δ for all $(i,j) \in A$.
Initialize $\mathcal{R} = \emptyset$.
Randomize the order of the commodities in K.

for every commodity k in K **do**
  $\mathcal{P}_k \longleftarrow$ Exact Route Algorithm (k, h, ζ, Δ). ▷ Section 4.4
  Set $\mathcal{R} = \mathcal{R} \cup \mathcal{P}_k$.

  for every arc (i,j) in route $\mathcal{P}_k$ **do**
    Set $\widetilde{\delta}_{i,j} = 0$.

$\vartheta \longleftarrow$ *createChromosome*(h, Δ).
$\varrho \longleftarrow$ *createIndividual*$(\vartheta, \mathcal{R})$.

Return ϱ.

To start, a potential hub is randomly opened. Then, each arc is assigned a temporary cost, denoted by $\widetilde{\delta}_{i,j}$. This cost is calculated by discounting the fixed cost of each arc with a random number between 0 and 1. By applying this discount, diversity is introduced in the population. To our knowledge, this discounting approach has not been investigated yet in the context of HLPs. The costs are updated throughout the algorithm, setting it to zero for an active arc.

The algorithm iteratively solves the Exact Route Algorithm for each commodity k in a randomized order to diversify the population. As mentioned earlier, this algorithm finds the lowest-cost route $\mathcal{P}_k$[2], considering the temporary costs and the maximum number of hub touches. Additionally, in each iteration, the temporary costs of arcs present in the lowest-cost route are set to zero, ensuring that other commodities are encouraged to use already active arcs.

The initialization method finishes by creating an individual. The function *createChromosome* takes a hub $h \in H_p$ and the temporary costs as inputs and constructs a chromosome. The first component of the chromosome is set to h. The genes in the second component are determined based on whether there is an active arc $(i,j) \in A$, defined by $\widetilde{\delta}_{i,j} = 0$, with the corresponding index. If such an arc exists, the gene is set to 1, otherwise it is set to 0.

[2] There exists a route for each commodity since we consider the entire network.

To evaluate the individual, its fitness must be determined. The generated chromosome is already decoded into a feasible solution within the method. This solution includes the lowest-cost routes for all commodities stored in $\mathcal{R}$. Given the subset of active arcs, these routes may not be optimal due to their dependency on the order of commodities and temporary costs. However, as explained in Section 4.2.3, such a solution is used to approximate the linehaul costs and reduce the computation time. The function *createIndividual* generates an individual, storing the chromosome and feasible solution provided as input. Finally, the fitness is determined using these two components, with the fitness function $\mathcal{F}(\varrho, \mathcal{R})$ being called as explained in Section 4.2.3.

4.6 Selection

The selection operator generates a set of individuals that are considered for crossover. This set comprises both elite individuals and mutants from the previous generation. Furthermore, tournament selection is employed as the selection procedure to expand the selection set and ensure diversity. This procedure involves randomly selecting t_s distinct individuals from the previous population and subsequently including the one with the lowest fitness value in the selection set. It is worth noting that individuals with identical chromosomes cannot be present in the selection set to ensure solution diversity. The tournament size t_s determines the number of individuals participating in each selection round, influencing overall diversity. The process is repeated until $s_p \times p_{size}$ individuals are chosen for crossover. By including mutants and elite individuals, together with applying tournament selection, the algorithm explores new solutions while still including the best traits from the previous generation.

4.7 Crossover

Crossover methods commonly produce offspring by combining the chromosomes of two parent individuals based on a predefined rule. This process typically involves alternating segments from the parent chromosomes to generate a new chromosome when dealing with binary chromosomes. However, such an approach may yield offspring that can not be decoded into a feasible solution. Hence, we present a crossover method similar to the initialization procedure. Again, a temporary cost is assigned to each arc, and an exact or heuristic approach is used to find (sub-)optimal routes. The proposed method, displayed by Algorithm 5, produces one or two offspring and is iteratively applied until the desired number of offspring is created.

First, two distinct parent individuals are randomly selected from the selection set. The potential hub that is open in the first parent is again selected. Then, the crossover operator assigns a temporary cost $\widetilde{\delta}_{i,j}$ to each arc based on whether the arc is active in either of the parent individuals. If the arc is active in at least one parent, the cost is set to a random percentage of the fixed cost of the arc. This percentage is consistent for all arcs to control the randomness while maintaining some degree of diversity. In addition, there is a mutation component where arcs that are inactive in both parents are also assigned such a cost with a probability p_{mut}. This mutation allows the

offspring to activate different arcs than their parents. The remaining arcs, which are neither active in the parents nor subject to mutation, are assigned an arbitrarily large value M.

Algorithm 5 Crossover

Input: selection set (Φ), max. hub touches (ζ), mutation probability (p_{mut}), heuristic proportion (h_p).
Output: individuals (ϱ_1, ϱ_2).

Randomly select 2 distinct parent individuals φ_1 and φ_2 from Φ.

for $q \longleftarrow 1$ to 2 **do**
  Set h to the open potential hub of φ_q.
  Draw a random number r_1 between 0 and 1.

  for every arc $(i,j) \in A$ **do**
    Draw a random number r_2 between 0 and 1.

    if arc (i,j) is active for φ_1 and/or φ_2 **then**
      Set $\widetilde{\delta}_{i,j} = \delta_{i,j} \times r_1$.
    else if $r_2 < p_{mut}$ **then**
      Set $\widetilde{\delta}_{i,j} = \delta_{i,j} \times r_1$.
    else
      Set $\widetilde{\delta}_{i,j} = M$.

  Put $\widetilde{\delta}_{i,j}$ in Δ for all $(i,j) \in A$.
  Initialize $\mathcal{R}_q = \emptyset$.
  $f \longleftarrow 1$.
  Randomize the order of the commodities in K.

  for every commodity k in K **do**
    if $f < h_p \times |K|$ **then**
      $\mathcal{P}_k \longleftarrow$ Exact Route Algorithm (k, h, ζ, Δ). ▷ Section 4.4
    else
      $\mathcal{P}_k \longleftarrow$ Heuristic Route Algorithm (k, h, ζ, Δ, (φ_1, φ_2)). ▷ Section 4.7.1

    Set $\mathcal{R}_q = \mathcal{R}_q \cup \mathcal{P}_k$.
    $f \longleftarrow f + 1$.

    for every arc (i,j) in route $\mathcal{P}_q$ **do**
      Set $\widetilde{\delta}_{i,j} = 0$.

  $\vartheta_q \longleftarrow$ *createChromosome*(h, Δ).
  $\varrho_q \longleftarrow$ *createIndividual*$(\vartheta_q, \mathcal{R}_q)$.

  if the open potential hub of φ_1 and φ_2 is identical **then**
    Return ϱ_q.

Return ϱ_1, ϱ_2.

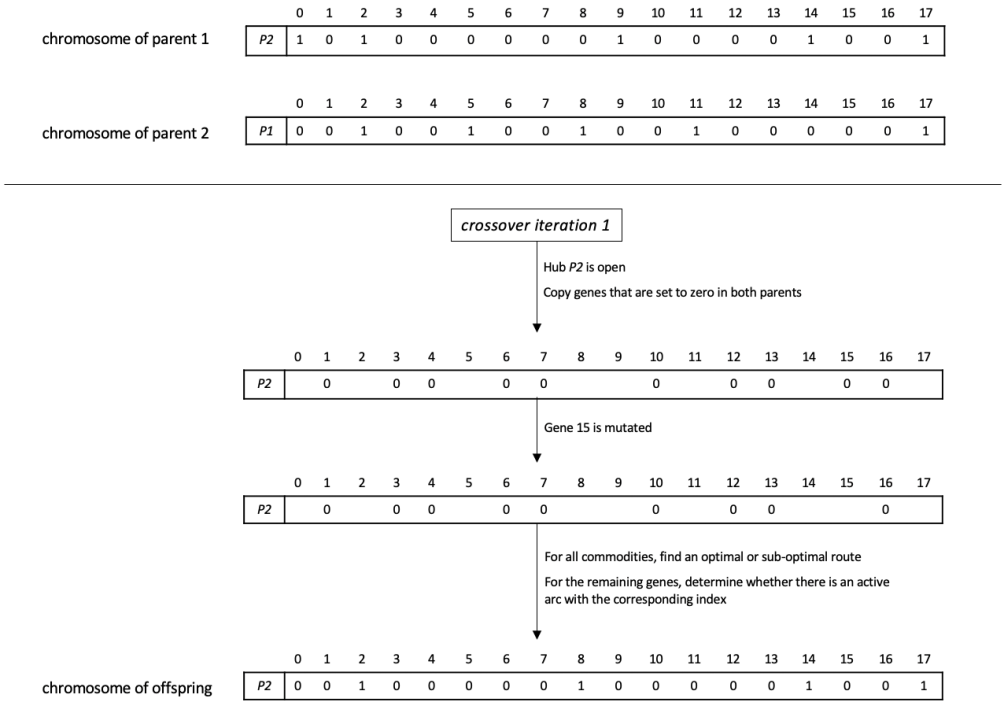
The operator then proceeds to iterate over the commodities in a randomized order. The algorithm determines the lowest-cost route using the Exact Route Algorithm if a fraction h_p of the commodities has yet to be considered. It is worth noting that the arcs with a temporary cost of M are never activated. In the Exact Route Algorithm, bound pruning leads to immediate pruning of a path that contains such an arc. Consequently, the number of considered routes is significantly reduced, leading to lower computation time compared to a scenario where all arcs are considered. Generally, we expect the number of active arcs to decrease significantly over the first generations, which accelerates the crossover operator.

Once a sufficient number of commodities have been processed, the algorithm switches to a heuristic approach. The Heuristic Route Algorithm, explained in Section 4.7.1, exploits the previously activated arcs to improve the computational efficiency further. In the worst-case scenario, the exact algorithm is called by the heuristic. The parameter h_p influences the quality of the solution since more sub-optimal routes may be selected when it is set low.

Each resulting route[3] is added to the feasible solution $\mathcal{R}$. Additionally, in each iteration, the temporary costs of arcs present in the lowest-cost route are set to zero, encouraging subsequent commodities to utilize those arcs. Finally, the crossover operator constructs offspring by creating a chromosome. Again, the constructed routes in $\mathcal{R}$ are used to approximate the linehaul costs, determining the offspring's fitness. If the open potential hub in the second parent is different from the first parent, the process is repeated for the other hub. However, one offspring is returned if the open potential hub is the same.

To clarify this operator, an example is given in Figure 3 where we consider the illustrative graph presented in Figure 2. The genes set to zero in both parents remain zero in the offspring, except for genes subject to mutation. The constructed routes determine the values of the remaining genes.

Figure 3: Crossover example.



[3] The existence of a route is ensured by the feasibility of both parents.

4.7.1 Heuristic Route Algorithm

The Exact Route Algorithm discussed in Section 4.4 optimizes the lowest-cost routes. However, it is worth noting that optimizing these routes may not always be the primary objective since it may lead to high computation times. Therefore, we propose an alternative approach, introducing a heuristic to reduce the running time of the GA. The Heuristic Route Algorithm is incorporated into the crossover operator, requiring two parent individuals as additional input.

The main idea is to swiftly identify routes of satisfactory quality whenever possible. This is achieved by leveraging the observation that when all arcs in a given route are already active, the route's cost tends to be relatively low, indicating its suitability. Algorithm 6 gives an overview of the heuristic. It examines the routes of commodity $k \in K$ in the solutions stored for both parent individuals. If all arcs in both routes have a temporary cost of zero, the algorithm returns the lowest-cost route. On the other hand, if all arcs in only a single route have a temporary cost of zero, the algorithm selects the respective route. However, if both routes do not meet this condition, the algorithm proceeds to solve the Exact Route Algorithm.

Algorithm 6 Heuristic Route Algorithm

Input: commodity (k), new hub (h), max. hub touches (ζ), temporary costs (Δ), individuals (ϱ_1, ϱ_2).
Output: best path ($\mathcal{P}_b$).

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the routes of commodity k in the solutions of ϱ_1 and ϱ_2, respectively.
Initialize *check1* = *check2* = *True*.

for every arc (i, j) in $\mathcal{P}_1$ **do**
  if $\widetilde{\delta}_{i,j} \neq 0$ **then**
    Set *check1* = *False*.
    break

for every arc (i, j) in $\mathcal{P}_2$ **do**
  if $\widetilde{\delta}_{i,j} \neq 0$ **then**
    Set *check2* = *False*.
    break

if *check1* == *check2* == *True* **then**
  Set $\mathcal{P}_b$ to lowest-cost route between $\mathcal{P}_1$ and $\mathcal{P}_2$.
else if *check1* == *True* **then**
  Set $\mathcal{P}_b = \mathcal{P}_1$.
else if *check2* == *True* **then**
  Set $\mathcal{P}_b = \mathcal{P}_2$.
else
  $\mathcal{P}_b \longleftarrow$ Exact Route Algorithm (k, h, ζ, Δ). ▷ Section 4.4

Return $\mathcal{P}_b$.

4.8 Mutation

In traditional mutation operators, random changes are typically introduced to individual genes with a predefined probability, or entirely new chromosomes are randomly generated. However, as mentioned in Section 4.5, these operators may not be applicable as they may result in chromosomes that cannot be decoded into feasible solutions. To overcome this problem, the algorithm creates mutants from scratch while utilizing elite individuals. This algorithm is repeatedly called until the desired number of mutants is achieved, expanding the solution space explored by the GA.

The mutation operator is presented in Algorithm 7. The process begins by randomly selecting an individual from the elite set. Subsequently, a closed hub is chosen associated with the elite individual. Opening this alternative hub for the mutant can be interpreted as a mutation of the first segment of the chromosome. However, directly copying the second segment is not possible, as it may lead to issues with the feasibility.

To assign temporary costs to each arc, the following rules are employed: inactive arcs in the elite individual are assigned their fixed cost, while active arcs have their fixed cost discounted by a random factor between 0 and 0.5. By reducing the cost of previously active arcs, we increase the likelihood of re-activating these promising arcs for the mutant. The algorithm proceeds by solving the Exact Route Algorithm for the commodities in a randomized order while updating the temporary costs of activated arcs. The algorithm terminates by creating the chromosome for the mutant, storing the resulting feasible solution, and determining its fitness.

Algorithm 7 Mutation

Input: elite individuals (Σ), max. hub touches (ζ).
Output: individual (ϱ).

Randomly select an individual φ_1 from Σ
Draw h from $H_p \setminus h^*$ where h^* represents the open potential hub of φ_1.
Draw a random number r between 0 and 0.5.

for every arc $(i, j) \in A$ **do**
  if arc (i, j) is active for φ_1 **then**
    Set $\widetilde{\delta}_{i,j} = \delta_{i,j} \times r$.
  else
    Set $\widetilde{\delta}_{i,j} = \delta_{i,j}$.

Put $\widetilde{\delta}_{i,j}$ in Δ for all $(i, j) \in A$.
Initialize $\mathcal{R} = \emptyset$.
Randomize the order of the commodities in K.

for every commodity k in K **do**
  $\mathcal{P}_k \longleftarrow$ Exact Route Algorithm (k, h, ζ, Δ). ▷ Section 4.4
  Set $\mathcal{R} = \mathcal{R} \cup \mathcal{P}_k$.

  for every arc (i, j) in route $\mathcal{P}_k$ **do**
    Set $\widetilde{\delta}_{i,j} = 0$.

$\vartheta \longleftarrow$ *createChromosome*(h, Δ).
$\varrho \longleftarrow$ *createIndividual*$(\vartheta, \mathcal{R})$.

Return ϱ.

4.9 Local Search

In previous sections, we discussed the genetic operators, which exploit constructed routes to approximate the linehaul costs associated with an individual. However, given the set of active arcs, these routes may be sub-optimal due to their dependency on the order of commodity consideration and the use of temporary costs. To address this, we incorporate a local search that aims to lower an individual's fitness. This is done by re-optimizing the routes given a fixed set of active arcs. As a result, routes are optimized independently, and the resulting fitness reflects the lowest possible linehaul costs given the corresponding chromosome.

The local search, presented in Algorithm 8, begins by initializing the temporary cost of each arc. For active arcs in the individual, the temporary cost $\widetilde{\delta}_{i,j}$ is set to zero. Conversely, for arcs that are inactive, the temporary cost is set to an arbitrarily large value M to ensure their inactivity and speed up[4] the local search. Subsequently, the Exact Route Algorithm is called on all commodities, and the resulting routes are saved in the set $\mathcal{R}$. Finally, the algorithm calls the function *updateIndividual*, which updates the stored solution and determines the new fitness of the individual.

Algorithm 8 Local Search

Input: max. hub touches (ζ), individual (ϱ).
Output: individual (ϱ).

Set h to open potential hub of ϱ.

for every arc $(i,j) \in A$ **do**
  if arc (i,j) is active in ϱ **then**
    Set $\widetilde{\delta}_{i,j} = 0$.
  else
    Set $\widetilde{\delta}_{i,j} = M$.

Put $\widetilde{\delta}_{i,j}$ in Δ for all $(i,j) \in A$.
Initialize $\mathcal{R} = \emptyset$.

for every commodity k in K **do**
  $\mathcal{P}_k \longleftarrow$ Exact Route Algorithm (k, h, ζ, Δ). ▷ Section 4.4
  Set $\mathcal{R} = \mathcal{R} \cup \mathcal{P}_k$.

$\varrho \longleftarrow$ *updateIndividual*$(\varrho, \mathcal{R})$.

Return ϱ.

4.10 Elite Set

The inclusion of an elite set within GAs serves the purpose of accelerating convergence towards (sub-)optimal solutions. This subset, with a size equal to $e_p \times p_{size}$, is updated every generation by identifying the individuals that have demonstrated superiority. This is reflected by their holding of a low fitness value. By selecting these individuals, the population preserves individuals that exhibit promising genetic traits. Importantly, it should be mentioned that the elite set is designed to maintain population diversity. To achieve this, individuals with identical chromosomes are excluded from the set.

[4] In general, the execution of local search on an individual requires less time compared to generating offspring using the Exact Route Algorithm within the same generation, due to the immediate pruning of a more significant number of routes.

5 Computational Results

In this section, we present the results regarding the GA. To begin, we tune the algorithm parameters, examining the effects of different settings on two specific instances. The results are given in Section 5.2. We apply these settings to all real-world networks, as outlined in Section 5.3. Here, the results of the GA are discussed in more detail. Moreover, four experiments are conducted to investigate the effects of various factors on the algorithm's performance and outcome, as shown in Section 5.4.

The genetic algorithm is implemented in Java, using the Eclipse IDE for Java Developers version 4.27.0. The experiments are run on a single thread of an 11th Gen Intel(R) Core(TM) i7-1185G7 CPU, with 32GB of RAM. Next, we elaborate on the test instances.

5.1 Test Instances

We consider three real-world networks characterized by depots and hubs operating independently. Additionally, we generate four experimental instances to evaluate the performance of the GA. In this section, we introduce the instances by discussing the sets and parameters related to the graphs representing the networks.

5.1.1 Real-world Instances

Table 2 provides an overview of the set cardinalities for each real-world network. The networks are classified based on their sizes, and we refer to them as *small*, *medium*, and *large*. This can be done since the cardinalities of the sets increase across the instances, except for the number of product types, which remains constant. The number of depots ranges between 46 to 197, while the number of existing hubs ranges from 5 to 21. Additionally, 46 to 185 potential hub locations are considered, and each hub is assigned a unique index.

Compared to the small network, the number of commodities increases by factors of 2.98 and 7.12 for the medium and large network, respectively. Consequently, we expect that the computational time required for generating a single offspring, mutant, or initial individual will increase across the instances. This is due to the larger number of commodities that need to be routed and the likelihood of having more routes to consider, given the greater number of existing hubs.

Table 2: The set cardinalities for real-world networks.

Network	$\|V\|$	$\|D\|$	$\|H_e\|$	$\|H_p\|$	$\|A\|$	$\|K\|$	$\|P\|$
small	97	46	5	46	5172	1836	4
medium	181	83	13	85	18634	5468	4
large	403	197	21	185	89354	13064	4

For each node and commodity, the associated parameters are provided. The time and cost of driving a single truck between any two nodes are also known. However, as mentioned earlier, the transportation costs are estimated through linearization. The variable arc cost is estimated by dividing the cost per truck by the given truck capacity. Furthermore, the fixed cost is set as a percentage of the cost per truck. This percentage m and the maximum number of hub touches ζ are treated as instance parameters, which will be further discussed in Section 5.2.

5.1.2 Experimental Instances

To further investigate the performance of our genetic algorithm, we design four additional networks: *m–A*, *m–B*, *l–A*, and *l–B*. The networks m–A and m–B are intended to be comparable in size to the small network. To achieve this, we derive these sub-networks from the medium network by randomly dividing the sets of depots, existing hubs, and potential hubs into two distinct subsets. To maintain feasibility, we exclusively consider commodities from the original set K that have both their origin and destination depots within the sub-network. This process results in two distinct sets of commodities for m–A and m–B. Similarly, the instances l–A and l–B are created to be comparable in size to the medium network and are obtained by utilizing the large network.

Table 3 provides an overview of the set cardinalities. In each experimental instance, parameter values are adopted from the real-world network, except for the maximum service time, which is set to infinity to avoid infeasibility problems.

Table 3: The set cardinalities for experimental networks.

Network	|V|	|D|	$\|H_e\|$	$\|H_p\|$	|A|	|K|	|P|
m–A	97	46	6	45	5262	2001	4
m–B	100	47	7	46	5668	1884	4
l–A	203	98	10	95	22786	5643	4
l–B	206	99	11	96	23408	5816	4

5.2 Parameter Tuning

In this section, we observe the effects of various parameters in the GA to enhance its performance. For clarity, the parameters ζ and m are associated with the instances rather than the algorithm itself. For this tuning process, we fix their values at $\zeta = 2$ and $m = 0.75$. However, we will investigate the influence of these instance parameters in our subsequent analyses.

In Section 4, we introduced eleven algorithm parameters, including three stopping criteria. These stopping criteria are set as follows: $G_{max} = \infty$, $S_{max} = 30$ and $T_{max} = 18.000$ seconds. These values are chosen to ensure a reasonable balance between computational efficiency and exploring potential solutions. Furthermore, we set the tournament size parameter t_s to three, which is relatively low. This setting increases the likelihood of selecting individuals with lower fitness values, promoting exploring the search space and maintaining diversity within the population. The selection proportion parameter s_p is set to 0.65, which is relatively high. This ensures that a sufficient number of non-elite individuals and non-mutants are considered for crossover.

As a result, we are left with six parameters that require tuning. Initially, these parameters are set the following values: $p_{size} = 20 \times |H|$, $e_p = 0.2$, $p_m = 0.05$, $m_p = 0.05$, $h_p = 1$, and $l_p = 0$. This implies that the Heuristic Route Algorithm and the local search are not applied. We allow a mild level of mutation by setting the two corresponding mutation parameters relatively low.

Optimizing all parameters simultaneously would lead to excessive experiments and unmanageable computation time. Therefore, we evaluate the parameters individually and iteratively update our model throughout the analysis. By systematically varying the value of each parameter, we observe its impact on the outcome of the GA. Based on these observations, we intend to make

informed decisions regarding their optimal values. We focus on optimizing the real-world small and medium network for practical reasons. It is important to acknowledge that there are limitations to this approach. First, not all possible parameter settings are considered, which limits our findings. Additionally, as we only examine two specific test instances, the obtained results may be sensitive to the characteristics of these instances. Nevertheless, this approach allows us to observe the effects of the GA parameters to the best of our capabilities within the allowed time.

5.2.1 Population Size

We let the population size in our genetic algorithm depend on the number of hubs in the network. The number of existing hubs impacts the possible routes, suggesting that a larger solution space may exist when more existing hubs are present. On the other hand, the population size also needs to depend on the number of potential hubs. For each potential hub, there should be individuals in the initial population where the hub is open. Therefore, we consider $p_{size} \in \{5 \times |H|, 10 \times |H|, 20 \times |H|, 30 \times |H|\}$.

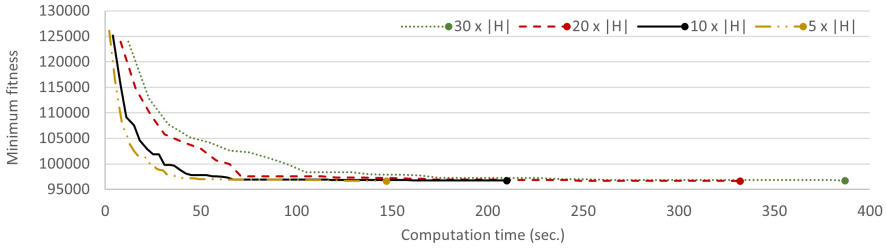
Table 4: GA results for different values of p_{size}.

Network	p_{size}	Minimum fitness value	Best hub	Number of generations	Comp. time (sec.)	Avg. comp. time generation (sec.)	Time % initialization
small	$5 \times \|H\|$	96650.10	65	139	191	1.35	1.05%
	$10 \times \|H\|$	96668.50	65	104	299	2.81	1.34%
	$20 \times \|H\|$	96608.99	65	83	492	5.76	1.63%
	$30 \times \|H\|$	96681.09	65	72	649	8.73	1.85%
medium	$5 \times \|H\|$	256215.08	133	131	2885	21.55	1.39%
	$10 \times \|H\|$	255700.54	133	131	5818	43.41	1.50%
	$20 \times \|H\|$	256421.93	133	112	9704	84.50	1.59%
	$30 \times \|H\|$	256082.42	133	128	16410	125.30	1.48%

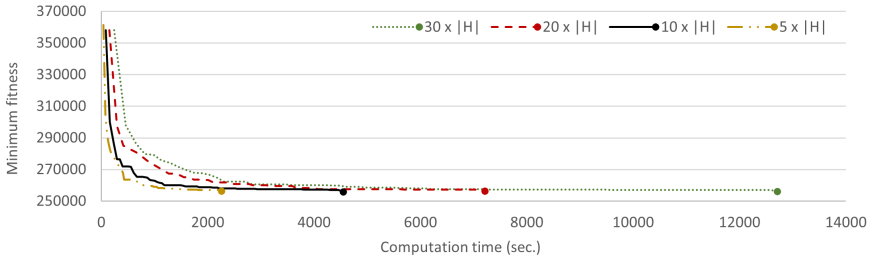
Table 4 presents the GA results for different values of p_{size}. Regardless of the population size, the same potential hub is open for both networks. For the small network, the lowest fitness value is found when $p_{size} = 20 \times |H|$, while for the medium network, $p_{size} = 10 \times |H|$ performs the best. It is, however, necessary to recognize that the differences are minimal. For the small and medium network the maximum variation in fitness is 0.06% and 0.28%, respectively. Furthermore, Figure 4 illustrates the progression of the minimum fitness value until the final stagnation phase. The GA converges slower for population sizes $20 \times |H|$ and $30 \times |H|$ compared to the other values.

As the population size grows, the average computation time per generation rises. These increases are proportional to each other, which is to be expected. Moreover, the initialization process takes longer for larger values of p_{size}. However, the percentage of time spent in the initialization heuristic remains relatively stable and is mainly influenced by the number of generations. Larger populations tend to result in fewer generations, particularly for the small network.

Figure 4: Evolution of the minimum fitness value until stagnation for different values of p_{size}.



(a) *small*



(b) *medium*

Considering the limitations of our parameter tuning approach, making an informed decision on the best parameter value is difficult since the differences are minor. Based on the convergence rate, we select $10 \times |H|$ as the population size. This setting allows the GA to explore a broader solution space compared to $5 \times |H|$ while maintaining a reasonably low computation time.

5.2.2 Elite Proportion

The elite proportion, denoted by e_p, is expected to influence the performance of the GA. Reducing the elite proportion results in more offspring per generation and more non-elite individuals considered for crossover, leading to higher diversity. However, there is a risk of losing valuable information regarding which arcs should be active and which hub should be open. Conversely, increasing the elite proportion reduces the number of offspring per generation. While preserving well-performing arcs and hubs, fewer non-elite individuals are selected for crossover. This may restrict the search space for potential solutions. We examine three values for e_p, namely 0.1, 0.2, and 0.3.

Table 5: GA results for different values of e_p.

Network	e_p	Minimum fitness value	Best hub	Number of generations	Comp. time (sec.)	Avg. comp. time generation (sec.)
small	0.1	96741.94	65	67	247	3.53
	0.2	96668.50	65	104	299	2.81
	0.3	96858.77	65	78	194	2.40
medium	0.1	256935.51	133	84	3928	45.24
	0.2	255700.54	133	131	5818	43.41
	0.3	256437.05	133	146	5659	38.03

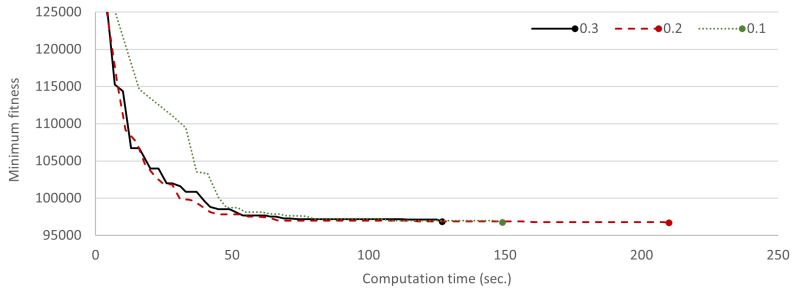
Table 5 presents an overview of the GA results, using 0.2 as the baseline for comparison. Across the different elite proportions, the fitness value varies slightly. The differences are at most 0.20% and 0.48% for the small and medium network, respectively. Again, the same hub is consistently

opened for both networks. Figure 5 illustrates the progression of the minimum fitness value until the final stagnation phase for both networks. Setting e_p to 0.1 for the small network leads to slower convergence, while for the medium network, all values of e_p exhibit a similar convergence pattern.

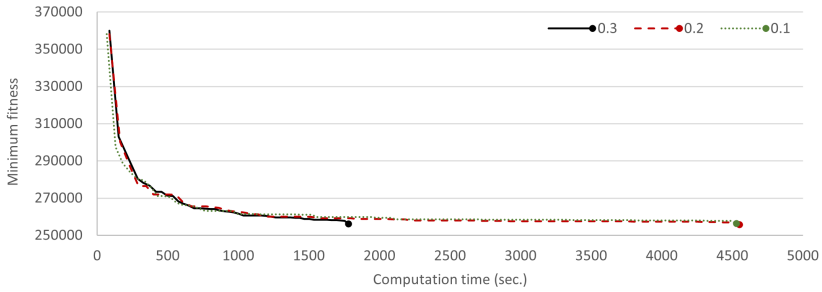
When e_p is decreased to 0.1, the average computation time per generation slightly increases due to the higher number of offspring created per generation. However, the total computation time decreases since the number of generations significantly reduces. This may indicate premature convergence when too few well-performing individuals are preserved. When e_p is set to 0.3, the computation time decreases, primarily because fewer offspring are created per generation.

Based on these results, we select $e_p = 0.2$ as it balances maintaining diversity and preserving well-performing individuals, resulting in the lowest fitness value for both networks. Again, it is necessary to emphasize that this decision is made while only observing the slightest differences in fitness.

Figure 5: Evolution of the minimum fitness value until stagnation for different values of e_p.



(a) *small*



(b) *medium*

5.2.3 Mutation Proportion and Mutation Probability

The GA incorporates two mutation parameters to preserve diversity in the active arcs and the open hub. The mutation probability (p_m) is incorporated into the crossover operator, and the mutation proportion (m_p) determines the number of mutants. These parameters are tuned simultaneously. Initially, a mild level of mutation is allowed by setting both parameters to 0.05. The results are summarized in Table 6, where the values 0, 0.05, and 0.1 are explored for these parameters.

The same hub is consistently selected for both networks across all parameter combinations. The maximum difference in fitness is 0.80% for the small network and 1.59% for the medium network. Interestingly, setting m_p to zero tends to lower the fitness compared to other m_p values. However, a notable exception exists within this pattern for the medium network, particularly when mutations are entirely excluded, i.e., $m_p = p_m = 0$. Furthermore, setting the mutation probability to zero could cause early stagnation, given that the number of generations is low for both networks for this parameter value. Setting p_m to 0.1 tends to increase the fitness.

As expected, increasing m_p results in a higher time percentage spent in the mutation operator. Additionally, it comes with an increase in the average computation time per generation. Generating mutants takes longer than producing offspring since fewer routes are immediately pruned. Furthermore, the average computation time per generation increases as p_m increases following the same reasoning. No clear pattern emerges regarding the total computation time, as the number of generations varies significantly across the different parameter settings.

Overall, the impact of the mutation parameters on fitness seems minimal, with the differences falling within the range of 1.59%. Therefore, it is difficult to recognize pronounced patterns. There is no evidence indicating that mutants lead to improved solutions. Since including mutants increases the average computation time per generation, it is suggested that $m_p = 0$ is preferable. Additionally, p_m is set to 0.05. This choice is motivated by the consideration that $p_m = 0$ might result in premature convergence, while $p_m = 0.1$ tends to result in higher fitness values.

Table 6: GA results for different values of p_m and m_p.

Network	p_m	m_p	Minimum fitness value	Best hub	Number of generations	Comp. time (sec.)	Avg. comp. time generation (sec.)	Time % crossover	Time % mutants
small	0	0	96448.83	65	61	146	2.29	96.58%	0%
		0.05	96651.16	65	65	175	2.57	86.86%	9.14%
		0.1	96608.99	65	65	188	2.79	79.26%	17.55%
	0.05	0	96219.68	65	103	252	2.38	98.02%	0%
		0.05	96668.50	65	104	299	2.81	88.29%	9.36%
		0.1	96668.50	65	104	285	2.68	81.05%	17.19%
	0.1	0	96608.99	65	58	279	2.51	97.49%	0%
		0.05	96815.65	65	69	181	2.54	88.95%	8.84%
		0.1	96985.76	65	88	296	3.28	81.08%	16.89%
medium	0	0	259769.66	133	75	2767	35.42	97.22%	0%
		0.05	256526.64	133	83	3286	38.14	84.94%	12.48%
		0.1	256169.79	133	158	6566	40.84	75.02%	23.82%
	0.05	0	255859.37	133	104	3672	34.31	98.04%	0%
		0.05	255700.54	133	131	5818	43.41	85.96%	12.50%
		0.1	256788.18	133	125	5386	42.19	76.59%	22.02%
	0.1	0	256634.06	133	152	5910	38.13	98.65%	0%
		0.05	256826.51	133	160	6848	42.11	87.78%	11.16%
		0.1	257623.40	133	136	6218	44.80	78.39%	20.28%

5.2.4 Heuristic Route Algorithm

Initially, the heuristic proportion (h_p) is set to 1, which ensures that the Exact Route Algorithm is applied to all commodities. Incorporating the Heuristic Route Algorithm aims to accelerate the crossover operator. By setting h_p below 1, the heuristic algorithm is invoked for $(1 - h_p)$ proportion of the commodities. Table 7 presents the results for $h_p \in \{0, 0.05, 0.1, 0.2, 0.3, 0.6, 1\}$, with $h_p = 1$ as the baseline.

Applying the heuristic algorithm successfully leads to a reduction in the average computation time per generation. Specifically, for the small and medium network, it is reduced by factors of 3.97 and 4.97, respectively, when h_p is lowered to zero. Generally, the lower the value of h_p, the quicker the crossover operator becomes. Consequently, the total computation time tends to decrease when h_p is set to a lower value.

Interestingly, there is no observable relation between the value of h_p and the minimum fitness value. For these instances, setting h_p below one can either cause an increase or reduction in fitness. However, the observed differences are relatively small. Moreover, setting h_p to zero for the medium network results in a different open hub than other values of h_p. The achieved fitness for hub 137 is very similar to that of hub 133, suggesting similar performance. This indicates that the algorithm may struggle to identify the best hub.

The objective of applying the heuristic is to effectively reduce the average generation time without significantly increasing the fitness value. Based on the observations, $h_p = 0.1$ is selected. This choice reduces the average computation per generation by factors of 3.22 and 3.75 for the small and medium network, respectively, while ensuring that the fitness does not increase by more than 0.10% for these instances.

Table 7: GA results for different values of h_p.

Network	h_p	Minimum fitness value	Best hub	Number of generations	Comp. time (sec.)	Avg. comp. time generation (sec.)
small	0	96096.88	65	99	64	0.60
	0.05	96631.80	65	72	58	0.70
	0.1	96054.75	65	113	89	0.74
	0.2	96054.71	65	89	99	1.00
	0.3	96608.99	65	78	95	1.14
	0.6	96030.87	65	132	252	1.86
	1	96219.68	65	103	252	2.38
medium	0	256235.15	137	111	880	6.91
	0.05	256095.08	133	108	1187	9.46
	0.1	256052.90	133	84	886	9.14
	0.2	255665.65	133	99	1290	12.09
	0.3	255490.13	133	129	1650	12.16
	0.6	255463.72	133	129	2871	20.84
	1	255859.37	133	104	3672	34.31

5.2.5 Local Search

The local search proportion (l_p) is tuned to investigate whether improving the fitness of individuals leads to better results. Among other things, this could be achieved by a better selection of individuals for crossover. The results for $l_p \in \{0, 0.05, 0.1, 0.2, 0.3, 0.6, 1\}$ are presented in Table 8.

Increasing the local search proportion tends to result in longer computation times as the average computation time per generation increases. This is a consequence of the additional time spent on this operator. When local search is applied to all offspring, the average computation time per generation increases by factors of 4.64 and 3.99 for the small and medium network, respectively. This increase is relatively high since local search relies on the Exact Route Algorithm, while the crossover operator benefits from applying the heuristic approach.

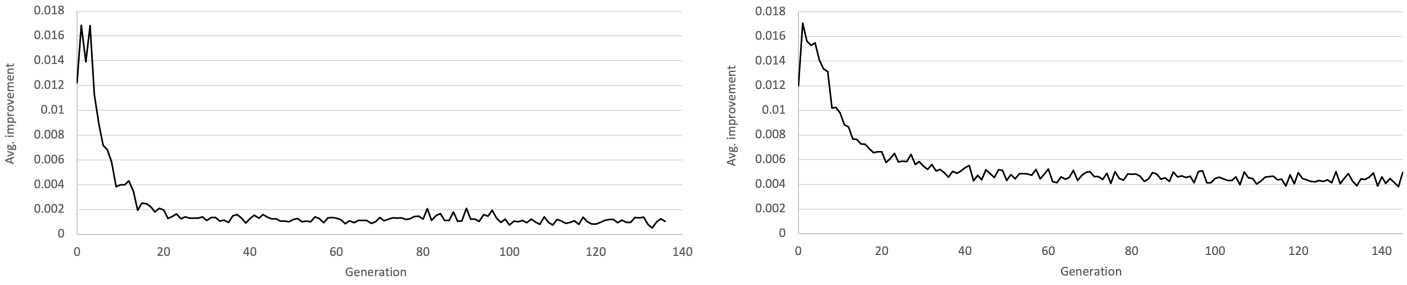
When we compare the obtained fitness values with $l_p = 0$, we conclude that including local search tends to improve the fitness. Specifically, local search on the top 5% of offspring already improves the GA outcome. However, the differences are small, and there is no guarantee that increasing l_p above 0.05 will further improve the fitness. Remarkably, the same solution is obtained for the small network in five out of seven GA runs. For the medium network, the lowest fitness is achieved for $l_p = 0.05$ and $l_p = 1$, with a difference of only 0.01%. These two solutions, however, correspond to different open hubs.

Table 8: GA results for different values of l_p.

Network	l_p	Minimum fitness value	Best hub	Number of generations	Comp. time (sec.)	Avg. comp. time generation (sec.)	Time % local search
small	0	96054.75	65	83	119	0.74	0%
	0.05	95988.70	65	135	126	0.87	12.70%
	0.1	95988.70	65	87	85	0.92	21.18%
	0.2	96608.99	65	69	95	1.29	34.74%
	0.3	95988.70	65	76	119	1.47	44.54%
	0.6	95988.70	65	89	233	2.54	63.52%
	1	95988.70	65	125	439	3.44	74.03%
medium	0	256052.90	133	84	886	9.14	0%
	0.05	255100.46	137	144	1503	9.88	14.84%
	0.1	255241.60	133	155	1631	10.00	25.51%
	0.2	255499.19	133	103	1401	12.76	39.47%
	0.3	255287.28	133	159	2785	16.88	50.02%
	0.6	255568.95	133	85	2474	27.65	63.38%
	1	255126.59	133	193	7163	36.48	77.51%

The local search proportion is set to 0.05 as it results in the lowest fitness value and increases the average computation time the least for both networks. Figure 6 provides the average improvement per generation the local search achieves. The improvement is most significant in the first generation, reaching around 1.70%. After approximately $20 - 30$ generations, it stabilizes around 0.15% for the small network and 0.45% for the medium network. This stabilization occurs as fewer arcs remain active, resulting in minimal differences that can be achieved by re-optimizing the routes of commodities. Alternatively, the local search could be applied only in the first few generations.

Figure 6: Average improvement per generation achieved by local search for $l_p = 0.05$.



(a) *small* (b) *medium*

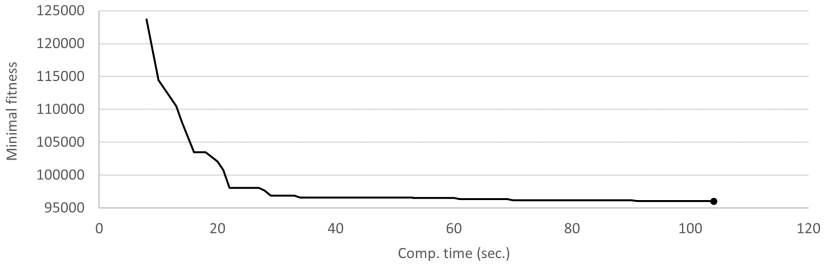
5.3 Results Real-world Instances

Table 9 presents the GA results for all real-world networks using the previously obtained parameter settings. The stopping criteria have been updated as follows: $G_{max} = \infty$, $S_{max} = 30$ and $T_{max} =$

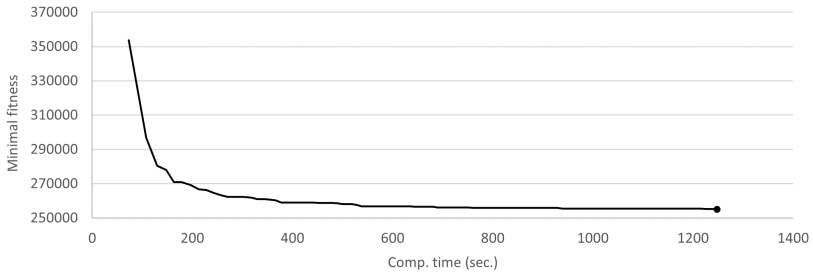
86.400 seconds. For the small network, the algorithm achieves a fitness of 95988.70 with hub 65 in just over two minutes. The same hub is consistently open in all GA runs during the parameter tuning phase. In the case of the medium network, the GA achieves a fitness of 255100.46 with hub 137 in just above 25 minutes. As observed earlier, the algorithm struggles to find the best hub, as at least two hubs have been selected while tuning the parameters. This issue will be further investigated in the subsequent subsection. Moving on to the large network, the GA requires nearly five hours to complete, resulting in an individual with a fitness of 523171.78, in which hub 393 is open.

Figure 7 illustrates the progression of the minimum fitness value for all networks until the final stagnation phase. A similar pattern is observed, with the most significant reduction occurring in the first $30 - 40\%$ of the computation time.

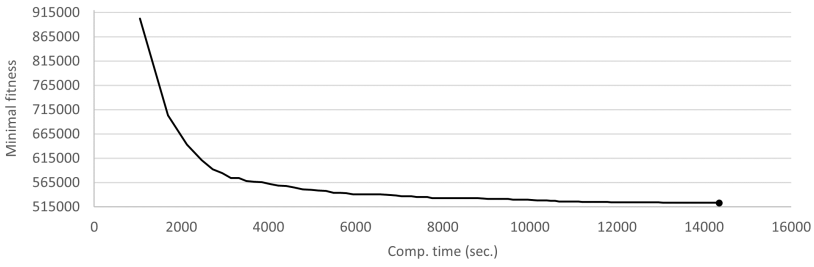
Figure 7: Evolution of the minimum fitness value until stagnation.



(a) *small*



(b) *medium*



(c) *large*

Table 9: GA results for real-world networks.

Network	Minimum fitness value	Best hub	Number of generations	Nr. active arcs	Avg. nr. hub touches	Comp. time (sec.)	Avg. comp. time generation (sec.)	Time % initialization	Time % crossover	Time % local search
small	95988.70	65	135	191	1.015	126	0.87	5.56%	80.16%	12.70%
medium	255100.46	167	114	443	1.387	1503	9.88	4.72%	80.37%	14.84%
large	523171.78	393	134	568	1.492	16930	118.01	5.93%	80.51%	13.53%

Furthermore, Figure 8 displays the average number of hub touches in the solution corresponding to the best individual across the generations. For all networks, this number initially increases significantly and then stabilizes. In the case of the small network, it stabilizes below the initial level. It is suspected that the medium and large network might benefit from allowing more hub touches, which will be further investigated in the following subsection.

The time percentage spent on initialization, crossover, and local search remains relatively stable across different networks. As mentioned earlier, the crossover operator speeds up throughout the algorithm. This can be seen in Figure 10, which depicts the computation time per generation for the large network. The most significant decrease occurs in the first 20 generations, mainly due to a significant decrease in active arcs per individual. As a result, more routes are immediately pruned in the Exact Route Algorithm. This is demonstrated by Figure 9 for the best individual in the large network.

Figure 8: Evolution of the average number of hub touches in the corresponding solution of the best individual until stagnation.

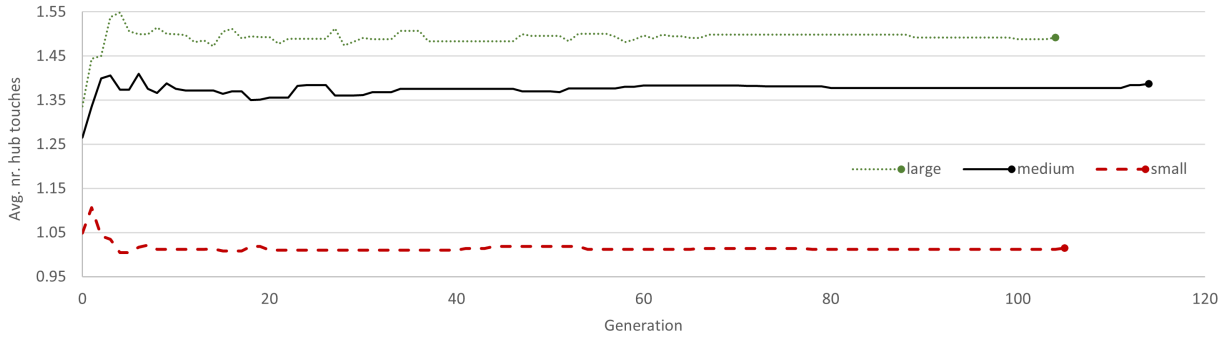


Figure 9: Evolution of the number of active arcs in the best individual for the large network until stagnation.

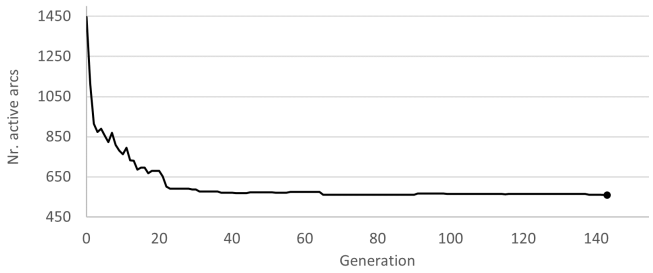
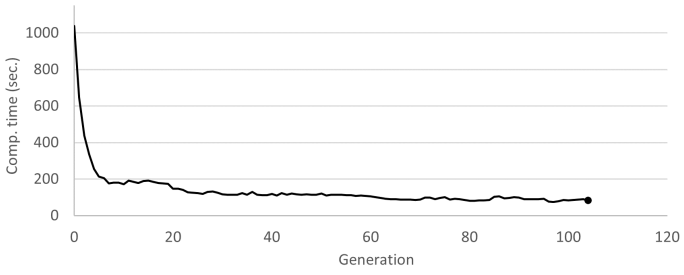


Figure 10: Computation time per generation for the large network until stagnation.



5.4 Experiments

We carry out four experiments to investigate the performance and stability of the GA. Since this problem has not been studied before, no alternative algorithms have been developed. Therefore, we cannot confirm whether the GA finds the optimal location. The two experiments in Sections 5.4.1 and 5.4.2 are designed to demonstrate the performance of the GA. These experiments focus on whether the GA consistently converges towards one location. When the GA cannot achieve this, it indicates that the algorithm cannot identify the optimal location. The last two experiments focus on two instance parameters related to the alternative modeling approach of the economies of scale. Specifically, in Section 5.4.3, we examine the impact of the maximum number of hub touches on the algorithm. In Section 5.4.4, we explore whether altering the size of the fixed cost relative to the variable cost impacts the GA solution.

5.4.1 Randomness

The initialization, crossover, selection, and mutation operators are subject to random factors. To assess the impact of randomness within the GA, we conduct 20 independent GA runs on each of the three real-world networks and the four experimental networks. The results obtained from these runs provide insights into the stability of the GA.

For practical reasons, specific stopping criteria are implemented. In the first generations, the most significant improvement is observed. Subsequently, this rate of improvement diminishes, eventually leading to algorithmic stagnation. As a result, a decision is made to restrict the number of generations. For small- and medium-sized networks, the rate of improvement becomes marginal after 50 generations. Therefore, the maximum number of generations (G_{max}) is set to 50 for these instances. For the large network, however, G_{max} is set to 65 due to the more significant number of generations before stagnation. The additional stopping parameters, S_{max} and T_{max}, are left unbounded. Notably, the deliberate choice of limiting the number of generations, rather than imposing a time limit, is made to maintain comparability across results. The aim is to prevent the total number of generations from influencing the outcome of the GA.

Table 10: Overview of 20 GA runs.

Network	Average fitness	Median fitness	Max. deviation from average (%)	Best hub across runs[1]	Avg. comp. time (sec.)[2]	Avg. comp. time generation (sec.)[2]
small	96322.02	96352.53	0.35%	65 (100%)	46	0.92
m–A	102235.49	102175.1	0.88%	58 (100%)	45	0.75
m–B	78410.49	78436.88	0.19%	81 (100%)	41	0.70
medium	256439.73	256388.95	0.43%	133 (85%)	587	10.20
l–A	222228.62	222221.20	0.13%	137 (95%)	601	10.14
l–B	225750.84	225929.88	1.55%	170 (100%)	760	14.01
large	527573.13	524980.32	1.40%	393 (95%)	10038	135.08

[1] Most frequent established hub across the runs, with the percentage of times it is chosen shown in parentheses.
[2] Network indicator computed by taking the average of the median runs based on the obtained fitness.

Table 10 presents an overview of the obtained results. The median and average fitness values for each network are shown. Furthermore, we determine the maximum deviation from the average fitness across the 20 GA runs. The maximum deviations for five out of seven networks are below 1%. However, the maximum deviation is slightly higher for experimental network l–B and the large network, at 1.55% and 1.40%, respectively.

The relatively small deviations indicate that randomness does not play a substantial role in the fitness obtained by the GA. Therefore, the algorithm demonstrates stability across the runs. However, since the differences within the parameter tuning process were often below 1%, the results and conclusions drawn from this analysis may be affected by the randomness to some degree.

Regarding hub selection, the GA consistently opens the same hub for all three small-sized networks. This consistency suggests that the algorithm is robust for these networks. However, when considering larger-sized networks, there was some variability in hub selection across the 20 runs. For instance, in the case of the medium network, hub 133 is selected 17 times, hub 137 is chosen twice, and hub 111 is established once. Given the minor maximum deviation of only 0.43% for this network, it suggests that these hubs likely exhibit similar performance. Further investigation confirms that these hubs are closely located, and the corresponding hub parameter values are comparable. Similar patterns are observed for the large network and experimental network l–A, where the selected hubs are in close proximity. We conclude that the GA is less stable when considering a large set of potential hubs, particularly when they are closely located.

5.4.2 Number of Potential Hubs

In this subsection, we investigate how the number of potential hubs ($|H_p|$) influences the performance of GA and whether it affects the hub selection. To observe this, we run the GA on subsets of H_p with proportions of 0.1, 0.25, and 0.5. These sets are designed to include the top-performing hubs.

We use the 20 independent GA runs described in Section 5.4.1 to create the subsets. For each run, we rank the potential hubs based on the lowest obtained fitness associated with the hubs. The ranks are assigned such that the best-performing hub receives rank 1, the second-best hub is given rank 2, and so forth. Subsequently, the average rank is computed for each potential hub across the runs. This average rank indicates which hubs exhibit superior performance. For the 0.1 subset, we select the 10% of hubs with the lowest average ranks. The subsets with proportions 0.25 and 0.5 are constructed similarly.

We execute ten independent GA runs for each (sub)set, applying the same stopping criteria as in the previous experiment. The results are presented in Table 11 for all real-world and experimental instances. It is important to note that the population size is kept constant across the different subsets, using the initial size of H_p.

For each instance, the changes in fitness are minimal when a smaller set of potential hubs is provided as input to the GA. Although the fitness tends to decrease for four instances, the differences are deemed insignificant. This conclusion is drawn by comparing the average fitness values with the range of observations for the baseline scenario in Section 5.4.1. Differences exceeding the corresponding maximum deviation outlined in Table 10 are considered significant since they do not appear to result from randomness. Notably, for the large network and experimental network l–B, the maximum deviation substantially decreases when considering fewer potential hubs. This observation indicates that while the size of H_p does not negatively affect the average GA outcome, it may lower its stability.

The size of H_p does not appear to affect the hub selection. For all instances, the most frequently selected hub remains consistent across the subsets with similar frequencies. We conclude that the outcome of the algorithm is not directly affected by the number of potential locations.

Table 11: Overview of ten GA runs for different subsets H_p.

Network	Proportion of H_p	Average fitness	Median fitness	Max. deviation from average (%)	Best hub across runs[1]	Avg. comp. time (sec.)[2]	Avg. comp. time generation (sec.)[2]
small	0.1	96426.86	96414.14	0.23%	65 (100%)	35	0.60
	0.25	96325.05	96323.41	0.35%	65 (100%)	30	0.54
	0.5	96348.69	96325.59	0.32%	65 (100%)	30	0.54
	1	96349.77	96417.65	0.27%	65 (100%)	43	0.78
m–A	0.1	102076.12	102039.18	0.14%	58 (100%)	23	0.40
	0.25	102192.22	102190.78	0.19%	58 (100%)	34	0.56
	0.5	102119.08	102141.13	0.16%	58 (100%)	36	0.62
	1	102271.86	102229.00	0.85%	58 (100%)	52	0.88
m–B	0.1	78393.47	78374.96	0.06%	81 (100%)	31	0.54
	0.25	78391.76	78374.96	0.06%	81 (100%)	30	0.52
	0.5	78409.11	78414.48	0.04%	81 (100%)	32	0.56
	1	78419.78	78436.88	0.06%	81 (100%)	41	0.70
medium	0.1	256132.67	256143.42	0.20%	133 (100%)	405	6.80
	0.25	255900.76	256247.77	0.99%	133 (90%)	368	6.26
	0.5	256115.11	256082.30	0.24%	133 (100%)	423	7.18
	1	256477.21	256467.99	0.17%	133 (90%)	709	12.18
l–A	0.1	222262.24	222234.64	0.10%	137 (100%)	248	4.10
	0.25	222224.14	222221.20	0.11%	137 (100%)	288	4.80
	0.5	222225.00	222215.03	0.09%	137 (100%)	356	6.00
	1	222203.01	222220.94	0.07%	137 (100%)	601	10.16
l–B	0.1	222506.65	222539.37	0.19%	170 (100%)	339	5.54
	0.25	222512.09	222515.63	0.06%	170 (100%)	401	6.62
	0.5	222681.86	222653.16	0.15%	170 (100%)	477	7.94
	1	225865.29	225929.88	1.53%	170 (100%)	872	14.74
large	0.1	526098.47	526329.04	0.27%	393 (100%)	4502	59.15
	0.25	526677.59	526683.37	0.08%	393 (100%)	5446	72.56
	0.5	527672.12	526920.90	1.22%	393 (100%)	6832	91.44
	1	526048.87	524216.50	1.65%	393 (100%)	10809	149.48

[1] Most frequent established hub across the runs, with the percentage of times it is chosen shown in parentheses.
[2] Network indicator computed by taking the average of the median runs based on the obtained fitness.

Including additional potential hubs causes certain computational implications. Specifically, it substantially increases the average computation time per generation. The most significant increase is observed for experimental network l–B, where this average increases by a factor of 2.66 when including all potential hubs instead of the top 10%. Two main factors cause this increase. First, we observe that having more potential hubs increases the number of active arcs per individual in the early generations. As a result, the crossover operator slows down. Secondly, the graph representing the network consists of more nodes and arcs when considering extra potential hubs. Even though routes including closed potential hubs are considered infeasible, pruning these routes requires computational effort. The underlying programming structure causes this computational inefficiency, and it presents an opportunity for an acceleration of the algorithm. One approach involves the creation of individual graphs for each potential hub, reducing the number of routes to consider within the Exact Route Algorithm.

5.4.3 Maximum Number of Hub Touches

One notable advantage of the algorithm's design is its flexibility in accommodating more than two hub touches. As mentioned, the maximum number of hub touches (ζ) is considered an instance parameter. In this section, we explore whether increasing ζ benefits the networks and how it influences the algorithm. Specifically, we compare scenarios where this parameter is set to two, three, and four, using two as the baseline scenario.

To conduct this experiment, ten independent GA runs are performed on the small and medium network, as well as the experimental instances. This layout limits the randomness while being practical in terms of computational resources. Similar to the procedure outlined in Section 5.4.1, identical stopping criteria are implemented.

Table 12: Overview of ten GA runs for different values of ζ.

Network	ζ	Average fitness	Median fitness	Max. deviation from average (%)	Best hub across runs[1]	Nr. active hub arcs[2]	Nr. active depot-hub arcs[2]	Avg. nr. hub touches[2]	Comp. time (sec.)[2]	Avg. comp. time generation (sec.)[2]
small	2	96349.77	96417.65	0.27%	65 (100%)	3	188	1.010	43	0.78
	3	96373.51	96356.74	0.40%	65 (100%)	3	188	1.013	51	0.92
	4	96273.43	96298.05	0.35%	65 (100%)	3	188	1.015	61	1.08
m–A	2	102271.86	102229.00	0.85%	58 (100%)	2	99	1.446	52	0.88
	3	102156.01	102140.39	0.50%	58 (100%)	2	100	1.432	59	0.98
	4	102073.56	102148.96	0.34%	58 (100%)	3	99	1.497	59	0.98
m–B	2	78419.78	78436.88	0.06%	81 (100%)	2	94	1.426	41	0.70
	3	78407.40	78409.62	0.04%	81 (100%)	2	100	1.432	50	0.84
	4	78542.17	78557.89	0.13%	81 (100%)	3	99	1.497	64	1.06
medium	2	256477.21	256467.99	0.17%	133 (90%)	9	415.5	1.383	709	12.18
	3	255478.48	255600.42	0.82%	133 (60%)	10	428.5	1.411	1109	19.01
	4	255487.17	255580.54	0.36%	137 (60%)	11.5	431.5	1.422	1545	26.33
l–A	2	222203.01	222220.94	0.07%	137 (100%)	6	192	1.434	601	10.16
	3	218266.31	218312.72	0.08%	137 (100%)	8	191.5	1.709	922	15.60
	4	217407.07	217458.33	0.27%	137 (100%)	9	193	1.859	1241	20.87
l–B	2	225865.29	225929.88	1.53%	170 (100%)	9	221	1.473	872	14.74
	3	222358.17	222425.71	0.18%	170 (100%)	7	220.5	1.622	1246	20.09
	4	222358.07	222405.97	0.12%	170 (100%)	7	221.5	1.621	1854	30.41

[1] Most frequent established hub across the runs, with the percentage of times it is chosen shown in parentheses.
[2] Network indicator computed by taking the average of the median runs based on the obtained fitness.

We observe no significant improvement or deterioration in fitness value for small-sized networks as ζ increases. Again, differences in average fitness exceeding the maximum deviations in Section 5.4.1 are considered significant. This result also holds for the medium network. However, the most frequently established hub changes for this instance. Specifically, we observed hub 137 being selected more often than hub 133 when $\zeta = 4$. For test instance l–A, the average fitness improves by 1.80% and 2.21% for ζ set to three and four, respectively. The improvement for network l–B is 1.58% for both values. This suggests that these networks benefit significantly from allowing more than two hub touches. Interestingly, in both cases, the same hub is selected most often for all values of ζ.

The average number of hub touches tends to rise as ζ increases. The maximum number of hub touches does not exhibit a correlation with the number of active arcs. The effect of increasing ζ on the number of active arcs between hubs and between hubs and depots relies on the specific network characteristic.

Table 13: GA results for the large network for different values of ζ.

Network	ζ	Minimum fitness value	Best hub	Nr. active hub arcs	Nr. active depot-hub arcs	Avg. nr. hub touches	Comp. time (sec.)	Avg. comp. time generation (sec.)
large	2	526054.60	393	17	558	1.491	10820	149.48
	3	509990.84	393	22	539	1.784	21921	287.43
	4	507588.06	393	31	553	1.941	36357	486.62

Finally, we analyze the large network to evaluate the impact of varying the maximum number of hub touches. A single GA run is executed for this network using $G_{max} = 65$, and the results are presented in Table 15. We observe a significant improvement when allowing more than two hub touches. This improvement is 3.15% and 3.64% for ζ equal to three and four, respectively. The same hub is consistently open regardless of the value of ζ. For this network, a greater number of arcs between hubs become activated, suggesting consolidation of commodities at these hubs. Again, the number of active arcs between depots and hubs does not exhibit a clear pattern.

To conclude, the GA can achieve lower fitness values by allowing more than two hub touches per commodity when it benefits the network. Additionally, there is no evidence that increasing this instance parameter may otherwise deteriorate the fitness value significantly. While the improvement is desirable, it results in a notable increase in computation time. This occurs due to fewer routes being pruned by the feasibility pruning process in the Exact Route Algorithm. Specifically, for these instances, the average computation time per generation increases by a factor of at most 3.26 when ζ is set to four.

5.4.4 Fixed Arc Costs

This section investigates the influence of varying the fixed arc costs relative to the variable arc costs. This is achieved by adjusting the parameter m, which determines the fraction of the truck cost designated as fixed cost. Initially, m is set to 0.75, and we extend our analysis by including two additional values: 0.5 and 1. By doing so, we aim to observe how changes in m affect the outcomes of the GA. As in the previous experiments, we conduct ten independent runs of the GA for each value of m, maintaining consistency in the stopping criteria to the previous experiments.

Table 14 summarizes the results for the small and medium-sized instances. The fixed arc costs are doubled when m is set to 1 compared to m being 0.5. As a result, the average fitness increases by $14.6-25.9\%$ since these costs significantly impact the fitness function. It is necessary to highlight that for experimental network l–B, when m is set to 1, the maximum deviation from the average is notably high at 2.44%.

Table 14: Overview of ten GA runs for different values of m.

Network	m	Average fitness	Median fitness	Max. deviation from average (%)	Best hub across runs[1]	Nr. active hub arcs[2]	Nr. active depot-hub arcs[2]	Avg. nr. hub touches[2]	Comp. time (sec.)[2]	Avg. comp. time generation (sec.)[2]
small	0.5	82105.35	82117.00	0.29%	65 (100%)	2.5	188	1.008	48	0.88
	0.75	96349.77	96417.65	0.27%	65 (100%)	3	188	1.010	43	0.78
	1	110651.59	110560.38	0.48%	65 (100%)	3	188	1.013	45	0.82
m–A	0.5	95274.93	95283.73	0.08%	58 (100%)	2	100	1.422	47	0.80
	0.75	102271.86	102229.00	0.85%	58 (100%)	2	99	1.446	52	0.88
	1	109218.77	109149.51	0.77%	58 (100%)	2	99	1.453	73	1.26
m–B	0.5	72157.30	72166.06	0.12%	81 (100%)	2	95	1.401	49	0.84
	0.75	78419.78	78436.88	0.06%	81 (100%)	2	94	1.426	41	0.70
	1	84547.35	84505.96	0.17%	81 (100%)	2	94	1.426	50	0.86
medium	0.5	226655.94	226236.27	0.33%	137 (70%)	11	438	1.370	836	14.58
	0.75	256518.85	256536.96	0.16%	133 (90%)	9	415.5	1.383	709	12.18
	1	285298.84	285370.20	0.44%	133 (90%)	10.5	416	1.393	821	14.21
l–A	0.5	202923.06	202938.46	0.10%	137 (90%)	6	195	1.430	690	11.72
	0.75	222203.01	222220.94	0.07%	137 (100%)	6	192	1.434	601	10.16
	1	241178.21	241205.90	0.09%	137 (100%)	6	190	1.450	684	11.61
l–B	0.5	204665.47	204124.42	1.20%	170 (100%)	6	224	1.444	890	14.90
	0.75	225865.29	225929.88	1.53%	170 (100%)	9	221	1.473	872	14.74
	1	246337.39	248401.72	2.44%	170 (100%)	9	216.5	1.486	749	13.00

[1] Most frequent established hub across the runs, with the percentage of times it is chosen shown in parentheses.
[2] Network indicator computed by taking the average of the median runs based on the obtained fitness.

Across the various instances, the hub selection remains relatively consistent, except for the medium-sized network. In the latter, hub 137 becomes the most frequently selected hub when m is set to 0.5. Interestingly, while the number of active hub arcs does not exhibit a clear correlation with m, we observe a slight decrease in the number of active arcs between depots and hubs as m increases. This relation is more pronounced in medium-sized instances. The reduction in active arcs can be attributed to the fact that, with higher m values, the threshold for activating an arc increases during initialization, crossover, and mutation. Specifically, the temporary cost of an arc becomes higher, leading to fewer activations, especially between depots and hubs where there is less consolidation of commodities.

Moreover, the average number of hub touches increases slightly as the fixed costs become relatively higher compared to the variable costs. This indicates increased consolidation at hubs. Regarding computational efficiency, the average computation time per generation remains relatively stable across different m values.

Table 15: GA results for the large network for different values of m.

Network	m	Minimum fitness value	Best hub	Nr. active hub arcs	Nr. active depot-hub arcs	Avg. nr. hub touches	Comp. time (sec.)	Avg. comp. time generation (sec.)
large	0.5	477714.20	393	16	567	1.475	11151	150.97
	0.75	526054.60	393	17	558	1.491	10820	149.48
	1	570501.26	393	19	544	1.513	9464	126.26

We extend our experiment to the large network by executing a single GA run for each m value with G_{max} set to 65. The results, showcased in Table 15, are in agreement with the observed patterns in smaller instances. Increasing the value of m reduces the number of active arcs between depots and hubs, accompanied by an increase in the average number of hub touches. Notably, the selected hub remains consistent across the different m values.

Making an informed decision regarding the relation between fixed and variable arc costs is important. As demonstrated by this experiment, it impacts which arcs are activated. While hub selection remains stable across varying m values, this cannot be guaranteed for all linehaul networks as the robustness relies on network characteristics.

6 Conclusion

In this research, we evaluate a limited number of possible hub locations by minimizing the linehaul costs for FedEx Express. To capture the complexity of the network, it is necessary to use an alternative approach to model the economies of scale related to the benefits of routing packages through hubs. We include fixed arc costs to determine whether transportation between two locations is allowed. We introduce our problem as the Multiple-Product Uncapacitated p-Hub Location and Arc Problem.

We propose a genetic algorithm to solve the MP-UpHLAP. The genetic information determines the location of the new hub and between which locations transportation is allowed. Generating an individual requires solving a routing problem for each commodity. We incorporate both an exact method and a heuristic approach to speed up the route construction. We fine-tune the GA parameters on two real-world instances and observe that including this heuristic improves the average computation time per generation by a factor of $3.22 - 3.75$.

Given the absence of prior research on this problem, alternative algorithms have yet to be established. Consequently, we cannot determine whether the GA can identify the optimal hub location. To demonstrate the performance of the GA, we conduct two experiments on three real-world instances and four experimental networks. For the largest instance, the GA takes just over three hours to terminate. We design these experiments to observe whether the algorithm consistently converges towards one hub location. In cases where the GA cannot accomplish this, it is an indication that the algorithm fails in finding the optimal location.

The first experiment focuses on the influence of randomness on the outcome of the GA. Across 20 GA runs, the fitness remains relatively constant, displaying a maximum deviation from the average of 1.59%. Furthermore, the GA consistently selects one location for smaller instances. The algorithm encounters challenges in converging towards a single location when dealing with a large network, including closely located hubs. The GA selects neighboring locations, resulting in similar fitness values. For these instances, the GA fails to find the exact optimal hub location, even though it identifies an area with high potential.

In the second experiment, we eliminate the worst-performing locations. Considering more locations can lower the stability of the GA, as randomness tends to have a larger effect on the obtained fitness. However, it does not affect the fitness on average or the hub selection. There is no evidence that the GA fails to find the optimal hub as a direct result of including too many locations. Instead, the challenge arises from the proximity of promising locations, as observed in the first experiment.

Two additional experiments investigate the instance parameters related to the alternative modeling approach of the economies of scale. The third experiment shows that allowing more than two hub touches can reduce linehaul costs, especially for larger instances. This is achieved by cost-effective routing to consolidate commodities further. Notably, there is a trade-off since increasing this maximum comes with a substantial increase in the computation time of the algorithm.

Finally, the last experiment shows that the relative magnitude of the fixed arc costs compared to the variable arc costs impacts the GA outcomes. Higher fixed costs lead to fewer arcs where transportation is permitted, especially between depots and hubs. There is an increase in the consolidation of commodities, and the increase in the number of hub touches aligns with this trend. We conclude that the fixed arc costs shape the design of the network. Therefore, it is crucial to establish a representative relation between these two cost components, which requires an analysis of the existing network.

In the context of these two experiments, it is notable that neither the maximum number of hub touches nor the magnitude of the fixed arc costs influence the established hub for most test instances. However, since this result relies on network characteristics, it cannot be generalized to all possible instances. These parameters may affect the chosen location by the algorithm, and FedEx Express should make an informed decision regarding their values.

6.1 Limitations and Future Research

There are several limitations to consider and recommendations to make for future research. To begin, one aspect that is not considered is the processing capacities of hubs. Each location may be subject to a limit on the volume of packages stored and handled per hour. Since we focus on strategic hub placement, the precise consolidation of packages is undetermined. As a result, the exact volume at a hub at any moment in time is unknown. However, these capacities likely influence the optimal location of a new hub. In future research, limiting the total volume processed through each hub would be possible. To include this constraint, one could focus on incorporating a penalty term into the fitness function or adjusting the routing algorithms to prevent violations from occurring.

Furthermore, the test instances contain commodities that must be routed through the network. FedEx Express estimates the volumes of these commodities, which are subject to stochastic factors. Future research should focus on incorporating this stochastic component into the GA to obtain a more robust solution. For instance, we could minimize the expected fitness value by considering various scenarios.

Shifting our focus to the solution approach, the GA contains two mutation components to preserve genetic diversity. The first component prevents situations where the ability to transport between two specific locations is lost entirely and cannot be reinstated in future generations. The second component goes further as it also avoids permanently eliminating potential hub locations. We observe that the two mutation components have a minimal effect on the outcome of the GA, and the second component is even deactivated after tuning the parameters. Future research should focus on alternative mutation methods to enlarge the search space. This could increase the stability of the GA regarding the hub selection as it may prevent premature convergence towards one location.

One approach, in particular, may increase the searched solution space. One or multiple restarts can be incorporated into the algorithm, where the worst-performing locations are deliberately removed from the network, and a new initial population is generated. By eliminating these locations, the GA can focus on the best-performing hub locations while reconsidering all remaining routes for each commodity.

Furthermore, we have three remarks regarding the performed analysis. First, we only consider five test instances, of which two are used in the parameter tuning process. To gain more conclusive insights, it is necessary to analyze the performance of the GA on more instances. Also, it would be interesting to compare the GA outcome on networks with strict and relaxed service time conditions.

The following remark is on the design of the parameter selection. When comparing the impact of randomness on the outcome of the GA with the observed differences in fitness in the parameter tuning process, we recognize that the selection of parameters is influenced by randomness. This is a limitation, and we recommend performing multiple GA runs for all parameter settings in future research for more reliable results. These runs allow us to compare the average fitness and the maximum deviation from the average. Consequently, the best and most stable parameter values can be selected. Additionally, we suggest adjusting the stopping criteria for this process by lowering the time limit or fixing the number of generations for a more objective comparison.

Lastly, as mentioned, the absence of alternative algorithms complicates the evaluation of the GA. To better understand the algorithm's performance, it is beneficial to develop an alternative method. Comparing the GA with an exact method on minor instances would provide information on whether the GA succeeds in selecting the optimal hub location. For larger instances, an exact method would allow us to compare the computation times and observe the expected time advantage of the GA.

Bibliography

Abdinnour-Helm, S. (1998). A hybrid heuristic for the uncapacitated hub location problem. *European journal of operational research*, 106(2-3):489–499.

Abyazi-Sani, R. and Ghanbari, R. (2016). An efficient tabu search for solving the uncapacitated single allocation hub location problem. *Computers & Industrial Engineering*, 93:99–109.

Alumur, S. A., Campbell, J. F., Contreras, I., Kara, B. Y., Marianov, V., and O'Kelly, M. E. (2021). Perspectives on modeling hub location problems. *European Journal of Operational Research*, 291(1):1–17.

Alumur, S. A., Kara, B. Y., and Karasan, O. E. (2009). The design of single allocation incomplete hub networks. *Transportation Research Part B: Methodological*, 43(10):936–951.

Brimberg, J., Mladenović, N., Todosijević, R., and Urošević, D. (2020). A non-triangular hub location problem. *Optimization Letters*, 14(5):1107–1126.

Campbell, J. F. (1994). Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, 72(2):387–405.

Contreras, I., Cordeau, J.-F., and Laporte, G. (2011a). Benders decomposition for large-scale uncapacitated hub location. *Operations research*, 59(6):1477–1490.

Contreras, I., Díaz, J. A., and Fernández, E. (2011b). Branch and price for large-scale capacitated hub location problems with single assignment. *INFORMS Journal on Computing*, 23(1):41–55.

Contreras, I. and O'Kelly, M. (2019). Hub location problems. *Location science*, pages 327–363.

Correia, I., Nickel, S., and Saldanha-da Gama, F. (2018). A stochastic multi-period capacitated multiple allocation hub location problem: Formulation and inequalities. *Omega*, 74:122–134.

Cunha, C. B. and Silva, M. R. (2007). A genetic algorithm for the problem of configuring a hub-and-spoke network for a ltl trucking company in brazil. *European Journal of Operational Research*, 179(3):747–758.

de Sá, E. M., Morabito, R., and de Camargo, R. S. (2018). Benders decomposition applied to a robust multiple allocation incomplete hub location problem. *Computers & Operations Research*, 89:31–50.

Ernst, A. T. and Krishnamoorthy, M. (1996). Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location science*, 4(3):139–154.

Farahani, R. Z., Hekmatfar, M., Arabani, A. B., and Nikbakhsh, E. (2013). Hub location problems: A review of models, classification, solution techniques, and applications. *Computers & industrial engineering*, 64(4):1096–1109.

Flores, S. D., Cegla, B. B., and Cáceres, D. B. (2003). Telecommunication network design with parallel multi-objective evolutionary algorithms. In *Proceedings of the 2003 IFIP/ACM Latin America conference on Towards a Latin American agenda for network research*, pages 1–11.

Ghaffarinasab, N. (2022). Stochastic hub location problems with bernoulli demands. *Computers & Operations Research*, 145:105851.

Ghaffarinasab, N., Jabarzadeh, Y., and Motallebzadeh, A. (2017). A tabu search based solution approach to the competitive multiple allocation hub location problem. *Iranian Journal of Operations Research*, 8(1):61–77.

Guan, J., Lin, G., and Feng, H.-B. (2018). A learning-based probabilistic tabu search for the uncapacitated single allocation hub location problem. *Computers & operations research*, 98:1–12.

Hartmanis, J. (1982). Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1):90.

Horner, M. W. and O'Kelly, M. E. (2001). Embedding economies of scale concepts for hub network design. *Journal of Transport Geography*, 9(4):255–265.

Katoch, S., Chauhan, S. S., and Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80:8091–8126.

Kimms, A. (2006). Economies of scale in hub & spoke network design models: We have it all wrong. *Perspectives on operations research: essays in honor of Klaus Neumann*, pages 293–317.

Klincewicz, J. G. (1992). Avoiding local optima in the p-hub location problem using tabu search and grasp. *Annals of Operations research*, 40(1):283–302.

Kratica, J., Stanimirović, Z., Tošić, D., and Filipović, V. (2007). Two genetic algorithms for solving the uncapacitated single allocation p-hub median problem. *European journal of operational research*, 182(1):15–28.

Kulturel-Konak, S. and Konak, A. (2008). A local search hybrid genetic algorithm approach to the network design problem with relay stations. *Telecommunications modeling, policy, and technology*, pages 311–324.

Labbé, M., Yaman, H., and Gourdin, E. (2005). A branch and cut algorithm for hub location problems with single assignment. *Mathematical programming*, 102(2):371–405.

Lin, C.-C., Lin, J.-Y., and Chen, Y.-C. (2012). The capacitated p-hub median problem with integral constraints: An application to a chinese air cargo network. *Applied Mathematical Modelling*, 36(6):2777–2787.

Lozano, L. and Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384.

Marić, M., Stanimirović, Z., and Stanojević, P. (2013). An efficient memetic algorithm for the uncapacitated single allocation hub location problem. *Soft Computing*, 17:445–466.

Meyer, T., Ernst, A. T., and Krishnamoorthy, M. (2009). A 2-phase algorithm for solving the single allocation p-hub center problem. *Computers & Operations Research*, 36(12):3143–3151.

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

Mokhtar, H., Krishnamoorthy, M., and Ernst, A. T. (2019). The 2-allocation p-hub median problem and a modified benders decomposition method for solving hub location problems. *Computers & Operations Research*, 104:375–393.

Nickel, S., Schöbel, A., and Sonneborn, T. (2001). Hub location problems in urban traffic networks. *Mathematical methods on optimization in transportation systems*, pages 95–107.

O'kelly, M. E. (1987). A quadratic integer program for the location of interacting hub facilities. *European journal of operational research*, 32(3):393–404.

O'Kelly, M. E. (1992). Hub facility location with fixed costs. *Papers in Regional Science*, 71(3):293–306.

O'Kelly, M. E., Campbell, J. F., de Camargo, R. S., and de Miranda Jr, G. (2015). Multiple allocation hub location model with fixed arc costs. *Geographical Analysis*, 47(1):73–96.

O'Kelly, M. E. and Bryan, D. (1998). Hub location with flow economies of scale. *Transportation research part B: Methodological*, 32(8):605–616.

Podnar, H., Skorin-Kapov, J., and Skorin-Kapov, D. (2002). Network cost minimization using threshold-based discounting. *European Journal of Operational Research*, 137(2):371–386.

Serper, E. Z. and Alumur, S. A. (2016). The design of capacitated intermodal hub networks with different vehicle types. *Transportation Research Part B: Methodological*, 86:51–65.

Tanash, M., Contreras, I., and Vidyarthi, N. (2017). An exact algorithm for the modular hub location problem with single assignments. *Computers & Operations Research*, 85:32–44.

Topcuoglu, H., Corut, F., Ermis, M., and Yilmaz, G. (2005). Solving the uncapacitated hub location problem using genetic algorithms. *Computers & operations research*, 32(4):967–984.

Van Essen, J. (2009). *Heuristics for the hub location and network designproblem with a mixed vehicle fleet*. Master thesis.

Yoon, M.-G. and Current, J. (2008). The hub location and network design problem with fixed and variable arc costs: formulation and dual-based solution heuristic. *Journal of the Operational Research Society*, 59(1):80–89.

Appendix A Complexity MP-UpHLAP

In this appendix, we present formal proof of the complexity of the MP-UpHLAP. To accomplish this, we start by outlining the problem description of the Uncapaciated r-Allocation Hub Location Problem (UrAHLP) with fixed hub locations. The formulation we employ is adapted from Campbell (1994) and Mokhtar et al. (2019), though we have made alterations to ensure consistency in notation.

Appendix A.1 UrAHLP with Fixed Hubs

The problem involves two disjoint sets, $\bar{D}$ and $\bar{H}$, representing the depots and hubs, respectively. Let $\bar{V} = \bar{D} \cup \bar{H}$ denote the complete set of nodes. The nodes $i \in \bar{V}$ and $j \in \bar{H}$ are connected bidirectionally such that $(i,j),(j,i) \in \bar{A}$ when $i \neq j$. Furthermore, we define $\bar{K}$ as the set of commodities, with $\bar{v}_k$ denoting the volume of commodity $k \in \bar{K}$. Each commodity $k \in \bar{K}$ has an origin depot $\bar{o}(k)$ and a destination depot $\bar{d}(k)$. It is important to note that there are no two commodities $k_1, k_2 \in \bar{K}$ such that $\bar{o}(k_1) = \bar{o}(k_2)$ and $\bar{d}(k_1) = \bar{d}(k_2)$.

The transportation of each commodity involves moving it from its origin to its destination via one or two hubs. Furthermore, each depot must be allocated to $\bar{r}$ hubs, and a total of $|\bar{H}|$ hubs are established. Consequently, all hubs $h \in \bar{H}$ are open and subject to a fixed cost $\bar{f}_h$. Additionally, a variable cost of $\bar{\tau}_{i,j}$ is imposed for each unit volume transported via arc $(i,j) \in \bar{A}$. For every $h \in \bar{H}$, the variable cost $\bar{\tau}_{h,h}$ is set to zero. To describe the problem formally, we introduce the following variables and parameters:

X_{k,h_1,h_2} = Fraction of commodity k transported via hubs h_1 and h_2, successively.

$Z_{i,h}$ = 1 if depot i is allocated to hub h, and 0 otherwise.

$C_{k,h_1,h_2} = \bar{\tau}_{\bar{o}(k),h_1} + \bar{\tau}_{h_1,h_2} + \bar{\tau}_{h_2,\bar{d}(k)}$

The problem is formulated as follows:

$$\text{Min:} \quad \sum_{k\in\bar{K}} \sum_{h_1\in\bar{H}} \sum_{h_2\in\bar{H}} \bar{v}_k \, X_{k,h_1,h_2} \, C_{k,h_1,h_2} + \sum_{h\in\bar{H}} \bar{f}_h \tag{1}$$

$$\text{St:} \quad \sum_{h_1\in\bar{H}} \sum_{h_2\in\bar{H}} X_{k,h_1,h_2} = 1 \qquad \forall\, k \in \bar{K} \tag{2}$$

$$\sum_{h\in\bar{H}} Z_{i,h} = \bar{r} \qquad \forall\, i \in \bar{D} \tag{3}$$

$$X_{k,h_1,h_2} \leq Z_{\bar{d}(k),h_2} \qquad \forall\, k \in \bar{K}, \forall\, h_1, h_2 \in \bar{H} \tag{4}$$

$$X_{k,h_1,h_2} \leq Z_{\bar{o}(k),h_1} \qquad \forall\, k \in \bar{K}, \forall\, h_1, h_2 \in \bar{H} \tag{5}$$

$$Z_{i,h} \in \{0,1\} \qquad \forall i \in \bar{D}, \forall h \in \bar{H} \quad (6)$$

$$0 \leq X_{k,h_1,h_2} \leq 1 \qquad \forall k \in \bar{K}, \forall h_1, h_2 \in \bar{H} \quad (7)$$

The objective function 1 aims to minimize the total transportation cost, considering both the variable costs associated with arcs and the fixed costs of hubs. Constraints 2 assure that every commodity is transported via one or two hubs. To guarantee that each depot is allocated to exactly $\bar{r}$ hubs, Constraints 3, 4, and 5 are imposed. Constraints 6 restrict the variables $Z_{i,h}$ to either zero or one. While the variables X_{k,h_1,h_2} should be binaries, we allow them to take any value between zero and one through Constraints 7, as there always exists an optimal solution where the variables become binaries (Campbell, 1994).

Appendix A.2 Prove $\mathcal{NP}$-hardness

The UrApHLP with fixed hubs has been proven to be $\mathcal{NP}$-hard by Mokhtar et al. (2019) for $2 \leq \bar{r} \leq |\bar{H}|$ and $|\bar{H}| \geq 3$. We aim to demonstrate that when $\bar{r} = |\bar{H}|$, the UrApHLP becomes a specific case of the MP-UpHLAP. We can construct an instance of the MP-UpHLAP from an instance of the original problem in polynomial time using the following steps:

1. Create a new graph $G = (V, A)$ by setting $V = \bar{V}$, $D = \bar{D}$, $H = \bar{H}$, $A = \bar{A}$, and $K = \bar{K}$.
2. Fix all hub locations by selecting a hub $h \in H$ such that $H_p = \{h\}$ and $H_e = H \setminus H_p$.
3. Each commodity $k \in K$ has the same origin and destination as the corresponding commodity $k \in \bar{K}$, i.e., $o(k) = \bar{o}(k)$ and $d(k) = \bar{d}(k)$.
4. Consider a single product type, i.e., $|P| = 1$ and $p(k) \in P$ for all $k \in K$.
5. Set the volume of each commodity $k \in K$ equal to the volume of the corresponding commodity $k \in \bar{K}$, i.e., $v_k = \bar{v}_k$.
6. Eliminate the time restrictions by setting $t_{i,j} = 0$ for all $(i,j) \in A$, $p_v = 0$ for all $v \in V$, and $s_k = \infty$ for all $k \in K$.
7. Allow up to two hub touches by setting $\zeta = 2$.
8. Enable all commodities to be routed through all hubs, i.e., $b_h^p = 1$ for all $h \in H$ and $p \in P$.
9. Set the fixed hub cost of each $h \in H$ equal to the fixed hub cost of the corresponding hub $h \in \bar{H}$, i.e., $f_h = \bar{f}_h$.
10. Eliminate the variable node costs by setting $\kappa_v^p = 0$ for all $v \in V$ and $p \in P$.
11. Consider no fixed arc costs by setting $\delta_{i,j} = 0$ for all $(i,j) \in A$.

12. Set the variable arc cost of each arc $(i, j) \in A$ equal to the variable cost of the corresponding arc $(i, j) \in \bar{A}$, i.e., $\tau_{i,j} = \bar{\tau}_{i,j}$.

By transforming the original problem into the MP-UpHLAP using these steps, we can establish an equivalence between the problems. Every solution to the original problem with $\bar{r} = |\bar{H}|$ can be used to construct a solution to the MP-UpHLAP, and vice versa. Specifically, due to the ability to connect each depot to every hub, the commodities can be routed identically in both problems, resulting in the same objective. Therefore, since the UrAHLP with $\bar{r} = |\bar{H}|$ and $|\bar{H}| \geq 3$ is known to be $\mathcal{NP}$-hard, it follows that the MP-UpHLAP with $|H| \geq 3$ is also $\mathcal{NP}$-hard.