Wageningen University - Department of Social Sciences

Operations Research & Logistics Group

The Multi-commodity Capacitated Discrete Location-allocation Model

Abstract

Location-allocation problems are crucial for optimizing facility placement and client allocation for diverse businesses and governments organizations. This thesis focuses on the multi-commodity capacitated discrete location-allocation model, modelling it, and creating a custom algorithm to solve the model. The algorithm's quality is assessed based on 100 runs on medium and large data sets, with a comparison against the commercial available solver, Gurobi. The precise formulation of the multi-commodity capacitated discrete location allocation model is presented, including unique elements. Notable additions include fixed costs for production lines, facility size constraints, and internal transport costs, including truck delivery constraints. The custom algorithm created in this study to solve the model comprises four steps: creating feasible solutions, a genetic algorithm, allocation of facilities and clients, product flows. The custom algorithm consistently outperformed Gurobi when applied to medium sized data sets, outperforming in terms of objective values and computational time. However, its performance on the large data sets showed mixed results in terms of speed and computing time.

Wordcount: 9420
September, 2023

Student name: Rick van Willigen
Student number: 1033338
Commissioner: Districon
Supervisor WUR: dr. Dmytro Krushynskyi
Supervisor Company: Jesse Bleij
2nd supervisor: dr. ir. Frits Claassen
Course code: ORL-80436

WAGENINGEN
UNIVERSITY & RESEARCH

Contents

1 Introduction

This thesis seeks to explore the modelling of the multi-commodity capacitated discrete location-allocation model. Furthermore, an innovative algorithm is developed to solve this model, and its performance is subsequently benchmarked against that of a commercial solver. This chapter begins by providing a background on the historical development, practical implications, and potential challenges regarding such a model. Subsequently, the problem statement, research objective, and research questions will be outlined.

1.1 Background

Most businesses encounter some form of the location allocation problem, since the problem of locating facilities and allocating clients covers the core components of distribution system design. Industrial companies must strategically choose the locations for their fabrication and assembly plants, as well as warehouses. Retail outlets need to carefully select the sites for their stores. Also, governments have to decide where to place offices, schools, fire stations, hospitals, etc. These are all cases where location-allocation models are applicable. The location of facilities influences both the quality and the cost of services offered to clients (Klose & Drexl, 2005). Additionally, the construction of new facilities requires considerable long-term investments, making appropriate location decisions critical for businesses (Melo et al., 2009). These various factors have led to the extensive study of the location-allocation models(Melo et al., 2009).

The first location theory was developed in 1909 by Alfred Weber and aimed at maximizing the position of a single warehouse in relation to its clients (Weber, 1909). Since then, a variety of models and methodologies have been developed, as evidenced by Drezner and Hamacher (2004), Klose and Drexl (2005), Melo et al. (2009), and ReVelle and Eiselt (2005). The models can be classified into four major groups: analytic models, continuous models, network models, and discrete location models (Sadjady & Davoudpour, 2012). All these models share common characteristics, such as a metric space for facilities, known client locations, and the need to determine the facility locations based on some objective function. Distance, time, and cost between clients and facilities are quantified by a given metric, which can be used to choose which facility should serve which client with the goal of minimizing total costs (Drezner & Hamacher, 2004). However, some location problems may emphasize other objectives, such as lowering the total length or mini-

mizing the maximum travel distance (Brandeau & Chiu, 1989).

In this research the focus lies on the group of discrete models. The group of discrete models can be further categorized into four types of models: median problems, which include uncapacitated and capacitated facility location problems; covering problems; center problems; and a group of problems that could be called supply chain network design (Melo et al., 2009). The first three groups are studied most in literature (Sadjady & Davoudpour, 2012). While the covering, center, and uncapacitated facility problems look at minimizing distance or cost from the facility to demand points, the capacitated facility location problem introduces facility capacities and limits the number of demand points each facility can serve (Owen & Daskin, 1998). The allocation-location problem builds on previously mentioned location theories and models. The problem is to simultaneously locate the facilities and determine an assignment of flows to clients (Scott, 1971).

The model in this thesis includes some other characteristics. Multi-commodity make it possible for clients to have demand for different products. The single source constraint enforces non-divisible flows to clients. The complete model can be explained as follows: given a set of clients with known locations and a known set of demand for commodities, the demand of the client can exist for multiple commodities, and each client can exclusively be served from a single facility adhering to the non-divisible flow constraint. Out of a set of known potential locations, facilities with known capacities for production lines can be opened. Production lines can produce different commodities with different capacities. To fulfil the demand of the client, transport of products between facilities is possible. The objective function is minimizing total costs. A simple schematic representation of the model can be seen in Figure 1.

The uncapacitated facility location problems (UFLP) are NP-hard, and thus the model used in this thesis is also NP-hard since it includes the UFLP (Yu & Lin, 2016). NP-hard meaning that the computational times required to solve reasonably sized instances of the model can grow prohibitively long since there is no polynomial-time algorithm for solving NP-hard problems. Making finding practical and realistic solutions difficult (Megiddo & Supowit, 1984; Sadjady & Davoudpour, 2012). Problems with this complexity usually use heuristic and meta-heuristic solution approaches to find an optimal solution as close as possible in a reasonable amount of time (Vincent et al., 2016).

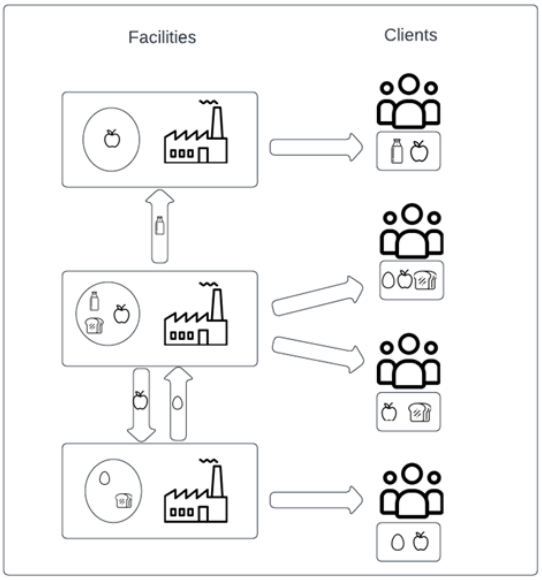

Figure 1: The model depicts facilities with production lines (circles) producing commodities. Clients, each having multi-commodity demand (square) and require supply from a single facility. As one facility might not fulfill all client demands, internal transport (arrow between facilities) is essential.

The exact model used in this thesis is not found in the literature. However, some models that are very similar are found in the survey of Ulukan and Demircioğlu (2015) including all research on the single source capacitated facility location problem. Which is solved mostly with the use of Lagrangian relaxation for initial solutions in combination with branch & bound, tabu search, or neighborhood search. Pirkul and Jayaraman (1998) formulated a multi-commodity, multi-plant capacitated facility location problem called the PLANWAR model. They used Lagrangian relaxation to find a lower bound combined with a custom heuristic to find good solutions to the problem.

1.2 Problem statement and research objective

The multi-commodity capacitated discrete location allocation model lacks specific literature covering both its formulation and specific solving algorithms. Especially the modelling of production lines at facilities poses its own unique constraints and could result in challenges for currently available solvers and algorithms. The objective of this thesis is to formulate the model and create an algorithm that is capable of finding good, feasible solutions within a reasonable amount of time compared to a commercially available solver.

1.3 Research questions

The main research question is:

How can the multi-commodity capacitated discrete location allocation model be modeled, and effectively solved, both in terms of the objective value and computing time compared to a commercially available solver?

Research sub-questions:

1. How is the multi-commodity capacitated discrete location allocation model precisely formulated?
2. What are the optimization techniques and algorithms that can be applied to solve the formulated model?
3. How does the chosen algorithm perform across diverse data sets in comparison to a commercially available solver?

2 Literature review

In this thesis, the focus is on the multi-commodity capacitated discrete location allocation model and exploring various solving methods to analyze their capabilities and efficiency. A comprehensive literature review is conducted to examine the existing research on these exact and similar models. Additionally, the literature review will highlight various solving methods employed in previous studies.

2.1 Model formulation

In this section of the literature review the model's components and alternative modeling approaches are researched.

2.1.1 Discrete, Continuous

In location modelling, a crucial consideration is whether to formulate the model as discrete or continuous. In a discrete model, all potential locations are defined as a set, typically denoted as $i \in I$, where i represents a possible location and I denotes the set of all locations. This formulation is reflected in the objective function as:

$$\sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{1}$$

where c_{ij} is a parameter representing the costs associated with delivering from point i to point j, and x_{ij} is a binary variable indicating whether point i delivers to point j (Conforti et al., 2014).

On the other hand, a continuous model aims to determine the optimal location within a continuous space. The objective function of a continuous model incorporates a variable distance associated with the cost of traveling a unit distance. This formulation takes the form of:

$$\sum_{i \in I} \sum_{j \in J} c_{ij} w_{ij} d(x_i, a_j). \tag{2}$$

Where c_{ij} represents the cost of transporting a unit flow per unit distance from facility i to client j, w_{ij} denotes the quantity of products transported from i to j, and d denotes the distance traveled. The distance calculation,

denoted as $d(x_i, a_j) = [|x_{i1} - a_{j1}|^r + |x_{i2} - a_{j2}|^r]^{\frac{1}{r}}$, is determined by the r-norm used in the model. The r-norm can range from 1 to ∞ and influences the calculation of the distance traveled: Manhattan distance ($r = 1$), Euclidean distance ($r = 2$), or Chebyshev distance ($r = \infty$) (Fernández et al., 2000).

An addition can be made to the objective function by incorporating a fixed opening cost for facilities. In discrete models, this additional opening cost is typically represented by a pre-defined parameter, given that the possible locations are known in advance. In continuous models, a standard opening cost can be applied, or a zone-dependent facility opening cost can be employed, as presented by Luis et al. (2015).

Sherali et al. (1994) presents a continuous capacitated location allocation model with Manhattan norm ($r = 1$) and transforms it into a discrete location allocation problem. The optimal locations are located at grid points formed by vertical and horizontal lines drawn through existing client locations, and the optimal location lies within the convex hull of clients, creating a finite set of candidates. Building upon this discrete approximation, Aras et al. (2007) developed a new efficient heuristic. Akyüz et al. (2012) also implements a discrete approximation but proposes the use of the $l_{1\infty}$ norm instead of the l_1 norm, demonstrating excellent accuracy and accommodating multiple commodities.

2.1.2 Single commodity, Multi commodity

In a single commodity model, the client's demand consists of only one commodity, and no adjustments need to be made to the objective function. To ensure that the demand is met, a constraint is formulated as follows:

$$\sum_{i \in I} w_{ij} = q_j \quad \forall j. \tag{3}$$

where w_{ij} represents the amount of product transported from facility i to client j, and q_j is the demand of client j .

However, in a multi-commodity model, the client's demand can consist of multiple commodities from a set denoted as $k \in K$. The objective function needs to be modified to incorporate this new set of commodities, resulting in the transformed expression(Naoto, 2015):

$$\sum_{k \in K} \sum_{i \in I} \sum_{j \in J} c_{kij} x_{kij}. \tag{4}$$

Furthermore, the demand constraint is modified to:

$$\sum_{i \in I} x_{kij} = q_{kj} \quad \forall j, k \tag{5}$$

Where x_{kij} represents the amount of commodity k transported from facility i to client j, and q_{kj} is the demand for commodity k from client j (Fernández et al., 2000).

Clearly, if the number of commodities is one, the multi-commodity model is reduced to the simpler single commodity model (Karkazis & Boffey, 1981). Based on this, Warszawski (1973) created two methods to solve the multi-commodity model. The first was a branch and bound method with $k + 1$ way branching at sub problem p on whether or not a selected site should be occupied or unoccupied by a facility of commodity k. By summing on the sub problems, a lower bound can be determined. Neebe and Khumawala (1981) proposes a stronger lower bound by allowing the facility i to be either fixed open at commodity k, fixed closed, or free. It is determined that a free warehouse will never be assigned to certain commodities in any optimal solution. Thus, such a warehouse can be fixed closed with respect to these certain commodities but left in a free status for the remaining commodities. This observation may lead to a stronger lower bound at successor nodes. However, Karkazis and Boffey (1981) doubts if this method proposed by Neebe and Khumawala (1981) is effective since no experimental results have been given.

The other solution Warszawski (1973) proposed was a heuristic since the branch and bound procedure could lead to excessive computations for larger problems. At any node p, the problem of a single commodity is solved for every commodity. If anywhere the constraint is violated, indicating that more facilities need to be opened at a single location, a new node will be formed by assigning the location k to a supply source from the commodity that violated the constraint for that location. This is done in such a way that the lower bound for the new node will be a minimum.

2.1.3 Capacitated and single source

Capacitated models implement a production capacity on facilities and imply that the total outflow of the product, if multi-commodity (including k different products), from location i to j does not exceed the capacity of facility

i. This is done through the constraints:

$$\sum_{j \in J} \sum_{k \in K} w_{jk} x_{kij} \leq a_{ki} y_i \quad \forall i \in I \tag{6}$$

Where a_{ki} represents the capacity of product k on facility i, and y_i is a binary variable indicating whether facility i is open (for discrete models). w_{jk} is the demand for product k from client j. By incorporating the capacity constraint into the model, it becomes necessary to open multiple facilities to meet the demands of clients (Naoto, 2015).

Single-source constraints impose that a client or demand point is only facilitated by a single facility, so called indivisible flow. This is implemented with the constraints (Ulukan & Demircioğlu, 2015):

$$\sum_{i \in I} \sum_{k \in K} x_{kij} = 1 \quad \forall\ j \in J \tag{7}$$

$$x_{kij} \in \{0, 1\} \quad \forall k \in K, i \in I, j \in J \tag{8}$$

2.2 Solvers

A mixed-integer programming (MIP) problem can include both integer and continuous variables. When the problem's objective function consists solely of linear expressions, it is categorized as a Mixed Integer Linear Programming (MILP) problem. This is formally represented as follows:

$$\begin{aligned} \text{Minimize:} \quad & c^T x \\ \text{Subject to:} \quad & Ax \leq b \\ & x \in \mathbb{R}^n \\ & x_j \in \mathbb{Z}, \quad \forall j \in I \end{aligned} \tag{9}$$

Should the objective involve a quadratic term, the problem transforms into a Mixed Integer Quadratic Program (MIQP). Furthermore, if any constraint within the model incorporates a quadratic term, regardless of the objective function, the problem is referred to as a Mixed Integer Quadratic Constrained Program (MIQCP) (Zhang et al., 2023).

In this thesis, all variables are formulated as integer variables. As a result, the primary focus of this thesis will be on the application of algorithms to solve MIP problems, although some of these algorithms may have applications in

other problem domains. The solvers are divided into exact algorithms and heuristics. Lastly, Gurobi will be discussed, which is the benchmark solver for this thesis.

2.2.1 Exact algorithm

Branch-and-cut is currently the most popular method for MIP solvers, which is formed by combining branch-and-bound and cutting-plane algorithms (Zhang et al., 2023). First, an overview of the general branch and bound algorithm is given, and the cutting plane algorithm is explained. After the branch-and-cut is explained.

Branch and bound operates on the principle of relaxing the MIP problem (refer to equation 9) into a series of linear programming (LP) problems (P), achieved by relaxing the constraints associated with integrality. The relaxed model takes on the following form:

$$\begin{aligned} \text{Minimize:} \quad & c^T x \\ \text{Subject to:} \quad & Ax \leq b \\ & x \in \mathbb{R}^n \end{aligned} \tag{10}$$

This relaxed formulation is easily solvable and gives the lower bound for minimization. Should the solution obtained consist solely of integer values for the decision variables, the algorithm stops and found the optimal solution. However, this scenario is often not the case. More frequently, fractional values are created. Then, two new problems are created called nodes. These nodes have constraints fixed to the nearest closed integer values. For instance, let's consider the variable $x_1 = 3.4$, leading to the formulation of two distinct MIP problems: P_1 with the constraint $x_1 \leq 3$, and P_2 with the constraint $x_1 \geq 4$. This process of branching persists until an integer solution is found, creating what is known as the search tree.

When an integer solution is found within a node, it becomes the upper bound. Consequently, that specific node ceases to branch (a process termed "fathoming"), as no better feasible solution can be found on that particular branch. The upper bound serves as the current best feasible solution. In instances where a solution in a node is not lower than the upper bound, the node is fathomed. However, if a lower integer solution is found, it becomes the new upper bound. A node will also fathom if it contains no feasible solution for the LP relaxation. This process continues until all nodes are evaluated.

The best upper bound is the optimal solution to the original MIP problem (Conforti et al., 2014; Zhang et al., 2023).

Cutting plane algorithms work on the basis that every MIP model can be relaxed to an LP problem, as can be seen in equation 9 and equation 10. Such LP problems can be solved by traditional efficient algorithms for LP problems, such as the Simplex and interior point method. The cutting plane algorithm adds cuts to the LP relaxation iteratively. The cuts are linear constraints that tighten the feasible area of the LP relaxation while keeping the MIP optimal solution (Conforti et al., 2014; Zhang et al., 2023). An example of the cutting plane algorithm is given in subsection 2.2.3. The reason why these constraints are added iteratively rather than all at the start is because a lot of these constraints can be formulated, making it computationally very hard to identify and include them all. Additionally, adding additional constraints makes the LP relaxation harder to solve. Therefore, constraints should only be added if their potential benefit is enough to make the search worthwhile (Gurobi Optimization, 2023).

Branch-and-cut method uses the cutting plane algorithm within the branch and bound tree to tighten the feasible region of the subproblems, thereby significantly speeding up the solving process. since less branching is required (Conforti et al., 2014; Zhang et al., 2023).

2.2.2 Heuristics

In practical scenarios, problems are often too big or too complex for exact methods to yield optimal solutions within a reasonable amount of time. In such instances, heuristics can be used. Heuristics aim to identify reasonably good feasible solutions in a relatively short amount of time, although the found solution is not guaranteed to be optimal (Claassen et al., 2006). However, very often exact methods succeed in finding a near optimal solution quickly but need a lot of time to reach an optimal one, or they succeed in finding the optimal solution but use a significant amount of time to prove optimality. For these reasons in some practical cases, exact methods are used as heuristics by stopping the solving procedure before getting a proof of optimality (e.g. imposing a CPU time limit, maximum number of iterations, node limit, etc.)(Hanafi & Todosijević, 2017).

Due to the limitations of exact methods, many heuristics are made. The advantage of heuristics is not only their ability to produce high-quality solu-

tions in a short time but also that they can easily be combined with exact methods to speed them up. For example, establishing a tight initial upper bound (for minimization) can considerably reduce the scale of a branch and bound tree. A distinction is drawn between common sense heuristics and metaheuristics. It should be noted that while these heuristics are examined in the context of MIP problems, they are not exclusively limited to MIP problems.

Commonsense heuristics use mostly practical rules or strategies that are based on common knowledge and intuition rather than rigorous mathematical or algorithmic techniques. Sometimes these heuristics are highly specific to the problem at hand and can yield exceptional solutions for that precise context. Yet, the drawback is that a slight alteration in problem characteristics could render the heuristic ineffective, necessitating the formulation of a new approach. Further classification of common sense are construction heuristics and improvement heuristics. A construction heuristic revolves around an iterative process: selecting a set of variables and refining variable bounds. Conversely, an improvement heuristic initiates with an initial solution and seeks improvements (Claassen et al., 2006; Zhang et al., 2023).

For instance, rounding is one such construction heuristic. It involves solving the LP relaxation of an MIP (seen in equation 10) to yield a fractional solution, which is subsequently rounded to an integer solution. Simple rounding preserves LP-feasibility, converting LP-feasible solutions into integer solutions while remaining LP-feasible (Conforti et al., 2014). The add heuristic, start with an empty solution, and keep adding decision variables. E.g., for an uncapacitaed facility location, open only 1 facility that supplies every client, the heuristic tries to find a cheaper solution by opening and extra depot whilst keeping the first depot open, all combinations are tried. The best improved solution is stored and from that solution the heuristics tries to again open new facilities to find a better solution, if no better solution is found the heuristic terminates (Claassen et al., 2006). This sort of heuristic falls under greedy algorithms. A greedy algorithm makes the best choice at the moment and solves the sub problem. The choice made may depend on previous choices but does not take future choices in consideration.

Improvement heuristics try to improve an existing solution, usually in an iterative way(Claassen et al., 2006). Most improvement heuristics work via a local search, that operates by exploring a structured neighborhood of solutions surrounding the best current feasible solution. When a better solution is found within this neighborhood, it replaces the current solution. This

iterative process continues until no further improvements can be made, at which point the heuristic terminates (Hansen & Mladenović, 2005). A well known variant of local search is the descent method, which exclusively accepts solutions that are better to the current one. However, a big downside of local search lies in the potential for the solution to become trapped in a local optimum, if the neighborhood is not big enough to climb out the local optimum (Claassen et al., 2006). An alternative improvement heuristic is the drop heuristic, which operates opposite to that off the add heuristic. This approach initiates by opening all facilities (for an UFLP). Subsequent iterations involve the closure of a facility and the reassignment of clients to other open facilities. Should a better solution be found, it replaces the existing one. This iterative process continuous until no further enhancements can be identified, leading to the termination of the heuristic (Salhi & Atkinson, 1995).

Metaheuristics often use techniques to explore the solution space in a more systematic way, allowing it to be applicable to a relatively wide range of problems, and thus have the advantage that they are more robust if the problem characteristics change. A disadvantage of meta heuristics is that they are often less effective than a tailor-made heuristic. Simulated annealing, tabu search, and genetic algorithm are very popular meta heuristics and will be explained.

Simulated annealing (Aarts et al., 2005; Claassen et al., 2006) starts with an initial solution and iteratively explores the solution space by considering neighboring solutions. The algorithm employs a temperature parameter that controls the likelihood of accepting solutions that are worse than the current one. As the algorithm progresses, the temperature gradually decreases, reducing the change to accept worse solutions. This controlled search of the solution space allows simulated annealing to escape local optima and discover better solutions that might be located out of the local optimum. The cooling schedule, which controls how the temperature decreases over iterations, is a important factor that influences the balance between exploration and exploitation .

Tabu Search (Claassen et al., 2006; Duarte et al., 2018) is a heuristic optimization algorithm that draws inspiration from social taboos to guide the search for optimal solutions in the solution space. This technique focuses on preventing the algorithm from revisiting previously explored solutions, by imposing a memory-based mechanism to diversify the search process. Tabu Search maintains a short-term memory, known as the "tabu list," which

records recently visited solutions and their attributes. This prevents the algorithm from getting stuck in local optima by prohibiting certain moves that lead back to previously explored areas. Additionally, the algorithm uses an aspiration criterion to relax tabu restrictions when a new solution exhibits significant improvement over the current best solution.

Tabu Search also employs a strategic exploration strategy by favoring moves that lead to unexplored regions or promising areas in the solution space. The algorithm's efficiency relies on the balance between diversification (exploration of new regions) and intensification (exploitation of solution space). Parameters such as tabu tenure, which defines the duration solutions remain on the tabu list, and neighborhood structures, which define the set of feasible moves, play crucial roles in shaping the search process. The algorithm iteratively refines solutions by considering and evaluating neighboring solutions while adhering to the tabu conditions.

The genetic algorithm (Claassen et al., 2006; Koza & Poli, 2005) is a heuristic optimization technique inspired by the process of natural evolution. The core concept behind a genetic algorithm lies in the principles of selection, crossover, and mutation observed in biological evolution.

The genetic algorithm starts by initializing a population of potential solutions, often referred to as individuals or chromosomes. Each individual represents a candidate solution to the problem at hand. Through a series of generations, the algorithm refines the population to converge towards an optimal or near-optimal solution. During each generation, a selection process is employed to choose individuals from the current population based on their fitness, which is determined by how well each solution performs based on the objective value to the problem.

Crossover and mutation are the two operators that drive the algorithm's exploration and exploitation capabilities. Crossover involves combining solutions from two parent individuals to create a new solution. Mutation introduces small random changes a variable in the new solution, helping to introduce diversity and prevent getting stuck in local optima.

As generations progress, individuals with higher fitness are more likely to be selected, while crossover and mutation try to prevent local optima. The genetic algorithm dynamically balances exploration and exploitation by maintaining a diverse population while giving better solutions a higher change to be chosen.The success of a genetic algorithm heavily depends on parameters such as population size, selection strategy, crossover rate, and mutation rate.

2.2.3 Gurobi

A large number of optimization software products are developed, all with their own strengths, and can be broadly classified into two categories based on licensing or open-source software. The three most known commercial optimization solvers are CPLEX, Gurobi, and XPRESS. The fundamental structure of an optimization solver comprises five steps. These steps are: problem formulation, model creation, model configuration, optimization, and solution generation. During problem formulation, the objective functions and constraints are defined. In the model creation phase, all identified functions and constraints are translated into a modelling language. The decision variables are then incorporated into the model. Following this, the model is fine-tuned based on the defined constraints and objective functions. The optimization process focuses on enhancing the objective functions within the model. Finally, optimized solutions are generated as a result. Anand et al. (2017) performed an analysis on all three solvers and found that both CPLEX and Gurobi provided competitive results in a real-life situation. XPRESS performs better than both CPLEX and Gurobi on complex and high-scalability problems due to its multi-threading capabilities. In this thesis, the Gurobi solver is used because of its easy installation and academic license.

The Gurobi solver uses the branch and bound algorithm described in subsection 2.2.1. Additionally, Gurobi includes presolve, cutting planes, heuristics, and parallelism which improve the solving time drastically and are explained below. All information and examples are retrieved from the Gurobi website (Gurobi Optimization, 2023).

Presolve constitutes a collection of problem reduction techniques commonly applied prior to commencing the branch-and-bound procedure. The primary objective of these techniques is to systematically reduce the problem's dimensions and refine its formulation for enhanced efficiency. A straightforward instance of dimension reduction involves scenarios where a problem is defined by constraints such as $x_1 + x_2 + x_3 \geq 15$, $x_1 \leq 7$, $x_2 \leq 3$, and $x_3 \leq 5$. Evidently, these constraints can only be simultaneously satisfied when $x_1 = 7$, $x_2 = 3$, and $x_3 = 5$. Consequently, substituting these values eliminates both the variables and their associated constraints, showing the potential of such reductions to significantly simplify problem structures.

This specific reduction demonstrates what is termed as an 'LP-presolve' reduction, distinguished by its independence from integrality constraints. Contrarily, an instance of an 'MIP-specific' reduction pertains to cases where x_1

and x_2 are non-negative integer variables, accompanied by a constraint like $2x_1 + 2x_2 \leq 1$. Dividing the constraint by 2 yields $x_1 + x_2 \leq \frac{1}{2}$. Given the integral nature of x_1 and x_2, this inequality implies $x_1 + x_2 \leq 0$, leading to the conclusion that $x_1 = x_2 = 0$ due to non-negativity. Consequently, both the variables and the constraint can be eliminated from the problem formulation. Notably, this reduction differs from the previous case as it refines the solution space of the LP relaxation, despite the integer feasible solution set remaining unchanged.

Cutting planes is already explained in subsection 2.2.1, but here an example is given. Suppose our formulation includes the constraint: $6x_1 + 5x_2 + 7x_3 + 4x_4 + 5x_5 \leq 15$, where x_1 through x_5 are constrained to be binary. The solved LP relaxation gives: $x_1 = 0$, $x_2 = 1$, and $x_3 = x_4 = x_5 = \frac{3}{4}$. This solution can be excluded because $7 + 4 + 5 = 16 > 15$, it is impossible for $x_3 = x_4 = x_5 = 1$. Thus, the new valid constraint can be added to the MIP: $x_3 + x_4 + x_5 \leq 2$. Given that $\frac{3}{4} + \frac{3}{4} + \frac{3}{4} = \frac{9}{4} > 2$, this new constraint reduces the current solution space.

Heuristics can be used to find a good incumbent for the branch-and-bound tree which can drastically reduce computing time. Because a good incumbent objective value increases the change of the value of an LP relaxation surpassing it (in minimization) leading to node pruning. Various heuristics are applied, such as the rounding heuristics detailed in the section 2.2.2. While additional heuristics are utilized, the information obtained from Gurobi does not specify which heuristic.

Parallelism is type of computing in which many calculations or processes are carried out simultaneously. Gurobi uses multiple cores exploring different nodes of the branch-and-bound tree at the same time. Effectively speeding up the search process (Gurobi Optimization, 2023)

3 Methodology

The aim of the thesis is to create a complete model formulation, algorithm and test the algorithm to the commercial solver Gurobi. The model formulation and the algorithm are based on the literature review in section 2. These are tested on custom made data sets and compared to the Gurobi solver from subsection 2.2.3.

3.1 Full Model

Based on the model formulations found in the literature review in section 2, a complete model can be formulated for the Multi-Commodity Capacitated Discrete Location Allocation Model with production lines. However, besides all the model elements already explained the model used in this thesis contains some unique elements. The interaction between the multi commodity, capacitated, and single source constraints makes it necessary to allow for internal transport, which means transport between facilities is allowed to fulfil demand for the clients. To accommodate this internal transport, flow constraints need to be formulated as well as an addition to the objective function to account for the extra transportation cost. The flow constraints are formulated in equation 11.

$$\sum_{l \in L} z_{kli} + \sum_{q \in I} w_{kqi} - \sum_{r \in I} w_{kir} - \sum_{j \in J} a_{jk} x_{ij} \geq 0 \quad \forall i \in I,\ k \in K \tag{11}$$

z_{kli} is the decision variable, indicating the amount of commodity $k \in K$ produced on production line $l \in L$ in facility $i \in I$. w_{kqi} is the decision variable for the amount of commodity $k \in K$ internally transported from facility $q \in I$ to facility i. On the other side, w_{kir} is the amount of commodities transported from facility i to facility $r \in I$, because these commodities are leaving the facility. Lastly, $a_{jk}x_{ij}$ is the outgoing flow to the client. This constraint ensures that all commodities that are transported from facility i are available at facility i.

The internal transport cost is formulated slightly differently compared to the allocation cost to clients. Due to the discrete formulation, the single source constraint, and known demand, it is possible to calculate the total cost of fulfilling all demand for every client j from facility i for all possible facilities. Therefore, it can be put in parameter C_{ij}. For internal transport, it is not known beforehand how many products are transported between facilities. A new decision variable is made called xx_{qr}, which indicates the

number of trucks that ride between facilities $q \in I$ and $r \in I$, multiplied by the cost of moving a single truck from facility q to r and ee is the number of commodities that can fit in a single truck. To ensure enough trucks are driven for the amount of product transported, constraint 12 is formulated. The addition to the objective function is formulated in equation 13.

$$\sum_{k \in K} \frac{w_{kqr}}{ee} \leq xx_{qr} \quad \forall q \in I, \forall r \in I \tag{12}$$

$$\sum_{q \in I} \sum_{r \in I} cc_{qr} xx_{qr} \tag{13}$$

The production lines have a fixed cost to place them in a facility. Additionally, a constraint needs to be formulated to ensure a facility has enough space to place a production line. The cost of setting up a production line is very similar to opening a facility. However, it is possible to place multiple production lines in the same facility, as seen in equation 14.

$$\sum_{l \in L} \sum_{i \in I} ff_l yy_{li} \tag{14}$$

Where ff_l is the fixed cost to place production line l, and yy_{li} is an integer value representing the number of production lines l placed in facility i

The constraint enforcing enough space in the facility for production lines is shown in equation 15.

$$\sum_{l \in L} e_l yy_{li} \leq b_i \quad \forall i \in I \tag{15}$$

Where e_l represents the size needed to place a single production line l. b_i represents the space available in facility i. Lastly, a constraint is formulated to ensure no production lines are placed if facility i is not open. Formulated in equation 16.

$$\sum_{l \in L} yy_{li} \leq M_1 y_i \quad \forall i \in I \tag{16}$$

Because $yy_l i$ is an integer value and can thus take a value higher than 1 and y_i is a binary variable. A big M constraint is required. A big M is a large positive constant, which ensures the possibility of opening more production lines than 1 if the facility is open. However, in practice high values of M lead to numerical instability and provide low-quality bounds (Bertsimas et al., 2021). Therefore, it is important to set M as small as possible.

Based on all the information provided in the literature review and section above, the complete model can be formulated as follows:

Sets:

$I = \{1, \ldots, m\}$ set of potential facilities
$J = \{1, \ldots, n\}$ set of clients
$K = \{1, \ldots, o\}$ set of commodities
$L = \{1, \ldots, p\}$ set of production lines

Decision variables:

$$y_i = \begin{cases} 1 & \text{if facility } i \text{ is open} \\ 0 & \text{if not} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if facility } i \text{ delivers to client } j \\ 0 & \text{if not} \end{cases}$$

yy_{li} = amount of production lines l in facility i
xx_{qr} = amount of trucks from facility q to facility r
z_{kli} = amount of commodity k produced on production line l in facility i
w_{kqr} = amount of commodity k internally transported from facility q to facility r

Parameters:

a_{jk} = demand of client j per commodity k per year	$\forall j \in J, \forall k \in K$
b_i = capacity of facility i in m^2 to place production lines l	$\forall i \in I$
bb_{kl} = capacity of production line l to produce commodity k per year	$\forall l \in L, \forall k \in K$
c_{ij} = total cost of transportation from facility i to client j per year	$\forall i \in I, \forall j \in J$
cc_{qr} = cost of moving a single truck from facility q to facility r	$\forall q \in I, \forall r \in I$
f_i = fixed cost to open facility i per year	$\forall i \in I$
ff_l = fixed cost to open production line l per year	$\forall l \in L$
d_{kl} = cost to produce commodity k on production line l	$\forall k \in K, \forall l \in L$
e_l = m^2 required per production line l	$\forall l \in L$
ee = amount of commodity transportable per truck	

Objective function:

$$\min\{\text{production costs} + \text{allocation costs} + \text{internal transport}\} \qquad (17)$$

Total production cost:

$$\sum_{i \in I} f_i y_i + \sum_{l \in L} \sum_{i \in I} f f_l y y_{li} + \sum_{k \in K} \sum_{l \in L} \sum_{i \in I} d_{kl} z_{kli} \tag{17a}$$

Total allocation cost:

$$\sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{17b}$$

Total internal transport cost:

$$\sum_{q \in I} \sum_{r \in I} cc_{qr} x x_{qr} \tag{17c}$$

Constraints:

$$\sum_{i \in I} x_{ij} = 1 \quad \forall j \in J \tag{18}$$

$$x_{ij} \leq y_i \quad \forall i \in I, \forall j \in J \tag{19}$$

$$\sum_{l \in L} y y_{li} \leq M_1 y_i \quad \forall i \in I \tag{20}$$

$$\sum_{l \in L} z_{kli} + \sum_{q \in I} w_{kqi} - \sum_{r \in I} w_{kir} - \sum_{j \in J} a_{jk} x_{ij} \geq 0 \quad \forall i \in I, \forall k \in K \tag{21}$$

$$\sum_{l \in L} e_l y y_{li} \leq b_i \quad \forall i \in I \tag{22}$$

$$z_{kli} \leq bb_{kl} y y_{li} \quad \forall i \in I, \forall k \in K, \forall l \in L \tag{23}$$

$$\sum_{k \in K} \frac{w_{kqr}}{ee} \leq x x_{qr} \quad \forall q \in I, \forall r \in I \tag{24}$$

$$y_i \in \{0, 1\} \quad \forall i \in I \tag{25}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, j \in J \tag{26}$$

$$y y_{li} \in \mathbb{Z}^+ \quad \forall i \in I, l \in L \tag{27}$$

$$x x_{qr} \in \mathbb{Z}^+ \quad \forall q \in I, r \in I \tag{28}$$

$$z_{kli} \in \mathbb{Z}^+ \quad \forall i \in I, k \in K, l \in L \tag{29}$$

$$w_{kqr} \in \mathbb{Z}^+ \quad \forall q \in I, r \in I, k \in K \tag{30}$$

3.2 The Algorithm

The algorithm used in this thesis is custom-made for the model described in section 3.1. The outcomes of the custom-made algorithm are compared to those of the Gurobi solver, which serves as a benchmark. Throughout the rest of the report, the custom-made algorithm will be referred to as 'The Algorithm,' and if the Gurobi solver is used, this will be specifically mentioned. The aim of The Algorithm is to find a better solution in terms of minimizing the objective function within a shorter time frame than the Gurobi solver can achieve. To accomplish this, The Algorithm utilizes various techniques, some of which are described in section 2. The Algorithm consists of four steps and cycles from step 4 to step 2 until the termination criteria is met, this process is illustrated in Figure 2. The four steps are:

1. Creating initial feasible solutions.
2. Genetic algorithm.
3. Allocating facilities and locations.
4. Product flows.

The algorithm is programmed in Python, and all runs are tested on the same laptop, whose specs can be found in Appendix A. Gurobi is also used for some parts of the algorithm. The reasoning for this is that Gurobi is much faster than a standard Python script, partly due to the parallelism capabilities of Gurobi (Gurobi Optimization, 2023).

Step 1: Feasible solution creation is the first step of The Algorithm. These feasible solutions are the starting stones for the rest of The Algorithm. Especially for the genetic algorithm it is important to provide a list with different feasible solutions to search the overall search space (Koza & Poli, 2005). To create the list of different feasible solutions the Gurobi solver is used, which directly makes the overall model easier to solve because of the presolve phase. When Gurobi solves the Branch-and-Bound tree, every time an incumbent is found the solution is stored in the solution pool. By implementing a callback function the solver will terminate if the solution pool reaches a specified amount, which in this thesis is 40 for the medium sized data set and 10 for the large sized data set. Lastly, all the different solutions, including the objective functions and values for the variables are retrieved and stored in a list.

Step 2: Genetic algorithm is the second step of The Algorithm. The genetic algorithm is only applied to the production line variable yy_{li}. The yy_{li} variables of two random feasible solutions from the solution pool are combined at a random cross over point, the better the solution is, the higher the change a solution is chosen out of the solution pool. Every variable has a change to mutate, depending on the state of the variable it can get +2/-2 or +1/-1 for a type of production line. A new list of values for the yy_{li} variable is created. After some testing a mutation rate of 0,1 showed the most promising. Meaning every variable has a 10% change to increase or decrease, which is very high for a genetic algorithm (Cavuoti et al., 2013).

Step 3: Allocation facilities and clients is done by a common-sense heuristic. Facilities are opened and closed based on the yy_{li} values gained from the genetic algorithm. If a production line is located at facility i, set $y_i = 1$. This is expressed by equation 31. After clients are assigned to the facility with the lowest transportation cost, denoted as c_{ij}. To prevent clients from being assigned to a closed facility, a multiplier of 10.000 is applied to their transportation costs from the closed facility. This ensures that it is never feasible for a client to be allocated to a closed facility.

$$y_i = \begin{cases} 1, & \sum_{l \in L} yy_{li} > 0 \\ 0, & \sum_{l \in L} yy_{li} = 0 \end{cases} \tag{31}$$

Step 4: Product flows is the last step of The Algorithm. At this phase of The Algorithm the variables y_i, yy_{li} and x_{ij} have a fixed value. The last variables, xx_{qr}, z_{kli} and w_{kqr} need to be calculated. Since the most important variables are already fixed, it is possible to fill in the values for the other variables. This is also done via the Gurobi solver, because there is a high chance that by fixing the variables, the model becomes infeasible. Gurobi is extremely fast at running infeasibility checks, making The Algorithm not spend much time on infeasible solutions. If a new feasible solution is found, it is directly added to the solution pool, which the genetic algorithm takes solutions out of. This implies the genetic algorithm does not have specific generations but a continuous generation on which only the best solutions are stored. The reason is that a single iteration takes relatively long to compute compared to a standard genetic algorithm. Therefore, it takes long to generate a whole generation based on 'bad' solutions. If, per change a solution already exists it does not get stored in the solution pool double, instead the solution gets discarded. The Algorithm will terminate after a

set number of iterations, if no better feasible solution is found. Infeasible solutions do not count toward this parameter, which is set to ten iterations. If this criteria is not yet reached, The Algorithm will loop back to step 2 and repeat the process.

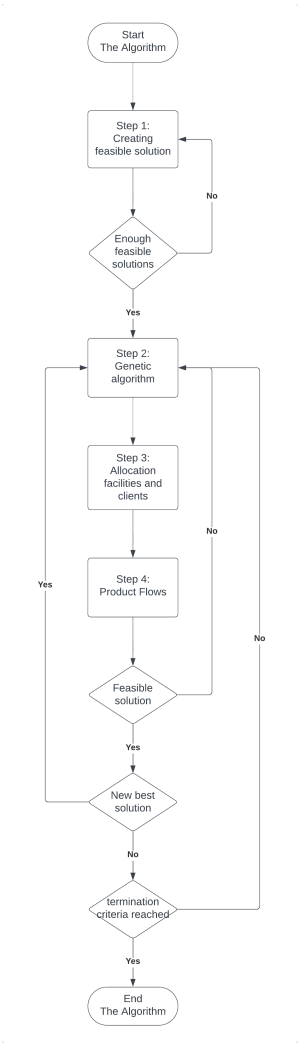

Figure 2: The Flowchart showing The Algorithm with the four steps: Creating feasible solution, Genetic Algorithm , Allocation of facilities and clients, Product flows. The first decision secures the creation of a enough feasible solutions to proceed to the second step. The decisions 'New best solution' and 'Termination criteria reached' guarantee enough generations of Steps 2, 3, and 4 before The Algorithm terminates.

3.3 Data creation

This model with the inclusion of production lines is not earlier used in research. Therefore, new data sets are specifically made for the tests. The data is based on geographical information focused on the Netherlands. The data sets are created and fine tuned by trial and error with short test runs with the Gurobi solver. Aiming to achieve 70% production costs (equation 17a) and 30% transport costs (equation 17b + 17c).

To create a realistic distribution in facility and client locations, the following method is used. Assumed is that facilities can be opened in large Dutch cities, e.g. if the dataset contains $I = 20$ the geographical coordinates (latitude and longitude) of the 20 largest cities in the Netherlands are retrieved from the Central Bureau of Statistics and used (CBS, 2022). To create the coordinates for clients (denotes as J) a list is created for every municipality in the Netherlands with the number of Albert Heijn's (AH, 2023), a major supermarket chain in the Netherlands, in that municipality. It is assumed that the more Albert Heijn's in a municipality, the more people live there. Based on the number of J, random coordinates from a municipality are retrieved from the list, the more Albert Heijn's the higher the chance of that municipality being picked, thereby preserving a distribution.

To create the C_{ij} and CC_{qr} parameters the Euclidean distance is calculated and multiplied by a factor of 1,2 to compensate for the fact that streets are not straight lines between locations. This is done for every possibility between the combinations I-J and I-I. This is the number of kilometers a truck has to drive for a one-direction trip. The cost per kilometer is €0,35 and a truck has to return so the distance is doubled. Additionally, a fixed cost of €50 is added to drive the truck. This is the cost for CC_{qr} since beforehand it is not known how many trucks are being driven between locations. For C_{ij} this is not the case since demand is known and the single source constraint enforcing a client can only be supplied from a single facility. It can be calculated how much it is going to cost to supply every client from every facility. By dividing total demand by ee (truck capacity) multiplied by the cost for a truck to drive to the client.

The values of a_{jk} are generated using a normal distribution. This distribution is centered around a mean of 4000 and has a deliberately high standard deviation of 7000. This high standard deviation is intentional; it serves the purpose of ensuring negative demand values appearing. In cases where demand might turn negative, it is adjusted to 0 instead. This approach ensures that not all clients will demand every commodity.To prevent the possibility of excessively high demand values due to the influence of the standard de-

viation, a random number ranging between 0 and 0,700 is generated. This random number is used to select a value from the inverse of the normal distribution function. This method helps in constraining the demand variation and keeping it within a reasonable range. In Figure 3 the demand of clients is shown for all commodities, everything to the left of the black dotted line is changed to a 0 in the used data sets.

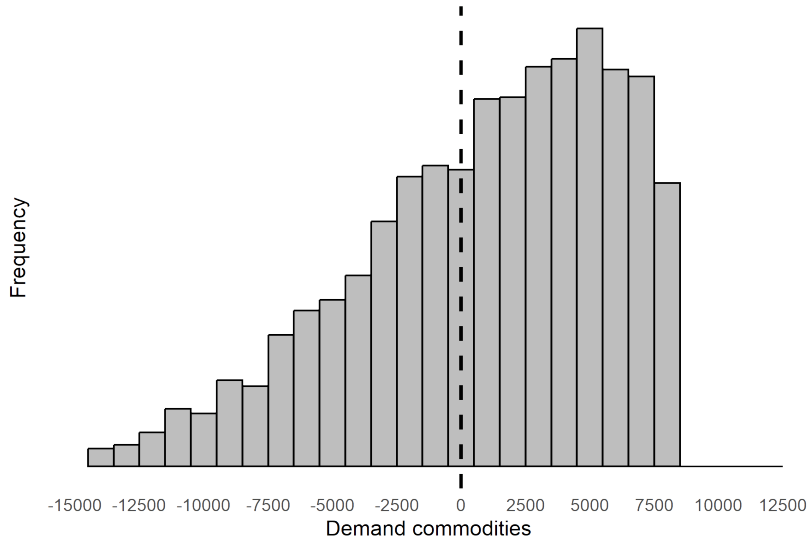

Figure 3: Visualization of the demand distribution: this histogram illustrates the distribution of demand amounts for commodities from clients. The X-axis represents the amount of demand, while the Y-axis displays the frequency of occurrence. The black dotted line marks the point of zero demand.

e_l is a small parameter and values are created manually for $L = 3$ the values are 400,600,800 for the medium data sets and $L = 4$ the values are 100,150,200,220 for the large data sets. b_i is created with a normal distribution. The concept here is that a regular facility can accommodate either 2 medium-sized production lines, 3 small ones, or a combination like 1 big and 1 small for the medium set. The large set needs more production lines per facility, since more commodities and more clients require both more production lines. The distribution has a mean of 1200.To allow for some difference in size, the standard deviation is set to 600. There is a small change that the facility will have a negative size, if this happens this facility will be generated again.

Each product can only be manufactured by a specific production line. The control over this allocation is determined by the parameter bb_{kl}. This pa-

rameter is created through a process trial and error, where various values are tried and assessed. If this parameter is set to low the model becomes infeasible or opens every location. If it is set to high only 1 of every production line is placed and no internal transport is required. It is found that right balance is with values ranging between 24.000 and 45.000.

The cost to open facility is depend on the available space in said facility. f_i is calculated by multiplying b_i with random value between 1.800 and 2.200. This way big facilities cost more than small facilities. The cost to open production line ff_l is in correlation with the space required e_l. For $L = 4$ the values are: 400.000, 500.000,600.000, 70.000 respectively.

For d_{kl} a small random number is chosen between 0,01 and 0,1.

3.4 Experiments

The Algorithm is tested on a medium and large data set. The reason to exclude a small data set is that most solvers are already capable of solving small problems within a minute, therefore a custom made algorithm is unnecessary. Three unique data sets are generated for each size category, resulting in a total of six data sets. Multiple data sets of the same size are required because the data is generated randomly, and the possibility exist an "easy" or "hard" to solve data set is created. The sizes of the data sets can be found in Table 1.

Table 1: Representation of sets in the medium and large size data sets.

	Medium	**Large**
Facilities (I)	20	40
Clients (J)	80	300
Production lines (L)	3	4
Commodities (K)	15	20

To assess the performance of The Algorithm, it undergoes ten separate runs on each data set due to its inherent randomness, including factors like crossover points and mutation changes. In contrast, Gurobi runs only once because it consistently follows the same path to reach a consistent solution. After The Algorithm terminates the computing time, the best objective function is stored. After the Gurobi solver is run till the same objective function is reached and the time taken is stored.

Every data set is run via the Gurobi solver for five hours, and the solution is stored. This solution is assumed to be the optimal solution to the problem

and used to calculate the MIP gap. MIP gap is an indication of how close the solver is to finding the true optimal solution, a small MIP gap indicates the solution is closer to optimally (Gurobi Optimization, 2023). This is calculated via equation 32.

$$MIP\ Gap = \frac{|Z_{\text{opt}}| - Z_{\text{best}}}{|Z_{\text{opt}}|} \times 100\% \tag{32}$$

Where:

- MIP Gap is the MIP gap percentage.
- Z_{best} is the objective value of the best solution found by the solver.
- Z_{opt} is the objective value of the proven optimal solution (if known), or the best lower bound found.

4 Results

The results contain information regarding a total of 100 runs, comprising 50 runs for five medium sized sets and another 50 runs for five large sized sets. The results obtained from these experiments are presented in tables 3 and 4 for the medium and large data sets, respectively.

For each set, the results show the best feasible solution out of the solution pool created by Gurobi in step 1 of The Algorithm. Within every data set, Gurobi comes to the same solution because it follows the same solving procedure every time before the termination criteria is reached, leading to the same outcome. Creating feasible solutions (sec) indicates the duration in seconds to create all feasible solutions for the first step in The Algorithm. Computing time Gurobi is the time it takes the standard Gurobi solver to find the same objective value as The Algorithm. Faster (sec) indicates how much faster The Algorithm is compared to that of the Gurobi solver by subtracting the Computing time Gurobi from Computing time total. If positive, The Algorithms is faster than Gurobi. If negative, Gurobi is faster than The Algorithm.

On all ten data sets, the Gurobi solver runs for five hours. The solutions, in table 2, are used as the optimal value for the MIP gap calculation seen in equation 32. In table 3 and 4 "MIP gap" references to the difference between the best found feasible solution and the optimal solution, and "New MIP gap" references to the difference between the solution found from The Algorithm and the optimal solution.

Table 2: Best feasible solutions for all datasets after 5 hours of solving via the Gurobi solver.

Scenario	**Medium Dataset**	**Large Dataset**
1	65.512.002	116.960.850
2	74.679.760	165.386.407
3	84.182.397	175.006.945
4	81.293.280	177.957.740
5	72.156.971	178.574.967

4.1 Medium Data Sets

For the first medium-sized set, all runs consistently discovered a better solution after The Algorithm terminated compared to the best feasible solution discovered by Gurobi during the initial step of The Algorithm. The MIP gap improved from 0,213 to an average of 0,082, in the best case to 0,080 and in the worst case to 0,95. Furthermore, every run was faster than the standard Gurobi solver in terms of finding the same objective function. On average, The Algorithm is 59,8 seconds faster than Gurobi. The largest time difference observed was 76 seconds, while the smallest difference was 34 seconds. 6 out of 10 solutions came to the same objective function of 71.211.570 and twice to 71.580.517.

For the second medium-sized set, all runs discovered better solutions than the initial feasible solution. The MIP gap improved from 0,134 to on average 0,091, in the best case to 0,040 and in the worst case to 0,101. However, only 6 out of the 10 runs were faster than the performance of the Gurobi solver. This indicates that Gurobi outpaced The Algorithm in finding the same objective function in 4 out of the 10 runs. On average outpaced The algorithm Gurobi with 6,6 seconds, the largest time difference observed was 72 seconds and the smallest was -27 seconds. The solution 83.086.000 is found four times and 81.856.834 twice.

For the third medium-sized set, all runs discovered better solutions than the best feasible solution it starts with. The MIP gap improved from 0,219 to on average 0,113, in the best case 0,106 and the worst case 0,133. The algorithm was in 7 out of the 10 runs faster than Gurobi, 2 times slower, and once took the same amount of seconds. The highest time difference was 24 seconds, the lowest was -30 seconds, and on average 5,1 seconds. Both 97.116.618 and 94.119.356 are found twice in the ten runs.

For the fourth medium-sized set, all runs discovered better solutions than the initial feasible solution. The MIP gap improved from 0,568 to on average 0,276, in the best case 0,259 and in the worst case 0,320. The Algorithm outperformed Gurobi in every run with an average of 27,6 seconds. The highest time difference was 48 seconds, and the lowest 24 seconds. Solutions 111.812.370, 112.766.206, 119.535.066, and 109.721.707 are found twice.

For the last medium-sized set, all runs discovered better solutions than the initial feasible solution. The MIP gap improved from 0,204 to on average 0,120, in the best case 0,063 and in the worst case 0,200. The Algorithm outperformed Gurobi in every run with an average of 66,1 seconds. The highest time difference was 114 seconds, and the lowest 4 seconds. Solution

77.013.508 is found five times, and solution 87.394.255 is found twice.

For all five medium-sized data sets, the variable w_{kqr} which is responsible for transport between facilities has a positive value. This implies that the model makes use of the internal transportation of commodities between facilities to fulfill the demand of clients.

Table 3: Results of 50 runs of The Algorithm on 5 different medium sets.

Run	Best Feasible solution	MIP Gap	Creating Feasible solutions (sec)	Best Solution The Algorithm	New MIP Gap	Computing Time Total (sec)	Gurobi Computing time (sec)	Faster
				Medium Set 1				
1	83.215.917	0,213	41	72.351.236	0,095	57	127	70
2	83.215.917	0,213	39	71.211.570	0,080	69	134	65
3	83.215.917	0,213	39	71.211.570	0,080	72	134	62
4	83.215.917	0,213	38	71.211.570	0,080	58	134	76
5	83.215.917	0,213	39	71.211.570	0,080	77	134	57
6	83.215.917	0,213	39	71.809.429	0,088	93	127	34
7	83.215.917	0,213	40	71.580.517	0,085	74	127	53
8	83.215.917	0,213	38	71.211.570	0,080	60	127	67
9	83.215.917	0,213	38	71.211.570	0,080	79	127	48
10	83.215.917	0,213	39	71.580.517	0,085	61	127	66
Average	83.215.917	0,213	38,9	71.459.112	0,082	70,0	129,8	59,8
				Medium Set 2				
1	86.191.985	0,134	30	82.526.267	0,095	63	84	21
2	86.191.985	0,134	35	83.086.000	0,101	92	84	-8
3	86.191.985	0,134	37	77.759.269	0,040	108	180	72
4	86.191.985	0,134	37	83.086.000	0,101	96	84	-12
5	86.191.985	0,134	37	83.086.000	0,101	80	84	4
6	86.191.985	0,134	37	82.041.010	0,090	96	84	-12
7	86.191.985	0,134	36	83.086.000	0,101	111	84	-27
8	86.191.985	0,134	36	81.856.834	0,088	81	93	12
9	86.191.985	0,134	37	83.086.000	0,101	82	84	2
10	86.191.985	0,134	38	81.856.834	0,088	79	93	14
Average	86.191.985	0,134	35,9	82.147.021	0,091	88,8	95,4	6,6
				Medium Set 3				
1	107.806.941	0,219	38	97.116.618	0,133	112	82	-30
2	107.806.941	0,219	36	95.213.602	0,116	94	94	0
3	107.806.941	0,219	38	94.119.356	0,106	77	94	17
4	107.806.941	0,219	37	97.025.553	0,132	75	82	7
5	107.806.941	0,219	36	93.675.691	0,101	94	105	11
6	107.806.941	0,219	37	94.119.356	0,106	70	94	24
7	107.806.941	0,219	36	94.827.780	0,112	97	94	-3
8	107.806.941	0,219	37	93.675.691	0,101	93	105	12
9	107.806.941	0,219	37	93.167.851	0,096	95	105	10
10	107.806.941	0,219	37	97.116.618	0,133	80	82	2
Average	107.806.941	0,219	36,8	95.005.811	0,113	88,6	93,7	5,1
				Medium Set 4				
1	188.087.633	0,568	12	111.812.370	0,273	47	75	28
2	188.087.633	0,568	9	111.812.370	0,273	30	75	45
3	188.087.633	0,568	11	119.535.066	0,320	38	66	28
4	188.087.633	0,568	12	112.766.206	0,279	56	66	10
5	188.087.633	0,568	12	116.363.116	0,301	62	66	4
6	188.087.633	0,568	11	119.535.066	0,320	37	66	29
7	188.087.633	0,568	13	112.899.232	0,280	37	66	29
8	188.087.633	0,568	12	101.615.913	0,200	46	94	48
9	188.087.633	0,568	11	109.721.706	0,259	61	94	33
10	188.087.633	0,568	12	109.721.706	0,259	72	94	22
Average	188.087.633	0,568	11,5	112.318.047	0,276	48,6	76,2	27,6
				Medium Set 5				
1	110.438.106	0,204	52	77.013.508	0,063	121	185	64
2	110.438.106	0,204	44	90.143.741	0,200	76	128	52
3	110.438.106	0,204	41	87.394.255	0,174	100	145	45
4	110.438.106	0,204	40	84.469.545	0,146	141	145	4
5	110.438.106	0,204	41	77.013.508	0,063	71	185	114
6	110.438.106	0,204	41	77.013.508	0,063	71	185	114
7	110.438.106	0,204	41	77.013.508	0,063	124	185	61
8	110.438.106	0,204	39	87.724.657	0,177	86	145	59
9	110.438.106	0,204	42	89.106.006	0,190	94	145	51
10	110.438.106	0,204	40	77.013.508	0,063	88	185	97
Average	110.438.106	0,204	42,0	82.390.575	0,120	97,2	163,3	66,1

4.2 Large Data Sets

For the first and third large-data set, The Algorithm failed to find any improved solutions compared to the best initial feasible solution generated during the first step of The Algorithm. This is evident from the average MIP Gap values, with the new MIP gap being higher than the initial MIP gap. As a result, The Algorithm is always slower than Gurobi since the initial feasible solution is found via Gurobi.

The second large data set shows a consistent improvement for the results compared to the initial best feasible solution. The MIP gap improves from 0,377 to on average 0,233, in the best case 0,203, and worst case 0,253. Every run is also faster compared to the standard Gurobi solver. with an average of 144,2 seconds, in the highest case 267 seconds and the lowest 13 seconds.

For the fourth large-data set, The Algorithm was able to find better solutions in 8 out of 10 runs. With an average MIP gap improvement from 0,158 to 0,130, in the best case to 0,108, and the worst case to that of the initial feasible solution 0,158. In the 8 runs which The Algorihm found better solutions it was also faster, with an average of 80,2 second, in the highest case 248 seconds and the lowest -132.

For the last large-data set, The Algorithm was only able to find a better solution in 1 out of 10 runs. The improvement in the MIP gap for this run is from 0,157 to 0,153. In all other runs the MIP gap decreased. On average a decrease from 0,157 to 0,292 and in the worst case from to 0,432. In all 10 runs The algorithm was slower than Gurobi with an average of 162,9 seconds slower, in the worst case 255 seconds slower and best case 66 seconds slower.

For both the first,third and fifth large-data set, the variable w_{kqr} takes a value. This is not the case for data set two and four, meaning these data sets do not have any internal transport between facilities.

Table 4: Results of 50 runs of The Algorithm on 5 different large sets.

Run	Best Feasible solution	MIP Gap	Creating Feasible solutions (sec)	Objective Function The Algorithm	New MIP Gap	Computing Time Total (sec)	GuRoBi Computing time (sec)	Faster
				Large Set 1				
1	133.341.381	0,123	337	163.202.420	0,283	390	337	-53
2	133.341.381	0,123	358	160.038.078	0,269	528	358	-171
3	133.341.381	0,123	495	160.502.026	0,271	646	495	-151
4	133.341.381	0,123	495	155.983.691	0,250	630	495	-136
5	133.341.381	0,123	330	155.983.691	0,250	410	330	-80
6	133.341.381	0,123	368	158.377.287	0,262	555	368	-187
7	133.341.381	0,123	365	162.142.774	0,279	427	365	-62
8	133.341.381	0,123	316	152.918.513	0,235	451	316	-135
9	133.341.381	0,123	419	156.489.144	0,253	629	419	-209
10	133.341.381	0,123	497	155.894.923	0,250	571	497	-75
Average	133.341.381	0,123	397,9	158.153.254	0,260	523,7	397,9	-125,9
				Large Set 2				
1	227.795.709	0,274	243	203.621.351	0,188	527	624	97
2	227.795.709	0,274	240	207.267.207	0,202	379	598	219
3	227.795.709	0,274	241	205.111.502	0,194	600	613	13
4	227.795.709	0,274	237	204.311.502	0,191	590	613	23
5	227.795.709	0,274	240	205.021.272	0,193	386	613	227
6	227.795.709	0,274	248	204.311.502	0,191	510	613	103
7	227.795.709	0,274	237	204.311.502	0,191	466	613	147
8	227.795.709	0,274	238	201.960.387	0,181	484	630	146
9	227.795.709	0,274	237	205.111.502	0,194	346	613	267
10	227.795.709	0,274	239	199.002.785	0,169	506	705	199
Average	227.795.709	0,274	239,5	204.003.051	0,189	479	623,5	144,2
				Large Set 3				
1	204.454.996	0,144	317	279.772.651	0,374	775	317	-457
2	204.454.996	0,144	354	270.018.868	0,352	1165	354	-811
3	204.454.996	0,144	356	255.343.965	0,315	1010	356	-655
4	204.454.996	0,144	354	227.219.018	0,230	1241	354	-887
5	204.454.996	0,144	355	277.275.585	0,369	917	355	-561
6	204.454.996	0,144	436	219.982.862	0,204	846	436	-410
7	204.454.996	0,144	328	271.992.183	0,357	1296	328	-968
8	204.454.996	0,144	335	217.409.089	0,195	1258	335	-923
9	204.454.996	0,144	411	233.947.811	0,252	816	411	-406
10	204.454.996	0,144	325	233.476.065	0,250	714	325	-389
Average	204.454.996	0,144	357,0	248.643.810	0,290	1003,8	357,1	-646,7
				Large Set 4				
1	211.262.463	0,158	301	200.762.011	0,114	580	625	45
2	211.262.463	0,158	277	199.587.416	0,108	567	815	248
3	211.262.463	0,158	301	203.032.806	0,124	461	600	139
4	211.262.463	0,158	251	207.077.053	0,141	443	625	182
5	211.262.463	0,158	265	206.910.593	0,140	521	598	77
6	211.262.463	0,158	320	211.262.463	0,158	452	320	-132
7	211.262.463	0,158	280	199.587.416	0,108	633	815	182
8	211.262.463	0,158	275	204.870.251	0,131	402	275	-127
9	211.262.463	0,158	273	211.262.463	0,158	457	630	173
10	211.262.463	0,158	255	200.762.011	0,114	610	625	15
Average	211.262.463	0,158	279,80	204.511.448	0,130	512,6	592,8	80,2
				Large Set 5				
1	211.744.789	0,157	141	217.739.361	0,180	330	141	-190
2	211.744.789	0,157	121	314.142.255	0,432	187	121	-66
3	211.744.789	0,157	121	217.114.024	0,178	259	121	-138
4	211.744.789	0,157	128	328.250.208	0,456	303	128	-175
5	211.744.789	0,157	122	217.739.361	0,180	242	122	-120
6	211.744.789	0,157	121	257.422.891	0,306	271	121	-150
7	211.744.789	0,157	121	210.796.658	0,153	380	125	-255
8	211.744.789	0,157	122	286.111.757	0,376	316	122	-194
9	211.744.789	0,157	153	240.454.098	0,257	352	153	-198
10	211.744.789	0,157	121	301.522.914	0,408	263	121	-141
Average	211.744.789	0,157	127,2	259.129.353	0,292	290,4	127,6	-162,9

5 Discussion

The findings from this study reveal a mixed pattern of effectiveness when evaluating The Algorithm proposed in this thesis against the Gurobi solver. Notably, within the medium-sized data sets, The Algorithm consistently shows better performance in terms of speed compared to Gurobi. Nevertheless, it is important to note that a significant portion of the overall run time is consumed by the generation of feasible solutions, relative to the other stages of The Algorithm. The remaining three stages of the algorithm are relatively short, indicating that these phases often discover better feasible solutions in the early iterations, yet The Algorithm struggles to enhance these solutions further and thus terminating fast. Furthermore, it frequently converges to the same solution. This can likely be attributed to the fact that a substantial proportion of the possible solutions generated by the genetic algorithm are infeasible, leading to the same variables being passed to subsequent stages. The latter steps, including the locating and allocating of facilities, as well as establishing the product flows (which is done by Gurobi) always follow the same path for the same variables, leading to the same solutions.

One more factor that can lead to the fast termination of The Algorithm is the fact that solutions with a lot of production lines and open facilities are faster feasible, due to the higher capacities to fulfill demand of the clients. However these solution can be sub-optimal by introducing inefficiencies in the form of unnecessary facility and production lines opening. Consequently, this can lead to The Algorithm terminating prematurely due to the lack of improvements. This scenario is most likely the case for the large data as they contain a substantially larger number of variables.

The quality of the objective function found for the medium set is quite stable floating around a MIP gap of 0,1. This means there is still some room to improve the solution, but as described above it is not able to find better solutions. However, the found solution after termination could also be used as a lower bound for an exact method such as branch-and-bound (Conforti et al., 2014). This would especially fit well since it is quite fast in finding these solutions for the medium sets compared to Gurobi. It is important to note that a speed comparison between Gurobi and The Algorithm is in favour of Gurobi. Considering that Gurobi uses multiple cores (parallelism) to compute the solutions while python itself is not able to do this, even though this would be very fitting for a genetic algorithm (Cavuoti et al., 2013). Additionally, Gurobi is programmed in C(Gurobi Optimization, 2023) while The Algorithm is programmed in phyton. Gurobi is automatically faster because C is on average a factor 30 faster than phyton (Nanz & Furia, 2015).

The second and fourth large data were the only large data sets where The Algorithm showed consistent improvements on both the objective value and computing time. These data sets did not contain any internal transport in their solutions, simplifying the model to the very similar single-source capacitated facility location problem. This most often solved via Lagrangian relaxation for the initial feasible solution, followed by variable neighborhood search or tabu search (Ulukan & Demircioğlu, 2015). However, both the variable neighborhood search and tabu search require a well-defined neighborhood structure (Claassen et al., 2006; Duarte et al., 2018). This requirement could potentially pose a challenge for the model proposed in this thesis due to the capacities introduced by facility sizes and production line capacities, which can lead to an infeasible solution when only a single variable is changed.

An essential part of meta heuristics is the parameter setting (Claassen et al., 2006). The Algorithm contains several important parameters including amount of feasible solutions created, mutation rate, termination criteria, time of filling in production flows. During this thesis, limited time hindered the complete optimization of these parameters, potentially leading to inefficiencies in The Algorithm's performance.

6 Conclusion

In conclusion, this thesis has presented an approach to model and solve the multi-commodity capacitated discrete location-allocation model with production lines. Together with a custom made algorithm, referred to as "The Algorithm", that is compared to the commercially available solver Gurobi. The study encompassed a total of 100 runs, consisting of 50 runs for 5 medium-sized data sets and another 50 runs for 5 large-sized data sets. The following research questions have been made and are answered.

Research question 1 , How is the multi-commodity capacitated discrete location allocation model precisely formulated? In this thesis, a literature review is performed regarding the different modeling elements and variants regarding the commonly used elements. In addition, new elements are formulated for the model. These additions are: fixed cost for production lines and a capacity constraint on facility size; a flow constraint ensuring every facility may only deliver to a client if enough commodities are present; internal transport cost, including a constraint for truck delivery. The complete mathematical formulation of the model can be seen in subsection 3.1.

Research question 2 , What are the optimization techniques and algorithms that can be applied to solve the formulated model? For solving such a model different techniques were researched and presented in the literature review. A distinction can be made between exact solvers and heuristics. Based on the found algorithms, a custom made algorithm is created, referred to as "The Algorithm". The Algorithm consists of four steps: 1, creating feasible solutions; 2, genetic algorithm; 3, allocating facilities and locations; and 4, product flows. The Algorithm loops from step 4 back to step 2 until the termination criteria of no improvements is reached. The Algorithm can be seen in subsection 3.2.

Research question 3 , How does the chosen algorithm perform across diverse data sets in comparison to a commercially available solver? For the five medium sized data sets, The Algorithm consistently outperformed Gurobi in terms of speed, with an average improvement of 59,8; 6,6; 5,1; 27,6; and 66,1 seconds, respectively. This speed advantage was accompanied by improvements in the MIP gap, reducing it from an initial 0,213; 0,134; 0,219; 0,568; 0,204 to an average of 0,082; 0,091; 0,113; 0,276; and 0,120, respectively. However, the algorithm often converged to the same solution in a local

optima and terminated fast after not finding any better feasible solutions.

On the other hand, the performance of The Algorithm on large sized data sets showed mixed results. For the first, third, and fifth data sets, The Algorithm failed to improve upon the initial feasible solutions created by Gurobi, automatically making it slower than Gurobi. The second and fourth large data sets showed consistent improvements in both MIP gap and computing time, with an average time difference of 144,2 and 80,2 seconds, respectively, and an improved MIP gap from the initial 0,274 and 0,158 to an average of 0,189 and 0,130, respectively. However, these data sets did not use internal transport to fulfill the demand for the clients, making the model easier to solve.

The findings suggest that The Algorithm has potential, particularly for medium-sized instances, where it can serve as a rapid, tight upper-bound solution for other exact algorithms such as branch-and-bound.

Limitations and future research revolve mainly around the genetic algorithm and parameter settings. The genetic algorithm is used because a neighborhood structure is not needed. Models similar to the researched model use heuristics like tabu search and variable neighborhood search built on this neighborhood structure. It would be interesting to research potential neighborhood structures for this model and implement these other algorithms proven to work on similar models. Research on the creation of the initial feasible solution could also improve the model since this takes a large portion of the total time of The Algorithm. Possibilities could be complex algorithms like Lagrangian relaxation, which was too complex and time-consuming to implement in this thesis. As well, research on parameter optimization took too much time to fully investigate.

References

Aarts, E., Korst, J., & Michiels, W. (2005). Simulated annealing. In E. K. Burke & G. Kendall (Eds.), *Search methodologies: Introductory tutorials in optimization and decision support techniques* (pp. 187–210). Springer US. https://doi.org/10.1007/0-387-28356-0_7

AH. (2023). *Location alber heijn's*. https://www.ah.nl/winkels

Akyüz, M. H., Öncan, T., & Altınel, İ. K. (2012). Efficient approximate solution methods for the multi-commodity capacitated multi-facility weber problem. *Computers & operations research*, *39*(2), 225–237.

Anand, R., Aggarwal, D., & Kumar, V. (2017). A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems*, *20*(4), 623–635.

Aras, N., Altınel, İ. K., & Orbay, M. (2007). New heuristic methods for the capacitated multi-facility weber problem. *Naval Research Logistics (NRL)*, *54*(1), 21–32.

Bertsimas, D., Cory-Wright, R., & Pauphilet, J. (2021). A unified approach to mixed-integer optimization problems with logical constraints. *SIAM Journal on Optimization*, *31*(3), 2340–2367.

Brandeau, M. L., & Chiu, S. S. (1989). An overview of representative problems in location research. *Management Science*, *35*, 645–674.

Cavuoti, S., Garofalo, M., Brescia, M., Pescape', A., Longo, G., & Ventre, G. (2013). Genetic algorithm modeling with gpu parallel computing technology. *Neural Nets and Surroundings: 22nd Italian Workshop on Neural Nets, WIRN 2012, May 17-19, Vietri sul Mare, Salerno, Italy*, 29–39.

CBS. (2022). *Inwoners per gemeente*. https://www.cbs.nl/nl-nl/visualisaties/dashboard-bevolking/regionaal/inwoners

Claassen, G., Hendriks, T., & Hendrix, E. (2006). 10. heuristcs. In *Decision science: Theory and applications* (pp. 227–255). Wageningen Academic Publishers.

Conforti, M., Cornuéjols, G., & Zambelli, G. (2014). Integer programming models. In *Integer programming* (pp. 45–84). Springer International Publishing. https://doi.org/10.1007/978-3-319-11008-0_2

Drezner, Z., & Hamacher, H. W. (2004). *Facility location: Applications and theory*. Springer Science & Business Media.

Duarte, A., Laguna, M., & Martí, R. (2018). Tabu search. In *Metaheuristics for business analytics: A decision modeling approach* (pp. 85–103). Springer International Publishing. https://doi.org/10.1007/978-3-319-68119-1_4

Fernández, J., Fernández, P., & Pelegrın, B. (2000). A continuous location model for siting a non-noxious undesirable facility within a geographical region. *European Journal of Operational Research*, *121*(2), 259–274.

Gurobi Optimization, L. (2023). Gurobi optimizer reference manual. https://www.gurobi.com

Hanafi, S., & Todosijević, R. (2017). Mathematical programming based heuristics for the 0–1 mip: A survey. *Journal of Heuristics*, *23*(4), 165–206.

Hansen, P., & Mladenović, N. (2005). Variable neighborhood search. In E. K. Burke & G. Kendall (Eds.), *Search methodologies: Introductory tutorials in optimization and decision support techniques* (pp. 211–238). Springer US. https://doi.org/10.1007/0-387-28356-0_8

Karkazis, J., & Boffey, T. (1981). The multi-commodity facilities location problem. *Journal of the Operational Research Society*, *32*(9), 803–814.

Klose, A., & Drexl, A. (2005). Facility location models for distribution system design. *European journal of operational research*, *162*(1), 4–29.

Koza, J. R., & Poli, R. (2005). Genetic programming. In E. K. Burke & G. Kendall (Eds.), *Search methodologies: Introductory tutorials in optimization and decision support techniques* (pp. 127–164). Springer US. https://doi.org/10.1007/0-387-28356-0_5

Luis, M., Salhi, S., & Nagy, G. (2015). A constructive method and a guided hybrid grasp for the capacitated multi-source weber problem in the presence of fixed cost. *Journal of Algorithms & Computational Technology*, *9*(2), 215–232.

Megiddo, N., & Supowit, K. J. (1984). On the complexity of some common geometric location problems. *SIAM journal on computing*, *13*(1), 182–196.

Melo, M. T., Nickel, S., & Saldanha-Da-Gama, F. (2009). Facility location and supply chain management–a review. *European journal of operational research*, *196*(2), 401–412.

Nanz, S., & Furia, C. A. (2015). A comparative study of programming languages in rosetta code. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, *1*, 778–788. https://doi.org/10.1109/ICSE.2015.90

Naoto, K. (2015). A combined capacity scaling and local branching approach for capacitated multi-commodity network design problem. *Far East Journal of Applied Mathematics*, *92*, 1–30. https://doi.org/10.17654/FJAMJul2015_001_030

Neebe, A. W., & Khumawala, B. M. (1981). An improved algorithm for the multi-commodity location problem. *Journal of the Operational Research Society*, *32*(2), 143–149.

Owen, S. H., & Daskin, M. S. (1998). Strategic facility location: A review. *European journal of operational research*, *111*(3), 423–447.

Pirkul, H., & Jayaraman, V. (1998). A multi-commodity, multi-plant, capacitated facility location problem: Formulation and efficient heuristic solution. *Computers & Operations Research*, *25*(10), 869–878.

ReVelle, C. S., & Eiselt, H. A. (2005). Location analysis: A synthesis and survey. *European journal of operational research*, *165*(1), 1–19.

Sadjady, H., & Davoudpour, H. (2012). Two-echelon, multi-commodity supply chain network design with mode selection, lead-times and inventory costs. *Computers & Operations Research*, *39*(7), 1345–1354.

Salhi, S., & Atkinson, R. (1995). Subdrop: A modified drop heuristic for location problems. *Location Science*, *3*(4), 267–273. https://doi.org/ https://doi.org/10.1016/0966-8349(96)00003-4

Scott, A. J. (1971). Dynamic location-allocation systems: Some basic planning strategies. *Environment and planning A*, *3*(1), 73–82.

Sherali, H. D., Ramachandran, S., & Kim, S.-i. (1994). A localization and reformulation discrete programming approach for the rectilinear distance location-allocation problem. *Discrete Applied Mathematics*, *49*(1-3), 357–378.

Ulukan, Z., & Demircioğlu, E. (2015). A survey of discrete facility location problems. *International Journal of Industrial and Manufacturing Engineering*, *9*(7), 2487–2492.

Vincent, F. Y., Jewpanya, P., & Redi, A. P. (2016). Open vehicle routing problem with cross-docking. *Computers & Industrial Engineering*, *94*, 6–17.

Warszawski, A. (1973). Multi-dimensional location problems. *Journal of the Operational Research Society*, *24*(2), 165–179.

Weber, A. (1909). *Über den standort der industrien. 1. teil: Reine theorie des standortes*. Tübingen, Germany.

Yu, V. F., & Lin, S.-Y. (2016). Solving the location-routing problem with simultaneous pickup and delivery by simulated annealing. *International Journal of Production Research*, *54*(2), 526–549.

Zhang, J., Liu, C., Li, X., Zhen, H.-L., Yuan, M., Li, Y., & Yan, J. (2023). A survey for solving mixed integer programming via machine learning. *Neurocomputing*, *519*, 205–217.

A Computer specifications

Processor: Intel(R) Core(TM) i5-8365U CPU @ 1.60GHz 1.90 GHz
Installed RAM: 8,00 GB (7,81 GB usable)
System type: 64-bit operating system, x64-based processor
Edition Windows: 10 Enterprise
Version: 22H2
OS build: 19045.3324
Gurobi: 10.0.2
Python: 3.8.10
Spyder IDE: 5.4.4

B GitHub

Link to GitHub or the URL: https://github.com/porkkobroodje/Thesis-Rick-van-Willigen