ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics

Bachelor Thesis Econometrics and Operations Research 2019-2020
Quantitative Logistics

Local Search Heuristics for the Mobile Facility Location Problem

Name student: Aimée Stuit
Student ID Number: 456737es

Supervisor: L. van Rijn
Second assessor: dr. T.A.B. Dollevoet

Date final version: 5 July 2020

Abstract

In this paper the Mobile Facility Location Problem (MFLP) is solved. This problem aims to find destinations for clients and facilities such that every client shares his/her destination with at least one facility while minimizing the total weighted distance that clients and facilities have to travel. This problem applies to many everyday situations such as the relocation of firefighters and ambulances, disaster relief logistics and supply chain problems. The MFLP is solved by means of an IP formulation and two local search heuristics given by Halper, Raghavan, and Sahin (2015). These local search heuristics search a neighborhood of possible solutions by swapping the destination locations. Additionally, we investigated schematic ways of how these swaps can be performed. Furthermore, a modification to these local search heuristics is explored in which we switch between searching the neighborhood of solutions by swapping one location and two locations. We find that the local search heuristic based on selecting the first improvement in the neighborhood search combined with a schematic way of swapping results in tight optimality gaps. Moreover, these optimality gaps are further improved by means of the modification to the local search heuristic.

Contents

1 Introduction

The Facility Location Problem (FLP) concerns the placement of facilities such that the total transportation time with associated costs is minimized. A variant of the FLP is the Mobile Facility Location Problem (MFLP) where facilities and clients are initially located at certain locations and are moving to a certain destination. The objective of this problem is to find a destination location for all clients and facilities, such that every client shares its destination with at least one facility while minimizing the total weighted distance travelled by every facility and client. The MFLP can be applied to many practical situations. For example, Demaine et al. (2009) discuss an application of the MFLP related to firefighters. Every firefighter should always be in contact with at least one other firefighter, so they should move to a location in such a way that they share the same destination, which in this case is a communication connection point.

This paper aims to answer the following question: *"How can we solve the MFLP which decides on the destination locations for clients and facilities such that they share a destination node while minimizing the total costs of moving the clients and facilities?"*. An Integer Programming Problem (IP) formulation, given in Halper, Raghavan, and Sahin (2015), will be used to solve the MFLP exactly. Furthermore, we will show that this IP formulation can be decomposed into the facility allocation problem and the client allocation problem where the destinations of all facilities and all clients are determined separately.

The IP formulation seems to be quite complex and NP-hard. Therefore Halper, Raghavan, and Sahin (2015) developed local search heuristics which solve the MFLP within polynomial time. Two of these local search heuristics will be studied in this research, which leads us to our second research question: *"Can we solve the MFLP by means of local search heuristics while achieving tight optimality gaps?"*. We refer to *optimality gaps* as the difference between the heuristic solution and the exact solution. The heuristics are searching for improvements in a neighborhood of solutions. These neighborhoods of solutions are created by means of *swapping*, which refers to replacing a set of locations from the set of destination locations with a set of locations from the remaining locations. Each swap thus creates a new set of destination locations, each associated with a unique solution. If this solution is better than the current best solution found we found an improvement. The first heuristic searches for the first improvement in the neighborhood of solutions and the second heuristic searches for the best improvement in the neighborhood of solutions.

Because of randomness in the local search heuristics we aim for stable results by trying to find a schematic way of deciding on which locations to swap. This leads to another research question: *"Can we develop a schematic way of swapping while maintaining or improving the performance of the heuristics?"*.

Furthermore, a modification will be applied to the most optimal heuristic. This modification starts with swapping one location and under a certain criterion it swaps two locations at once. The goal here is to prevent the heuristic to end up in a local optimum too quickly, which

leads to our next research question: *"Can we prevent the heuristics to end in an local optimum?"*.

Our first key finding is that we can solve the MFLP exactly by means of the IP formulation. Secondly, we can solve the MFLP approximately with tight optimality gaps. Thirdly, we show that randomly changing the set of destinations results in inconsistent results for the heuristics, which motivates to find a schematic way of swapping the destination locations and we find a scheme which improves the performance of the heuristics. The final key finding is that using the modification will prevent many instances to end up in a local optimum too quickly.

The paper will be organized as follows. In Section 2 relevant applications of the MFLP and necessary literature to underpin the study in this paper are represented. The problem will be described in Section 3. Next, the methodology will be explained in Section 4, which consists of the IP formulation, its decomposition, the local search algorithms, the different schemes for the swaps and the modification on the heuristic. The used data will be discussed in Section 5. The results of the different approaches and applications will be described in Section 6. Section 7 will contain the conclusion of this research. We will end this paper with a discussion in Section 8.

2 Literature

This section will discuss a number of applications of the MFLP. The origin of the MFLP can be found in the paper of Demaine et al. (2009). This paper refers to the relocation of firefighters as a movement problem. Moreover, they look at this problem as a MinMax problem in which the maximum movement is minimized in order to attain a connection. The paper gives more variations on the MinMax problem, such as the problem of minimizing the total movement and minimizing the number of moving objects. The paper also discusses the problem in which nearby pairs have to be formed such that they can exchange information. The MFLP can thus be seen as a MinMax problem which minimizes the total weighted distance while providing perfect matching for each pair of destinations and clients and each pair of destinations and facilities.

Yet another example is given in the paper of Gendreau, Laporte, and Semet (2001) in which constantly moving ambulances need to be relocated. In this application it is necessary that each client should be connected to an ambulance in such a way that the total time and distance travelled from the ambulance to the client is minimized and such that all ambulances achieve a good coverage level. Another problem to solve here is to determine which ambulance should be connected to which client in order to minimize the waiting time for each client.

Another application of the MFLP deals with disaster relief logistics (Halper, Raghavan, and Sahin (2015)). Imagine there is a natural disaster and an aid organisation wishes to distribute supplies to the areas in need. In this case distribution points should be determined for every warehouse. Next, the supplies should be transported from the distribution points to the aid stations. Here the aim is to provide the supplies at minimum transport costs.

Furthermore, the MFLP can also be applied to supply chain problems. This application is

used in the paper of Friggstad and Salavatipour (2011). The decision which arises here is to decide whether a plant will facilitate multiple retail stores directly or to build a distribution center for each plant in order to distribute those supplies. These distribution centers may be located closer to a certain set of retail stores, which results in transport cost reductions. This example can be seen as an MFLP in which they aim to choose the new locations of facilities and clients such that in the end each client has the same destination as some facility, while the total costs of relocation are minimized.

The paper of Friggstad and Salavatipour (2011) looks at the *k-median* problem and the *k-facility* location problem as problems related to the MFLP. The *k-median* problem is characterized by the fact that it has to open at most k facilities with no fixed costs to open a facility while the facilities serve the demand of all clients. The MFLP and the *k-median* problem have in common that they have a fixed amount of facilities to serve the clients. The difference is that for the MFLP holds that not only the costs of serving clients need to be minimized, but also the costs of moving facilities. According to Arya et al. (2004) the best way to design an approximation algorithm for *k-median* problems is to use local search heuristics. This motivated Friggstad and Salavatipour (2011) to use this local search algorithm to develop an algorithm for the MFLP. Another similar problem is the *k-facility* location problem. This problem is characterized by the fact that at most k facilities can be opened and that there is a price for opening a facility. The best algorithm for *k-facility* location problems seems to be a local search algorithm (Zhang (2007)). However, Friggstad and Salavatipour (2011) noticed that applying the same local search to a problem where facilities are mobile, results in a too large locality gap, so they propose local search operations which produce a large neighborhood of solutions. Halper, Raghavan, and Sahin (2015) notice that it is not possible to find the optimal solution for the MFLP with this set of local search operations, so they introduce other local search heuristics inspired by this heuristic.

Previous work on facility location problems used the Hungarian Algorithm for solving the assignment problem. In this research we also use this algorithm in to solve the facility assignment problem. The Hungarian Algorithm has its origins from Kuhn (1955) in which a General Assignment Problem (GAP) of matching individuals to jobs, for which they are qualifying for, is solved. To solve this GAP the Hungarian Algorithm is defined and this method has been used for many variations of the MFLP thereafter. The paper of Berman and LeBlanc (1984), for example, also deals with relocation of mobile facilities and uses the Hungarian Algorithm to look for the least cost relocation of all facilities to other locations.

3 Problem Description

The MFLP is represented on a graph $G(V, E)$. In this graph V denotes the set of vertices and E denotes the set of edges. The set of vertices consists of a subset of client vertices $C \subseteq V$ and a subset of facility vertices $F \subseteq V$. The per unit distance cost of each client $i \in C$ is denoted by u_i, to which we will also refer to as the weight of a client. The per unit distance cost of each facility $j \in F$ is denoted as $w_j > 0$, to which we will also refer to as the weight of

a facility. A vertex of destination locations for each client and facility is given by $v(i)$ and $v(j)$ respectively. For all $i \in C$ it must hold that $v(i) = v(j)$ for some $j \in F$. This means that each client should have a destination location which is the destination of at least one facility such that the client's demand can be satisfied. The objective of this problem is to minimize the total travelling distance between facilities and destinations, and between clients and destinations and the associated costs which are proportional to the distance.

The following assumptions are made for the MFLP. Firstly, clients and facilities are allowed to have the same vertex as origin and destination so that they stay at the same location. Of course this only holds if every client shares this destination with at least one facility. We also allow for clients and facilities to have the same origin which would imply that $C \cap F \neq \emptyset$. Furthermore, it is assumed that two clients that have the same initial vertex also have the same destination vertex, because it is most cost-efficient that they will both be served by the destination location which is the closest, otherwise one of the clients could have been served for less costs. Lastly, it is assumed that two facilities will not start at the same vertex. To conclude, a client and a facility can both be uniquely defined by their original vertex.

If we define $d_{jv(j)}$ as the minimal distance between vertex j and its destination vector $v(j)$ and $d_{iv(i)}$ as the minimal distance between vertex i and its destination vector $v(i)$ the objective of the MFLP can be formulated as the following:

$$\min \qquad \sum_{j \in F} w_j d_{jv(j)} + \sum_{i \in C} u_i d_{iv(i)} \tag{1}$$

4 Methodology

In this section the methods used to solve the MFLP will be explained. Firstly, the IP formulation will be given. Secondly, this formulation will be decomposed into a facility assignment problem and a client assignment problem. Next, the local search algorithms from the paper of Halper, Raghavan, and Sahin (2015) will be described which is followed by schematic ways of choosing the locations to swap. Lastly, a modification to these heuristics will be made.

4.1 Integer Programming Formulation

The following variables are defined in order to formulate the Integer Programming problem:

$$x_{iv} = \begin{cases} 1, & \text{if the destination of client } i \in C \text{ is vertex } v \in V \\ 0, & \text{otherwise} \end{cases}$$

$$y_{jv} = \begin{cases} 1, & \text{if the destination of facility } j \in F \text{ is vertex } v \in V \\ 0, & \text{otherwise} \end{cases}$$

$$z_v = \begin{cases} 1, & \text{if vertex } v \in V \text{ is the destination of some facility} \\ 0, & \text{otherwise} \end{cases}$$

In order to solve the MFLP in which the destination of both facilities and clients has to be decided, the following formulation (2)-(7) from Halper, Raghavan, and Sahin (2015) is used in this paper:

$$\min \quad \sum_{j \in F} \sum_{v \in V} w_j d_{jv} y_{jv} + \sum_{i \in C} \sum_{v \in V} u_i d_{iv} x_{iv} \tag{2}$$

$$\text{s.t.} \quad \sum_{v \in V} x_{iv} = 1 \qquad \forall i \in C \tag{3}$$

$$\sum_{v} y_{jv} = 1 \qquad \forall j \in F \tag{4}$$

$$\sum_{j \in F} y_{jv} - z_v \leq 0 \qquad \forall v \in V \tag{5}$$

$$x_{iv} - z_v \leq 0 \qquad \forall i \in C, v \in V \tag{6}$$

$$y_{jv}, x_{iv}, z_v \in \{0, 1\} \qquad \forall i \in C, j \in F, v \in V \tag{7}$$

The objective (2) minimizes the sum of the total weighted distance travelled for every facility and the total weighted distance travelled for every client. Constraints (3) ensure that for every client $i \in C$ there is exactly one destination vertex. Constraints (4) make sure that every facility $j \in F$ has exactly one vertex as its destination. Constraints (5) guarantee that a facility cannot move to vertex $v \in V$ if this vertex is not a destination of a facility. This condition also ensures that if vertex $v \in V$ is the destination of a facility then there is exactly one facility that has this vertex as a destination. Constraints (6) ensure that if the vertex $v \in V$ is not a destination of any facility, there cannot be a client moving to this vertex. Constraints (7) define the domains of all variables as binary variables.

4.2 Decomposition of the Integer Programming Formulation

In order to make use of a local search heuristic for our problem, we define the subset $Z \subseteq V$ which has p vertices. This subset exists to ensure that each facility has a destination which is

in this subset Z. For our IP formulation this means that we can set $z_v = 1$ for all $v \in Z$ and $z_v = 0$ for all $v \in V \backslash Z$. So the number of vertices in subset Z is equal to:

$$p = \sum_{v \in V} z_v = |F| \tag{8}$$

Now we know all variables z_v for a given subset Z, we conclude that the IP Formulation can be decomposed into two sub-problems, which are given in Sections 4.2.1 and 4.2.2.

4.2.1 Facility Assignment Problem

For a given set Z we can thus set all variables $z_v = 1$ for all $v \in Z$ and $z_v = 0$ for all $v \in V \backslash Z$, because every facility must have a unique destination in Z. The Facility Assignment Problem (FA(Z)), which is given in (9)-(12), decides on a least cost bipartite matching between the facilities and the facility destinations. The exact formulation of the FA(Z) is as follows:

$$\min \quad \sum_{j \in F} \sum_{v \in Z} w_j d_{jv} y_{jv} \tag{9}$$

$$\text{s.t.} \quad \sum_{v \in Z} y_{jv} = 1 \qquad \forall j \in F \tag{10}$$

$$\sum_{j \in F} y_{jv} = 1 \qquad \forall v \in Z \tag{11}$$

$$y_{jv} \geq 0 \qquad \forall j \in F, v \in Z \tag{12}$$

The integrality of the y_{jv} variables is relaxed, because of the fact that the constraint matrix of this problem is total unimodular. Objective (9) ensures that the total weighted distances between facilities and their destinations is minimized. Constraints (10) and (11) ensure that for every facility holds that it is moved to one destination and that for every destination location there is exactly one facility that is moving there. Constraints (12) define the domain of the variables. Because of the totally unimodular constraint matrices, the FA(Z) can be solved by means of the Hungarian Algorithm which is given in Appendix A.

4.2.2 Client Assignment Problem

Like for the FA(Z) we know the values of z_v for a given set Z. Furthermore, we set $x_{iv} = 0$ for all $v \in V \backslash Z$, so that Constraints (6) are redundant for all $v \in V \backslash Z$. The Client Assignment Problem (CA(Z)), which is given in (13)-(15), decides on the destinations of the clients such that the distance between the clients and the destinations is minimized and is formulated as:

$$\min \quad \sum_{i \in C} \sum_{v \in Z} u_i d_{iv} x_{iv} \tag{13}$$

$$\text{s.t.} \quad \sum_{v \in Z} x_{iv} = 1 \qquad \forall i \in C \tag{14}$$

$$x_{iv} \geq 0 \qquad \forall i \in C, v \in Z \tag{15}$$

The constraint matrix is totally unimodular and therefore we relax the integrality of the x_{iv} variables. Objective (13) minimizes the total weighted distance between clients and their destinations. Constraints (14) ensure that every client is moved to exactly one destination location. Constraints (15) define the domain of the variables. As a result of the totally unimodular property the CA(Z) problem can be solved by sending each client $i \in C$ to its closest destination in Z.

It can be observed that each subset $Z \subseteq V$ can be uniquely solved by optimizing each individual subproblem so that we minimize the sum of both solution values. With this insight Halper, Raghavan, and Sahin (2015) designs a local search heuristic in order to find the subset Z that minimizes the sum of both subproblems. This heuristic will be explained in Section 4.3.

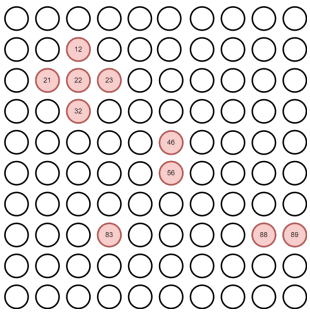
4.3 Local Search Algorithm

In this section the *n-OptSwap*, which is used in order to solve the MFLP, is explained. Each subset $Z \subseteq V$ of size p denotes a set of destination locations. The set of destination locations Z is initially set equal to the set of facility locations F. Every subset Z is associated with a unique solution which can be obtained by solving assignment problems $FA(Z)$ and $CA(Z)$. The local search heuristic aims to find the subset Z that minimizes $FA(Z) + CA(Z)$. For a set Z, we take a subset of k destinations in Z and replace them with a subset of k locations in $V \backslash Z$, $(1 \leq k \leq n)$, which we will refer to as *swapping*. These swaps create a neighborhood of solutions of size $\sum_{k=1}^{n} \binom{n}{k} \binom{|V|-|F|}{k}$. After each swap of destination locations, the $FA(Z)$ and $CA(Z)$ are solved again. If a swap results in a smaller solution value than the current best solution, we found an improvement.

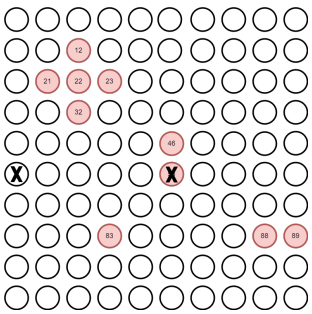
There are two versions of the *n-OptSwap* heuristic which are investigated in this research and given in Algorithm 1. The first one is called the *n-OptSwapFI* (FI) heuristic. This heuristic selects the first improvement that it finds during the local search. If it finds a first improvement, the current best solution is updated to this first improvement. For the following iterations the current best solution is used to find a next first improvement. A replacement is made by randomly swapping k locations from set Z with k locations from set $V \backslash Z$. The latter step is made until the first improvement is found. The solution of this heuristic is the current best solution after searching through the entire neighborhood. The stopping criteria for the case $n = 1$ is that if we do not find an improvement for $|F| \times (|V| - |F|)$ iterations, then the current best solution will be reported as the solution. As a default the k locations are chosen randomly. However, this method will be extended later by investigating a number of systematic schemes to choose the k locations to be swapped.

The second type of local search heuristic that is investigated is called the *n-OptSwapBI* (*BI*) heuristic. This version looks through the entire neighborhood to find the best improvement. So we again start with an initial set Z which is initially equal to set F. Then we replace k locations in Z with k locations in $V \backslash Z$ until we have a neighborhood of size $\sum_{k=1}^{n} \binom{n}{k}\binom{|V|-|F|}{k}$. Thereafter we search this neighborhood for the best improvement. This best improvement solution will be the current best solution and set Z will be updated. The stopping criteria for this heuristic is that if we cannot find another improvement for ($|F|$) iterations, the current best solution will be reported as the solution.

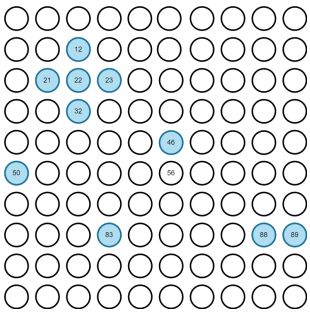
To demonstrate how a swap works, we take the first instance, which is given in the data and is explained in Section 5, as an example. This instance has ten facilities in F which are originally located on locations $\{12, 21, 22, 23, 32, 46, 56, 83, 88, 89\}$. Set Z will be initialized as set F. This set will then be associated with its unique solution $FA(Z) + CA(Z)$. Next, we will replace k destination locations in Z with k locations in $V \backslash Z$. In this paper the FI and BI heuristics will be implemented for $n = 1$, so one destination location will be replaced in every iteration. This will result in a neighborhood of solutions which we will search in order to find the optimal set of destination locations Z. In this example every one of the 10 locations $v \in Z$ can be replaced by one of the 90 other locations $v \in V \backslash Z$. So that in total we will have a neighborhood of $\sum_{k=1}^{n} \binom{n}{k}\binom{|V|-|F|}{k} = \binom{10}{1}\binom{90}{1} = 10 \times 90 = 900$ solutions. An illustration of how one swap in the 1-*OptSwap* heuristic works is given in Figure 1.



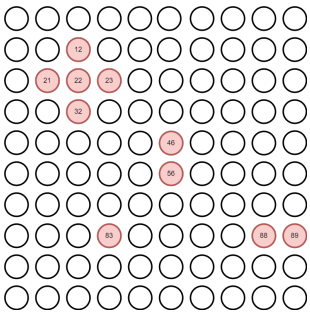
(a) Initialize Z=F and calculate $f(Z) = FA(Z)+CA(Z)$



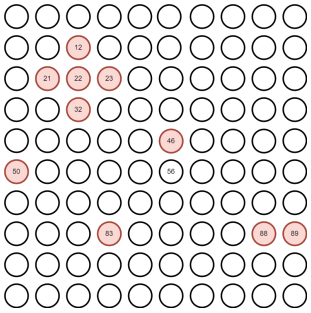
(b) Choose $k \in Z$ and $k \in V \backslash Z$ randomly



(c) Set Z' as the set after the swap and calculate $f(Z') = FA(Z')+CA(Z')$



(d) If $f(Z') \geq f(Z)$ go back to (b) with the original set Z



(e) If $f(Z') < f(Z)$ we set $Z = Z'$, go back to (b) with the new set Z

Figure 1: Example of one swap of the 1-*OptSwap* heuristic

Algorithm 1: The k-OptSwap Heuristic

Define set F as the set of initial facility locations which are given. Define set C as the set of initial client locations which are given.

Define set Z as the set of facility destinations. Initialize Z as $Z = F$.

Associate this set of facility destinations Z with its unique solution $FA(Z) + CA(Z)$ and add this set with associated solution to the neighborhood of solutions. Solve $FA(Z)$ with the Hungarian Algorithm given in Algorithm 2. Solve $CA(Z)$ by assigning each client $i \in C$ to the closest facility destination $v \in Z$.

Define s as the size of the neighborhood: $s = \sum_{k=1}^{n} \binom{n}{k}\binom{|V|-|F|}{k}$

for *n-OptSwapFI* **do**

  while *We can find an improvement in the neighborhood of swaps of size s* **do**

    Until we find an improvement: replace k $(1 \leq k \leq n)$ facility destination locations in Z (the current best solution) with k locations not in Z. Associate this new set of facility destinations Z' with its solution $FA(Z') + CA(Z')$ if $FA(Z') + CA(Z') < FA(Z) + CA(Z)$

    The current best solution will be updated to this first improvement solution $FA(Z') + CA(Z')$. Set $Z = Z'$

  The solution of this heuristic equals the current best solution

for *n-OptSwapBI* **do**

  while *We found less than* $|F|$ *best improvements* **do**

    while *We can find an improvement in the neighborhood of swaps of size s* **do**

      Replace the k $(1 \leq k \leq n)$ facility destination locations in Z (the current best solution) with k locations not in Z associate this new set of facility destinations Z' with its solution $FA(Z') + CA(Z')$ and update the current best solution to this improvement solution $FA(Z') + CA(Z')$. Set $Z = Z'$

    The best improvement will be the current best solution

  The solution of this heuristic equals the current best solution

4.4 Finding a schematic way of choosing the locations to swap

For the *n-OptSwap* heuristics k locations in Z are replaced by k locations in $V \backslash Z$. Now the question arises of how to decide on which k locations to swap. In Halper, Raghavan, and Sahin (2015) it is unclear how they selected the k swap locations. One could decide on choosing k locations randomly, which is applied as default in the previous section. However, these results are not expected to be consistent due to randomness in the algorithm. This gives motivation to perform investigation on how to select the k locations to swap.

The first scheme of choosing k locations is a fixed scheme. First, we define an ordered list of locations in Z and an ordered list of locations in $V \backslash Z$. To start with, the first k locations in Z are replaced by the first k locations in $V \backslash Z$. The first scenario is that a swap results in an improvement. If this is the case the lists of locations in Z and $V \backslash Z$ are updated and the next step is to replace the same k locations in Z with the next k locations in $V \backslash Z$. The second

scenario is that a swap results in a worse outcome. If this is the case the lists of locations in Z and $V\backslash Z$ will not be updated and the next k locations in Z will be replaced by the same k locations in $V\backslash Z$. If all possible swaps for a certain set of locations in Z are considered, we continue with replacing the next set of locations in Z. If all possible swaps are considered we start at the beginning of both lists again, as the lists are changed as a result of the swaps. In the continuation of this paper this scheme will go by the name $S1$.

For the second scheme we take the distances between certain locations into consideration. Three variations, on choosing which locations to swap, are investigated. For each of these variations the k locations in Z to replace are chosen sequentially, e.g. the first k locations in Z are swapped first, then the following k locations, etcetera. The k locations in $V\backslash Z$ that are replacing these k locations in Z are chosen in three different ways which will be explained promptly.

Firstly, we choose to swap the k locations that maximize the coverage level. To achieve this we will use the distances between the locations in $V\backslash Z$ and all other locations in $V\backslash Z$ given in (16). We define $d(lm)$ as the distance between location $l \in V\backslash Z$ and location $m \in V\backslash Z$. For the locations that are added to Z, the locations which are closest to all other locations in $V\backslash Z$ are chosen, such that these locations in Z are able to facilitate many clients within the least distance. The k locations that will be added to Z are chosen by selecting l_i k times while ignoring the locations in Z that already have been selected. This selection procedure is given in equation (17). In the continuation of this paper, this scheme will go by the name $S2a$.

$$D(S2a) = \begin{bmatrix} d(1,1) & \dots & d(1,|V|-|Z|) \\ \vdots & \ddots & \vdots \\ d(|V|-|Z|,1) & \dots & d(|V|-|Z|,|V|-|Z|) \end{bmatrix} = d_{S2a}(lm) \quad \forall l \in V\backslash Z, m \in V\backslash Z \tag{16}$$

$$l_i = \arg\min_{l} \; d_{S2a}(lm) \quad \text{for} \quad i = 1,..,k \text{ and } l_i \in V\backslash Z \tag{17}$$

Secondly, we aim for a cost reduction by looking at the location in $V\backslash Z$ which is the farthest away from all current location destinations in Z. To achieve this we will use the distances between the locations in $V\backslash Z$ and all locations in Z given in (18). Here we define $d(lm)$ as the distance between location $l \in V\backslash Z$ and location $m \in Z$. This location $l \in V\backslash Z$ is the most expensive one to facilitate, so by making this location a destination location we save the travel costs of this expensive client. The k locations that will be replaced are chosen by selecting l_i k times while ignoring the locations in Z that already have been selected. This selection procedure is given in equation (19). In the continuation of this paper, this scheme will go by the name $S2b$.

$$D(S2b) = \begin{bmatrix} d(1,1) & \dots & d(1,|Z|) \\ \vdots & \ddots & \vdots \\ d(|V|-|Z|,1) & \dots & d(|V|-|Z|,|Z|) \end{bmatrix} = d_{S2b}(lm) \quad \forall l \in V\backslash Z, m \in Z \tag{18}$$

$$l_i = \arg\max_{l} \; d_{S2b}(lm) \quad \text{for} \quad i = 1, .., k \text{ and } l_i \in V \backslash Z \tag{19}$$

Lastly, we investigate what happens if we swap the k locations in $V \backslash Z$ of which the sum of distances to all other locations in Z is the closest to the average of the distances between all locations in $V \backslash Z$ and Z. For this we will use the vertex given in (20). We define $d(l)$ as the total distance from location $l \in V \backslash Z$ to all locations in Z. The intuition here is that a compromise is made between the distance that the facility and client both have to travel. The k locations that will be swapped from $V \backslash Z$ are chosen by selecting l_i k times while ignoring the locations in Z that already have been selected. This selection procedure is given in equation (21). In the continuation of this paper, this scheme will go by the name $S2c$.

$$D(S2c) = \begin{bmatrix} \sum_{m=1}^{|Z|} d_{S2b}(1m) \\ \sum_{m=1}^{|Z|} d_{S2b}(2m) \\ \vdots \\ \sum_{m=1}^{|Z|} d_{S2b}(|V| - |Z|m) \end{bmatrix} = d_{S2c}(l) \quad \forall l \in V \backslash Z \tag{20}$$

$$l_i = \arg\min_{l} \; \Big(d_{s2c}(l) - \big(\frac{1}{|V| - |Z|} \sum_{l=1}^{|V|-|Z|} d_{dS2c}(l) \big) \Big)^2 \quad \text{for} \quad i = 1, .., k \text{ and } l_i \in V \backslash Z \tag{21}$$

4.5 Modification to the local search algorithm

In this paper, the FI and BI heuristics will be solved for the case $n = 1$ so that we search the neighborhood of swapping one location. As a variation on these heuristics from Halper, Raghavan, and Sahin (2015) given in Section 4.3, we propose the so-called $1to2$-$OptSwap$. The aim of the $1to2$-$OptSwap$ heuristic is to prevent the heuristic to end in a local optimum, by searching the neighborhood of swapping one and two locations if we meet the criterion that the heuristic solution does not improve for $|F|$ times. The heuristic starts the same as the FI and BI heuristics and thus starts searching the neighborhood of swapping one location in Z with one location in $V \backslash Z$. If the criterion is met, we also swap two destinations in the following search, after which we restart to swap one location until the criteria is met again. Recall that for the n-$OptSwap$ heuristic we have a neighborhood size of $\sum_{k=1}^{n} \binom{|F|}{k} \binom{|V|-|F|}{k}$, so the neighborhood of the $1to2$-$OptSwap$ heuristic is significantly larger.

5 Data

The data that is used for this paper is retrieved from S. Raghavan (2019). The data consists of homogeneous instances, for which holds that all facilities have the same weight, and heterogeneous instances, for which holds that all facilities have different weights. In these homogeneous and heterogeneous instances a distinction is made between large C *over* F instances and small C *over* F instances. For the large C *over* F instances the $|C|/|F|$ ratio is larger than 10 and for the small C *over* F instances the $|C|/|F|$ ratio is smaller than or equal to 10. In the continuation of

this paper these sets of instances will be referred to as: hom_largeC, hom_smallC, het_largeC and het_smallC respectively.

In this paper, the MFLP will be solved for 28 different instances from hom_smallC which contains the following data. Firstly, the number of locations $|V|$ is given. This is followed by a $|V| \times |V|$ distance matrix $D = (d_{ij})$ which shows the shortest path distances between all locations. Next, the weights of all facilities are given by $w_j \quad \forall j = 1, .., |V|$. Furthermore, the weights of all clients are given by $u_i \quad \forall i = 1, ..., |V|$. If $w_j > 0$, there is a facility initially located on location j. If $u_i > 0$, there is a client initially located on location i. For these 28 instances from hom_smallC it holds that there is a client located on every location point. The numbers of locations and clients vary between 100 and 900 and the number of facilities take up between 10% and 33% from the number of clients and locations. For these 28 instances the number of clients is equal to the number of locations such that on every location there is a client present. More details about the data are provided in the paper of Raghavan, Sahin, and Salman (2019)

6 Results

In this section the results of solving the MFLP will be given and we will conclude that the FI heuristic in combination with the use of scheme $S1$ will result in the best approximate solution. This section will be organized as follows: First, the results of the IP formulation will be discussed. Next, the results from the Local Search Algorithms will be given. Thirdly, the findings on using different schemes to select the swaps will be given. Lastly, the results of the application to the FI heuristic with the best schematic approach will be provided.

All results are obtained on an *Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz and 8.00GB RAM running 64-bit Windows 10*. The IP formulation is implemented in *Java Version "1.8.0-24"64-Bit* using *CPLEX Studio IDE 12.8.0* and the code is included in a ZIP file with package name *BachelorThesis*. This package contains all instances, the *cplex.jar* library, *main.java* which reads all instances and *model.java* which builds the model and uses the *CPLEX* solver to solve it. All heuristics are implemented in *Java Version "1.8.0-24"64-Bit* and the code is included in a ZIP file with package name *BachelorThesisHeuristic*. Class *main.java* reads all instances and when the code runs the user can select which heuristic is used. All heuristics use a class ["*heuristicname*".*java*] that updates the locations in Z and the locations in $V \backslash Z$ and a class [$OptSwapHeuristic.java$] that performs the swapping. Furthermore, the heuristics contain classes which perform the Hungarian Algorithm [$HungarianAlgorithm.java$] and the Client Assignment [$ClientAssignment.java$] in order to associate each set of destination locations with its solution value. The code for the Hungarian Algorithm is retrieved from Aalmi (2018).

6.1 Results - Integer Programming Formulation

The results of the IP formulation show that the objective value varies significantly between the instances. Namely, the smaller the $|C|/|F|$ ratio, the smaller the objective value. Table 1 shows the results from the IP formulation given in Section 4.1 and these results match the results from

Halper, Raghavan, and Sahin (2015). Table 1 shows the number of locations, facilities and clients for each instance. More importantly, the IP objective value and the runtime are displayed for the 28 instances. The IP objective value varies between 2170.568 and 7151.863. The objective value is smaller for instances where the ratio of clients to facilities is smaller. Intuitively, this makes sense as you have more facilities that are able to serve the client and thus it is more likely that a solution can be found that has a smaller distance. The results also show that the larger the instance, the longer it takes for the problem to be solved. The smallest instance takes 0.48 seconds and the largest instance takes 890.23 seconds to run. Also noticeable is the fact that in most cases it holds that for instances with the same number of locations, the runtime is lower if there are more facilities available.

	Instance	Locations	Facilities	Clients	IP Objective	IP RT(sec)	IP-LP gap(%)
1	pmed2v3	100	10	100	4914.029	0.48	0.00
2	pmed3v3	100	10	100	5792.737	0.47	0.00
3	pmed4v3	100	20	100	4748.112	0.41	0.00
4	pmed5v3	100	33	100	2170.568	0.42	0.00
5	pmed8v3	200	20	200	6077.759	3.51	0.00
6	pmed9v3	200	40	200	4348.864	3.27	0.00
7	pmed10v3	200	67	200	2377.398	3.16	0.00
8	pmed13v3	300	30	300	5487.019	15.19	0.01
9	pmed14v3	300	60	300	3963.368	10.92	0.00
10	pmed15v3	300	100	300	2642.845	10.00	0.00
11	pmed18v3	400	40	400	5844.415	35.77	0.00
12	pmed19v3	400	80	400	4229.188	29.56	0.00
13	pmed20v3	400	133	400	3178.699	29.78	0.00
14	pmed23v3	500	50	500	6756.927	74.58	0.00
15	pmed24v3	500	100	500	4782.641	66.13	0.00
16	pmed25v3	500	167	500	3033.293	66.97	0.00
17	pmed28v3	600	60	600	6133.262	174.67	0.01
18	pmed29v3	600	120	600	4756.740	145.19	0.01
19	pmed30v3	600	200	600	3151.897	123.61	0.00
20	pmed33v3	700	70	700	6740.803	228.05	0.00
21	pmed34v3	700	140	700	4507.579	251.09	0.02
22	pmed34v3-1	700	233	700	3214.991	249.48	0.01
23	pmed37v3	800	80	800	6296.171	457.97	0.02
24	pmed37v3-1	800	160	800	4538.410	357.75	0.00
25	pmed37v3-2	800	267	800	3327.008	330.61	0.00
26	pmed40v3	900	90	900	7151.863	890.23	0.08
27	pmed40v3-1	900	180	900	5261.134	641.03	0.01
28	pmed40v3-2	900	300	900	3753.487	588.77	0.00
tot						4789.07	0.14
avg						171.038	0.005

Table 1: Results of the IP formulation of instances hom_smallC (from CPLEX)

6.2 Results - Local Search Algorithm

This section will discuss the results of the 1-$OptSwapFI$ and 1-$OptSwapBI$ heuristics which are given in Algorithm 1. The results of the FI and BI heuristics show that on average the FI heuristic will be preferred over the BI heuristic. The results also indicate that randomly choosing the locations to swap results in noteworthy different optimality gaps. Due to limited time the larger instances were not able to run to its full capacity. For these instances the runtimes of the FI and BI heuristics were limited to 1800 and 3600 seconds respectively. Nevertheless, we ran the largest instance of the heuristic to confirm that the heuristic is also able to solve the largest instance. For the facility assignment the Hungarian Algorithm is used which is very time consuming because it runs in $O(n^3)$ time. In this research an attempt is made to achieve lower runtimes by not calculating the assignment problem of all sets of destination locations if it is already clear that a lower objective value cannot be achieved. Namely, Halper, Raghavan, and Sahin (2015) give a rule of when a better solution cannot be made. However, this attempt to increase the speed of the heuristics did not succeed. In order to ensure clarification, only results of the first eight instances will be discussed. For more results we refer to Table B of Appendix B in which is shown that the FI heuristic results in a tight optimality gap for the largest instance as well. Table C of Appendix C shows more results of the BI heuristic. Note that for these results the problem was solved once. The results of the FI and BI do not match the results of Halper, Raghavan, and Sahin (2015). This can be caused by different ways of choosing the locations to swap.

Table 2 shows the results of the FI heuristic with a random choice of the locations to swap. This heuristic is solved 100 times in order to get insight on the distribution of the outcomes. Table 2 shows the average objective value (OV), optimality gap and runtime. Here, the average optimality gap equals 0.51%. The table also shows that on average the worst case scenario gives an optimality gap of 2.32% and in the best case scenario the average optimality gap equals 0.00%. Figure D.1 of Appendix D shows the distribution of outcomes for the first instance of the FI heuristic, which shows that the optimal solution is achieved 26 times and the worse solutions are not so frequently achieved.

Table 3 shows the results of the BI heuristic with a random choice of the locations to swap which is also solved 100 times. Here, the average optimality gap equals 0.66%. In the worst case scenario the average optimality gap is 2.59% and in the best case scenario the average optimality gap is 0.02%. Figure D.2 of Appendix D shows the distribution of outcomes for the first instance of the BI heuristic, which shows that the optimal solution is achieved 15 times and the worse solutions are not so frequently achieved.

Comparing the results of the first eight instances of the FI and BI heuristic indicates that the FI heuristic is preferred over the BI heuristic. Namely, the FI heuristic results in an average gap of 0.51% while the BI heuristic results in an average gap of 0.66%. When running all instances only the first 12 instances could run without time limit and here optimality gap of the BI was only 8% better than for the FI heuristic. However, the runtime of the BI was 58 times

larger, because of the fact that the BI heuristic has to investigate its whole neighborhood to find the best improvement. Therefore, we still draw the same conclusion that the FI heuristic is preferred here. Lastly, we conclude that the results of randomly choosing the locations to swap are significantly different and this gives motivation to find a schematic way of choosing the locations to swap, which will be discussed in the next section.

Instance	avg OV	avg Gap(%)	min OV	min Gap(%)	max OV	max Gap(%)	std. dev.
1	4941.28	0.55	4914.03	0.00	5038.84	2.54	29.94
2	5839.11	0.80	5792.74	0.00	5971.56	3.09	51.04
3	4782.55	0.73	4748.11	0.00	4887.43	2.93	32.66
4	2182.17	0.54	2170.57	0.00	2273.23	4.73	15.33
5	6092.20	0.24	6077.76	0.00	6145.52	1.11	17.14
6	4376.34	0.63	4349.86	0.02	4428.28	1.83	16.45
7	2385.86	0.36	2377.40	0.00	2411.58	1.44	5.86
8	5497.67	0.19	5487.02	0.00	5534.01	0.86	10.18
avg		0.51		0.00		2.32	22.33

Table 2: Statistics on the 1-OptSwapFI with random choice of k

Instance	avg OV	avg Gap(%)	min OV	min Gap(%)	max OV	max Gap(%)	std. dev.
1	4951.39	0.76	4914.03	0.00	5099.43	3.77	35.77
2	5855.66	1.09	5792.74	0.00	5990.59	3.42	43.42
3	4798.83	1.07	4748.11	0.00	4989.56	5.09	43.32
4	2189.61	0.88	2170.57	0.00	2230.89	2.78	13.23
5	6105.71	0.46	6077.76	0.00	6266.09	3.10	28.97
6	4375.31	0.61	4349.80	0.02	4430.85	1.89	15.03
7	2383.73	0.27	2380.87	0.15	2387.04	0.41	1.55
8	5492.50	0.10	5487.02	0.00	5502.61	0.28	5.34
avg		0.66		0.02		2.59	23.33

Table 3: Statistics on the 1-OptSwapBI with random choice of k

6.3 Results - Finding a schematic way of choosing the locations to swap

As a result of choosing the swap of the facility destinations randomly, the results are spread out over a range of 2.32% for the FI heuristic and over a range of 2.57% for the BI heuristic (see Tables 2, 3). These differences, together with the standard deviations of 22.33 and 23.33 respectively, reflect the inconsistency of the outcomes and thus it seems reasonable to evaluate a scheme which decides on the locations to swap. Even though the optimal solution was found 26% of the time by the FI heuristic and 15% of the time BI by the BI heuristic, finding a schedule for the swaps has added value for the reliability of the solutions. This section will give the results of using the different schemes in order to select the locations to swap. From these results we conclude that the FI heuristic in combination with scheme $S1$ will give the best approximate result. We also notice that whether a scheme is beneficial depends on the instance. One scheme may work very well for one instances and may work poorly for others.

The results of the first eight instances of the FI and BI heuristics with the different schemes $S1$, $S2a$, $S2b$ and $S2c$ are displayed in Table 4 and Table 5. Comparing these results to the random case indicates that for the FI heuristic $S1$ slightly outperforms the random choice of the swaps on average with 0.50% versus 0.51% and significantly outperforms $S2a$, $S2b$ and $S2c$ which give average optimality gaps of 1.10%, 1.04% and 0.91% respectively. However, the random case will result in the optimal value more often than $S1$ and the maximum optimality gap of $S1$ equals 1.45% while for the random case this is 0.80%. The latter shows that scheme $S1$ works better for some instances, while it results in worse outcomes for others. Nevertheless, our conclusion will be based on the average performance and consistency will be prioritized. The runtimes of $S1$ are higher than the random case for almost all instances. This can be caused by the fact that the pre-determination of the order of swaps takes longer to achieve a first improvement. Using the different schedules for the BI heuristic results in average optimality gaps of 0.66% for $S1$, 1.12% for $S2a$, 2.07% for $S2b$ and 0.97% for $S2c$. The application of $S1$ results in the same average gap as the random case and all schemes result in larger average gaps than for the FI heuristic, so the preferred heuristic is still the FI heuristic. Noticeable is that where the application of $S1$ to the FI heuristic results in larger runtimes, the application of $S1$ to the BI heuristic results in smaller runtimes for most instances. This can be explained by the fact that the pre-determination of the order of swaps does impact the time to find the next best improvement as the BI heuristic always has to search the whole neighborhood regardless. More results of the FI and BI heuristic with $S1$ are given in Table E and Table F of Appendix E and Appendix F respectively.

In short, after comparing all results we can conclude that on average the FI heuristic with $S1$ results in the best approximate solution. Despite the fact that the random case results in the optimal solution more often, consistency is prioritized. Another observation is that for every instance another schedule is the most beneficial. In the next section we will try to improve the FI heuristic with $S1$ even further.

Instance	Random		S1		S2a		S2b		S2c	
	Gap	RT	Gap	RT	Gap	RT	Gap	RT	Gap	RT
1	0.55	1.61	0.47	1.88	0.39	1.60	0.00	1.74	0.00	1.48
2	0.80	1.80	0.64	1.96	1.50	1.07	0.00	1.65	0.00	1.46
3	0.73	2.02	0.16	3.34	1.25	1.93	2.25	1.63	2.67	1.83
4	0.54	4.33	0.58	3.90	1.19	3.45	0.65	2.85	1.05	2.61
5	0.24	20.07	0.60	27.27	0.00	9.82	0.00	10.22	0.03	13.84
6	0.63	43.37	0.08	74.96	3.41	11.57	4.77	13.49	2.96	12.27
7	0.36	75.49	1.45	81.32	0.46	74.38	0.39	59.20	0.39	46.14
8	0.19	92.04	0.00	156.65	0.59	37.98	0.23	51.11	0.18	61.94
avg	0.51	27.23	0.50	43.91	1.10	17.73	1.04	17.74	0.91	17.70

Table 4: Optimality gaps in % and runtimes in seconds of the 1-OptSwapFI

Instance	Random		S1		S2a		S2b		S2c	
	Gap	RT	Gap	RT	Gap	RT	Gap	RT	Gap	RT
1	0.76	5.23	0.39	3.34	1.98	1.72	1.26	2.52	0.06	2.53
2	1.09	5.89	2.28	2.85	0.00	1.58	2.13	2.59	1.92	1.77
3	1.07	18.41	0.32	5.98	1.64	8.48	1.86	7.40	1.79	8.15
4	0.88	24.56	1.08	19.06	0.58	19.32	0.75	22.49	0.55	30.16
5	0.46	182.01	0.10	106.86	0.25	97.42	1.70	75.97	0.84	77.57
6	0.61	700.29	0.29	522.26	3.17	232.10	5.72	136.03	2.08	293.60
7	0.27	799.33	0.78	2230.03	0.35	813.30	1.32	1503.93	0.23	2291.68
8	0.10	1375.94	0.02	1583.76	1.00	602.86	1.80	425.10	0.27	1254.71
avg	0.66	388.96	0.66	559.27	1.12	222.10	2.07	272.00	0.97	495.02

Table 5: Optimality gaps in % and runtimes in seconds of the 1-OptSwapBI

6.4 Results - Modification to the local search algorithm

From the previous section we conclude that the FI heuristic with $S1$ results in the best optimality gap for the first eight instances. Therefore the $1to2\text{-}OptSwap$ heuristic will also use $S1$ to select the swaps. The results indicate that the $1to2\text{-}OptSwapFI$ heuristic gives even tighter optimality gaps than the FI. Table 6 shows the results of the first twelve instances of the FI heuristic with $S1$ and the results of the $1to2\text{-}OptSwapFI$ heuristic with $S1$. The extension of searching the neighborhood of both one and two swaps for a certain criteria results in an average optimality gap of 0.37% versus an average optimality gap of 0.53%, which is an improvement of 43.24%. Half of the instances achieve a tighter optimality gap with the $1to2\text{-}OptSwapFI$ heuristic and only 2 instances perform worse. The intuition of this extension is to prevent the heuristic to end up in a local optimum, however it seems that it is still possible to end up in the same or an even worse local optimum. Nevertheless the application turned out to improve the FI heuristic significantly and still boosts the local search out of a local optimum for most instances.

Instance	1-OptSwapFI with S1		1to2-OptSwapFI with S1	
	Gap	RT	Gap	RT
1	0.47	1.88	0.00	2.70
2	0.64	1.96	0.64	98.19
3	0.16	3.34	0.16	474.61
4	0.58	3.90	0.00	497.06
5	0.60	27.27	0.60	642.17
6	0.08	74.96	0.28	2752.96
7	1.45	81.32	1.14	9032.72
8	0.00	156.65	0.13	7929.76
9	1.35	285.74	0.74	13901.22
10	0.64	485.55	0.45	17122.93
11	0.11	450.13	0.07	15872.12
12	0.23	1481.20	0.23	28001.42
avg 1-8	0.50	43.91	0.37	2759.03
avg 1-12	0.53	254.49	0.37	8080.83

Table 6: Optimality gaps in % and runtimes in seconds of the 1-OptSwapFI and 1to2-OptSwapFI with S1

7 Conclusion

This research aimed to decide on the destination locations for clients and facilities such that all clients and facilities share a destination location, while the moving costs for both clients and facilities were minimized. This MFLP is solved exactly by means of an IP formulation and approximately by means of two local search heuristics given by Halper, Raghavan, and Sahin (2015). One of the local search heuristics searches a neighborhood of solutions for the first improvement and the other searches for the best improvement. Furthermore, this research aimed to find effective schedules in order to make the results consistent while maintaining or improving the heuristic solutions. These schedules decide the order in which locations are swapped. Lastly, an application is made to the most favorable heuristic in order to prevent the local search heuristic to end up in a local optimum. The results of the IP formulation implied that the objective value is smaller for instances where the ratio of clients to facilities is smaller. The solving time is increasing with the number of locations as well. The results of the two local search algorithms showed that the runtimes were much higher than for the IP formulation, so the first eight instances are studied in this paper. These two local search algorithms searched a neighborhood by randomly swapping locations in the set of destinations and the remaining locations. From these results we can conclude that selecting the first improvement is preferred over selecting the best improvement when searching in the neighborhood of solutions. We found optimality gaps of 0.51% on average for the FI heuristic and 0.66% on average for the $BIheuristic$ for the first eight instances so these heuristics can be used to solve the MFLP while achieving tight optimality gaps. We also saw that the solutions are inconsistent when choosing the swaps randomly which gave motivation to find a schematic way of choosing the next locations to swap. Consequently four different schemes were investigated and these results indicated that one of the schedules in combination with selecting the first improvement in the neighborhood resulted in the best approximate solution, namely an average optimality gap of 0.50% for the first eight instances. The implementation of the schemes also showed that some schemes work very well for certain instances while others do not. A modification was made to the heuristic of selecting the first improvement combined with the best schedule, where for a certain criterion the neighborhood of swapping both one and two locations was searched instead of swapping only one location. As a consequence the average optimality gap decreased to 0.37% and many instances were prevented from ending up in a local optimum too quickly.

8 Discussion

The runtimes of the heuristics in this paper were significantly higher than in the paper of Halper, Raghavan, and Sahin (2015) which resulted in not being able to run all instances to its optimal potential. Methods in order to improve the speed of the implementation of these heuristics can be applied in future research, so that the larger instances can run to its full potential as well. For future research one could also do some more research on how to define the best schedule to select the locations to swap. In this paper, it was noticeable that a certain scheme worked beneficial for some instances and worked poorly for others. Furthermore, the discrepancies between the results in this paper and the results of Halper, Raghavan, and Sahin (2015) indicate that there

were differences in the way of choosing the next locations to swap. More specific criteria could be defined taking all characteristics like the number of facilities, clients and locations of each instance into consideration.

Although the runtimes in this research were significantly high, we showed that the MFLP can be solved approximately by means of the local search heuristics of Halper, Raghavan, and Sahin (2015) and that the solutions are even closer to optimal when applying a schematic way of swapping and when the neighborhood was increased for a certain criterion. These heuristics can thus be used to solve the MFLP or variations on this problem. Many fields could use these heuristics to solve their everyday problems like departments relocating firefighters and ambulances, but they are also useful for disaster relief logistics and supply chain problems.

References

Aalmi (2018). *"HungarianAlgorithm"*. URL: `https://github.com/aalmi/HungarianAlgorithm`.

Arya, Vijay et al. (2004). "Local search heuristics for k-median and facility location problems". In: *SIAM Journal on computing* 33.3, pp. 544–562.

Berman, Oded and B LeBlanc (1984). "Location-relocation of mobile facilities on a stochastic network". In: *Transportation science* 18.4, pp. 315–330.

Demaine, Erik D et al. (2009). "Minimizing movement". In: *ACM Transactions on Algorithms (TALG)* 5.3, pp. 1–30.

Friggstad, Zachary and Mohammad R Salavatipour (2011). "Minimizing movement in mobile facility location problems". In: *ACM Transactions on Algorithms (TALG)* 7.3, pp. 1–22.

Gendreau, Michel, Gilbert Laporte, and Frédéric Semet (2001). "A dynamic model and parallel tabu search heuristic for real-time ambulance relocation". In: *Parallel computing* 27.12, pp. 1641–1653.

Halper, Russell, S Raghavan, and Mustafa Sahin (2015). "Local search heuristics for the mobile facility location problem". In: *Computers & Operations Research* 62, pp. 210–223.

Kuhn, Harold W (1955). "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2, pp. 83–97.

Raghavan, S. (2019). *Instances for the Capacitated Mobile Facility Location Problem*. DOI: `10.17632/JTV74HKMFG.1`. URL: `https://data.mendeley.com/datasets/jtv74hkmfg/1`.

Raghavan, S, Mustafa Sahin, and F Sibel Salman (2019). "The capacitated mobile facility location problem". In: *European Journal of Operational Research* 277.2, pp. 507–520.

Zhang, Peng (2007). "A new approximation algorithm for the k-facility location problem". In: *Theoretical Computer Science* 384.1, pp. 126–135.

Appendices

A Hungarian Algorithm

Algorithm 2: Hungarian Algorithm

1 Define a $m \times n$ matrix $C = (c_{ij})$ denoting the costs of moving from facilities/clients to destinations. For the facilities these costs are denoted by $d_{ij} * w_j$. For the clients these costs are denoted by $d_{ij} * u_i$. Have the facilities/clients you have to assign on the left and the possible destination locations along the top.

2 If $m = n$, the matrix is square and go to step 3.
If $m \neq n$, the matrix is not square, so add dummy rows/columns. The value c'_{ij} of these dummy rows/columns are set equal to the largest number in the matrix.

3 Look for the minimum value $rowmin_i = \min_j c_{ij}$ of each row i and subtract this value from every value of that row. $c_{ij} = c_{ij} - rowmin_i \quad \forall i = 1, .., m; j = 1, ..., n$

4 Look for the minimum value $colmin_j = \min_i c_{ij}$ of each column j. For every column without a zero: subtract this value from every value of these column.
$c_{ij} = c_{ij} - colmin_j \quad \forall i = 1, ..., m; j = 1, ..., n$

5 Cover the zero elements with the minimum number of lines it is possible to cover them with. If the number of lines is equal to the number of rows, then go to step 9.

6 Add the minimum uncovered element to every covered element. If an element is covered twice, add the minimum element to it twice.

7 Subtract the minimum element from every element in the matrix:
$d_{ij} = d_{ij} - \min d_{ij} \quad \forall i = 1, .., m; j = 1, ..., n.$

8 Cover the zero elements again. If the number of lines covering the zero elements is not equal to the number of rows, return to step 6.

9 Select a matching by choosing a set of zeros so that each row or column has only one selected.

10 Apply the matching to the original matrix, disregarding dummy rows. Adding all distances will give the total minimum distances

B Results of FI for choosing the swaps randomly

Instance	Objective Value	Gap(%)	Runtime (sec)
1	4933.21	0.39	1.01
2	5792.74	0.00	1.45
3	4788.23	0.85	2.29
4	2170.57	0.00	4.30
5	6077.76	0.00	16.37
6	4368.11	0.46	30.87
7	2381.79	0.19	70.16
8	5488.05	0.02	91.36
9	3964.60	0.03	275.24
10	2656.50	0.52	384.69
11	5859.25	0.25	169.51
12	4242.94	0.33	848.332
13	3186.77	0.25	2032.65
14	6764.79	0.12	672.10
15	4786.53	0.08	2360.04
16	3051.59	0.60	1438.82
17*	6143.44	0.17	1970.12
18*	4768.20	0.24	1902.40
19*	3239.99	2.80	1881.23
20*	6779.20	0.57	1886.43
21*	4571.42	1.42	2016.48
22*	3353.36	4.30	1800.77
23*	6347.74	0.82	1804.22
24*	4711.74	3.82	1830.08
25*	3477.60	4.53	2119.66
26*	7262.73	1.55	1812.01
27*	5500.290	4.38	1862.40
28	3775.03	0.57	133458.20
avg		1.05	5894.40
**		0.27	8479.85
***		0.25	157.97

Table B: Results from the 1-OptSwapFI heuristic of instances hom_smallC

* These instances have a runtime limit of 1800 seconds

** The average gaps and runtimes excluding the instances with runtime limit

*** The average gaps and runtimes for the first 12 instances

C Results of BI for choosing the swaps randomly

Instance	Objective Value	Gap(%)	Runtime(sec)
1	4914.03	0.00	4.05
2	5972.74	0.00	4.71
3	4748.11	0.00	14.19
4	2183.25	0.58	21.10
5	6077.76	0.00	136.67
6	4365.96	0.39	583.78
7	2391.03	0.57	894.51
8	5487.84	0.02	1136.36
9	3964.13	0.02	9869.87
10	2655.40	0.48	15684.79
11	5853.55	0.16	10186.39
12	4253.73	0.58	70582.14
13*	3627.35	14.11	3600.02
14*	7056.00	4.43	3600.02
15*	5507.40	15.15	3600.03
16*	3493.10	15.19	3600.03
17*	6747.09	10.01	3600.02
18*	5593.12	17.58	3600.01
19*	3731.00	18.37	3600.27
20*	8026.52	19.07	3600.02
21*	5416.48	20.16	3600.01
22*	3915.93	21.80	3600.26
23*	7470.02	18.64	3600.03
24*	5803.57	27.88	3600.03
25*	3976.57	19.52	3600.15
26*	8890.52	24.31	3600.04
27*	6644.57	26.30	3600.08
28*	4464.50	18.94	3600.76
avg		10.51	5956.10
**		0.23	9097.42
***		0.14	271.55

Table C: Results from the 1-OptSwapBI heuristic of instances hom_smallC

* These instances have a runtime limit

** The average gaps and runtimes excluding the instances with runtime limit

*** The average gaps and runtimes for instances 1, 2, 3, 4, 5, 6 and 8

D Distribution of solutions for the FI and BI heuristics for Instance 1

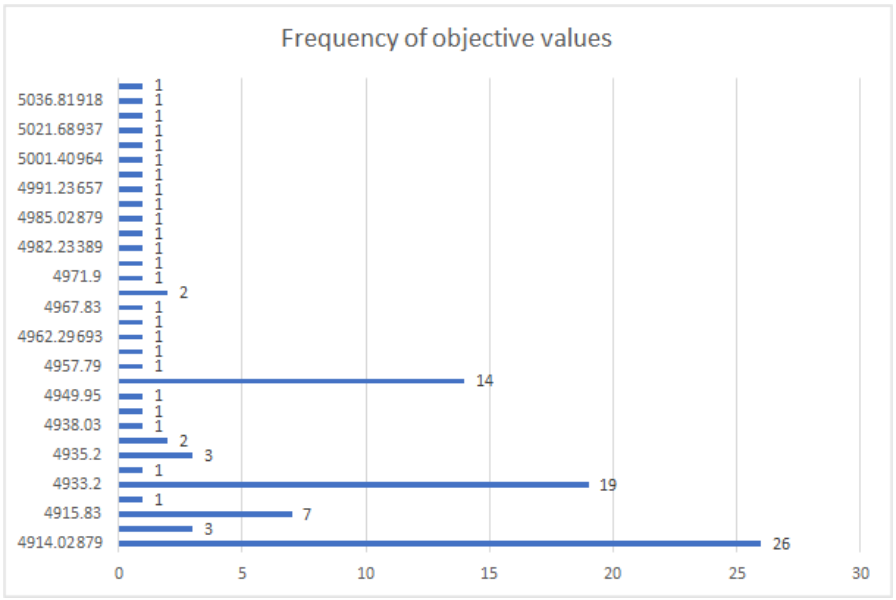


Figure D.1: Distribution of instance 1 outcomes 1-OptSwapFI with random choice of k

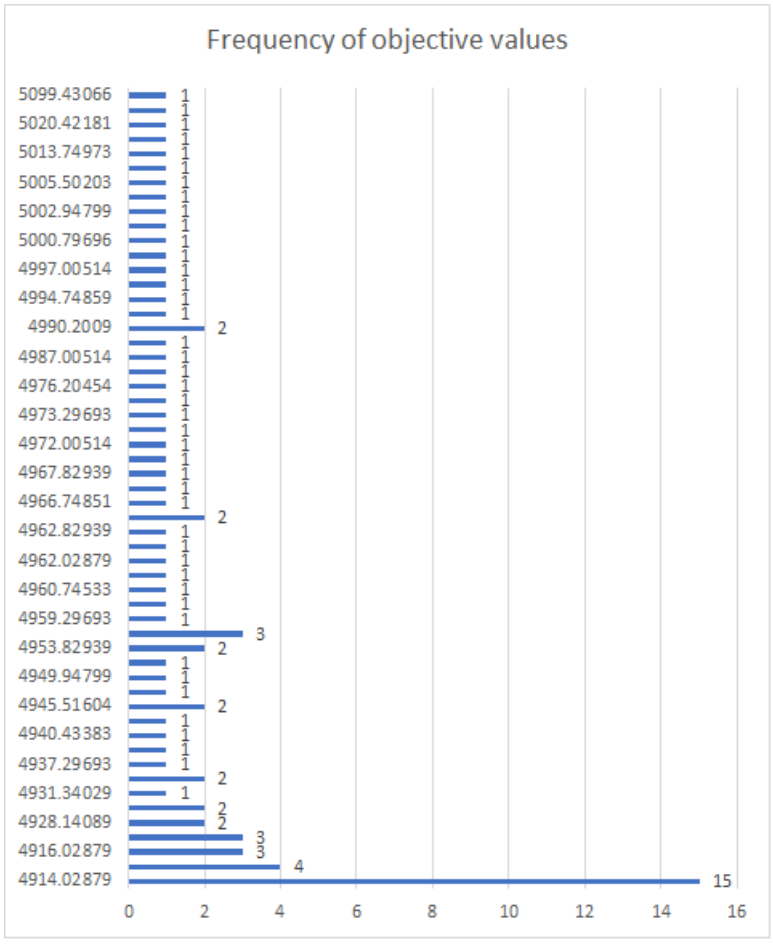


Figure D.2: Distribution of instance 1 outcomes 1-OptSwapBI with random choice of k

E Results of FI with S1

Instance	Objective Value	Gap(%)	Runtime (sec)
1	4937.30	0.47	1.88
2	5829.54	0.64	1.96
3	4755.91	0.16	3.34
4	2183.25	0.58	3.90
5	6114.35	0.60	27.27
6	4352.39	0.08	74.96
7	2411.86	1.45	81.32
8	5487.10	0.00	156.65
9	4016.80	1.35	285.74
10	2659.72	0.64	485.55
11	5851.05	0.11	450.13
12	4238.86	0.23	1481.20
13	3193.47	0.47	2091.10
14	6783.42	0.39	934.19
15	4811.75	0.61	1847.90
16	3145.73	3.71	1655.95
17*	6144.95	0.19	1802.22
18*	4916.52	3.36	1814.82
19*	3393.50	7.66	1808.46
20*	6917.28	2.26	1898.61
21*	4855.57	7.72	1835.66
22*	3627.47	12.83	1828.56
23*	6524.72	3.63	1813.81
24*	5051.23	11.29	1817.68
25*	3726.38	12.00	1962.12
26*	8068.74	12.82	1823.52
27*	5928.04	12.68	1822.96
28	4270.19*	13.76*	1863.35*
avg		3.99	1131.24
**		0.72	598.94

Table E: Results from the 1-OptSwapFI heuristic of instances hom_smallC

* These instances have a runtime limit of 1800 seconds

** The average gaps and runtimes excluding the instances with runtime limit

F Results of BI with S1

Instance	Objective Value	Gap(%)	Runtime(sec)
1	4933.14	0.39	3.45
2	5924.90	2.28	2.85
3	4763.07	0.32	5.98
4	2194.06	1.08	19.06
5	6083.53	0.10	106.86
6	4361.26	0.29	522.26
7	2395.94	0.78	2230.03
8	5488.05	0.02	1583.76
9*	4640.80	17.09	3600.03
10*	3003.91	13.66	3600.02
11*	7249.64	24.04	3600.01
12*	5326.45	25.95	3600.07
13*	3847.26	21.03	3600.05
14*	7113.18	5.27	3600.05
15*	5311.44	11.06	3600.05
16*	3493.10	15.19	3600.03
17*	6697.41	9.20	3600.05
18*	5709.03	20.02	3600.04
19*	3731.00	18.37	3600.13
20*	7839.34	16.30	3600.06
21*	5416.48	20.16	3600.04
22*	3915.93	21.80	3600.30
23*	8031.650	27.56	3600.04
24*	5803.57	27.88	3600.06
25*	3976.97	19.53	3600.06
26*	9271.96	29.64	3600.04
27*	6644.57	26.30	3600.14
28*	4464.50	18.94	3600.18
avg		14.82	2780.20
**		0.64	270.12

Table F: Results from the 1-OptSwapBI heuristic of instances hom_smallC

* These instances have a runtime limit

** The average gaps and runtimes excluding the instances with runtime limit