A Tabu search approach to the mobile facility location problem

Supervisor Lisanne van Rijn
Second assessor Twan Dollevoet

Vivian Yeung (454989)

Bachelor Thesis Econometrie en Operationele Research

July 5, 2020

Abstract

Facility location problems appear extensively in the academic literature in different forms. One specific problem is the mobile facility location problem. Previous work on the mobile facility location problem has been conducted by Halper, Raghavan, and Sahin (2015). They define local search heuristics to obtain high quality solutions to the problem in reasonable time. This paper builds upon these local search heuristics. We use Tabu search procedures to improve the local search heuristics. Using a simple Tabu search we find small improvements in the optimality gaps compared to their local search heuristic counterparts defined by Halper et al. (2015).

ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

The views stated in this thesis are those of the author and not necessarily those of the supervisor, second assessor, Erasmus School of Economics or Erasmus University Rotterdam.

Contents

List of Figures

List of Tables

1 Introduction

The academic literature covers multiple different variants of the facility location problem. One variant is the mobile facility location problem (MFLP), first introduced by Demaine et al. (2009). This problem holds facilities and clients at initial locations and assigns these to destination locations in a way that every client is assigned to a location that at least one facility is also assigned to. This is done while minimising the total weighted distance of the facilities and clients to their destination locations. Friggstad and Salavatipour (2011) paint a scenario in which the MFLP comes into play. In this scenario manufacturing plants fabricate products that must be delivered to customers with minimal cost. This can be accomplished by sending the products directly from manufacturing plants to customers. A potentially less costly alternative would be using distribution centers that collect products from a plant for multiple customers, and send them to the individual customers. These distribution centers could be placed far away from the manufacturing plants and closer to the clients if this is desirable. This way the shipping cost can be reduced as the larger shipment from the manufacturing plants to the distribution centers can be less costly and if the distribution centers are located closer to the customers, these smaller shipments are less costly because of the smaller distance. Another example would be disaster relief logistics, as described by Halper et al. (2015). After a disaster has taken place, supplies need to be send to aid stations. This can be done efficiently through distribution points. These distribution points for warehouses need to be determined and would function as the facilities of the MFLP. Supplies will then be send to aid stations, which function as the clients. It is therefore of interest to find an efficient solution method to the MFLP that can be used in practice.

Some work on the MFLP has already been carried out. In particular, Halper et al. (2015) build on the work by Friggstad and Salavatipour (2011), discussing two integer problems (IPs) and local search heuristics. This paper extends the work of Halper et al. (2015). We test the state-of-the-art meta-heuristic Tabu search (TS) on the local search heuristics to try and avoid getting caught in local optima. Several different TS procedures are defined, two simple TS heuristics and two additional methods that further search the solution space. Through this we answer the research question: *"What is the effect of a Tabu search procedure on the results of local search heuristics for the MFLP?"*. We find that one of the IPs has a shorter average running time compared to the other IP. For the local search heuristics, we find high quality solutions within a reasonable running time. Applying the TS procedures, we see that the simple TS heuristics produce smaller optimality gaps compared to the local search heuristics of Halper et al. (2015), but it does not guarantee the optimal solution every time. Using the two additional methods, we generally do not find improvements in the TS heuristics.

The remainder of this paper is structured as follows. We first discuss the MFLP in more detail in Section 2 and discuss the relevant literature in Section 3. Section 4, 5 and 6 discuss the exact models, local search heuristics of Halper et al. (2015) and TS respectively. Section 7 provides the

results and in Section 8 we give a conclusion with suggestions for further research.

2 Problem description

This section gives a mathematical formulation to the MFLP. We use the framework introduced by Halper et al. (2015) to formulate the problem. This formulation sets the MFLP on a graph $G(F \cup V \cup C, A)$, with F, C and V denoting the set of facility, client and destination location vertices respectively, and A the edges between facilities and clients to destination locations. If a facility $j \in F$ or a client $i \in C$ also functions as a destination location, a new (identical) vertex is created to denote a destination location $v \in V$ so that F, C and V are three disjoint sets. It holds that the initial location of a facility is also a possible destination location, as it allows for a facility to not travel. Facilities and clients are then reassigned to a location, while minimising the total weighted travel cost, in other words the sum of the travel cost of the facilities to their location destinations and the travel cost of the demand of supplies of the clients from a facility on their destination location to the client. It is required that the destination location of a client also needs to be a destination location for at least one facility.

To calculate the total weighted distance we use the following parameters:

- $d_{j,v}$: the distance between facility $j \in F$ and destination location $v \in V$
- $d_{i,v}$: the distance between client $i \in C$ and destination location $v \in V$
- u_i: the per unit distance cost of satisfying the demand of client $i \in C$
- w_j: the per unit distance cost of relocating mobile facility $j \in F$

The weights u_i and w_j are positive, as the travel costs are positive. The total travel cost for one client (or facility) is subsequently calculated by multiplying the distance between the initial and destination location with the client (or facility) weight. Note that in this formulation multiple clients at a vertex can be seen as one aggregated client with a weight equal to the sum of their individual weights. This is because multiple clients at the same vertex will have the same destination location. If this was not the case, the total demand satisfaction cost of client $i \in C$, that has a destination location that lays further from the client vertex compared to another client $\hat{i} \in C$, can be reduced by switching the destination location of i to the closer destination location of $\hat{i}$. This results in the two clients having the same destination location $v \in V$ with total demand satisfaction cost $d_{i,v}u_i + d_{\hat{i},v}u_{\hat{i}}$, and since the two clients are at the same vertex ($d_{\hat{i},v} = d_{i,v}$) this results in the total demand satisfaction cost $d_{i,v}(u_{\hat{i}} + u_i)$. For facilities it holds that multiple facilities are not located at the same initial vertex. If multiple facilities are located at a single vertex, a facility vertex can be created for each separate facility with their respective costs and the same distances $d_{i,v}$ to vertex v for all $v \in V$ as the original vertex.

A feasible solution to this problem would be a selection of destination locations $v_j \in V$ for each facility $j \in F$ and a destination location $v_i \in V$ for a client $i \in C$ so that $v_i = v_j$ for $\forall i$

and $\exists j$, such that the constraint of a client being assigned to a location with an open facility is satisfied.

3 Literature review

Demaine et al. (2009) introduce the MFLP among movement problems and uses pebbles to illustrate these movements. For the MFLP these pebbles have an initial configuration and are moved to a target configuration, which satisfies a certain property. This can be done for any particular objective function. The pebbles represent the facilities and clients. The paper also provides a 2-approximation algorithm for the variant of the MFLP that minimises the maximum movement of a facility or client. Friggstad and Salavatipour (2011) give an IP formulation for the MFLP and present an 8-approximation algorithm through rounding off a solution of a LP relaxation. Halper et al. (2015) build upon the IP formulation of Friggstad and Salavatipour (2011) and present a new IP formulation that uses less memory and is thus applicable to larger instances. They also present multiple local search heuristics. In particular, they show that the MFLP can be decomposed into assigning the facilities to destination locations and the clients to destination locations separately, resulting in two subproblems. Using this key insight they design three local search heuristics. The first variant of the local search heuristic defines a neighbourhood as swapping destination locations of certain facilities (and minimising the client assignment cost). The second one swaps destination locations and minimises both the facility and client assignment cost. The last one also swaps destination locations, but only reassigns the facilities for which their previous destination locations are swapped out. They find that the last local search heuristic is most efficient, as it results in small optimality gaps, while still having a relatively short running time. Ahmadian, Friggstad, and Swamy (2013) show that the first heuristic, which swaps the destination locations of facilities, has an unbounded approximation ratio, because a constant number of facilities are swapped in a neighbourhood, and the second heuristic, which swaps destination locations and optimises the facility assignment, has an asymptotic approximation ratio of 3, when the maximum number of destination locations swapped is large.

Local search heuristics always find a local minimum to the problem, but it is not guaranteed that the global minimum (and thus the optimal solution) is found. To escape local optima and to perform a robust search of the solution space, meta-heuristics can be used (Gendreau & Potvin, 2010). There are many meta-heuristics that can be used in this regard, e.g. Simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), Guided local search (Voudouris & Tsang, 2003) and Late acceptance hill climbing (Burke & Bykov, 2008). One state-of-the-art meta-heuristic is Tabu search, introduced by Glover and Laguna (1998). This heuristic allows acceptance of non-improving moves to get out of a local optimum and uses memories of past visited solutions to prevent cycling back to this local optimum. Glover and Samorani (2019) state that there are three fundamental components for all of these global search meta-heuristics: intensification, di-

versification and learning. Glover and Laguna (1998) discuss these components. Intensification entails searching a particular promising part of the search space more thoroughly. Diversification makes sure a large enough part of the solution space is searched. This is done through forcing the heuristic to search unexplored areas. Learning is the way the heuristic uses the information that becomes available during the search for further progress.

There are also related problems to the MFLP found in the literature. Raghavan, Sahin, and Salman (2019) define the capacitated mobile facility problem, which is similar to the MFLP, but also has to keep in mind that the facilities are capacitated and therefore only a limited number of clients can be served per facility. The mobile facility routing problem by Halper and Raghavan (2011) is set on a time horizon on which the demand of clients change over time. Therefore the facilities can change locations over time to satisfy the demand of the clients.

4 Exact solution approaches

In this section we describe two IP formulations from the literature, using the formulation described in Section 2. These IPs are used to obtain the optimal solution of a MFLP. For both IPs we use a time limit of five hours during the computation.

4.1 Integer problem formulation 1

We discuss integer problem formulation 1 (IP1) as introduced by Friggstad and Salavatipour (2011). IP1 uses two decision variables:

- $x_{i,v} \in \mathbb{B}$: equal to 1 if client $i \in C$ has destination location $v \in V$, 0 otherwise
- $y_{j,v} \in \mathbb{B}$: equal to 1 if facility $j \in F$ has destination location $v \in V$, 0 otherwise

This gives us the following IP formulation:

$$\min \sum_{j \in F} \sum_{v \in V} w_j d_{j,v} y_{j,v} + \sum_{i \in C} \sum_{v \in V} u_i d_{i,v} x_{i,v} \tag{1}$$

$$\sum_{v \in V} x_{i,v} = 1 \qquad \forall i \in C \tag{2}$$

$$\sum_{v \in V} y_{j,v} = 1 \qquad \forall j \in F \tag{3}$$

$$\sum_{j \in F} y_{j,v} \geq x_{i,v} \qquad \forall i \in C, \forall v \in V \tag{4}$$

$$y_{j,v}, x_{i,v} \in \mathbb{B} \qquad \forall i \in C, \forall j \in F, \forall v \in V \tag{5}$$

Objective function (1) minimises the total weighted distance travelled by taking the sum of all the weighted distances of the facilities and the clients to the destination locations they are assigned to. Constraints (2) make sure all clients $i \in C$ are assigned to exactly one destination location and constraints (3) do the same for all facilities $j \in F$. Lastly constraints (4) ensure that a client

$i \in C$ is only allowed to be assigned to a destination location $v \in V$ if at least one facility is assigned to v. Halper et al. (2015) note that in this formulation the variable $x_{i,v}$ can be relaxed to a continuous variable ($x_{i,v} \in [0,1]$), which reduces the running time.

4.2 Integer problem formulation 2

The integer problem formulation 2 (IP2), as introduced by Halper et al. (2015), will now be discussed. First note that no two facilities will have the same destination location. If multiple facilities would have the same destination location $v \in V$, the solution could be improved by changing the destination location of the facilities that are the furthest away equal to the destination location $\hat{v} \in V$ that is a duplicate of their own facility vertex. This procedure still ensures that the destination location v has an open facility. Besides this, it was discussed in Section 2 that each facility vertex has one facility, and we thus know that the destination location $\hat{v}$ is a destination location to only one facility. If this is not the case, the same procedure can be applied until it does hold. If it is the case that each facility will have their own destination location, then we know that a total of $|F|$ vertices will have an open facility located on it. Using this knowledge, Halper and Raghavan (2011) formulate IP2. IP2 is almost identical to IP1. We first introduce an extra decision variable next to $x_{i,v}$ and $y_{j,v}$, namely $z_v \in \mathbb{B}$, which is equal to 1 if destination location $v \in V$ is the destination location of some facility. Using this decision variable we formulate the following IP formulation.

$$\min \sum_{j \in F} \sum_{v \in V} w_j d_{j,v} y_{j,v} + \sum_{i \in C} \sum_{v \in V} u_i d_{i,v} x_{i,v} \tag{6}$$

$$\sum_{v \in V} x_{i,v} = 1 \qquad \forall i \in C \tag{7}$$

$$\sum_{v \in V} y_{j,v} = 1 \qquad \forall j \in F \tag{8}$$

$$\sum_{j \in F} y_{j,v} - z_v = 0 \qquad \forall v \in V \tag{9}$$

$$x_{i,v} - z_v \leq 0 \qquad \forall i \in C, \forall v \in V \tag{10}$$

$$y_{j,v}, x_{i,v}, z_{i,v} \in \mathbb{B} \qquad \forall i \in C, \forall j \in F, \forall v \in V \tag{11}$$

We see that objective function (6) and constraints (7) and (8) are identical to IP1. Next to this we replace constraints (4) with the constraints (9) and (10). Constraints (9) ensure that if a facility $j \in F$ is assigned to destination location $v \in V$ ($y_{j,v} = 1$), there is a facility open at v ($z_v = 1$). We know that $\sum_{j \in F} y_{j,v} \leq 1$, since a destination location holds up to one facility. Constraints (10) make sure that a client $i \in C$ can only be assigned to a destination location $v \in V$ ($x_{i,v} = 1$), if a facility is open at v ($z_v = 1$). Halper et al. (2015) show that the variables $x_{i,v}$ and $y_{j,v}$ can be relaxed to continuous variables ($x_{i,v}, y_{j,v} \in [0,1]$) to shorten the running time.

4.3 Linear relaxations for a lower bound

To construct a lower bound as a benchmark for the MFLP, we use a linear relaxation, similarly to Halper et al. (2015). Halper et al. (2015) show that the linear relaxation (LP) of IP1 and IP2 are identical to one another. It is therefore only necessary to compute one of them. Since the LP serves as a benchmark, the running time is not relevant and thus the results of LP1 and LP2 do not differ in quality for this paper. Therefore only LP2 will be executed.

5 Local search heuristic solution approaches

Halper et al. (2015) implement three local search heuristics for the MFLP which we evaluate. These local search heuristics all decompose the MFLP into two separate subproblems, the facility assignment subproblem and client assignment subproblem. Therefore we first explain this decomposition of the problem. After this the three local search heuristics are introduced. For all the local search heuristics, we use a time limit of five hours and use the best found solution within those five hours if the time limit is reached.

5.1 Decomposition of the MFLP

Section 4.2 explains that a total of $|F|$ destination locations have an open facility. Looking at IP2, if Z is the set of $|F|$ destination locations $\tilde{v} \in Z$ with an open facility $f \in F$ and Z is known, we know that $z_{\tilde{v}} = 1$ for all $\tilde{v} \in Z$ and $z_{\hat{v}} = 0$ for all $\hat{v} \in V \setminus Z$. Setting these values for z_v for all $v \in V$, we obtain an IP that can be decomposed into two separate subproblems, because the decision variables $x_{i,v}$ and $y_{j,v}$ are both only dependent upon z_v. These two subproblems are named the facility assignment subproblem and the client assignment subproblem and assign each facility and each client to a distinct destination location in Z respectively. This is done while minimising the weighted travelled distance of the two subproblems.

The facility assignment subproblem entails assigning facilities $j \in F$ to destination locations $\tilde{v} \in Z$, given Z and $|Z| = |F|$. We therefore know that $y_{j,\hat{v}} = 0$ for $j \in F$ and $\hat{v} \in V \setminus Z$ in IP2, because it is given that the vertices $\hat{v}$ do not have an open facility assigned to them. This results in constraints (8) simplifying to constraints (13) and, together with setting $z_{\tilde{v}} = 1$ for $\tilde{v} \in Z$, this results in constraints (9) simplifying to constraints (14), which gives us the following IP formulation for the facility assignment subproblem.

$$\min \quad FA(Z) = \sum_{j \in F} \sum_{\tilde{v} \in Z} w_j d_{j,\tilde{v}} y_{j,\tilde{v}} \tag{12}$$

$$\sum_{\tilde{v} \in Z} y_{j,\tilde{v}} = 1 \qquad \forall j \in F \tag{13}$$

$$\sum_{j \in F} y_{j,\tilde{v}} = 1 \qquad \forall \tilde{v} \in Z \tag{14}$$

$$y_{j,\tilde{v}} \geq 0 \qquad \forall j \in F, \forall \tilde{v} \in Z \tag{15}$$

This IP is identical to the formulation of a least cost weighted bipartite matching problem, which matches facilities with destination locations. We see that the constraint matrix is unimodular, since constraints (13) assign one destination location per facility, and constraints (14) assigns one facility per destination location. Consequently, we can relax the integrality constraints of $y_{j,\tilde{v}}$ and the facility assignment subproblem can be solved in polynomial time using, for example, the Hungarian algorithm by Kuhn (1955).

The client assignment subproblem assigns each client $i \in C$ to a destination location $\tilde{v} \in Z$, while minimising their total weighted travelled distance. Since we know that $z_{\hat{v}} = 0$ for $\hat{v} \in V \setminus Z$ and $z_{\tilde{v}} = 1$ for $\tilde{v} \in Z$, if we set $x_{i,\hat{v}} = 0$ for $i \in C$, we can disregard constraints (10), which gives us the following IP formulation for the client assignment subproblem.

$$\min \quad CA(Z) = \sum_{i \in C} \sum_{\tilde{v} \in Z} u_i d_{i,\tilde{v}} x_{i,\tilde{v}} \tag{16}$$

$$\sum_{\tilde{v} \in Z} x_{i,\tilde{v}} = 1 \qquad \forall i \in C \tag{17}$$

$$x_{i,\tilde{v}} \geq 0 \qquad \forall i \in C, \forall \tilde{v} \in Z \tag{18}$$

Again, the integrality constraints of $x_{i,\tilde{v}}$ is relaxed, because the constraint matrix is unimodular. It can also be seen that the client assignment subproblem is essentially assigning individual clients to destination locations independently from each other. We can therefore solve the client assignment subproblem by selecting, for each client separately, the closest destination location in Z.

This gives us two subproblems of the MFLP that can be solved optimally independently from each other, given a set Z of destination locations with open facilities. Using this insight Halper et al. (2015) define three local search heuristics that searches neighborhoods and minimises the total weighted travelled distance, namely $FA(Z) + CA(Z)$.

5.2 *n-Swap* local search heuristic

The *n-Swap* heuristic of Halper et al. (2015) is based on an operation defined by Friggstad and Salavatipour (2011). The operation selects $k \leq n$ facilities and unoccupied destination locations and changes the destination location of each of the selected facilities to the unoccupied destination locations in order. There are two versions of the heuristic, best improvement (BI) and first improvement (FI). *n-SwapBI* goes through the whole neighbourhood and selects the best improved solution (largest improvement) and *n-SwapFI* selects the first improved solution. The *n-Swap* heuristic is described in Algorithm 1. We see that for every step we select a neighbour through first selecting a set of facilities $j_1, ..., j_k$, then selecting a set of unoccupied destination locations $v_1, v_2, v_3, ..., v_k$ and afterwards selecting an order of the selected destination locations, e.g. $v_2, v_1, v_3, ..., v_k$. The neighbour is composed by swapping the current destination location of j_1 with v_2, the current destination location of j_2 with v_1 etc., and solving the client assignment problem optimally to obtain the objective value of this neighbour.

Algorithm 1: The *n-Swap* local search heuristic

Input : $d_{i,v}$, $d_{j,v}$, w_j and u_i for all $i \in C$, $j \in F$ and $v \in V$

Output: A feasible solution with a local minimum value of the total weighted travelled distance

1 Initialise the current objective value $curr\_obj = FA(Z) + CA(Z)$, current facility assignment $curr\_fac$ and the occupied destination locations Z. And if performing BI, initialise $best\_obj = curr\_obj$, $best\_fac = curr\_fac$, $Z_{best} = Z$. Go to step 2.

2 Select k facilities $j_1, ..., j_k$ for $k \leq n$. If all combinations of k facilities for all $k \leq n$ are searched, go to Step 8 if performing FI and Step 9 if performing BI. If not all combinations of k facilities for all $k \leq n$ are searched, go to Step 3.

3 Select k unoccupied destination locations from $V \setminus Z$. If all combinations of k unoccupied destination locations searched go to Step 2, otherwise go to Step 4.

4 Select an order of the selected unoccupied destinations $v_1, ..., v_k$. If all permutations of the k unoccupied destination locations are searched go to Step 3, otherwise go to Step 5.

5 Set $Z_{found} = Z$ and remove all destination locations of j_l for all $l = 1, ..., k$ from Z_{found} and add v_l for all $l = 1, ..., k$. Set $found\_fac = curr\_fac$ and swap the destination location of j_l to v_l for $l = 1, ..k$ in $found\_fac$. Set $found\_obj = FA(Z_{found}) + CA(Z_{found})$. If performing FI, go to step 6, if performing BI go to step 7.

6 If $found\_obj < curr\_obj$, go to Step 8, otherwise go to Step 4.

7 If $found\_obj < best\_obj$ set $best\_obj = found\_obj$, $best\_fac = found\_fac$ and $Z_{best} = Z_{found}$, otherwise go to Step 4.

8 If $found\_obj < curr\_obj$ set $curr\_obj = found\_obj$, $curr\_fac = found\_fac$ and $Z = Z_{found}$ and go to Step 2, otherwise STOP and return $curr\_obj$.

9 If $best\_obj < curr\_obj$ set $curr\_obj = best\_obj$, $curr\_fac = best\_fac$ and $Z = Z_{best}$ and go to Step 2, otherwise STOP and return $curr\_obj$.

An example is given for three facilities and $k = 2$ for *2-Swap* in Figure 1. We select for the solution on the left the facilities j_1 and j_2 and the unoccupied destination locations v_1 and v_2. The selected order of the destination locations is v_2, v_1, which results in the relocation of the facilities to the destination locations shown on the right of Figure 1.

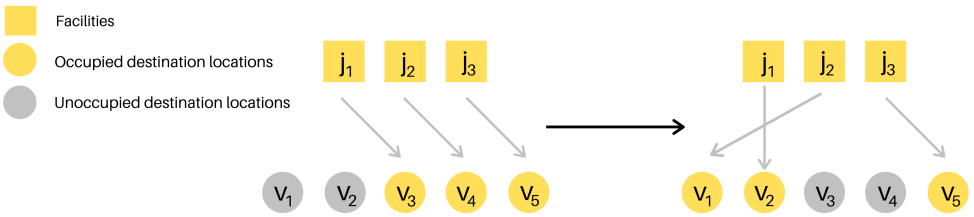

Figure 1: Example of a swap for the *2-Swap* heuristic

For this paper only *1-Swap* is implemented. We use the initial vertex of a facility as their destination location as the intial solution, so that no facility travels in this solution. The stopping criterion described in Algorithm 1, which stops if a local minimum is found, is adapted from Ahmadian et al. (2013). The client assignment subproblem is solved as described in Section 5.1. Observe that when solving the client assignment subproblem during the search, if a client was

previously assigned to a location that is still part of the set of occupied destination locations, it is sufficient to only look for a potential improvement for the client in the set of newly added occupied destination locations.

It should be noted that for the implementation of *n-SwapFI* the order of searching the neighbourhood influences the results. For our paper the order is obtained through recursively selecting subsets of facilities, starting with $k = 1$ and ending at $k = n$, and for every selected subset of unoccupied destination locations, recursively selecting subsets of k unoccupied destination locations. The subsets are recursively selected starting from the first facility or unoccupied destination location in their respective sets. The order of the k unoccupied destination locations is again recursively chosen through swapping a destination location with all possible positions it can take, starting with the last destination location in the subset.

5.3 *n-OptSwap* local search heuristic

n-OptSwap of Halper et al. (2015) defines a neighbour as a swap between k occupied destination vertices and k unoccupied destination vertices. We therefore do not search the permutations in Step 4 of the *n-Swap* heuristic in Algorithm 1, but instead we solve the assignment of the facilities through the facility assignment subproblem. This is done through the implementation of the Hungarian algorithm with a $O(|X|^3)$ complexity, introduced by Edmonds and Karp (1972) and adapted by Halper et al. (2015). Therefore a neighbour can be characterised with the set of occupied destination locations only. A swap between two neighbours is shown in Figure 2. We see that the occupied destination locations are initially vertex 3, 4 and 5. The swap is between the locations 3 and 4, and 1 and 2, which results in the solution on the right.

Figure 2: Example of a swap for the *2-OptSwap* heuristic

The Hungarian algorithm solves, given an edge weighted graph $G(F \cup Z, E)$, the least cost weighted bipartite matching problem. In this algorithm we label the facilities l_j for all $j \in F$ and the unoccupied vertices l_v for all $v \in Z$. For a feasible labeling it holds that $l_j + l_v \leq d_{j,v}$. We also define the tight graph $G_L(V, E)$ as the graph containing all the tight edges given a labeling L, namely for which $l_j + l_v = d_{j,v}$. This algorithm essentially tries to find a matching on a graph with only tight edges, which gives us a least cost matching, because given the definition of the labels $\sum_{j \in F} l_j + \sum_{v \in Z} l_v$ is a lower bound on the cost of the perfect matching. Therefore if the total sum of the labeling is equal to the cost $(\sum_{j \in F} \sum_{v \in Z} d_{j,v})$, we obtain a minimum cost perfect matching. A pseudocode for the Hungarian algorithm can be found in Appendix A, Algorithm A1. Halper et al. (2015) also take three different steps to further shorten the running time of the Hungarian algorithm (and *n-OptSwap*), which we elaborate more on in Appendix A.1.

We again implement both *n-OptSwapBI* and *n-OptSwapFI*. The order in which the neighbourhood is searched is identical to the order of the selected subset of facilities and unoccupied vertices for the *n-SwapFI* heuristic. The initial solution of *n-OptSwap* and the stopping criterion is identical to that of *n-Swap*. Note that the initial solution is identical to selecting the facility vertices as the occupied vertices and solving the facility assignment subproblem, since holding the facilities on their initial facility vertices, gives a total weighted travelled distance of zero for the subproblem. Therefore the Hungarian algorithm does not have to be applied on the initial solution.

5.4 *n-SmartSwap* local search heuristic

n-SmartSwap of Halper et al. (2015) also swaps occupied destination vertices with unoccupied destination vertices, but the facility assignment subproblem is not solved in its entirety. Only for the facilities for which their assigned destination vertices are swapped, we perform the Hungarian algorithm, discussed in Section 5.3, with the selected unoccupied destination vertices. Similarly to *n-OptSwap* we reduce the running time using the steps described in Appendix A.1.

An example of a swap is given in Figure 3. We see that the occupied destination locations 3 and 4 get swapped with the unoccupied destination locations 1 and 2. Therefore facilities 1 and 2 do not have an assigned destination location anymore and the Hungarian algorithm will be applied on the facilities 1 and 2 and destination locations 1 and 2, to obtain the minimum matching between the facilities and destination locations.

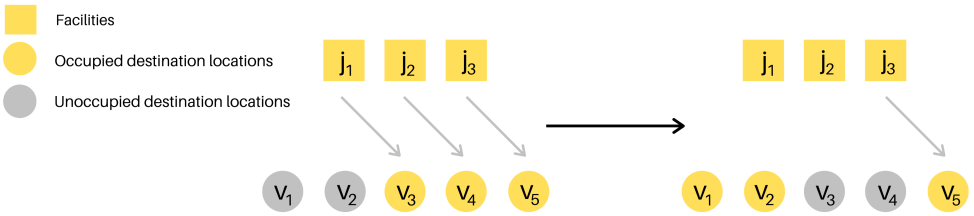

Figure 3: Example of a swap for the *2-SmartSwap* heuristic

We search for a solution in the neighbourhood until the first (FI) or best (BI) solution is attained. After this solution is obtained, the facility assignment subproblem is solved in its entirety for the facilities and occupied destination locations of the selected neighbour and the objective function $FA(Z) + CA(Z)$ is recalculated to form the final solution of one iteration. The initial solution and the stopping criterion is identical to the ones used so far for the other local search heuristics. Note that it is possible that the best solution of the neighbourhood does not improve the current solution during the search of the neighbourhood, but only after solving the facility assignment subproblem entirely it gives an improvement and consequently is still accepted. Again, observe that the initial solution is identical to selecting the set of facility vertices as the occupied destination locations and solving the facility assignment subproblem.

6 Tabu search solution approaches

This section uses Tabu search (TS) procedures to improve the local search heuristics of Halper et al. (2015). TS, introduced by Glover and Laguna (1998), is a meta-heuristic that guides local search heuristics to try and escape local optima. This is done through allowing non-improving solutions. Cycling back to local optima is prevented by memories, which are called the tabu lists. We refer to the implementation of TS for the three heuristics as *n-SwapTS*, *n-OptSwapTS* and *n-SmartSwapTS*. Again a five hours time limit is used for all of the heuristics and the best found solution in those five hours is used as the found solution if the time limit is reached. We first introduce two variants of the simple TS. After this we suggest several ways to implement a more thorough search of the solution space through two diversification techniques.

6.1 Two simple Tabu searches

For simple TS a few basic components are needed (Gendreau, 2003). First are the search space and neighbourhood structure. We use the search space and neighbourhood structures of *n-Swap*, *n-OptSwap* and *n-SmartSwap* introduced in Section 5. Secondly, tabus need to be determined. Tabus are certain solutions or moves that are prohibited. We implement two different tabus, which we call TS1 and TS2. For TS1 we consider solutions as a whole. We characterise these solutions through the facility assignment for *n-Swap* and the set of occupied destination locations for *n-SmartSwap* and *n-OptSwap*. These tabus are saved in tabu lists. The maximum size of these lists are set in advance. This ensures that cycling back to solutions will be avoided and a local optimum can be escaped. Gendreau (2003) state that besides cycling one of the properties of the tabus is that it encourages diversification. This can be done through defining a stricter tabu and prohibiting all solutions that have a component that is similar to a previous visited solution. Therefore we define for TS2 the following tabus: for *n-Swap* we define a separate tabu list for each facility, and regard each destination location that is previously swapped out for the particular facility as tabu for that particular facility. For *n-SmartSwap* and *n-OptSwap* we prohibit all previous destination locations that are swapped out from the occupied destination location set. Sometimes the tabu restriction defined is too strong. This especially holds for TS2. Therefore aspiration criteria are often used in TS. This is a criterion that, if it holds, a tabu active solution can still be selected and visited. Gendreau (2003) state that the most frequent implemented aspiration criterion is to allow for solutions that improve the best found solution so far. We implement this same aspiration criterion.

Simple TS searches the whole neighbourhood of a current solution and finds the best solution that is not tabu or satisfies the aspiration criterion, which will then be set as the new current solution. We continue the search from this current solution even if it does not improve our best found solution. If this solution does improve the best found solution, the solution is also saved as

the current best found solution. We evaluate the solutions through the total weighted travelled distance of a neighbour, which we want to minimise. We initialise the heuristic similarly to the local search heuristics described in Section 5. For TS1, we save the initial solution in the tabu list. Since TS2 considers swapped destination locations, this is not needed and we start with an empty tabu list. The stopping criteria we use for the heuristic is equal to the one used by Al-Sultan and Al-Fawzan (1999) and also described by Gendreau (2003) as the most generally used in the literature. We run the heuristic until the best found solution is not improved for a number of iterations or all solutions in the neighbourhood are tabu and do not satisfy the aspiration criterion. A more detailed description of the heuristic can be found in Appendix A, Algorithm A2.

6.1.1 Parameter testing

Two parameters need to be determined for these heuristics, the size of the tabu list and the maximum number of non-improving iterations. The parameters are set through experimental analysis. For the maximum number of non-improving iterations r it holds that the larger this number is, the better the found solution is, but the longer the running time. Therefore a trade-off between a high quality solution and a shorter running time has to be made. We only choose for an increase of r, if the proportional increase of the running time is smaller than the proportional decrease of the optimality gap in percentages. That is if $\frac{RT_1}{RT_2} < \frac{gap\%_2}{gap\%_1}$, we choose r_2 over r_1.

For the tabu list size l, it holds that we have to find a size that is large enough to avoid cycling, but small enough so that it is not too strong and drives the search to a different part of the solution space without searching the current part thoroughly. Al-Sultan and Al-Fawzan (1999) elaborate that a small tabu list size focuses more on intensification and a large size more on diversification. Therefore some intermediate values for the tabu list size must be found that produces the smallest optimality gap. Burke and Bykov (2008) state that the increase of the length of the tabu list always causes a higher computational cost per iteration, because more tabu solutions need to be checked. This does not mean that the total running time will necessarily be longer, because the number of iterations until termination could still differ.

We set the parameters of TS1 and TS2 independently from each other. This is because according to Glover and Taillard (1993) the appropriate size for the tabu list is dependent on the strength of the tabu restrictions. Lastly, the parameters for TS are set using half of the available instances, so overfitting is avoided, as recommended by Gendreau (2003).

6.2 Diversification strategies

Gendreau (2003) states that one of the main problems of TS is that it tends to be too "local", and therefore only focuses on a small portion of the solution space. This could cause the heuristic to miss the global optimum. Diversification methods try to avoid this, by forcing the heuristic to explore previously unexplored areas. In addition, Gendreau (2003) also mentions that

intensification methods are not necessarily needed in TS. Therefore we decide to only focus on diversification methods in this paper. We introduce two different diversification methods based on the diversification methods mentioned by Gendreau (2003) as two of the most commonly used.

6.2.1 Restart diversification

A restart diversification method is proposed by Soriano and Gendreau (1996). The restart diversification method forces a not yet visited component into a current solution and proceeds the search from this newly formulated solution. For our restart method, we formulate an entirely new solution if possible. This is done through selecting $|F|$ destination locations out of the set of not yet visited locations V^* and setting the set of occupied locations Z equal to the set of these selected locations. The selected destination locations $\hat{v}^* \in V^*$, are selected based on proximity to the clients. We choose the destination location with the smallest average distance to clients, calculated with $\frac{1}{|C|}\sum_{i \in C} d_{i,v^*}$ for all destination location $v^* \in V^*$. For *n-Swap* we assign $\hat{v}^*$ to the closest (not yet assigned) facility to obtain a solution. For *n-OptSwap* and *n-SmartSwap* Z gives an unique solution in the solution space and we assign the locations $\hat{v}^*$ to the facilities using the Hungarian algorithm, discussed in Section 5.3. If the number of not yet visited locations is smaller than the number of facilities ($|V^*| < |F|$), we assign for *n-Swap*, after assigning all selected locations $\hat{v}^*$, the not assigned facilities to the same destination location as the previous solution and for *n-OptSwap* and *n-SmartSwap* we select out of the previous occupied locations set, the locations with the smallest average distance to the facilities, calculated with $\frac{1}{|F|}\sum_{j \in F} d_{j,v}$ for all destination locations $v \in Z$ of the last visited solution.

We implement the restart diversification method after the simple TS is finished for the stopping criteria defined in Section 6.1. After constructing a new solution we proceed back to the simple TS (with the previous filled tabu list). This is done until all destination locations are visited at least once. We update the set of not yet visited destination locations V^* after a solution is selected from the neighbourhood and also each time a new solution is constructed. A detailed description can be found in Appendix A.2, Algorithm A3. We use the same parameters for simple TS as previously set. This is done similarly to Sarmady (2012), as he tests the diversification methods independently from all other settings in his TS.

6.2.2 Continuous diversification

The continuous diversification method performs a more subtle diversification procedure, as it does not restart from a whole new solution. Instead it modifies the function used to evaluate the solution, so that it guides the heuristic to previously unexplored areas. This is done through penalising the objective function if the occupied destination location that is swapped into the solution had been previously present a number of times in a visited solution. We use objective

function (19) for this purpose, with $numPres_{\tilde{v}}$ being the number of times that $\tilde{v}$ has been present in a visited solution and $\tilde{v}$ the currently swapped destination location. A logarithmic function is used for the penalisation in this objective function. The log is used so that the added term does not have a too large effect on the objective value, as discussed by Soriano and Gendreau (1996). Similarly to the restart method, we use the previously set parameters for simple TS.

$$\min \sum_{j \in F} \sum_{v \in V} w_j d_{j,v} y_{j,v} + \sum_{i \in C} \sum_{v \in V} u_i d_{i,v} x_{i,v} + \log(1 + numPres_{\tilde{v}}) \tag{19}$$

7 Computational Results

This section presents the results of the exact models, local search heuristics followed by the TS approaches. All computational results are obtained with an Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz and 16GM RAM running 64-bit Windows 10. The programs are coded in Java and compiled using Eclipse IDE 2018-12. A detailed description of the code can be found in Appendix C. The IP and LP formulations are solved using IBM ILOG CPLEX 12.6. The Wilcoxon signed-rank test by Wilcoxon (1992) is used to test whether the difference between the results are significantly different, which is a non-parametric test to compare the mean of two samples. We use a significance level of 0.05 throughout the analysis. We first introduce the test instances that are used to evaluate the models and heuristics.

7.1 Test instances

The data that is used for the implementation of this paper is retrieved from Raghavan et al. (2019). This data set contains identical instances to the 40 instances used by Halper et al. (2015), called *pmed* instances. For the purpose of their paper, Raghavan et al. (2019) generate a total of five extra instances. These are removed from the data set, so we end up with the 40 *pmed* instances of Halper et al. (2015). Table 1 gives an overview of the the number of destination locations $|V|$, facilities $|F|$ and clients $|C|$ per instance and the client weight u_i over all the instances, as well as the average client weight per instance. All location destinations are initial locations of a facility or client. The facility weights are all equal to 1.0 for the instances. In Table B1 of Appendix B.1 the number of destination locations, facilities, clients and average client weight per instance is given.

For TS we use the instances with an uneven number to set the parameters and the instances with an even number are used to evaluate the heuristics. The alternated selection is chosen so that we obtain a variety of instances that are representative of the whole *pmed* data set.

Table 1: Statistics of *pmed* data set

	$\|V\|$	$\|F\|$	$\|C\|$	u_i	**Avg.** u_i
Min.	100	5	100	1.0	1.09
Max.	900	200	900	1.90	1.35
Avg.	460	46.13	460	1.22	1.21

7.2 Exact solution approaches

The results for the two IPs described in Section 4 show that IP2 has a shorter running time on average compared to IP1. A summary of the results is provided in Table 2. Out of the 40 instances, instance 36 did not provide a solution for IP1 within five hours. Therefore we extract this instance for the comparison of the models. We see that the average running time of IP2 is more than four times as short as the average running time of IP1. Table 3 gives the gap (in percentages of the optimal value) between the LP2 objective value and the optimal value. It can be seen that LP2 provides the optimal (integer) solution for some of the instances as the minimum gap found is 0.00%. This is the case for a total of 19 out of 40 instances. The largest found gap is 1.19% with an average of 0.21%. All of these results largely agree with the results of Halper et al. (2015), except for the overall longer running times, likely caused by the use of a different hardware and software. Appendix B.2, Table B2 shows the objective values, running times and gaps of IP1, IP2 and LP2 for each instance. Since the results show that IP2 has a shorter average running time as opposed to IP1, we use the running time of IP2 as a benchmark for the running times of the heuristics in the remainder of our analysis.

Table 2: Statistics on the objective and running times (sec.) of IP1, IP2 for the instances $1-35$ and $37-40$

	IP1		**IP2**	
	Objective	**RT**	**Objective**	**RT**
Min.	2170.57	0.94	2170.57	0.31
Max.	14810.14	11554.07	14810.14	2262.73
Avg.	7057.23	1133.21	7057.23	272.41

Table 3: Statistics on the gap (%) and running times (sec.) of LP2

	LP2	
	Gap	**RT**
Min.	0.00	0.15
Max.	1.19	506.34
Avg.	0.21	97.04

7.3 Heuristic solution approaches

From the three local search heuristics described in Section 5 the *1-OptSwap* heuristics result in the smallest optimality gaps, but with a relative long running time compared to the other heuristics. The running times of *1-Swap* and *1-SmartSwap* are similar, but *1-SmartSwap* results in considerably smaller optimality gaps. The optimality gaps (in percentages) and the running times are given in Table 4, together with the running times of IP2. We see that both *1-SwapBI* and *1-SmartSwapBI* provide significantly smaller gaps compared to *1-SwapFI* and *1-SmartSwapFI*

respectively, with a p-value of 0.00. The smallest average gap is given by *1-OptSwapFI* with an optimality gap of 0.17%. After testing, we conclude that *1-OptSwapFI* and *1-OptSwapBI* significantly outperform all other four local search heuristics, as well as that *1-SmartSwapFI* outperforms *1-SwapFI*, and *1-SmartSwapBI* outperforms both *1-SwapBI* and *1-SwapFI*. It should be noted that the average running time of the *1-OptSwap* heuristics are large compared to the other heuristics and similar to the average running time of IP2.

Notably, the running times of all the local search heuristics are longer compared to the running times of Halper et al. (2015). This is likely because of the difference in hardware and software used and small differences in implementation of the heuristics. The average gaps of some of the heuristics differ from the results of Halper et al. (2015), with a decrease of the gap of 0.03% and 0.34% for 1-SmartSwapBI and 1-SwapFI respectively, and an increase of 0.01% and 0.25% for 1-OptSwapFI and 1-SmartSwapFI respectively. For the FI heuristics these discrepancies could be caused by the order in which the neighbourhood is searched. For *1-SmartSwapBI* the only identified potential cause of the difference in gaps is the use of a different initial solution. Appendix B.3 provides the optimality gaps and running times for every instance in Table B3. Summarised we conclude that *1-OptSwapFI* results in the smallest optimality gap, and *1-SmartSwapBI* results in a quality solution with a relatively short running time.

Table 4: Statistics on the optimality gap (%) and running times (sec.) of the local search heuristics and IP2

(a) Results of the BI heuristics

	1-SwapBI		***1-OptSwapBI***		***1-SmartSwapBI***		**IP2**
	Gap	**RT**	**Gap**	**RT**	**Gap**	**RT**	**RT**
Min.	0.00	0.03	0.00	0.04	0.00	0.04	0.31
Max.	2.82	547.83	1.02	2686.19	1.80	521.64	2262.73
Avg.	0.87	62.77	0.19	183.34	0.40	63.37	272.41

(b) Results of the FI heuristics

	1-SwapFI		***1-OptSwapFI***		***1-SmartSwapFI***		**IP2**
	Gap	**RT**	**Gap**	**RT**	**Gap**	**RT**	**RT**
Min.	0.00	0.03	0.00	0.05	0.00	0.03	0.31
Max.	7.79	1055.51	0.86	3314.52	9.21	745.23	2262.73
Avg.	2.64	86.03	0.17	269.10	1.15	65.25	272.41

A remark on the results presented in Table 4 can be the longer running times of the FI heuristics compared to the BI heuristics. This can be explained by the number of iterations each heuristic performs to reach a local minimum. Table 5 shows that the average number of iterations for each FI heuristic is large compared to the number of iterations for their BI counterpart. It is important to note that the average running time of one iteration for each FI heuristic is shorter compared to their BI heuristic. This is in line with the definition of FI, as it stops an iteration directly when an improvement is found as opposed to searching through all solutions as BI does.

Table 5: Statistics on the number of iterations and average running time (sec.) per iteration of the heuristics

(a) Results of the BI heuristics

	1-SwapBI		*1-OptSwapBI*		*1-SmartSwapBI*	
	#Iterations	**Avg. RT**	**#Iterations**	**Avg. RT**	**#Iterations**	**Avg. RT**
Min.	5	0.00	5	0.00	5	0.00
Max.	75	7.30	79	42.64	75	6.96
Avg.	26.85	1.07	28.10	3.01	27.35	1.06

(b) Results of the FI heuristics

	1-SwapFI		*1-OptSwapFI*		*1-SmartSwapFI*	
	#Iterations	**Avg. RT**	**#Iterations**	**Avg. RT**	**#Iterations**	**Avg. RT**
Min.	23	0.00	23	0.00	24	0.00
Max.	508	2.08	745	11.88	399	1.90
Avg.	128.45	0.31	174.80	0.80	105.08	0.30

7.4 Tabu search solution approaches

This section presents the results of the TS heuristics. We first discuss the parameter settings, determined using the odd numbered instances, and continue with the results of the even numbered instances for simple TS1, TS2 and the TS heuristics with the diversification methods.

7.4.1 Parameter settings

Through experimental analysis, discussed in Section 6.1.1, we obtain the parameter settings shown in Table 6. We see that TS1 requires less non-improving iterations compared to TS2. The tabu list size of TS1 is generally larger than for TS2. This is to be expected, since the tabu restriction of TS2 is stricter than the one of TS1. We found that a larger tabu list did not necessarily result in a longer running time. Therefore multiple appropriate intermediate values are tested and the one that resulted in the shortest running time is selected.

Table 6: Maximum number of non-improving iterations r and tabu list size l for simple TS

	Simple TS1			**Simple TS2**		
	1-Swap	*1-OptSwap*	*1-SmartSwap*	*1-Swap*	*1-OptSwap*	*1-SmartSwap*
r	3	6	6	44	10	60
l	20	200	53	5	100	14

7.4.2 Simple TS solution approaches

We evaluate our TS heuristics using the even numbered instances. The simple TS heuristics are a variant of the BI heuristics, since it searches the whole neighbourhood for the best (non-tabu) solution. Therefore we mainly compare the TS heuristics with the BI variants. We show the statistics of the results of the even instances for the BI heuristics and IP2 in Table 7.

Table 7: Statistics on the optimality gap (%) and running times (sec.) of the local search heuristics and IP2 for the even instances

	1-SwapBI		*1-OptSwapBI*		*1-SmartSwapBI*		**IP2**
	Gap	**RT**	**Gap**	**RT**	**Gap**	**RT**	**RT**
Min.	0.01	0.07	0.00	0.04	0.00	0.04	0.31
Max.	2.01	547.83	1.02	2686.19	1.60	521.64	3384.20
Avg.	0.89	80.47	0.23	248.66	0.44	79.67	455.54

From the results of the simple TS heuristics, we see a small decrease of the optimality gaps compared to their BI counterpart for all TS heuristics. On top of that, the TS2 heuristics results also show a smaller gap compared to their TS1 counterpart, but with a longer running time. We present the results of the simple TS heuristics in Table 8. It can be seen that for all except *1-OptSwapTS2* the average running time is still shorter than that of IP2. We note, though, that the maximum running time of *1-OptSwapTS1* is actually longer than the maximum running time of IP2. Moreover, we see that *1-SwapTS1* gives an improvement of 0.04% to *1-SwapBI*, with a slightly shorter running time. Besides that, all the TS heuristics have a smaller average gap compared to their BI counterpart. When we test the hypothesis whether the TS heuristics produce a smaller gap compared to their BI counterpart heuristics, it is significant for all the simple TS heuristics except for *1-SwapTS1*. This shows that the TS procedures do indeed improve the optimality gap. Table 8 further shows that TS2 results in a smaller gap compared to TS1 for *1-Swap* and *1-SmartSwap*, both with a 0.02% decrease. The difference between the two *1-SmartSwap* heuristics is also significant. We remark that the average running times of the TS2 heuristics are longer than the TS1 heuristics. Lastly, we want to mention that the *1-OptSwapTS* heuristics both improve the previous local search heuristic with the smallest gap (*1-OptSwapFI*). Appendix B.4 shows the results for each separate instance in Table B4. Following this section we analyse the two diversification methods and their effects on the two simple TS heuristics.

Table 8: Statistics on the optimality gap (%) and running times (sec.) of simple TS1 and simple TS2 heuristics for the even instances

(a) Results of the TS1 heuristics

	1-SwapTS1		**1-OptSwapTS1**		**1-SmartSwapTS1**	
	Gap	**RT**	**Gap**	**RT**	**Gap**	**RT**
Min.	0.01	0.07	0.00	0.06	0.00	0.06
Max.	2.01	388.63	0.84	3402.69	1.60	476.44
Avg.	0.85	74.37	0.13	310.60	0.37	91.47

Table 8: Statistics on the optimality gap (%) and running times (sec.) of simple TS1 and simple TS2 heuristics for the even instances, continued.

(b) Results of the TS2 heuristics

	1-SwapTS2		**1-OptSwapTS2**		**1-SmartSwapTS2**	
	Gap	**RT**	**Gap**	**RT**	**Gap**	**RT**
Min.	0.01	0.14	0.00	0.09	0.00	0.16
Max.	2.01	661.20	0.84	4735.80	1.55	729.19
Avg.	0.83	113.11	0.13	642.93	0.35	118.37

7.4.3 TS with diversification solution approaches

We now present the results of TS1 and TS2 heuristics with the two diversification methods discussed in Section 6.2. We show that the two diversification methods generally do not result in a smaller optimality gap despite both diversification methods having a longer running time compared to their simple TS heuristic counterparts.

Table 9 presents the optimality gap and running time of the TS heuristics using the restart diversification method, with TS1 in Table 9a and TS2 in Table 9b. It can be seen that all of the running times are longer compared to their simple TS heuristic counterparts. Additionally, a significant difference in the optimality gap is only seen in *1-SwapTS2*, with a decrease of 0.08%. However the running time of *1-SwapTS2* with the restart diversification method is more than four times the running time of simple *1-SwapTS2* and also longer than the average running time of IP2. Appendix B.4 displays the results of the restart diversification method per instance in Table B5.

Table 9: Statistics on the optimality gap (%) and running times (sec.) of TS1 and TS2 heuristics with the restart diversification method for the even instances

(a) Results of the TS1 heuristics

	1-SwapTS1		**1-OptSwapTS1**		**1-SmartSwapTS1**	
	Gap	**RT**	**Gap**	**RT**	**Gap**	**RT**
Min.	0.01	0.12	0.00	0.31	0.00	0.38
Max.	2.01	651.61	0.84	18000.19	1.60	4957.69
Avg.	0.85	106.60	0.13	1221.47	0.36	399.13

(b) Results of the TS2 heuristics

	1-SwapTS2		**1-OptSwapTS2**		**1-SmartSwapTS2**	
	Gap	**RT**	**Gap**	**RT**	**Gap**	**RT**
Min.	0.00	0.74	0.00	1.32	0.00	0.77
Max.	2.01	2604.03	0.84	18000.82	1.55	5957.69
Avg.	0.75	482.68	0.13	4297.67	0.34	904.94

Table 10 gives a summary of the results of TS1 and TS2 using the continuous diversification method. The results per instance can be found in Appendix B.4, Table B6. The running times

are generally longer compared to the running times of the simple TS methods, but shorter than the restart diversification method heuristics. We see a significant decrease of 0.07% in the gap for *1-SmartSwapTS2* compared to simple *1-SmartSwapTS2*, with a running time that is almost half of the average running time of IP2, but double the running time of simple *1-SmartSwapTS2*. We therefore conclude that out of the two diversification methods, only *1-SmartSwapTS2* with the continuous diversification method provides a relevant improvement of the optimality gap.

Table 10: Statistics on the optimality gap (%) and running times (sec.) of TS1 and TS2 heuristics with the continuous diversification method for the even instances

(a) Results of the TS1 heuristics

	1-SwapTS1		**1-OptSwapTS1**		**1-SmartSwapTS1**	
	Gap	**RT**	**Gap**	**RT**	**Gap**	**RT**
Min.	0.01	0.05	0.00	0.12	0.00	0.08
Max.	2.01	505.67	0.46	15598.40	1.57	569.95
Avg.	0.85	81.10	0.10	954.47	0.38	88.93

(b) Results of the TS2 heuristics

	1-SwapTS2		**1-OptSwapTS2**		**1-SmartSwapTS2**	
	Gap	**RT**	**Gap**	**RT**	**Gap**	**RT**
Min.	0.01	0.14	0.00	0.11	0.00	0.12
Max.	2.01	895.81	0.84	18000.42	1.55	1699.44
Avg.	0.82	137.02	0.13	1506.78	0.28	232.32

8 Conclusion

In this paper we have researched solution approaches to the mobile facility location problem (MFLP). We build upon the work of Halper et al. (2015), implementing the two exact models and local search heuristics of their paper. We observe that one of the local search heuristics produces high quality solutions with a running time that is four times less than the running time of the fastest exact model. Additionally, we have defined two simple tabu search (TS) procedures. Answering our research question, *"What is the effect of a Tabu search procedure on the results of local search heuristics for the MFLP?"*, we see that both simple TS procedures improve the local search heuristics, but generally also have a longer running time. Although, one of the simple TS heuristics actually produces a smaller gap with a similar running time to the local search heuristic without TS. We also have tested two methods to further try and improve the TS procedure. These methods force the TS to explore previously unexplored areas of the solution space. This generally did not improve the solution in spite of the longer running time. We conclude that the TS procedures show promise in improving the local search heuristics of Halper et al. (2015). These TS heuristics can still be developed in multiple ways, both obtaining lower optimality gaps and having a shorter running time.

8.1 Further research

The parameters that are currently used are set using a small sample out of the whole data set. These instances might not be representative for either the data set or the average MFLP problem. Halper et al. (2015) recognised similar problems, and also evaluated the heuristics on larger instances. More research into appropriate parameters using a larger and more representative sample will therefore be beneficial. Moreover, it is seen that the sizes of the instances and the number of facilities per instance differ over a large range. Focusing on groups of instances separately with similar characteristics may result in more favorable parameters.

More research into the TS procedure is also necessary. A suggestion would be to only consider part of the neighbourhood for each iteration to shorten the running time or to use a variant tabu list size which will generally result in higher quality solutions, as Glover and Taillard (1993) discuss. The diversification methods should also be a point of research, especially since this is one of the main problems of TS (Gendreau, 2003). Other diversification methods, like strategic oscillation, can be considered, but we can also improve the two defined diversification methods, e.g. modifying the added term of the continuous method. Also adding an intensification phase to the TS would likely be valuable. This is especially true for the two heuristics that do not perform a thorough search of all the solutions for a particular occupied destination location set. An idea would be to use the heuristic that goes through all the solutions for a given set for the intensification phase of the previous two heuristics, so that when a promising region is found, all the potential optimal solutions are searched in that region. Lastly we suggest not limiting to TS and also exploring other meta-heuristics that can potentially improve the local search heuristics.

References

Ahmadian, S., Friggstad, Z., & Swamy, C. (2013). Local-search based approximation algorithms for mobile facility location problems. In *Proceedings of the twenty-fourth annual acm-siam symposium on discrete algorithms* (pp. 1607–1621).

Al-Sultan, K. S., & Al-Fawzan, M. A. (1999). A tabu search approach to the uncapacitated facility location problem. *Annals of Operations Research*, *86*, 91–103.

Burke, E. K., & Bykov, Y. (2008). A late acceptance strategy in hill-climbing for exam timetabling problems. In *Patat 2008 conference, montreal, canada* (pp. 1–7).

Demaine, E. D., Hajiaghayi, M., Mahini, H., Sayedi-Roshkhar, A. S., Oveisgharan, S., & Zadimoghaddam, M. (2009). Minimizing movement. *ACM Transactions on Algorithms (TALG)*, *5*(3), 1–30.

Edmonds, J., & Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, *19*(2), 248–264.

Friggstad, Z., & Salavatipour, M. R. (2011). Minimizing movement in mobile facility location problems. *ACM Transactions on Algorithms (TALG)*, *7*(3), 1–22.

Gendreau, M. (2003). An introduction to tabu search. In *Handbook of metaheuristics* (pp. 37–54). Springer.

Gendreau, M., & Potvin, J.-Y. (2010). *Handbook of metaheuristics* (Vol. 2). Springer.

Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization* (pp. 2093–2229). Springer.

Glover, F., & Samorani, M. (2019). Intensification, diversification and learning in metaheuristic optimization. *Journal of Heuristics*, *25*(4-5), 517–520.

Glover, F., & Taillard, E. (1993). A user's guide to tabu search. *Annals of operations research*, *41*(1), 1–28.

Halper, R., & Raghavan, S. (2011). The mobile facility routing problem. *Transportation Science*, *45*(3), 413–434.

Halper, R., Raghavan, S., & Sahin, M. (2015). Local search heuristics for the mobile facility location problem. *Computers & Operations Research*, *62*, 210–223.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671–680.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, *2*(1-2), 83–97.

Raghavan, S., Sahin, M., & Salman, F. S. (2019). The capacitated mobile facility location problem. *European Journal of Operational Research*, *277*(2), 507–520.

Sarmady, S. (2012). An investigation on tabu search parameters. *School of Computer Sciences, Universiti Sains Malaysia*, *11800*.

Soriano, P., & Gendreau, M. (1996). Diversification strategies in tabu search algorithms for the maximum clique problem. *Annals of Operations Research*, *63*(2), 189–207.

Voudouris, C., & Tsang, E. P. (2003). Guided local search. In *Handbook of metaheuristics* (pp. 185–218). Springer.

Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics* (pp. 196–202). Springer.

A Algorithm details

A.1 Local search heuristics

Algorithm A1: The Hungarian Algorithm

Input : An edge weighted graph $G(F \cup Z, E)$ with a feasible labeling of l_j for all $j \in F$ and l_v for all $v \in Z$ and an initial matching M

Output: A minimum weighted perfect matching

while *A perfect matching M is not yet found* **do**
  Find an unmatched vertex $j \in F$ and set $S := j$ and $T := \emptyset$
  Set $slack_v := max_{j \in S}\{l_j + l_v - d_{j,v}\}$ for all $v \in Z$
  while *M is not changed* **do**
    if $J_{G_L}(S) \neq T$ *with* $J_{G_L}(S) = \{v : (j, v) \in E_l, j \in S\}$ **then**
      Set $\delta := max_{v \in Z \setminus T} slack_y$
      Update the labels $l_j - \delta$ for all $j \in S$ and $l_v + \delta$ for all $v \in T$
      Update G_L to contain the additional tight edges under the new labeling
      Update $slack_v := slack_v - \delta$ for each $v \in Z$
    end
    Choose a $v \in J_{G_L}(S)$ and set $\hat{v}$ equal to the selected v
    if *$\hat{v}$ is matched in M with any vertex $\hat{j} \in F$* **then**
      Update $S := S \cup \{\hat{j}\}$ and $T := T \cup \{\hat{v}\}$
      Update $slack_v := max\{slack_v, l_j + l_v - d_{j,v}\}$ for all $v \in Z$
    else
      Find an augmenting path P from $j \in F$ to $v \in Z$ that alternates between edges in M and edges in $E_L \setminus M$ and for which $j, v \notin M$
      Change M by setting $M := (M \setminus P) \cup (P \setminus M)$
    end
  end
end

Reducing the running time of the local search heuristics

For *n-OptSwap* we take three steps, taken from Halper et al. (2015), to reduce the running time of the heuristic. The three steps are discussed below. We also implement Step 1 during the neighbourhood search and Step 2 when solving the facility assignment sub-problem in its entirety for *n-SmartSwap*.

1. Before solving the facility assignment sub-problem, solve the client assignment sub-problem first. If Z is the set of occupied vertices from the current solution and Z' is the set for the neighbour currently being searched, it holds that if $CA(Z') \geq FA(Z) + CA(Z)$ it can be concluded that the solution will not be an improvement on the current solution. In that case the floor assignment sub-problem is not solved to save computation time and the neighbour

is disregarded. This is often the case, because usually the number of clients is relatively large compared to the number of facilities.

2. Use the solution of the last found solution to initialise the labels and matching for the Hungarian algorithm of the searched neighbour and therefore "not starting from scratch". Since the goal is to obtain a labeling that is equal to the distance on the edge, we want to start with the largest feasible labeling (and thus the closest to a distance on an edge) found. To obtain this labeling note that to obtain the new neighbour Z', it entails swapping $v \in Z$ with $\hat{v} \in Z' \setminus Z$ and $d_{j,v}$ to $d_{j,\hat{v}}$ on the edges for every $j \in F$ to obtain the new weighted graph $G(F \cup Z', E)$. Therefore a feasible labeling would be $l_{\hat{v}} = l_v + min_{j \in F}\{d_{j,\hat{v}} - l_j - l_v\}$ for the destination locations $\hat{v} \in Z' \setminus Z$. Afterwards we increase the labels of all $j \in F$ by setting with $l_j = l_j + min_{v \in Z'}\{d_{j,v} - l_j - l_v\}$. Obtaining the initial labeling we set the initial matching equal to the matching of the last found solution and remove the edges that are no longer tight with the new labeling.

3. Check during the execution of the Hungarian Algorithm, whether the searched neighbour would provide an improved solution to the best found solution so far in the neighbourhood. This is done by checking whether $FA(Z'') + CA(Z'') - CA(Z') > \sum_{j \in F} l_j + \sum_{v \in Z'} l_v$ after every update of the labels (in line 7 of Algorithm A1), with Z'' being the neighbour with the best solution found so far in the neighbourhood of the current solution and Z' the searched neighbour. If this inequality does not hold, we can conclude that no improved solution will be found and the Hungarian Algorithm can be terminated. To justify this note that an improved solution is found if $FA(Z'') + CA(Z'') > FA(Z') + CA(Z')$ and that $FA(Z') \geq \sum_{j \in F} l_j + \sum_{v \in Z'} l_v$, since the labels form a dual feasible solution to the facility assignment sub-problem. Therefore it has to hold that $FA(Z'') + CA(Z'') - CA(Z') > \sum_{j \in F} l_j + \sum_{v \in Z'} l_v$ to find an improved solution.

We initialise the labeling of the initial solution as $l_v = 0$ for all $v \in Z$ and $l_j = \max_{v \in Z} d_{j,v}$ for all $j \in F$, so it can be used to solve the neighbourhood solutions. When running the Hungarian algorithm partially for *n-SmartSwap* we use an initial labeling $l_{\hat{v}} = 0$ for all $\hat{v} \in Z' \setminus Z$ and $l_{\hat{j}} = \max_{\hat{v} \in Z' \setminus Z} d_{\hat{j},\hat{v}}$ for all $\hat{j} \in F'$, with F' being the set of facilities with an assigned destination location after the swap. An initial matching is obtained through selecting for each facility $\hat{j} \in F'$, a $\hat{v} \in Z' \setminus Z$ for which the edge between those two vertices is tight and $\hat{v}$ is not yet assigned to another facility. If no such $\hat{v}$ exists, the facility $\hat{j}$ is not matched for the initial matching.

A.2 Tabu search

Algorithm A2: Simple Tabu search

Input : Initial solution S_0, tabu list size $maxSize_{TL}$, maximum number of non-improving iterations $MaxIter$

Output: Best found solution $S*$

```
1  Current solution S := S_0
2  Best found solution S* := S
3  while numNoImpr < MaxIter do
4  |  Find the best solution in the neighbourhood that is not tabu or fits the aspiration criteria S_iter
5  |  if no S_iter is found then
6  |  |  break
7  |  end
8  |  S := S_iter
9  |  if S < S* then
10 |  |  S* := S
11 |  |  numNoImpr = 0
12 |  else
13 |  |  numNoImpr = numNoImpr + 1
14 |  end
15 |  Update the tabu list
16 |  if size_TL > maxSize_TL then
17 |  |  remove earliest entry in tabu list
18 |  end
19 end
20 return S*
```

Algorithm A3: Restart diversification method

Input : Initial solution S_0, tabu list size $maxSize_{TL}$, maximum number of non-improving iterations $MaxIter$

Output: Best found solution $S*$

Current solution $S := S_0$
Best found solution $S^* := S$
The set of not yet visited destination locations $V^* := V \setminus Z$, with Z the set of occupied destination locations of S and V the set of destination locations
while $V^* \neq \emptyset$ **do**
    Execute the TS algorithm (in Algorithm A2), additionally perform $V^* := V^* \setminus Z$ for every solution S
    if $V^* \neq \emptyset$ **then**
        Find a new solution S_{new}, using V^*
        $S := S_{new}$
        $V^* := V^* \setminus Z$
        if $S < S^*$ **then**
            $S^* := S$
            $numNoImpr = 0$
        else
            $numNoImpr = numNoImpr + 1$
        end
        Update the tabu list if needed
        if $size_{TL} > maxSize_{TL}$ **then**
            remove earliest entry in tabu list
        end
    end
end
return S^*

B Computational results per instance

B.1 Test instances analysis

Table B1: Statistics per instance of the *pmed* data set

Instance	$\|V\|$	$\|F\|$	$\|C\|$	**Avg.** u_i
1	100	5	100	1.19
2	100	10	100	1.09
3	100	10	100	1.25
4	100	20	100	1.27
5	100	33	100	1.20
6	200	5	200	1.14
7	200	10	200	1.24

Continued on next page

Table B1 – *Continued from previous page*

Instance	$\|V\|$	$\|F\|$	$\|C\|$	**Avg.** u_i
8	200	20	200	1.19
9	200	40	200	1.30
10	200	67	200	1.22
11	300	5	300	1.09
12	300	10	300	1.33
13	300	30	300	1.12
14	300	60	300	1.10
15	300	100	300	1.12
16	400	5	400	1.18
17	400	10	400	1.16
18	400	40	400	1.10
19	400	80	400	1.20
20	400	133	400	1.27
21	500	5	500	1.19
22	500	10	500	1.25
23	500	50	500	1.33
24	500	100	500	1.35
25	500	167	500	1.14
26	600	5	600	1.34
27	600	10	600	1.32
28	600	60	600	1.26
29	600	120	600	1.33
30	600	200	600	1.11
31	700	5	700	1.24
32	700	10	700	1.13
33	700	70	700	1.31
34	700	140	700	1.21
35	800	5	800	1.18
36	800	10	800	1.28
37	800	80	800	1.12
38	900	5	900	1.34
39	900	10	900	1.09
40	900	90	900	1.28

B.2 Exact solution approaches

Table B2: Objective values or gap (%) and running time (sec.) of all instances for IP1, IP2 and LP2

	IP1		**IP2**		**LP2**	
Instance	**Objective**	**RT**	**Objective**	**RT**	**Gap**	**RT**
1	7231.77	0.94	7231.77	1.12	0.63	0.28
2	4914.03	1.11	4914.03	0.31	0.00	0.18
3	5792.74	4.45	5792.74	3.02	0.00	0.18
4	4748.11	2.38	4748.11	0.38	0.00	0.19
5	2170.57	2.98	2170.57	0.50	0.00	0.15
6	8958.09	4.38	8958.09	5.59	0.02	6.62

Continued on next page

Table B2 – *Continued from previous page*

	IP1		**IP2**		**LP2**	
Instance	**Objective**	**RT**	**Objective**	**RT**	**Gap**	**RT**
7	7241.09	2.23	7241.09	3.14	0.00	3.27
8	6077.76	2.98	6077.76	2.64	0.00	2.63
9	4348.86	4.28	4348.86	3.55	0.00	1.69
10	2377.40	5.41	2377.40	1.36	0.00	11.15
11	8444.63	14.04	8444.63	24.37	0.13	15.73
12	9219.27	12.05	9219.27	16.72	0.06	12.65
13	5487.02	15.14	5487.02	13.89	0.01	7.20
14	3963.37	14.46	3963.37	5.83	0.00	4.24
15	2642.84	18.65	2642.84	3.59	0.00	1.83
16	9655.15	56.44	9655.15	72.22	0.87	47.59
17	8300.98	56.51	8300.98	52.83	0.42	34.50
18	5844.42	32.82	5844.42	21.89	0.00	20.16
19	4229.19	32.67	4229.19	16.82	0.00	16.26
20	3178.70	57.18	3178.70	16.71	0.00	7.36
21	10908.32	23.24	10908.32	63.82	0.00	54.99
22	10856.89	453.75	10856.89	194.70	0.60	141.69
23	6756.93	50.05	6756.93	20.22	0.00	40.23
24	4782.64	163.31	4782.64	32.98	0.00	23.30
25	3033.29	1667.34	3033.29	47.36	0.00	10.80
26	13314.31	386.94	13314.31	344.54	0.71	179.02
27	11199.97	174.67	11199.97	258.01	0.12	192.04
28	6133.26	144.48	6133.26	52.51	0.02	47.30
29	4756.74	1038.09	4756.74	168.08	0.02	29.81
30	3151.90	7990.85	3151.90	105.52	0.00	13.96
31	12524.16	336.79	12524.16	634.25	0.64	210.86
32	10743.65	499.98	10743.65	627.45	0.12	343.47
33	6740.80	270.09	6740.80	76.28	0.00	86.03
34	4507.58	5168.37	4507.58	257.02	0.02	53.03
35	12408.12	962.31	12408.12	1044.25	0.98	298.22
36	-	-	12943.36	3384.20	0.02	151.62
37	6296.17	2582.40	6296.17	245.30	1.19	498.25
38	14810.14	3538.20	14810.14	2262.73	1.09	460.08
39	10329.38	6849.13	10329.38	2216.95	0.55	506.34
40	7151.86	11554.07	7151.86	1705.51	0.09	346.67

B.3 Heuristic solution approaches

Table B3: Gap (%) and running time (sec.) of all instances for the local search heuristics

	1-SwapBI		*1-SwapFI*		*1-OptSwapBI*		*1-OptSwapFI*		*1-SmartSwapBI*		*1-SmartSwapFI*	
Instance	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**
1	0.32	0.07	0.16	0.07	0.00	0.06	0.00	0.05	0.00	0.09	0.38	0.04
2	0.31	0.07	3.56	0.07	0.00	0.06	0.00	0.06	0.00	0.09	0.00	0.06
3	0.00	0.03	3.31	0.03	0.00	0.04	0.00	0.05	0.00	0.05	1.28	0.03
4	1.30	0.07	4.31	0.07	0.31	0.10	0.16	0.17	0.82	0.08	2.46	0.08
5	0.58	0.09	6.79	0.08	0.58	0.27	0.00	0.29	0.58	0.12	4.98	0.10
6	0.84	0.09	0.50	0.11	0.00	0.04	0.00	0.05	0.00	0.04	0.00	0.05
7	0.94	0.13	0.83	0.30	0.00	0.13	0.00	0.28	0.00	0.13	0.00	0.27
8	2.01	0.38	3.80	0.49	0.03	0.43	0.00	0.65	0.03	0.45	2.62	0.30
9	1.97	1.69	5.31	1.55	0.24	2.05	0.52	3.48	0.08	1.70	0.39	1.53
10	1.60	2.49	7.79	2.33	0.22	9.51	0.15	10.59	1.60	2.43	2.86	2.11
11	0.86	0.07	1.24	0.12	0.86	0.07	0.86	0.08	0.86	0.08	0.00	0.11
12	0.26	0.41	1.25	1.42	0.00	0.44	0.00	0.95	0.00	0.42	0.00	0.72
13	2.05	1.96	1.42	4.56	0.02	3.21	0.00	5.47	0.38	2.15	0.25	3.16
14	1.57	10.61	4.96	8.02	0.63	20.68	0.40	20.77	1.06	10.08	1.14	7.39
15	0.00	16.60	2.52	11.13	0.00	36.45	0.22	37.36	0.00	16.83	1.84	11.46
16	0.09	0.19	0.36	0.35	0.00	0.25	0.00	0.27	0.06	0.16	0.00	0.24
17	0.36	0.48	1.02	1.77	0.00	0.72	0.00	1.40	0.00	0.47	0.00	0.77
18	0.29	11.00	3.06	18.57	0.05	13.30	0.30	23.50	0.08	10.38	0.83	11.34
19	2.82	36.43	5.99	36.26	0.36	72.44	0.23	106.59	1.80	37.00	2.19	37.23
20	0.97	60.49	5.45	39.87	0.84	289.14	0.40	349.70	0.97	61.74	3.46	37.15
21	0.04	0.25	0.22	0.56	0.00	0.30	0.00	0.50	0.00	0.27	0.00	0.40
22	0.99	1.55	1.20	3.48	0.00	1.90	0.72	2.43	0.00	1.58	0.72	1.60
23	1.98	34.82	4.17	62.87	0.09	47.96	0.35	90.82	0.49	35.13	0.59	42.52
24	1.77	96.72	5.30	112.83	0.28	239.29	0.28	320.53	0.65	101.52	1.30	107.20

Continued on next page

Table B3 – *Continued from previous page*

	1-SwapBI		*1-SwapFI*		*1-OptSwapBI*		*1-OptSwapFI*		*1-SmartSwapBI*		*1-SmartSwapFI*	
Instance	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**
25	0.96	168.28	3.59	124.48	0.45	1130.84	0.41	1216.14	0.86	178.31	1.55	127.65
26	0.60	0.42	0.55	0.56	1.02	0.47	0.48	0.52	0.48	0.42	0.48	0.44
27	0.00	1.72	0.82	5.68	0.00	1.95	0.08	3.25	0.00	1.72	0.08	3.24
28	1.40	76.75	2.26	153.39	0.02	99.66	0.00	228.29	0.54	71.48	9.21	33.31
29	1.23	227.06	3.53	223.53	0.17	506.91	0.27	706.53	1.02	232.65	1.25	191.01
30	0.37	376.81	3.97	285.11	0.23	2686.19	0.20	3314.52	0.33	383.70	2.09	271.26
31	0.02	0.51	0.15	1.69	0.02	0.60	0.00	1.48	0.02	0.53	0.00	1.34
32	0.26	2.19	0.67	10.20	0.02	2.40	0.02	8.10	0.02	2.08	0.02	5.72
33	1.44	149.43	3.43	309.28	0.10	202.75	0.28	428.32	0.73	149.41	0.49	189.92
34	0.78	417.05	3.55	469.69	0.31	881.86	0.26	1180.63	0.60	420.88	2.01	400.30
35	0.01	0.95	0.00	3.30	0.00	1.03	0.00	1.59	0.00	0.92	0.00	1.52
36	0.72	2.68	0.52	13.46	0.34	3.78	0.07	12.98	0.53	2.65	0.07	10.61
37	1.17	255.72	4.22	456.13	0.02	347.16	0.20	909.07	0.48	279.11	0.82	350.34
38	0.01	1.65	0.05	4.09	0.00	1.77	0.00	3.10	0.00	1.63	0.00	2.84
39	0.37	4.93	0.41	18.40	0.00	5.57	0.00	12.41	0.00	4.82	0.00	9.64
40	1.70	547.83	3.52	1055.51	0.31	721.97	0.03	1760.88	1.07	521.64	0.69	745.23

B.4 Tabu search solution approaches

Table B4: Gap (%) and running time (sec.) of even instances for simple TS1 and TS2

	1-SwapTS1		**1-SwapTS2**		**1-OptSwapTS1**		**1-OptSwapTS2**		**1-SmartSwapTS1**		**1-SmartSwapTS2**	
Instance	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**
2	0.31	0.13	0.31	0.14	0.00	0.12	0.00	0.09	0.00	0.06	0.00	0.23
4	1.30	0.07	1.30	0.29	0.16	0.23	0.16	0.27	0.82	0.14	0.82	0.16
6	0.84	0.11	0.61	0.16	0.00	0.06	0.00	0.09	0.00	0.09	0.00	0.34
8	2.01	0.43	2.01	1.42	0.03	0.52	0.03	0.65	0.03	0.68	0.03	2.11
10	1.60	2.51	1.55	3.64	0.12	14.58	0.12	30.68	1.60	2.76	1.55	3.11

Continued on next page

Table B4 – *Continued from previous page*

	1-SwapTS1		1-SwapTS2		1-OptSwapTS1		1-OptSwapTS2		1-SmartSwapTS1		1-SmartSwapTS2	
Instance	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**
12	0.26	0.49	0.26	2.04	0.00	0.54	0.00	0.88	0.00	0.58	0.00	2.06
14	1.57	10.44	1.57	28.21	0.07	25.65	0.07	61.77	1.06	11.17	1.03	18.16
16	0.09	0.25	0.09	1.69	0.00	0.37	0.00	0.61	0.00	0.45	0.00	2.37
18	0.29	12.62	0.29	30.33	0.05	14.20	0.05	21.35	0.08	12.38	0.08	40.24
20	0.97	57.63	0.97	158.23	0.84	355.02	0.84	1931.30	0.97	70.02	0.87	99.92
22	0.99	1.84	0.99	6.86	0.00	2.19	0.00	2.59	0.00	2.48	0.00	8.96
24	1.77	94.51	1.77	164.70	0.28	293.45	0.28	836.54	0.65	111.76	0.65	186.80
26	0.23	0.79	0.23	3.26	0.00	0.92	0.00	1.15	0.00	1.11	0.00	4.34
28	1.40	71.93	1.40	127.48	0.02	107.64	0.02	113.88	0.54	73.76	0.54	146.42
30	0.37	343.68	0.29	463.14	0.13	3402.69	0.23	4735.80	0.33	449.57	0.26	439.11
32	0.24	2.98	0.24	9.94	0.00	3.43	0.00	4.13	0.00	3.66	0.00	14.04
34	0.78	388.63	0.78	661.20	0.29	1154.90	0.31	3682.94	0.60	476.44	0.60	729.19
36	0.22	5.93	0.22	16.55	0.34	4.78	0.34	5.97	0.00	7.18	0.00	22.42
38	0.01	2.09	0.01	9.80	0.00	2.62	0.00	3.41	0.00	2.78	0.00	13.29
40	1.70	490.43	1.68	573.18	0.28	828.11	0.23	1424.52	0.80	602.41	0.52	634.07

Table B5: Gap (%) and running time (sec.) of even instances for TS1 and TS2 with the restart diversification method

	1-SwapTS1		1-SwapTS2		1-OptSwapTS1		1-OptSwapTS2		1-SmartSwapTS1		1-SmartSwapTS2	
Instance	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**
2	0.31	0.12	0.31	0.89	0.00	0.47	0.00	1.67	0.00	0.45	0.00	1.33
4	1.30	0.13	1.30	0.74	0.16	0.31	0.16	1.32	0.82	0.38	0.62	0.77
6	0.84	0.92	0.00	7.93	0.00	1.28	0.00	2.77	0.00	1.54	0.00	10.69
8	2.01	1.07	2.01	8.00	0.03	2.03	0.03	6.92	0.03	1.63	0.03	12.02
10	1.60	2.76	1.55	7.48	0.12	583.06	0.12	5079.60	1.60	3.80	1.55	9.64
12	0.26	2.79	0.26	30.30	0.00	6.98	0.00	14.00	0.00	5.35	0.00	40.60
14	1.57	11.67	1.57	40.85	0.07	46.15	0.07	136.80	1.06	16.49	1.03	64.82
16	0.09	6.27	0.00	78.53	0.00	19.03	0.00	26.07	0.00	14.49	0.00	107.75

Continued on next page

Table B5 – *Continued from previous page*

	1-SwapTS1		1-SwapTS2		1-OptSwapTS1		1-OptSwapTS2		1-SmartSwapTS1		1-SmartSwapTS2	
Instance	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**
18	0.29	16.88	0.29	90.72	0.05	46.61	0.05	82.00	0.08	28.24	0.08	145.00
20	0.97	65.77	0.97	179.26	0.84	541.67	0.84	18000.02	0.97	84.52	0.87	415.00
22	0.99	14.82	0.45	166.18	0.00	36.92	0.00	119.56	0.00	36.68	0.00	221.27
24	1.77	107.21	1.77	309.72	0.28	542.72	0.28	18000.31	0.65	183.71	0.65	569.31
26	0.23	23.43	0.00	296.21	0.00	62.90	0.00	120.16	0.00	59.53	0.00	403.13
28	1.40	97.94	1.40	504.45	0.02	236.23	0.02	695.93	0.54	163.87	0.54	716.46
30	0.37	420.71	0.29	788.72	0.13	18000.19	0.23	18000.82	0.00	4957.69	0.26	5957.69
32	0.24	41.72	0.20	507.56	0.00	100.90	0.00	164.95	0.00	82.84	0.00	699.10
34	0.78	488.02	0.78	1568.63	0.29	2044.43	0.31	18000.07	0.60	747.35	0.60	1976.46
36	0.22	70.65	0.10	994.01	0.34	173.04	0.33	224.00	0.00	182.34	0.00	1298.83
38	0.01	107.56	0.00	1469.40	0.00	278.85	0.00	315.29	0.00	277.77	0.00	1971.99
40	1.70	651.61	1.68	2604.03	0.28	1705.57	0.23	6961.19	0.80	1133.91	0.52	3477.04

Table B6: Gap (%) and running time (sec.) of even instances for TS1 and TS2 with the continuous diversification method

	1-SwapTS1		1-SwapTS2		1-OptSwapTS1		1-OptSwapTS2		1-SmartSwapTS1		1-SmartSwapTS2	
Instance	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**
2	0.31	0.08	0.31	0.14	0.00	0.26	0.00	0.27	0.00	0.19	0.00	0.33
4	1.30	0.07	1.30	0.33	0.16	0.31	0.16	0.48	0.82	0.18	0.82	0.12
6	0.84	0.05	0.61	0.15	0.00	0.12	0.00	0.11	0.00	0.08	0.00	0.34
8	2.01	0.40	2.01	1.42	0.03	0.70	0.03	0.77	0.03	0.71	0.00	3.08
10	1.60	2.34	1.57	7.89	0.12	16.22	0.17	16.46	1.57	3.59	1.55	13.03
12	0.26	0.47	0.26	1.55	0.00	0.62	0.00	0.69	0.00	0.55	0.00	1.98
14	1.58	9.40	1.58	21.39	0.07	26.25	0.07	44.01	1.08	11.71	0.69	13.68
16	0.09	0.32	0.09	1.26	0.00	0.38	0.00	0.45	0.00	0.47	0.00	2.64
18	0.29	13.85	0.29	22.65	0.05	15.26	0.05	15.42	0.08	11.77	0.04	36.79
20	0.97	83.28	0.97	118.15	0.46	659.71	0.84	1117.13	0.97	70.61	0.67	133.77
22	0.99	2.36	0.99	5.24	0.00	2.31	0.00	3.35	0.00	2.12	0.00	8.84

Continued on next page

Table B6 – *Continued from previous page*

	1-SwapTS1		1-SwapTS2		1-OptSwapTS1		1-OptSwapTS2		1-SmartSwapTS1		1-SmartSwapTS2	
Instance	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**	**Gap**	**RT**
24	1.77	94.50	1.77	161.83	0.28	318.53	0.28	750.26	0.70	105.63	0.70	270.73
26	0.23	0.78	0.23	3.19	0.00	0.96	0.00	1.16	0.00	1.05	0.00	5.31
28	1.40	67.31	1.40	126.09	0.02	113.48	0.02	109.82	0.54	71.83	0.33	237.46
30	0.37	400.26	0.33	709.25	0.07	15598.40	0.13	18000.42	0.33	445.50	0.30	992.35
32	0.24	2.81	0.24	9.90	0.00	4.60	0.00	4.17	0.00	3.22	0.00	16.11
34	0.78	430.37	0.64	627.28	0.34	1123.15	0.31	7899.47	0.60	470.52	0.35	1176.35
36	0.22	5.62	0.22	17.01	0.34	5.05	0.34	6.02	0.00	6.42	0.00	21.16
38	0.01	2.04	0.01	9.88	0.00	2.76	0.00	3.47	0.00	2.57	0.00	12.95
40	1.70	505.67	1.53	895.81	0.11	1200.31	0.17	2161.60	0.81	569.95	0.12	1699.44

C Code description

We present brief descriptions of the five Java packages used in this paper and their classes in Table C7. The codes are available upon request.

Table C7: Codes descriptions of the classes in the Java packages

(a) Classes in the *data_analysis* package

Class name	**Description**
Main	Main class to analyse the data. Mainly gives the minimum, maximum, overall average and average per instance client and facility weights.
Instance	Instance object containing the distance (between vertices) and weights (of facilities and clients) data of the instance.

(b) Classes in the *exact_models* package

Class name	**Description**
Main	Main class to solve the two IPs (using CPLEX) and make a .csv file containing the results for all the instances in a particular folder.
MainLP	Main class to solve the LP relaxation of IP2 (using CPLEX) and make a .csv file containing the results for all the instances in a particular folder.
IP	Model for the two IPs and LP2. Using input 1 and input 2 gives the model for IP1 and IP2 respectively. Using input 3 gives the LP relaxation of IP2 (LP2).
Instance	Instance object containing the distance (between vertices) and weights (of facilities and clients) data of the instance.

(c) Classes in the *swap* package

Class name	**Description**
Main	Main class to run the heuristics for n-Swap.
Graph	Graph object containing the clients and facilities with their distance (to vertices) and weights. Also calculates the objective value given the destination and initial vertices of clients and facilities.
Solution	Solution object containing the client and facility assignment, occupied and unoccupied locations and corresponding objective value for given graph.
Swap	Implements n-Swap, BI or FI, for a given graph and initial solution.
TabuSearch	Implements n-SwapTS1 for a given graph and initial solution.
StrictTabuSearch	Implements n-SwapTS2 for a given graph and initial solution.
TabuSearch1	Implements n-SwapTS1 with restart diversification for agiven graph and initial solution.
StrictTabuSearch1	Implements n-SwapTS2 with restart diversification for a given graph and initial solution.
TabuSearch2	Implements n-SwapTS2 with continuous diversification for a given graph and initial solution.
StrictTabuSearch2	Implements n-SwapTS2 with continuous diversification for a given graph and initial solution.

Table C7: Codes descriptions of the classes in the Java packages, continued.

(d) Classes in the *optswap* package

Class name	**Description**
Main	Main class to run the heuristics for n-OptSwap.
Graph	Graph object containing the clients and facilities with their distance (to vertices) and weights. Also calculates the objective value given the destination and initial vertices of clients and facilities.
Solution	Solution object containing the client and facility assignment, occupied and unoccupied locations and corresponding objective value for given graph.
OptSwap	Implements n-Swap, BI or FI, for a given graph and initial solution.
HungarianAlg	Class that executes the Hungarian Algorithm between (all) facilities and (selected) destination locations. It uses a previous assignment as the initial labeling to shorten the running time.
TabuSearch	Implements n-OptSwapTS1 for a given graph and initial solution.
StrictTabuSearch	Implements n-OptSwapTS2 for a given graph and initial solution.
TabuSearch1	Implements n-OptSwapTS1 with restart diversification for a given graph and initial solution.
StrictTabuSearch1	Implements n-OptSwapTS2 with restart diversification for a given graph and initial solution.
TabuSearch2	Implements n-OptSwapTS2 with continuous diversification for a given graph and initial solution.
StrictTabuSearch2	Implements n-OptSwapTS2 with continuous diversification for a given graph and initial solution.

(e) Classes in the *smartswap* package

Class name	**Description**
Main	Main class to run the heuristics for n-SmartSwap.
Graph	Graph object containing the clients and facilities with their distance (to vertices) and weights. Also calculates the objective value given the destination and initial vertices of clients and facilities.
Solution	Solution object containing the client and facility assignment, occupied and unoccupied locations and corresponding objective value for given graph.
AllHungarianAlg	Class that executes the Hungarian Algorithm between (all) facilities and (selected) destination locations. It uses a previous assignment as the initial labeling to shorten the running time.
SubHungarianAlg	Class that executes the Hungarian Algorithm between (selected) facilities and (selected) destination locations.
SmartSwap	Implements n-SmartSwap, BI or FI, for a given graph and initial solution.
TabuSearch	Implements n-SmartSwapTS1 for a given graph and initial solution.
StrictTabuSearch	Implements n-SmartSwapTS2 for a given graph and initial solution.
TabuSearch1	Implements n-SmartSwapTS1 with restart diversification for a given graph and initial solution.
StrictTabuSearch1	Implements n-SmartSwapTS2 with restart diversification for a given graph and initial solution.
TabuSearch2	Implements n-SmartSwapTS2 with continuous diversification for a given graph and initial solution.
StrictTabuSearch2	Implements n-SmartSwapTS2 with continuous diversification for a given graph and initial solution.