Erasmus University Rotterdam

Erasmus School of Economics

Thesis in Quantitative Logistics

A covering tour approach to the location of satellite distribution centers to supply humanitarian aid

Author

Matthijs van der Kaaden (464381)

Supervisor

Lisanne van Rijn

Second assessor

Dr. Wilco van den Heuvel

July 4, 2021

Abstract

This paper will extend the research of Naji-Azimi et al. (2012). It concerns locating satelite distribution centers (SDCs) to supply humanitarian aid in disaster areas. People are instructed to visit an SDC in order to obtain the survival goods, as relief teams can not visit everyone. The SDCs need to be located such that every home is within walking distance of at least one SDC. The SDCs need to be supplied from a central depot, using a heterogenous and capacitated vehicle fleet. The problem is formulated as a generalization of the covering tour problem and a heuristic approach is proposed by Naji-Azimi et al. (2012). The heuristic approach obtains lower quality solutions, but better computation times. This paper will also regard an extension to the heuristic, which obtains better solutions than the original heuristic. There is a trade-off between quality and computation time.

Contents

1 Introduction

The amount of natural disasters has been growing over the past few years, which caused the interest in emergency logistics to explode in the last ten years (Naji-Azimi et al. 2012). It is difficult, but crucial to create an efficient aid delivery network. There are several tasks that need to be considered for an efficient emergency network: establishing a command center, gathering information on the affected area, identifying appropriate sites for shelter, creating an evacuation plan, delivering relief material and installing the emergency facilities (fire, medical and emergency construction).

In this paper, the research of Naji-Azimi et al. (2012) will be replicated and extended. The main focus of this paper is on the distribution of survival goods to the inhabitants of the disaster area. "Distribution networks for humanitarian aid are often compared to classic industrial distribution networks, replacing suppliers with humanitarian agencies and distribution centers with public sites that are temporarily adapted to store and handle goods" (Naji-Azimi et al. 2012). Also, the retailers are replaced by mobile distribution booths that are placed in locations with easy and public access. Finally, the objective function for a classic industrial distribution network is to maximize overall profit, while the objective of the humanitarian aid problem is to attain the highest level of efficiency and fairness for aid delivery (Naji-Azimi et al. 2012).

The distribution network for humanitarian aid consists of multiple *central depots* (CDs). Each of these CDs is responsible for aid delivery in the area that it is located in. In this paper we consider optimizing the efficiency for one of these CDs, assuming that a CD has been opened for the given region.

The CDs do not deliver aid directly to the affected people. They supply *satellite distribution centers* (SDCs) where citizens can go to to receive aid. Emergency managers have to choose several locations from a set of potential locations, the chosen locations will be used as SDCs. The victims from the affected areas need easy access to these locations. Therefore, all the victims' homes should be within a maximum distance of the SDCs. This maximum distance is set by the emergency managers, the maximum distance is from now on referred to as the *covering distance*. The locations for the SDCs have to be chosen in such a way that all demand points (victims' homes) are covered.

The demand of the SDC (the demand of all victims that are covered by that SDC) is supplied by a fleet of heterogenous and capacitated vehicles. The fleet departs from the central depot and makes sure that every SDC can supply aid to all the victims assigned to it. It is possible for multiple vehicles to visit the same SDC. The problem at hand is how to choose the optimal locations for the SDCs, the allocation of the vehicles to the SDCs and how much the vehicles supply to a certain SDC.

This paper provides a mathematical model to determine the number and location of the SDCs. The model follows an exact algorithm and is for this reason not applicable in real-size instances. Therefore, a heuristic is proposed by Naji-Azimi et al. (2012) to provide an approximation of the optimal value. The heuristic does an initialization and then a local search to try and find the minimum objective value. We found the heuristic to obtain, on average, higher objective values than the exact algorithm. Also, the computation times of the heuristic were significantly longer than the computation times found by Naji-Azimi et al. (2012). To improve these results, the heuristic was extended. An additional local search operator was added to the heuristic and the heuristic will be stopped if it finds no improvement after a (local search) iteration. This resulted in lower computation times and lower objective values. The goal of this paper is to provide a useful tool for emergency managers in their decision making about the SDC locations.

The rest of this paper is structured as follows. Section 2 reviews relevant literature on the subject of covering problems. Section 3 defines the problem and mathematical formulation. Section 4 presents the methods used to solve the problem as proposed by Naji-Azimi et al. (2012). Section 5 describes how the data was obtained and what the data consists of. Section 6 shows the results of the paper and compares the results of this paper to the results of Naji-Azimi et al. (2012). Section 7 discusses the extension that is done. Section 8 concludes the paper.

2 Literature review

The problem at hand is a covering problem. There are multiple options to formulate a covering problem. In this section the options will be discussed and distinguished. The most important covering problems are the covering salesman problem, the covering tour problem and the median cycle problem.

The *Covering Salesman Problem* (CSP) aims to minimize the cost of a tour through a subset of p cities. All other cities should be within a certain covering distance of a city in the subset. Arkin and Hassin (1994) introduced a geometric version of the CSP. Each of the n salesman's clients specified a neigbourhood in which they were willing to meet. The goal was to compute the shortest tour that intersects all of the buyers' neighbourhoods. The major difference between this problem and the problem discussed in this paper, is that each client has a different covering distance, compared to the constant covering distance in this paper. Current and Schilling (1989) also used the CSP to obtain routes in disaster areas. Healthcare teams had to visit a subset of the affected villages while the rest of the affected villages should be within walking distance of one of the visited sites.

Two bi-objective variants to the CSP were introduced by Current and Schilling (1994); the *Median Tour Problem* (MTP) and the *Maximal Covering Tour Problem* (MCTP). The first objective is to minimize the length of a tour through only p of all the villages. This is the same for both problems. The second objective for the MTP is to minimize the total distance between each unvisited village and the nearest visited village. The second objective for the MCTP is to maximize total demand that can be assigned to a tour stop, where demand can be assigned if the demand points are within a predetermined maximum distance of the stop.

The *Covering Tour Problem* (CTP) was studied by Gendreau et al. (1997). The following sets were defined: W_1, W_2 and $\bar{W} \subset \mathrm{W}_1$. W_1 is a set of vertices that are optional for visits, W_2 is a set of vertices that have to be covered and $\bar{W}$ is a set of vertices that must be on the route. The objective of the CTP is to find a Hamiltonian cycle with minimum length over a subset of W_1, such that all vertices in W_2 are covered and all vertices in $\bar{W}$ are visited. To solve the problem, an exact branch-and-cut algorithm and an algorithm combining a set covering and TSP heuristic were proposed. In extension to this, Hachicha et al. (2000) investigated three heuristics to solve a multi-vehicle covering tour problem. In this paper, a common application for the delivery of health care by mobile units in developing countries is discussed, which resembles the purpose of this paper.

The *Median Cycle Problem* (MCP) can be studied in two versions. The MCP1 and MCP2 version. In MCP1, the objective is to minimize the length of the cycle and the assignment distance of the vertices not in the cycle, where the assignment distance is the distance to the nearest vertex in the cycle. This problem is also referred to as the ring star problem (Kedad-Sidhoum and Nguyen 2010).

The MCP2 aims to minimize the length of the route, where the assignment distance has an upper bound. Both versions of the problem were solved by Pérez et al. (2003) and Renaud et al. (2004). Pérez et al. (2003) used a variable neighbourhood tabu search heuristic and Renaud et al. (2004) used an evolutionary algorithm.

In Table 1 we can find the characteristics of the discussed problems. All characteristics of our problem are discussed, except for the amount of products greater than one. As every problem has at least one similarity to our problem, all discussed literature might prove to be useful. However, none of the problems discussed, are completely the same as the problem in this paper.

Problem name	Objective Function	No. of vertices in the subtour	Kinds of node	Covering distance	Nodes with demand	No. of products	No. of vehicles
CSP *	Min. cost	Fixed, p	One	Yes	No	1	1
CSP **	Min. distance	Fixed, p	One	Yes	No	1	1
MTP	Min. distance and assignment cost	Fixed, p	One	No	Yes	1	1
MCTP	Min. distance and max. demand within a covering distance	Fixed, p	One	Yes	Yes	1	1
CTP	Min. distance while covering nodes of W_2	Free	W_1 can be visited, some must be visited and W_2 must be covered	Yes	No	1	1
m-CTP	Min. distance while covering nodes of W_2, subject to maximum number of nodes and maximum distance per route	Maximum of p	W_1 can be vistied, some must be visited and W_2 must be covered	Yes	No	1	m
MCP1	Min. distance and assignment cost	Free	One	No	No	1	1
MCP2	Min. distance subject to a maximum assignment cost	Free	One	No	No	1	1
This paper	Min. distance	Free	J can be visited and I must be covered	Yes	Yes	s	m

The CSP was solved * by Arkin and Hassin (1994) and ** by Current and Schilling (1989).

Table 1: Related problems

3 Problem formulation

In this paper we use the following definitions for the problem formulation. Let $G = (V, A)$ be a complete directed graph. The set V contains all vertices, $V = \{0\} \cup I \cup J$, where 0 is the central depot; $I = \{1, ..., n\}$ the set of demand points and $J = \{1, ..., m\}$ the set of potential SDC's. Every

vertex in I must be covered while the covering of vertices in J is optional. The set A contains all arcs, $A = \{(v_i, v_j) : v_i, v_j \in V\}$. A distance matrix is defined over A with c_{ij} the distance between vertex i and $j \in V$. The amount of aid type $s(s = 1, ..., t)$ required at demand point $i \in I$ is d_{is}. The weight of each aid unit s is equal to w_s. As stated before, there is a fleet of l heterogenous vehicles available. The vehicles' capacity is defined as Q_k for each vehicle $k = 1, ..., l$. A $n * m$ matrix α is defined where α_{ij} is equal to 1 if demand point i is within covering distance τ from SDC j and 0 otherwise.

For the model, the following decision variables are defined:

D_{isjk}	quantity of demand type s at demand point i supplied by vehicle k while visiting SDC j
x_{ijk}	equals 1 if arc (i, j) is visited by vehicle k, 0 otherwise
y_{jk}	equals 1 if SDC j is visited by vehicle k, 0 otherwise
u_{ik}	a free variable used in the sub-tour elimination constraints

The model is formulated as follows:

$$\min \quad \sum_{i=0}^{m}\sum_{j=0}^{m}\sum_{k=1}^{l} c_{ij}x_{ijk} \tag{1}$$

$$\text{s.t.} \quad \sum_{i=0}^{m} x_{ijk} = y_{jk} \qquad j \in \{1,...,m\}, k \in \{1,...,l\} \tag{2}$$

$$\sum_{i=0}^{m} x_{jik} = y_{jk} \qquad j \in \{1,...,m\}, k \in \{1,...,l\} \tag{3}$$

$$\sum_{j=0}^{m} x_{0jk} = 1 \qquad k \in \{1,...,l\} \tag{4}$$

$$\sum_{j=0}^{m} x_{j0k} = 1 \qquad k \in \{1,...,l\} \tag{5}$$

$$\sum_{j=1}^{m}\sum_{k=1}^{l} \alpha_{ij} D_{isjk} \geq d_{is} \qquad i \in \{1,...,n\}, s \in \{1,...,t\} \tag{6}$$

$$\sum_{i=1}^{n}\sum_{s=1}^{t} w_s D_{isjk} \leq Q_k y_{jk} \qquad k \in \{1,...,l\}, j \in \{1,...,m\} \tag{7}$$

$$\sum_{s=1}^{t}\sum_{i=1}^{n}\sum_{j=1}^{m} w_s D_{isjk} \leq Q_k \qquad k \in \{1,...,l\} \tag{8}$$

$$u_{ik} - u_{jk} + (m+1)x_{ijk} \leq m \qquad i,j \in \{1,...,m\}, k \in \{1,...,l\} \tag{9}$$

$$x_{ijk} \in \{0,1\} \qquad i,j \in \{1,...,m\}, k \in \{1,...,l\} \tag{10}$$

$$y_{jk} \in \{0,1\} \qquad j \in \{1,...,m\}, k \in \{1,...,l\} \tag{11}$$

$$u_{ik} \geq 0 \qquad i \in \{1,...,m\}, k \in \{1,...,l\} \tag{12}$$

$$D_{isjk} \geq 0 \qquad i \in \{1,...,n\}, s \in \{1,...,t\}, j \in \{1,...,m\}, k \in \{1,...,l\} \tag{13}$$

The objective (1) is to minimize the travel distance of all vehicles combined. Constraints (2) and (3) make sure that for each SDC j and each vehicle k, there are the same amount of incoming arcs as outgoing arcs. Constraints (4) and (5) ensure that each vehicle leaves from the central depot and returns to the central depot. Constraints (6) ensure that all demand points i have their demand fulfilled. Constraints (7) link the distribution variables D_{isjk} to the use of vehicle k for a delivery to demand point j. Constraints (8) ensure that each vehicle is not loaded with more weight than its capacity allows for and constraints (9) are the sub-tour elimination constraints as proposed by Miller et al. (1960). Constraints (10) - (13) are the decision variable definitions.

4 Methodology

This section describes the heuristic approach used to solve the problem. The first step in the heuristic is the *Preprocessing* step. This step checks whether there are certain possible locations for SDCs that have to be used. Next up is the *Initialization* to provide an initial solution. Then, a *Local search* algorithm is used to find the optimal solution. The final step is the *Diversification*, this is done in order to give the solution some flexibility. The pseudocode of the heuristic can be found in Figure 1 at the end of this section.

4.1 Preprocessing step

The aim of the preprocessing step is to find demand points that have only one SDC within the covering distance. If this is the case, it means that the demand point can only be supplied from one SDC. Therefore, this SDC must be selected in every feasible solution. For each SDC j, a subset T_j is constructed containing all demand points within the maximum covering distance from that SDC.

4.2 Initialization

The initialization finds an initial feasible solution. This step randomly selects a vehicle k, an SDC $j \in J$ and a demand point $i \in T_j$. If the total demand from the demand point is less than or equal to the remaining capacity of the vehicle, demand point i is assigned to SDC j. When all demand points in T_j are assigned, the next SDC on route k is selected as the SDC closest to j where the set T_{j+1} contains at least one demand point that is not yet assigned to an SDC.

When there is not enough capacity left in vehicle k to supply all demand from the next demand point, the vehicle is filled with as many products as possible for this demand point. After visiting the last SDC, the vehicle returns to the central depot. A new vehicle is randomly selected and the process continues with the demand point that was not fully supplied. This algorithm continues until all demand points are assigned to an SDC. The obtained feasible solution is now used in the next step to get an approximation of the optimal solution.

4.3 Local search

The *Local search* algorithm consists of four procedures, each designed to deal with specific characteristics of the current solution. These four procedures are repeated for a given number of iterations

(*#_LocalSearch_iterations*).

4.3.1 Delete-Redundant-SDC

The *Delete-Redundant-SDC* procedure tries to remove unnecessary SDCs from routes. A stop is unnecessary if after removing the stop, the solution is still feasible. This is the case if every demand point assigned to an SDC j, can be supplied by another SDC in the solution. The *Delete-Redundant-SDC* procedure considers each route and for each SDC j on that route, it evaluates if the SDC can be removed. Verifying the feasibility of the solution after removing SDC j can be done by checking if all demand points served by SDC j can be served by another SDC on the same route. If the solution remains feasible, the SDC can be removed. The next step is to assign all demand points previously supplied by SDC j to another SDC. The algorithm tries to assign as many demand points as possible to the closest SDC to j. If not all demand points are covered by the closest SDC, the algorithm moves on to the second closest SDC to j, this continues until all demand points are assigned to other SDCs on the same route.

4.3.2 Swap

The *Swap* procedure tries to reduce the length of a route. This can be achieved by changing the order of visits within a route or even by swapping SDCs between different routes. Changing the order of visits within a route will always result in a feasible solution, as no changes are made to the fulfilled demand. However, swapping SDCs between routes will result in a different allocation of demand, which might exceed the vehicles' capacity. Therefore, this is not always possible.

First, each duo of SDCs on a route is considered for a swap. If the swap results in a shorter route length, the swap is performed. This procedure continues for all routes. Next, swaps between different routes are considered. As stated before, this will not always result in a feasible solution, which makes it a more difficult procedure. Each SDC serves different demand points, which results in different amounts of demand assigned to different SDCs. Therefore, if the remaining capacity of a vehicle was close to zero before the swap, the remaining capacity could become negative and would thus become infeasible after the swap.

To illustrate this problem, consider a swap between SDC j_1 and SDC j_2 visited by vehicles k_1 and k_2 respectively. The remaining capacities of the vehicles are denoted as R_1 and R_2 respectively. The

capacity required for the SDCs is denoted as d_1 and d_2 ($d_1 = \sum\sum D_{isj_1k_1}w_s$, $d_2 = \sum\sum D_{isj_2k_2}w_s$ for all s and i). To remain feasible, the following condition needs to be satisfied:

$$\min(R_1 + d_1 - d_2; R_2 + d_2 - d_1) \geq 0$$

The procedure checks potential swaps for each SDC j on each route k. A list of the closest SDCs to j is created. All swaps between SDC j and SDCs in the top Φ of the list are considered for a swap. If the feasibility condition holds, the total length of the two routes before and after the swap are computed. If the total length after the swap is shorter, the swap is performed.

4.3.3 Drop & Add

The *Drop & Add* procedure tries to replace a currently chosen location in a non-optimal route by one or more unchosen locations. It is possible that visiting more and different SDCs will reduce the length of the route. Another possibility occurs when looking at an SDC that is visited on multiple routes, the route length might be reduced by replacing this SDC with two or more SDCs in the neighbourhood.

Each visited SDC is considered to be dropped. If an SDC j gets dropped, some demand points become uncovered. The algorithm tries to reassign these demand points to other SDCs on the route. If not all demand points can be reassigned to another SDC on the route, the algorithm considers adding unvisited SDCs to the route, starting with the one closest to SDC j. If an unvisited SDC covers at least one of the uncovered demand points, the SDC is added to the route such that the route length is minimized. Next, the algorithm tries to assign as many uncovered demand points to the newly added SDC. This proces continues until all demand points are covered. When the proces is done, the total route length is computed. The removal of the SDC which leads to the shortest total route length, will be performed, even if the total route length is increased in comparison to the current solution.

4.3.4 Extraction-Insertion

The last operator is the *Extraction-Insertion*. This procedure tries to move an SDC j currently on route k to another route k', in order to optimize vehicle capacity. To check whether it is possible to insert SDC j in route k', the volume required ($d_j = \sum\sum D_{isjk}w_s$) to transport demand for SDC j

should be less than or equal to the remaining capacity of vehicle k'. If SDC j can be supplied by vehicle k', the algorithm inserts the SDC in route k', such that the route length increase is minimized. For each route, each SDC is considered for a transfer to every other route. The procedure is only performed if it reduces the total route length. As long as improvements are found, the procedure is repeated.

4.4 Diversification

After all iterations of the local search operations are completed, there is one last *Diversification* procedure. The goal of this procedure is to allow the local search operations in the next iteration to perform changes. The *Diversification* randomly selects ϕ unvisited SDCs and inserts them into the routes of the current solution. This is done in such a way that the increase in total route length is minimized. The combination of the *Local search* and the *Diversification* is repeated for a certain number of iterations (*#_iterations*).

Figure 1: Pseudocode heuristic

```
For r=1 to # of restarts
  Preprocessing
  Initialization
  For i=1 to #_iterations
    For j=1 to #_LocalSearch_iterations
      Delete-Redundant-SDC
      Swap
      Drop & Add
      Extraction-Insertion
    next j
    Diversification
  next i
next r
```

5 Data

The data used in this paper is not the same as the data used in Naji-Azimi et al. (2012). The results might be different for this reason. The data used in this paper is provided by the Erasmus University of Rotterdam. For the parameter testing 20 instances were generated, each with 100 demands points, 20 SDCs, 2 product types and 2 vehicle types. For the computational results, the amount of demand points, SDCs, product types and vehicle types varied. The amount of demand

points is either 20, 30 or 40. The amount of SDCs is either 4, 6, 8, 9, 12 or 16. The amount of product types and vehicle types ranges from 2 to 4. For each of the used combinations, five instances were generated. In total, 100 instances were generated.

The data files contain the following information: the maximum covering distance, distance matrix, covering matrix, weights of the aid units, demand at demand points per product type and the vehicle fleet with corresponding capacities.

6 Results

In this section the best combination of parameters will be tested. Once this is done, the best combination will be used to calculate the objective values of the instances. The objective values will be calculated using the heuristic as described by Naji-Azimi et al. (2012). These objective values will then be compared to the exact solution. All codes are written in Java, using Eclipse IDE 2018-12. Algorithms are executed on a computer with an Intel Core i5-8250U CPU at 1.60GHz and 8 GB RAM.

6.1 Parameter setting

The number of restarts for the parameter testing is fixed to five. The following parameter combinations were considered:

Number of iterations:	10, 20
Number of local search iterations:	10, 15, 20
Number of SDCs considered in swap procedure (Φ) :	4, 6, 8
Number of SDCs considered in diversification (ϕ) :	3, 5, 7

These combinations were tested on instances with 100 demand points, 20 SDCs, 2 types of products and 2 vehicle types. The results can be found in Table 2. The results range from 2152 to 2315, where the best result was obtained using 20 iterations, 20 local search iterations, ϕ equal to 3 and Φ equal to 8. This combination of parameters will be used for the computational results. Especially the parameter for the amount of swaps seems to have a big impact on the results. As we can see, the objective values decrease as Φ increases.

Table 2: Heuristic parameters setting

# iterations	# local search iterations	ϕ	Φ		
			4	6	8
10	10	3	2277	2231	2184
		5	2262	2262	2205
		7	2270	2210	2170
	15	3	2283	2214	2193
		5	2290	2194	2201
		7	2315	2216	2155
	20	3	2297	2229	2180
		5	2276	2235	2169
		7	2281	2245	2168
20	10	3	2289	2249	2197
		5	2280	2248	2181
		7	2272	2220	2156
	15	3	2289	2253	2174
		5	2269	2237	2198
		7	2277	2200	2169
	20	3	2272	2225	2152
		5	2261	2202	2165
		7	2268	2217	2177

6.2 Computational results

In this sub-section, the performance of the heuristic is evaluated. To evaluate the performance, the objective values and computation times of the heuristic will be compared to those of the exact algorithm. For the exact algorithm, a time limit of 1800 seconds per instance was set. For each combination of demand points, SDCs, products and vehicles, the objective is the average of the five instances. The computation time is the sum of the five computation times.

For the heuristic, the following parameters were used: 3000 restarts, 20 iterations, 20 local search iterations, ϕ equal to 3 and Φ equal to 8. In Table 3 we can see that the heuristic, on average, finds a higher objective value than the exact algorithm (average gap is positive). The average computation time for the heuristic is shorter than for the exact algorithm, but when we look at Table 3 again, we can see that this is mostly caused by an outlier for the instances with 8 SDCs, 4 product types and 4 vehicles. Most of the other heuristics' computation times are significantly longer than the computation times of the exact algorithm. Therefore, the exact algorithm outperforms the heuristic on small instances.

For the larger instances (30 demand points or more), the computational times for the heuristic are, on average, almost a third of the computational times for the exact algorithm (Table 4). However, an average optimality gap of 4.62% is found. This indicates that the heuristic does not consistently perform for the greater instances.

Table 3: Computational results for smaller instances

Demand points	SDCs	products	vehicles	Exact		Heuristic		
				Objective	Seconds	Objective	Seconds	Gap(%)
20	4	2	2	385.2	4.92	403.8	312.7	4.83
		2	3	328.0	0.59	328.0	264.3	0.00
		2	4	294.2	0.26	294.2	206.7	0.00
		3	2	449.6	47.35	449.8	548.7	0.04
		3	3	422.8	2.10	424.0	459.9	0.28
		3	4	306.2	3.61	303.8	503.3	-0.78
		4	2	779.6	37.20	795.4	899.4	2.03
		4	3	423.8	3.00	423.8	610.5	0.00
		4	4	478.8	14.29	491.0	549.6	2.55
	6	2	2	420.6	69.96	422.8	472.0	0.52
		2	4	285.4	1.35	286.2	294.1	0.28
		4	2	680.0	2410	710.0	1009	4.41
		4	4	334.2	29.27	342.2	767.1	2.39
	8	2	4	385.8	444.0	385.0	453.8	-0.21
		4	2	723.4	7242	703.8	1241	-2.71
		4	4	419.0	1972	422.6	828.8	0.86
Average					767.6		588.8	0.91

Table 4: Computational results for larger instances

Demand points	SDCs	products	vehicles	Exact			Heuristic		
				Objective	Gap(%)	Seconds	Objective	Seconds	Gap(%)
30	9	4	4	735.8	42.15	9000	741.4	2880	0.76
	12	4	4	638.8	41.01	9000	655.2	2634	2.57
40	12	4	4	833.6	47.92	9000	915.2	3795	9.79
	16	4	4	866.4	58.01	9000	913.0	4119	5.37
Average					47.27	9000		3357	4.62

6.3 Discussion results replication

The results obtained through the heuristic by Naji-Azimi et al. (2012) are better than the results obtained in this paper. Their heuristic returned lower objective values than the exact algorithm, which can not be said about the results in this paper. This might be caused by the fact that the triangle inequality does not hold for some of the instances used in this paper, where Naji-Azimi et al. (2012) stated that it holds for their instances. This was tested on a single instance, on which the effect was not too great, so only a small portion of the difference can be explained by this.

Furthermore, the implementation could be different, which can also result in different objective values.

The heuristic of Naji-Azimi et al. (2012) also had lower computation times. There could be differences in the implementation of the heuristic, which could explain the difference in computation time. Another possible argument is that it is unknown how many restarts they did for the heuristic. In this paper the amount of restarts was 3000, based on the fact that Naji-Azimi et al. (2012) did 3000 restarts for their parameter testing. However, it is not clear whether or not they changed the amount of restarts for the computational results.

7 Extension

The extension of the original paper will consist of three parts. First, the heuristic parameter Φ considered will not be constant, where it was set to eight for the heuristic. Second, the heuristic will be stopped early if no improvement is found for two iterations. Third, a *3-Opt-Exchange* will be added to the heuristic. The *3-Opt-Exchange* operator picks three SDCs on a route and tries to rearrange them in such a way that the route length is reduced. The pseudocode for the extended heuristic can be found in Appendix A.

7.1 Variable Swap Parameter

As stated above, the parameter was fixed to eight in the heuristic. Allowing Φ to vary, can reduce computation times. At the start of the extension, the parameter is set to four, but when the improvement after a local search iteration is not greater than ρ, the amount of swaps considered will increase by two. From the parameter setting, we concluded that Φ had a significant impact on the quality of the solutions. Therefore, a relatively big ρ of at least 10% was chosen. With a relatively big ρ, Φ will increase fast and the objective values will not be affected by this. The increase of two is inspired by the possibilities suggested for the parameter setting by Naji-Azimi et al. (2012). Once a whole iteration has passed (20 local search iterations), Φ is set back to four.

$$\rho = objectiveValue/10 + 2$$

The goal of this part of the extension is to reduce computation times. In the result section we found the computation times for the heuristic to be longer than the computation times for the exact

algorithm. Limiting the amount of SDCs considered for the *Swap-procedure* until the improvement is too small, can reduce computation time.

7.2 Stopping early

The heuristic will be stopped when no improvement is found for two iterations, the heuristic has found a local optimum. If this is the case while not all iterations have been performed, the computation time can be reduced by simply stopping the heuristic. Since the heuristic is already in a local optimum, no further improvements would have been found.

7.3 3-Opt-Exchange

As we saw in the results section, the results of the exact algorithm are, on average, better than the results obtained through the heuristic. To improve the objective values of the heuristic, an additional local search operator can help to find solutions that would not have been found without this operator.

The *3-Opt-Exchange* picks three SDCs on a single route and tries to rearrange the order of visits in such a way that the route length is minimized. As the SDCs are still visited by the same vehicle, the remaining capacities remain unchanged. Therefore, the new solution will always be feasible. The procedure calculates the current length of the route and the route lengths of the possible new routes. All possible routes are compared and the one with the shortest route length is set as the new route. The idea was to also check possibilities for exchanging three SDCs between routes, but due to the computation times increasing exponentially, this could not be implemented.

7.4 Results

We can still see in Table 5 that the computation times for small instances are often higher for the extended heuristic. However, the computation times of the extended heuristic for the large instances are only a fraction of the computation times from the exact algorithm. Compared to the original heuristic, the computation times are significantly shorter with an average reduction in computation time of 86.98% (Table 6). The goal to reduce computation times was achieved. We can also see that the objective values of the extension are on average lower than the replications' objective values. Overall, the extended heuristic performs much better than the original heuristic.

On average, the extended heuristic still obtains higher objective values than the exact algorithm. The second goal of the extension was to obtain lower objective values through the extra local search operator. This goal was achieved, but not in significant measures, as the exact algorithm still obtains lower values.

Table 5: Extension

Demand points	SDCs	products	vehicles	Exact		Extension		
				Objective	Seconds	Objective	Seconds	Gap(%)
20	4	2	2	385.2	4.92	403.2	20.48	4.67
		2	3	328.0	0.59	328.0	15.64	0.00
		2	4	294.2	0.26	294.2	13.20	0.00
		3	2	449.6	47.35	449.6	27.57	0.00
		3	3	422.8	2.10	424.6	57.74	0.43
		3	4	306.2	3.61	302.2	33.76	-1.31
		4	2	779.6	37.20	792.0	110.5	1.59
		4	3	423.8	3.00	423.8	74.12	0.00
		4	4	478.8	14.29	484.4	34.01	1.17
	6	2	2	420.6	69.96	422.8	66.25	0.52
		2	4	285.4	1.35	286.2	43.34	0.28
		4	2	680.0	2410	717.2	165.9	5.47
		4	4	334.2	29.27	341.2	97.54	2.09
	8	2	4	385.8	444.0	383.8	77.95	-0.52
		4	2	723.4	7242	687.8	212.2	-4.92
		4	4	419.0	1972	422.8	137.5	0.91
30	9	4	4	735.8	9000	746.2	348.6	1.41
	12	4	4	638.8	9000	653.0	457.0	2.22
40	12	4	4	833.6	9000	919.2	742.5	10.26
	16	4	4	866.4	9000	896.8	1185	3.51
Average					2051		196.0	1.39

8 Conclusion

This paper examined a heuristic approach to locating satellite distribution centers to deliver humanitarian aid in disaster areas. However, in general, the heuristic did not find better results than the exact approach and the computation times were far greater than the exact approach for small instances. For the extended heuristic, the results were also not as good as the ones provided by the exact approach, but the computation times were only a fraction of the computation times of the exact approach when applied to larger instances. On average, the extended heuristic supplied lower objective values than the replicated heuristic. Therefore, the extended heuristic would definitely be preferred over the replicated heuristic. The choice between the exact algorithm and the extension will be a trade-off between objective values and computation times.

Table 6: Comparison between heuristic and extended heuristic

Demand points	SDCs	products	vehicles	Heuristic		Extension		Improvement	
				Gap(%)	Seconds	Gap(%)	Seconds	Gap	Seconds reduction(%)
20	4	2	2	4.83	312.7	4.67	20.48	0.16	93.45
		2	3	0.00	264.3	0.00	15.64	0.00	94.08
		2	4	0.00	206.7	0.00	13.20	0.00	93.61
		3	2	0.04	548.7	0.00	27.57	0.04	94.98
		3	3	0.28	459.9	0.43	57.74	-0.15	87.45
		3	4	-0.78	503.3	-1.31	33.76	0.53	93.29
		4	2	2.03	899.4	1.59	110.5	0.44	87.71
		4	3	0.00	610.5	0.00	74.12	0.00	87.86
		4	4	2.55	549.6	1.17	34.01	1.38	93.81
	6	2	2	0.52	472.0	0.52	66.25	0.00	85.96
		2	4	0.28	294.1	0.28	43.34	0.00	85.26
		4	2	4.41	1009	5.47	165.9	-1.06	83.56
		4	4	2.39	767.1	2.09	97.54	0.30	87.28
	8	2	4	-0.21	453.8	-0.52	77.95	0.31	82.82
		4	2	-2.71	1241	-4.92	212.2	2.21	82.90
		4	4	0.86	828.8	0.91	137.5	-0.05	83.41
30	9	4	4	0.76	2880	1.41	348.6	-0.65	87.90
	12	4	4	2.57	2634	2.22	457.0	0.35	82.65
40	12	4	4	9.79	3795	10.26	742.5	-0.47	80.43
	16	4	4	5.37	4119	3.51	1185	1.86	71.23
Average				1.65	1142	1.39	196.0	0.26	86.98

References

[1] Esther M Arkin and Refael Hassin. "Approximation algorithms for the geometric covering salesman problem". In: *Discrete Applied Mathematics* 55.3 (1994), pp. 197–218.

[2] John R Current and David A Schilling. "The covering salesman problem". In: *Transportation science* 23.3 (1989), pp. 208–213.

[3] John R Current and David A Schilling. "The median tour and maximal covering tour problems: Formulations and heuristics". In: *European Journal of Operational Research* 73.1 (1994), pp. 114–126.

[4] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. "The covering tour problem". In: *Operations Research* 45.4 (1997), pp. 568–576.

[5] Mondher Hachicha, M John Hodgson, Gilbert Laporte, and Frédéric Semet. "Heuristics for the multi-vehicle covering tour problem". In: *Computers & Operations Research* 27.1 (2000), pp. 29–42.

[6] Safia Kedad-Sidhoum and Viet Hung Nguyen. “An exact algorithm for solving the ring star problem”. In: *Optimization* 59.1 (2010), pp. 125–140.

[7] Zara Naji-Azimi, Jacques Renaud, Angel Ruiz, and Majid Salari. “A covering tour approach to the location of satellite distribution centers to supply humanitarian aid”. In: *European Journal of Operational Research* 222.3 (2012), pp. 596–605.

[8] José A Moreno Pérez, J Marcos Moreno-Vega, and Inmaculada Rodrıguez Martın. “Variable neighborhood tabu search and its application to the median cycle problem”. In: *European Journal of Operational Research* 151.2 (2003), pp. 365–378.

[9] Jacques Renaud, Fayez F Boctor, and Gilbert Laporte. “Efficient heuristics for median cycle problems”. In: *Journal of the Operational Research Society* 55.2 (2004), pp. 179–186.

Appendix A

Pseudocode Extension

```
prevObj = Integer.MAX_VALUE
For r=1 to # of restarts
  Preprocessing
  Initialization
  For i=1 to # of iterations
    obj = this.getObjective
    For j=1 to # of local search iterations
      Delete-Redundant-SDC
      Swap
      3-Opt-Exchange
      Drop & Add
      Extraction-Insertion
      curObj = this.getObjective
      If prevObj-curObj < ρ
        Φ+ = 2
      prevObj = curObj
    next j
    Φ = 4
    Diversification
    If obj = this.getObjective
      iter++
      If iter >= 2
        return
      Else
        iter = 0
  next i
next r
```

Appendix B

The Extension.java file contains the code for the extended heuristic. First it gives an overview of what the heuristic will do. Next, it describes the preprocessing method and the initialization. Then all local search operators and the diversification method are described. After that, a few methods used in the previously mentioned code are described.

The Heuristic.java file contains the code for the replicated heuristic. First, it describes the preprocessing method and the initialization. Next, all local search operators and the diversification method are described. After that, a few methods used in the previously mentioned code are described.

The Main.java file contains the code used for running the heuristics and the model. It also contains a method that reads the data.

The Model.java file contains the code for the exact algorithm. First, the variables are created and the process of doing this is described. Next, all constraints are created. Then the objective is created and after that there are a few methods that are helpful in printing the results.

A more extensive description is given in the comments of the java files.