Master's degree thesis

LOG950 Logistics

Routing in Smart Waste Management

Preben Kiønig Senland

Number of pages including this page: 60

Molde, 31.10.2019

Molde University College
Specialized University in Logistics

Mandatory statement

Each student is responsible for complying with rules and regulations that relate to examinations and to academic work in general. The purpose of the mandatory statement is to make students aware of their responsibility and the consequences of cheating. Failure to complete the statement does not excuse students from their responsibility.

Please complete the mandatory statement by placing a mark ***in each box*** *for statements 1-6 below.*		
1.	**I/we hereby declare that my/our paper/assignment is my/our own work, and that I/we have not used other sources or received other help than mentioned in the paper/assignment.**	☒
2.	**I/we hereby declare that this paper** 1. Has not been used in any other exam at another department/university/university college 2. Is not referring to the work of others without acknowledgement 3. Is not referring to my/our previous work without acknowledgement 4. Has acknowledged all sources of literature in the text and in the list of references 5. Is not a copy, duplicate or transcript of other work	Mark each box: 1. ☒ 2. ☒ 3. ☒ 4. ☒ 5. ☒
3.	I am/we are aware **that any breach of the above will be considered as cheating, and may result in annulment of the examination and exclusion from all universities and university colleges in Norway for up to one year, according to the Act relating to Norwegian Universities and University Colleges, section 4-7 and 4-8 and Examination regulations** section 14 and 15.	☒
4.	**I am/we are aware that all papers/assignments may be checked for plagiarism by a software assisted plagiarism check**	☒
5.	**I am/we are aware that Molde University College will handle all cases of suspected cheating according to prevailing guidelines.**	☒
6.	**I/we are aware of the University College's rules and regulation for using sources**	☒

Personal protection

Personal Data Act

Research projects that processes personal data according to Personal Data Act, should be notified to Data Protection Services (NSD) for consideration.

Have the research project been considered by NSD? ☐yes [✓ ☒no]

- If yes:

Reference number:

- If no:

I/we hereby declare that the thesis does not contain personal data according to Personal Data Act.: ☒

Act on Medical and Health Research

If the research project is effected by the regulations decided in Act on Medical and Health Research (the Health Research Act), it must be approved in advance by the Regional Committee for Medical and Health Research Ethic (REK) in your region.

Has the research project been considered by REK? ☐yes [✓ ☒no]

- If yes:

Reference number:

Publication agreement

ECTS credits: 30

Supervisor: Arild Hoff

Agreement on electronic publication of master thesis

Author(s) have copyright to the thesis, including the exclusive right to publish the document (The Copyright Act §2).

All theses fulfilling the requirements will be registered and published in Brage HiM, with the approval of the author(s).

Theses with a confidentiality agreement will not be published.

I/we hereby give Molde University College the right to, free of charge, make the thesis available for electronic publication:	[✓ yes] ☐no
Is there an agreement of confidentiality?	☐yes [✓ no]
(A supplementary confidentiality agreement must be filled in)	
- If yes:	
Can the thesis be online published when the period of confidentiality is expired?	☐yes ☐no

Date: 31.10.2019

Page unintentionally left blank.

Summary

In this work two models and a matheuristic method are developed for routing problems in smart waste management. The solution methods are analyzed with regards to their quality and time use. A simulation is done with one of the models to evaluate its usefulness in long-term real cases.

Contents

1.0 Introduction

It is estimated that by 2050, two thirds of the global population will live in urban areas (Marques et al. 2018). Cities growing larger gives an increased stress on their infrastructure. One approach to deal with this, is to make the infrastructure more efficient by introducing Internet of Things (IoT) devices into the infrastructure. An IoT device is an electronic device, usually equipped with a sensor, that is connected to other devices via the internet. When IoT is used in the context of a city, the term Smart city is often used.

A vital part of any city’s infrastructure is the collection and disposal of waste. If waste isn't collected as it's produced, it will accumulate, increasing the chance for the spread of disease and harm to the environment. In addition to this, and efficient collection of waste allows for more materials to be recycled, reducing the amount of raw materials needed to be extracted from the earth. Because of this, waste collection in urban areas has gained attention from researchers and practitioners in the last decade (Xue et al., 2019).

Typically, the local municipality is responsible for the collection, but it can also be performed be private companies. Regardless of who is performing the collection, it is their interest to minimize the travel distance of garbage trucks to reduce costs, both monetary and environmental.

The rest of this thesis is structured in the following way: Section 2 gives a verbal description of the problem studied in this work. Relevant background literature is covered in section 3. In section 4, the mathematical models for the problem are developed, while the matheuristic method is described in section 5. Section 6 describes how the test data is generated and how the testing is performed. The results from the tests are presented in section 7, and section 8 concludes.

2.0 Problem Description

This section states the objective of this work and a verbal description of the problem setting. Considerations regarding cost elements of the problem are also discussed.

2.1 Smart Waste Management

The objective of this thesis is to develop decision-making tools for the Smart Waste Collection Problem. The problem concerns the collection of waste from bins that are equipped with sensors to measure their fill level. This information is made available to the decision-maker, to utilize in the planning of the emptying schedule and routing of the garbage trucks.

Traditionally, waste is collected at fixed intervals, say weekly or bi-weekly. If the accumulation of waste in the bins is deterministic, i.e. waste accumulates at a known, constant rate, this method can be adequate. One only needs to make sure that each bin is emptied at least every $capacity/daily\ fillrate$ days. The real world is not that simple, and in most cases we are dealing with stochastic processes.

By equipping the bins with sensors, the decision-maker knows whether each bin must be emptied on that day or if it can be emptied sometime in the future. We don't have perfect information about how waste will accumulate in the bins, but we can track their levels, and make inferences about the current and future fill-rate. One strategy could be to only empty bins as they approach their maximum capacity, in an attempt to reduce the number of times they are emptied. This approach could be too naive, as it will in many cases lead to having to make an additional trip for only a few number of bins. A better approach could be to, even though these bins are only partially filled, empty them with trucks that are already in their close proximity.

2.2 Cost Elements

There are several cost elements that need to be considered when modelling waste management. Some cost elements are shared by many VRP variants, such as *total distance* and *duration of routes*. In some cases it is not enough to consider only the total distance travelled, because the cost may not be proportional to the distance. The duration of routes needs to be considered in cases where drivers are paid at a higher wage when they exceed a certain number of hours worked in a single day. There could also be constraints on the duration of routes stemming from regulation, be it government regulation, union rules, or self-imposed rules

Other cost elements are more specific to the problem of waste management. If a full bin isn’t emptied, people might start leaving their waste next to it. This is of course not desirable, but it may be permissible. There needs to be a cost associated with the bins overflowing. In an attempt to avoid any overflowing, this cost could be set very high, or it could be set at a level that incorporates the marginal costs of the lower service level.

3.0 Literature

The problem studied in this thesis incorporates elements from the Vehicle Routing Problem. The Vehicle Routing Problem (VRP) was first formulated by Dantzig and Ramser in a paper called 'The Truck Dispatching Problem', as the problem of optimizing the routing of a fleet of gasoline delivery trucks between a bulk terminal and a large number of gas stations (Dantzig & Ramser 1959). It is a generalization of the Travelling Salesman Problem (TSP), where the objective is to find the shortest path for a salesman travelling between cities, ending up in the city of origin. The origins of the Travelling Salesman Problem aren't fully known, but Dantzig, Fulkerson and Johnson (1954) were among the first to formalize and offer a solution to the problem. TSP and VRP are combinatorial optimization problems that are known to be NP-hard (Archettia et al. 2009). As a consequence, applications to real-world problems are often impossible to solve to optimality within a reasonable time frame. The VRP has been extensively studies over the years, and several heuristics and meta-heuristics have been developed.

3.1 Logistics Concepts

This section covers two fundamental areas of logistics, the vehicle routing problem and inventory routing.

3.1.1 The Vehicle Routing Problem

Laporte et al. (2000) gives an overview of classical heuristics and TABU Search based meta-heuristics for the VRP. The two main techniques used in classical heuristics are to merge existing routes using a savings criterion, or to use an insertion cost to gradually assign vertices to vehicle routes. The different categories of classical heuristics presented are Savings algorithms, sequential improvement methods, the sweep algorithm, petal algorithms, cluster first - route second algorithms, and improvement heuristics.

The Clarke & Wright savings algorithm (Clarke & Wright 1964) is a well-known heuristic for VRPs where the number of vehicles is a decision variable. The first step of the algorithms is to compute the savings (reduced distance) of including node i and j in the same route. Savings are computed as $s_{ij} = c_{i0} + c_{0j} - c_{ij}\ \forall i, j = 1..n, i \neq j$. The next step is to merge routes together. This can be done in a parallel or sequential order.

Mole and Jameson (1976) and Christofides (1979) have proposed the two most well-known insertion methods for the VRP, but neither of them are as efficient as alternative methods.

The sweep method is easy to understand from its name. One imagines a line originating in the depot, and rotates it. Nodes are added to the routes as they are passed by the line, as long as there is available capacity in the vehicle. Some implementations also include a post-optimization phase after this initial clustering, where nodes are exchanged between the clusters and the cluster-routes are reoptimized (Laporte et al. 2000). Petal algorithms are an extension of the sweep algorithm, where several routes are generated and then selected by solving a set partitioning problem (Laporte et al. 2000).

3.1.2 The Inventory Routing Problem

The Inventory Routing Problem (IRP) merges the VRP with the Inventory Management Problem. The objective of the IRP is to simultaneously optimize the scheduling of deliveries, the routing of vehicles making said deliveries, and the inventory policies. IRPs can be classified by looking at the seven main attributes time horizon, structure, routing, inventory policy, inventory decisions, fleet composition, and fleet size. The time horizon can be either finite or infinite. The structure describes the number of suppliers and customers, and is typically one-to-one, one-to-many or many-to-one. When there is one customer per route, the routing is described as direct, and when there are more than one it is called multiple. A problem has continuous routing if there isn't a central depot. Inventory policies are the decision rules for how customer inventories should be replenished. How inventory management is modeled is described by the inventory decisions, and deals with issues such as negative inventory, back-orders, stock-outs and lost sales. Fleet composition and size describe the vehicles available. A fleet can be comprised of a single vehicle, multiple homogenous or heterogenous vehicles, or unconstrained (Coelho, Cordeau & Laporte 2013).

When looking at approaches to solving industrial cases of the IRP, Andersson et al. (2010) note that there are two broad classes of methodologies, heuristics and metaheuristics on one side, and exact methods that are terminated before optimality is proven. This is due to the fact that the complexity of the problem and the size of real-world instances requires an unreasonable long time to solve and verify.

3.2 Metaheuristics

The heuristics mentioned above have been classical heuristics of either the construction or improvement category. By using improvement heuristics, we can find better solutions than by using construction heuristics on their own. However, for certain problems this is not adequate. Improvement heuristics stop when they reach local optima, and, depending on the problem topology, this can be quite far from the global optimum. Metaheuristics are methods designed to search for a good solution beyond a local optimum (Hvattum 2017).

The term metaheuristic was coined by Glover (1986), and since then many definitions have been proposed. Metaheuristics has been defined both as a framework and as the implementation of the framework. Sorensen and Glover include both these perspectives in their definition:

> "A metaheuristic is a high-level problem-independent algorithmic
> framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a
> heuristic optimization algorithm according to the guidelines expressed in such a framework." (Sörensen & Glover, 2013)

Other definitions focus solely on the implementation of the framework:

> "A metaheuristic is an iterative master process that guides and modifies
> the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or

low) level procedures, or a simple local search, or just a construction method." (Voß et al, 1999)

There are several ways to classify metaheuristics, such as nature-inspired vs non-nature inspired, population-based vs single-point search, one vs various neighborhood structures, and memory usage vs memory-less methods (Blum & Roli 2003). Some of the most well-known metaheuristics are Simulated Annealing, TABU Search, GRASP and Genetic Algorithms.

3.2.1 Genetic Algorithms

Genetic Algorithms (GA), also called Evolutionary Search or Evolutionary Computation algorithms, are, as the name suggests, metaheuristics inspired by the theory of evolution. The idea is to mimic the evolutionary process found in biological systems and apply it to optimization problems. The first evolution-based metaheuristics were developed in the 1960s and 1970s (Blum & Roli 2003) by Fogel (1962), Rechenberg (1973), and Holland (1975) who developed the Genetic Algorithm. Using the scheme of Blum & Roli (2003), we can classify genetic algorithms as nature-inspired, population-based, one neighborhood structure and memoryless methods.

The population in a genetic algorithm is a set of chromosomes. Each chromosome is a representation of a solution. Each chromosome can be divided into several genes that represent different elements of the solution. The position of the gene within the chromosome is called the locus, and the different states of each gene are called alleles. For binary problems the chromosome is usually a bit-string (Mitchell 1998). In the bit-string 101010, each of the bits is a gene, with the alleles 0 and 1, and represents each decision variable in the optimization problem. For a VRP the chromosome can be a list of the nodes in the order they are visited, e.g. [A, B, C] represents a route where the vehicle drives from the depot to node A, from node A to node B, and from node B to Node C, before it returns to the depot. An alternative decoding of [A, B, C], in the presence of limitations on vehicle capacity or route length, could be that the vehicle drives from the depot to node A and then back to the depot again, before it drives to nodes B and C.

Chromosomes are evaluated by a fitness function. The fitness-function can simply be the cost function, or it can have additional elements included, to measure how good this chromosome is.

The basic version of a genetic algorithm consists of three operators: selection, crossover and mutation. The selection-operator chooses two “parent” chromosomes from the population, usually with a bias for better solutions. Genes from these parent chromosomes are then combines, according to the crossover-operator, to create offspring, i.e. new chromosomes (Mitchell, 1998). Using a single point (middle) crossover-operator. The two parent chromosomes 000000 and 111111 can create the two offspring 000111 and 111000. Finally, the mutation-operator is applied to increase the diversity of the population. Common mutations are to flip one bit, or to swap the alleles of two genes. Applying the flip operator on the first locus of the chromosome 000111 gives the chromosome 100111, while applying the swap operator on the 3rd and 4th loci gives the chromosome 001011. For VRPs, the typical chromosome is a sequence of nodes in the order they are visited in the solution. The crossover then needs to be modified, to only have one occurrence of each node in the chromosome. With the parent chromosomes [1, 2, 3, 4, 5, 6] and [6, 5, 4, 3, 2, 1], and a crossover point in the middle, we start by copying the genes preceding the crossover point from one parent, e.g. [1, 2, 3] from parent one, and then continue copying the genes from the other parent. Because all the nodes from the second half ([3, 2, 1]) already are in the offspring chromosome, we skip them and start copying from the beginning of the parent chromosome until the crossover point ([6, 5, 4]), giving the offspring [1, 2, 3, 6, 5, 4]. The other offspring will then be [6, 5, 4, 1, 2, 3]. To decode the chromosome, we need to insert the depot in the beginning and in the end. If we are dealing with more than one route, we also need divide the sequence into separate routes and insert the depot at the beginning and end of each of them. One way to separate the routes is to split sequence before the cumulative load breaches the vehicle capacity. When choosing the mutation operator for a genetic algorithm for the VRP, the most obvious choice is to swap two nodes, as it is not possible to flip a non-binary value.

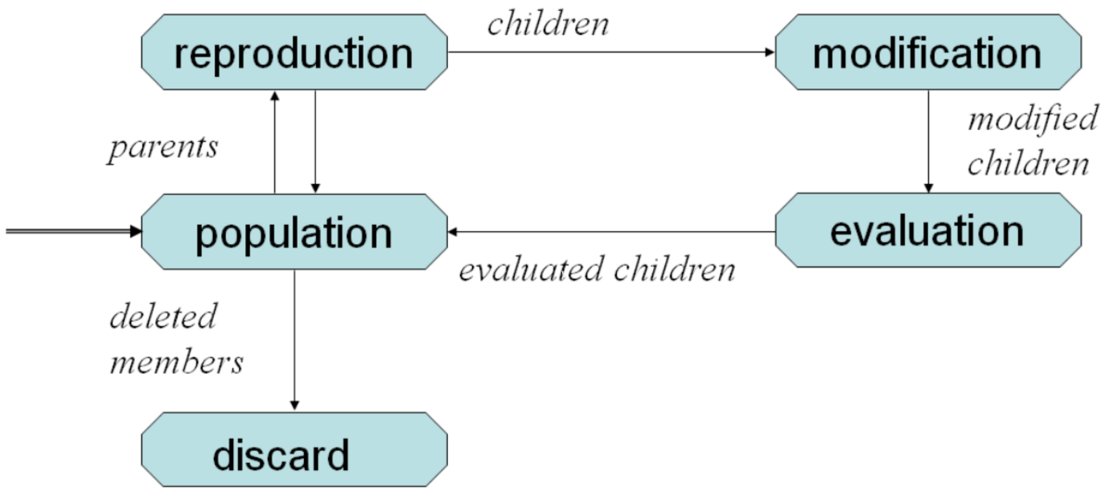

Figure 1: Genetic algorithm scheme, from Hvattum (2017)

The diagram above illustrates the general framework of genetic algorithms. Parent chromosomes are selected from the population and used in the reproduction of children, i.e. offspring chromosomes. These children are then modified or mutated in some way, before they are evaluated and introduced into the general population. To avoid "over population", parts of the population are discarded. In general, one wants to discard chromosomes with the worst evaluation, but some might be kept in order to maintain a certain level of diversity in the population.

The pseudo code for a genetic algorithm by Reeves (2010) is given below:

```
1: Choose an initial population of chromosomes
2:     while stopping criterion not met do
3:         while sufficient offspring has not been created do
4:             if recombination condition satisfied then
5:                 Select parent chromosomes
6:                 Choose recombination parameters
7:                 Perform recombination
8:             end if
9:             if mutation condition satisfied then
10:                Choose mutation points
11:                Perform mutation
```

```
12:                 **end if**
13:                 Evaluate fitness of offspring
14:     **end while**
15:     Select new population
16: **end while**
```

This is a very general description of a genetic algorithm, that leaves out the decisions that need to be made for each step when it is implemented.

3.2.2 Matheuristics

Matheuristics are heuristic algorithms that are made by combining metaheuristics and mathematical programming. (Boschetti et al. 2009). Recently, Matheuristics have been applied to routing problems in a variety of ways. This development is due to advances both in hardware and in the solvers used for mixed integer linear models (Archetti & Speranza 2014).

3.3 Waste Management

The field of waste management is concerns questions relating to the collection, disposal and recycling of waste. Here we are specifically addressing the issue of collecting solid waste, in an urban area. Urban areas have a high population density, and therefore also a high production of waste that needs to be removed from that area. Several methods for waste management have been studies, and more recently, as technology has progressed, smart management has gained interest from both practitioners and academics.

3.3.1 Classical Waste Management

Buhrkal, Larsen & Ropke (2012) study the waste collection vehicle routing problem with time windows (WCVRPTW). The element of time windows is introduced because of limitations on the customer side, and labor laws regarding rest time for the drivers. The WCVRPTW they study is different from the tradition VRPTW because the vehicles have to drive to disposal sites to empty their load before they start another route or return to the depot. They propose an adaptive large neighborhood search (ALNS) metaheuristic to solve the problem. They use a greedy heuristic to generate the initial solution. The ALNS uses several destroy and repair methods to improve the solution. The test their method on instances from Kim et al. (2006) and their own instances from the Copenhagen area in Denmark.

Karadimas, Papatzelou & Loumos (2007) performed a case study on a small area in Greece. Their objective is to minimize the time, and not the total distance, which is usually the case. They first cluster bins in a way that makes sure each cluster is below the vehicle capacity. This lets them reduce the problem to a asymmetric TSP. Their genetic algorithm finds routes that have low durations, but they stress that it should be tested on bigger instances.

3.3.2 Smart Waste Management

In a paper on the smart waste collection routing problem, the authors tackle the problem with three different approaches, dubbed “limited”, “smart” and “smarter” (Ramos, de Morais and Barbosa-Póvoa 2018). In the limited approach, they make receive daily readings of the bins’ fill-evels, and then make routing decisions for that day. First, they employ a heuristic to decide which bins to empty. The heuristic is simply to empty all bins that are filled over a certain level. After removing all bins below this level from the problem, it is modelled as a capacitated vehicle routing problem, using the two-commodity network flow formulation of Baldacci, Hadjiconstantinou, and Mingozzi (2004) to minimize the transportation cost. In the smart approach, they simultaneously choose which bins to empty and the routing of the vehicles. This approach allows that a certain percentage of bins may overflow. The number of vehicles used is decided by the model, and there is piecewise penalty for using vehicles. They solve the problem as a mixed integer linear programming, by adapting the formulation used in the limited approach. The objective function of the model is the profit, calculated as revenues from selling garbage collected from the bins, less the penalty for using vehicles and a variable cost that is a linear function of the distance travelled. In the smarter approach, they use a heuristic to determine when the model from the smart approach should be run to maximize profit. They perform a case study for a company with a homogenous vehicle fleet collecting waste in Portugal

Mes, Schutten & Rivera (2014) model the smart waste management problem as an IRP, and develop a heuristic method to solve it. The heuristic divides the bins into three groups, based on whether they need to be emptied on that day, may be emptied on that day, or should not be emptied on that day. The heuristic makes short-term plans, but also considers the long-term performance. They test their heuristic on a real-world instance from a Dutch waste collection company, and show that costs can be reduced by 40%.

Sharifyadzi and Flygansvær (2015) studied the effect of availability of fill-rate data from sensors installed in waste containers in the Oslo municipality. Using simulation techniques, they evaluated two systems for waste collection. The base case, where no sensor data was available and routes and schedules were fixed, and the case where containers had sensors. The data for the study consisted of 1003 containers, of 5 different sizes, in 851 locations and 2 trucks. In the base case a static route is made by using the nearest neighbor algorithm. This is the same method as the operating company was using. For the case with sensors, they use a modified version of the Clarke & Wright savings algorithm. The two systems are evaluated on the metrics total haulage distance, service level, and capacity utilization. The dynamic system with sensors outperformed the base case on all three metrics, with a reduction in haulage distance of 8.2%. The authors conclude that there are both environmental and financial benefits to be gained from the use of the sensors, stating that "it is likely that investment on the sensors pay off" (Sharifyazdi & Flygansvær, 2015). However, they do not look at set up and maintenance costs of the sensors. They also don't investigate if there are any opportunities to further optimize the routing without using sensors, e.g. by using different routing algorithms.

4.0 Mathematical Models

In this section two mixed integer programming models are developed. Both models are 3-index vehicle flow models. The second model represents the real-world situation more accurately, but as it is more complex it has computationally harder to solve. The models are meant to be solved at the beginning of each workday. They will give the routes for the current day, and a plan for the ones to come. On the next day, the model is solved again using updated data from the sensors, and the schedule and routes are updated.

4.1 Basic Model

The basic model is a three-index vehicle flow model for smart waste management. The current bin levels should be combined with the growth rate to feed the model a schedule for which days each bin can be emptied.

4.1.1 Notation

4.1.1.1 Sets

V	the set of all nodes, where the depot is located at node 0.
D	the set of all days included in the planning horizon.

4.1.1.2 Variables

x_{ijd}	a binary variable taking the value 1 if the edge between vertex *i* and *j* is included in a route on day *d*.
u_{id}	a continuous variable representing the vehicle's load when departing from node *i*.

4.1.1.3 Parameters

Q	the vehicle capacity.
c_{ij}	the cost (distance) of traversing the edge between nodes *i* and *j*.
a_{id}	a parameter set to 1 if node *i* is allowed to be serviced on day *d*, 0 otherwise.
b	the capacity for the number of routes per day.

q_{id}	the fill level of the bin located in node *i* on day *d*. For *d* = 0 this level is the actual level measured by the sensor, while for *d* > 0 it is the expected level, calculated as $q_{id} = q_{i0} + g_i * d$, where g_i is the expected daily production of waste.

4.1.2 Formulation

$$minimize \sum_{i\in V}\sum_{j\in V}\sum_{d\in D} c_{ij}x_{ijd} \qquad (1)$$

$$subject\ to$$

$$\sum_{j\in V} x_{0jd} \le b \qquad \forall d \in D \qquad (2)$$

$$\sum_{j\in V}\sum_{d\in D} \mathrm{x}_{\mathrm{ijd}} = 1 \qquad \forall \mathrm{i} \in \mathrm{V}\backslash\{0\} \qquad (3)$$

$$\sum_{j\in V} x_{ijd} = \sum_{j\in V} x_{jid} \qquad \forall j \in V, d \in D \qquad (4)$$

$$\sum_{i\in V} x_{ijd} \le a_{jd} \qquad \forall j \in V\backslash\{0\}, \mathrm{d} \in D \qquad (5)$$

$$u_{id} - u_{jd} \ge q_{jd} - Q(1 - x_{ijd}) \qquad \forall i \in V\{0\}, j \in V\{0\}, d \in D \qquad (6)$$

$$0 \le u_i \le Q \qquad \forall i \in V\backslash\{0\} \qquad (7)$$

$$x_{ijd} \in 0,1 \qquad \forall i \in V, j \in V, d \in D \qquad (8)$$

4.1.3 Model Description

The objective function (1) is the total distance travelled. The first constraint (2) limits the number of routes for each day. The limit on the number of trips was chosen over a duration constraint, because in most real-world applications, most of the driving time is between the landfill and zone where the bins are located. Constraints (3) and (4) ensure that each node is visited once during the planning horizon, and that the balance of flow is maintained at each node. Constraint (5) is used to limit the days when a node can be visited. It can be used to exclude non-working days, but the primary purpose is to enforce that the bins are emptied before they overflow. Constraint (6) eliminates sub-cycles and, together with (7), ensures that the load stays within the vehicle capacity. (8) is the binary-constraint for the *x*-variable.

4.2 Extended Model

This model is an extension of the basic model, with more flexibility. It incorporates elements such as overflow, additional routes and multiple visits of the nodes in the planning horizon. Because of this it is more complex and computationally more demanding to solve. The model has also been included in Hrabec et al. (2019).

4.2.1 Notation

4.2.1.1 Sets

V	the set of all nodes where bins are located.
V_0	the set off all nodes, including the depot located in node 0.
D	the set of all days in the planning horizon.
D^0	a subset of D, where day 0 has been excluded.

4.2.1.2 Variables

x_{ijd}	a binary variable taking the value 1 if the edge between vertex *i* and *j* is included in a route on day *d*.
y_{id}	a binary variable taking the value 1 if the bin located in node *i* is serviced on day *d*.
u_{id}	a continuous variable representing the vehicle's load when departing from node *i*.
q_{id}	a continuous variable representing the expected fill level at node *i* on day *d*.
w_{id}	a continuous variable representing the level of overflow at node *i* on day *d*.
v_{id}	a continuous variable used in the calculation of bin levels.
t_d	an integer variable representing additional vehicles used.

4.2.1.3 Parameters

c_{ij}	the cost (distance) of traversing the edge between nodes *i* and *j*.
a_d	a parameter set to 1 if node *i* is allowed to be serviced on day *d*, 0 otherwise.
Q	the vehicle capacity.

q_i^{sensor}	the fill level (utilization) of the bin located in node i on day 0, obtained from the sensors.
q_i^{max}	the capacity of the bin located at node *i*.
b	the capacity for the number of routes per day.
f	the minimum number of times the bins should be emptied during the planning horizon.
d	the penalty of an overflowing bin, per unit per day.
e	the cost of one addition route, representing overtime or outsourcing from another company.
M	big M.
g_i	the expected daily production rate at each node.

4.2.2 Formulation

$$minimize \sum_{i \in V_0} \sum_{\substack{j \in V_0 \\ i \neq j}} \sum_{d \in D} c_{ij} x_{ijd} + \sum_{i \in V} \sum_{d \in D^0} d w_{id} + \sum_{d \in D^0} e t_d \qquad (9)$$

$$subject\ to$$

$$\sum_{j \in V} x_{0jd} \leq b a_d + t_d \qquad \forall\, d \in D \qquad (10)$$

$$\sum_{\substack{i \in V \\ i \neq j}} \sum_{\mathrm{d} \in \mathrm{D}} \mathrm{x}_{\mathrm{ijd}} \geq f \qquad \forall j \in V \qquad (11)$$

$$\sum_{\substack{j \in \mathrm{V}_0 \\ i \neq j}} x_{ijd} = \sum_{\substack{j \in \mathrm{V}_0 \\ i \neq j}} x_{jid} \qquad \forall i \in V_0\,, d \in D \qquad (12)$$

$$u_{id} - u_{jd} \geq q_{jd} - Q\left(1 - x_{ijd}\right) \qquad \forall i,j \in V\ i \neq j, d \in D \qquad (13)$$

$$q_{i0} = q_i^{sensor} \qquad \forall i \in V \qquad (14)$$

$$y_{jd} = \sum_{\substack{j \in V \\ i \neq j}} x_{ijd} \qquad \forall j \in V, d \in D \qquad (15)$$

$$v_{id} \leq M(1 - y_{id-1}) \qquad \forall i \in V, d \in D^0 \qquad (16)$$

$$v_{id} \leq q_{id-1} \qquad \forall i \in V, d \in D^0 \qquad (17)$$

$$v_{id} \geq q_{id-1} - M y_{id-1} \qquad \forall i \in V, \mathrm{d} \in D^0 \qquad (18)$$

$$q_{id} = v_{id} + g_i \qquad \forall i \in V, d \in D^0 \tag{19}$$

$$w_{id} \geq q_{id} - q_i^{max} \qquad \forall i \in V, d \in D^0 \tag{20}$$

$$0 \leq u_{id} \leq Q \qquad \forall i \in V, d \in D \tag{21}$$

$$0 \leq q_{id} \qquad \forall i \in V, d \in D \tag{22}$$

$$x_{ijd} \in \{0,1\} \qquad \forall\, i, j \in V_0, i \neq j, d \in D \tag{23}$$

$$a_{id} \in \{0,1\} \qquad \forall i \in V, d \in D \tag{24}$$

$$0 \leq w_{id} \qquad \forall i \in V, d \in D^0 \tag{25}$$

$$0 \leq v_{id} \qquad \forall i \in V, d \in D^0 \tag{26}$$

$$y_{id} \in \{0,1\} \qquad \forall i \in V, d \in D^0 \tag{27}$$

$$t_d \in \mathbb{N}_0 \qquad \forall\, d \in D^0 \tag{28}$$

4.2.3 Model Description

The objective function (9) consists of three parts: the first one is the total distance travelled, while the second and third are penalty costs. The first penalty is related to the overflow of bins, and is the product of the penalty coefficient and the unit-days of overflow. The second penalty is the number of routes above the allowed number of daily routes. The number of additional routes is multiplied with the cost of having the crew work overtime or the cost of outsourcing the job. If these aren't viable options, the cost can be set to a sufficiently big number, to avoid additional routes. Note however that this may lead to infeasibility of the model.

Constraint (10) is similar to constraint (2) from the basic model, in that it limits the number of routes for each day. By introducing the variable t_d we also allow for additional routes to be created, at an extra cost. Constraint (11) is the same as constraint (3) from the basic model, but the instead of stating that each bin must be emptied at least once during the planning horizon, it allows us to require a different number. Depending on the type of waste and the length of the planning horizon, it might be reasonable to change this number. For example, organic waste might need to be emptied more frequently than paper and cardboard, due odor from the decomposing material. If it is suitable, the model can easily be changed to have the *f*-parameter set individually for each bin. The balance constraint (12) is the same as (4) in the basic model. The load constraint (13) is also similar to that of the basic model, but in this model q_{id} is a variable for the bin levels. This is done to allow for multiple visits through the horizon. Constraint (14) initializes the day zero bin level variables to the values collected from the sensors. Constraint (15) is just used to increase the readability of the model, as it sets the variable y_{jd} to 1 if bin j is emptied on day d. Constraints (16) through (19) are used to reset the bin level if the bin is emptied the previous day. This is necessary to have solutions with multiple visits in the solution space. Constraint (20) accounts for the overflow, so that it can be included in the objective function. The constraints (21) through (28) set the domains for the decision variables.

5.0 Matheuristic Implementation

The problem of waste management can be split into two main sub-problems. Decisions about Scheduling – at what point in time the bins should be emptied, and routing – in which order the garbage truck should empty the bins. The matheuristic method developed uses a genetic algorithm to solve the scheduling, and then the routing is solved exactly using mixed integer linear programming. Because the schedule is solved first, the complexity of the routing problem is greatly reduced, and exact methods are more feasible.

5.1 A Genetic Algorithm for the Scheduling Sub-Problem

5.1.1 Chromosome Representation

The table below represents the scheduling for a waste management problem with 10 nodes and a 5-day planning horizon.

Node\day	0	1	2	3	4
1	1	0	1	0	1
2	1	0	0	1	0
3	0	1	1	0	0
4	0	1	0	0	0
5	0	0	1	0	0

Each row represents the 5-day emptying schedule for the given bin. A value of 1 means the bin is scheduled to be emptied, and 0 means it is not. Bin 1, for example, is scheduled to be emptied today (day 0), in two days, and again in 4 days.
To represent the schedule in a way that is better suited for genetic algorithms, we can flatten it. To flatten it simply means we change the representation from 2-dimensional to 1-dimensional. This is done by moving each row to the right of the one that's originally above it, giving the following 1-dimensional array:

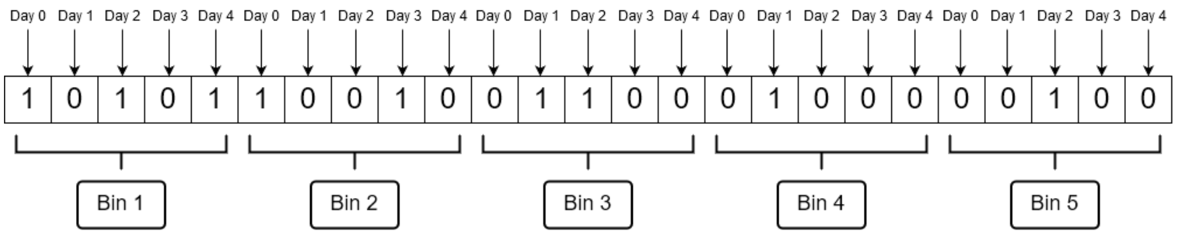

Figure 2: Chromosome representation

This representation of the solution is suitable as a chromosome in the genetic algorithm framework.

5.1.2 Population Initialization

The population is initialized with a total of $4n$ chromosomes. The population is divided into evenly sized groups. All chromosomes in a group have the same number of 1-values, but they are shuffled around. The first group has n 1-values, and the next group has an increasing amount. The idea behind this is that in a feasible solution, each bin must be emptied at least once. An alternative approach is to initialize the population with random solutions.

5.1.3 Genetic Operators

In the reproduction phase, two parent-chromosomes are selected at random from the population. The crossover-operator is then applied on these two parents, spawning two child-chromosomes. The crossover-operator chooses a random crossover-point. One child inherits the genetic information before the crossover point from one parent and the information after it from the other parent. The other child does the same, but with the parents swapped. This process is illustrated below:

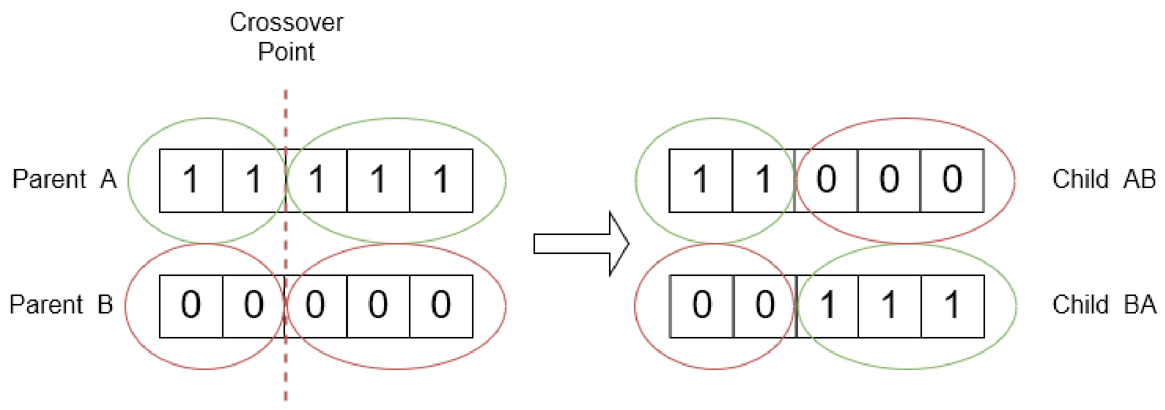

Figure 3: Crossover

After the crossover, there is a 50% chance of a gene to be mutated. The mutation is done by taking a gene and changing it from 1 to 0 or from 1 to 0. In the illustration below, the fifth gene is mutated:

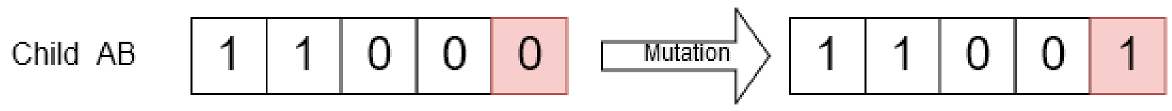

Figure 4: Mutation

The reproduction phase continues until the population is doubled in size, and then the solutions are evaluated by the evaluation function.

5.2 An Exact Method for the Routing Sub-Problem

Each chromosome from the genetic algorithm represents a schedule of which bins are emptied on which days. This schedule is passed to the model from chapter 4.2. With the schedule being fixed, the complexity of the model is greatly reduced, and it can much easier be solved exactly. The model is solved in AMPL, using the CPLEX solver. After solving the model, the objective function value is returned to the genetic algorithm, as the fitness of the chromosome.

5.3 Inefficiencies of the Implementation

The matheuristic is implemented in the Python programming language, which is drastically slower than compiled languages such as C and C++. Python was chosen for its readability and because it allows for a short way between idea and prototype. There are libraries available to increase the speed of python code, but this implementation is done using the standard library. The interaction with the CPLEX solver is done using the AMPL API.

Apart from the choice of language, an obvious optimization is parallelization. Because the evaluation of a chromosome is independent of other chromosomes, the evaluation function could be parallelized.

6.0 Data and Testing

This section describes how the test instances are generated, and how the methods are tested.

6.1 Test Instances

The computational experiments are conducted on synthetic data. We generate coordinates for the nodes by sampling *n+1* pseudo random numbers for both axes. The first pair of coordinates are those of the depot. The fill-levels at the nodes are generated in a similar fashion.

An example of the data for a test instance is shown in the table:

Node	0	1	2	3	4	5	6	7	8	9	10
X	58	19	97	92	47	66	21	22	29	69	21
Y	97	7	19	9	77	37	47	33	64	40	21
Utilization	0	55	59	38	27	14	83	79	87	42	85

And a graphical representation of the instance is given below. The green hexagon is the depot, and the other ten markers are the locations of the bins. The size of the inner, blue circle illustrates the fill-level relative to the size of the bin capacity, shown in pink.

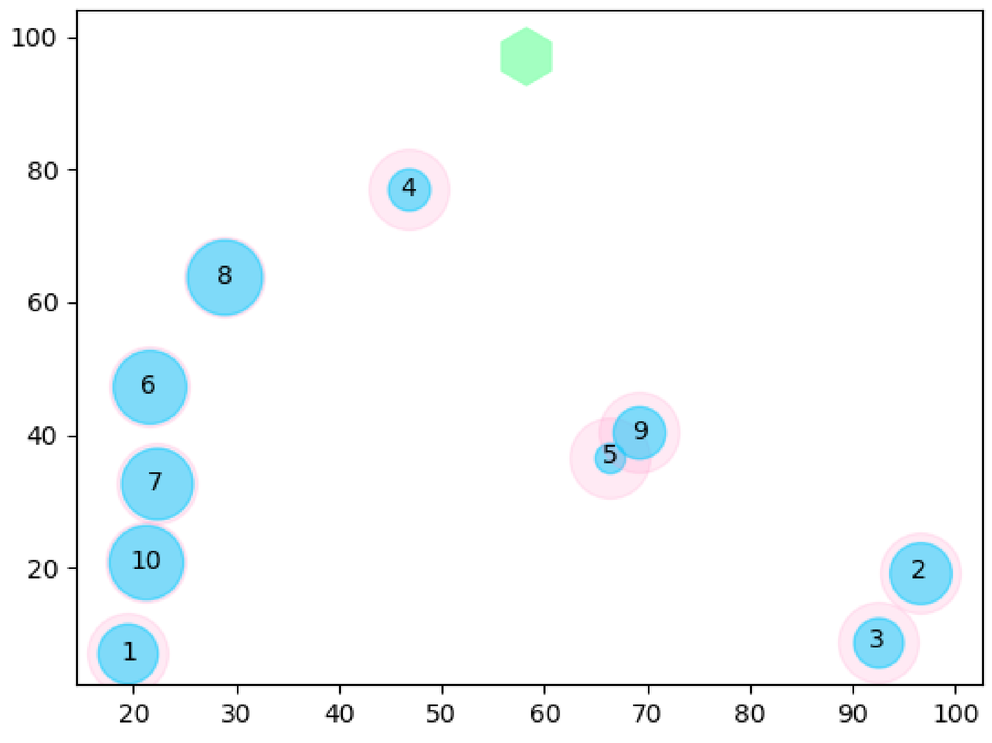

Figure 5: Instance map

The test instances are named according to the number of bins and the number of allowed daily routes. The instance 8-1, for example, has 8 bins and we allow 1 route each day.

6.2 Distances

We use the Euclidean distances between nodes, i.e. $c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, giving the following symmetrical distance matrix for the example test instance:

	0	1	2	3	4	5	6	7	8	9	10
0	0	98	86.7	94.7	23.1	61	62	73.8	44.4	57.8	84.7
1	98	0	78	72.9	75.1	55.4	40.2	25.7	57.5	59.9	13.9
2	86.7	78	0	11.2	76.2	34.8	80.1	75.5	80.9	34.5	75.3
3	94.7	72.9	11.2	0	82	38.1	80.6	74.1	83.9	39.1	72.2
4	23.1	75.1	76.2	82	0	44.9	39.1	50.7	22.2	43	61.7
5	61	55.4	34.8	38.1	44.9	0	46.1	44.4	46.3	4.7	47.8
6	62	40.2	80.1	80.6	39.1	46.1	0	14.6	18.2	48.3	26.3

7	73.8	25.7	75.5	74.1	50.7	44.4	14.6	0	31.8	47.7	11.8
8	44.4	57.5	80.9	83.9	22.2	46.3	18.2	31.8	0	46.7	43.6
9	57.8	59.9	34.5	39.1	43	4.7	48.3	47.7	46.7	0	51.8
10	84.7	13.9	75.3	72.2	61.7	47.8	26.3	11.8	43.6	51.8	0

6.3 Testing

The models are run on the test instances, to see how they compare to each other with regards to the time it takes to solve them and the quality of the solutions. When the optimal solution isn't verified within the time limit, the gap to the lower bound is analyzed.

6.4 Simulation

To find out how the whether the solution methods are suitable for real world application, they must be tested over time. To do this, the process is simulated. The model is solved, and the bins chosen to be emptied on the first day have their levels reset. Then the bin-levels are updated with sampled growth numbers, and the model is solved again. This is done using a program written in the Python programming language. The simulation is done using the extended model, and an instance with 5 nodes.

7.0 Results

In this section the results from the tests are presented. Results the quality and runtime of the models is presented, and then the matheuristic and simulation results.

7.1 Solution Methods

7.1.1 Exact Methods

This section present findings obtained when using the two models to solve test instances. CPLEX version 12.9.0.0 is used as the solver for all instances. The solver is given a time limit of 10 minutes, but other than that all settings are at the default values. The models are solved with a horizon of five days.

7.1.1.1 Solution Quality

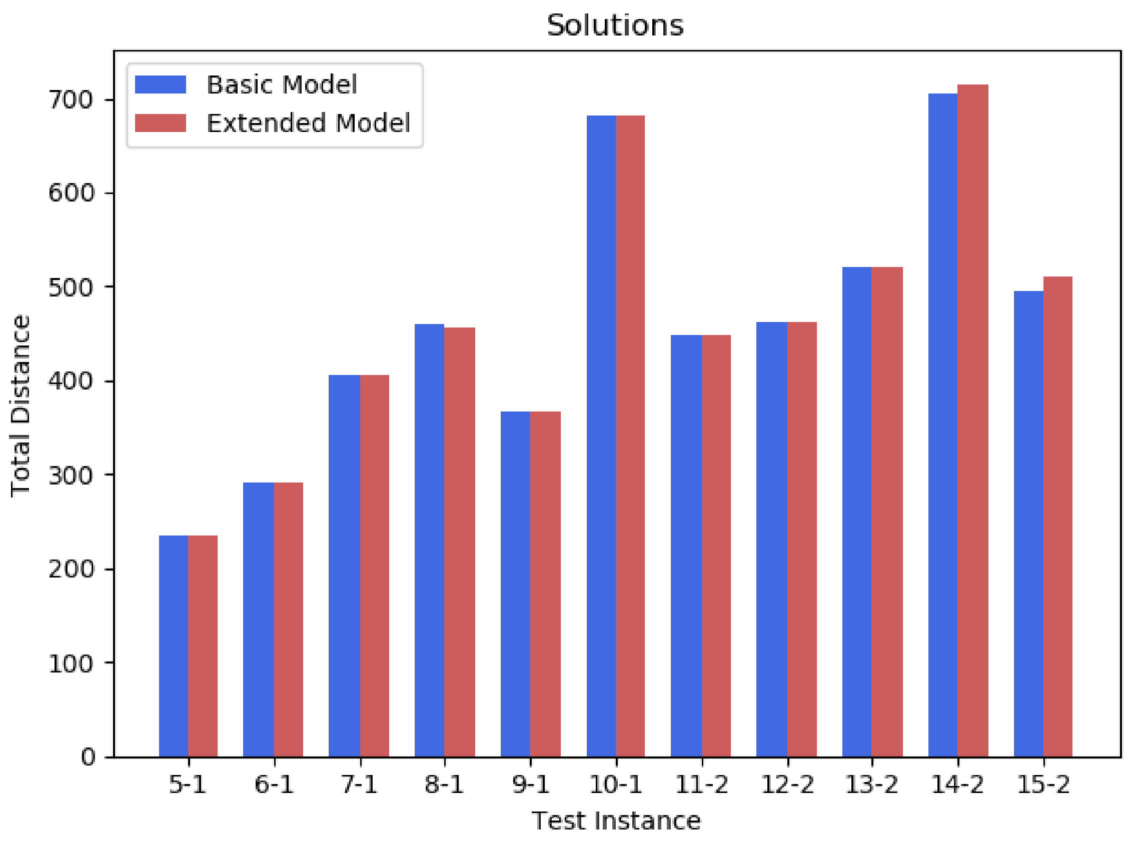

Figure 6: Solution Quality

Both models were used to solve 11 different test instances. The instances range in size from 5 to 15 bins. For the instances with 5 to 10 bins we allow for one route each day. To ensure feasibility, the instances with more than 10 bins were solved with up to two daily routes. For most of the instances the two models find the same optimal solution. For instance 8-1, a slightly better solution is found using the extended model. Because the solution space for the basic model is a subspace of that of the extended model, the extended model's optimum is always the same as or better than the basic model's optimum. For instances 14-2 and 15-2 we get better solutions using the basic model because the allotted 10-minute runtime isn't enough for the extended model to find the optimum. The reason why the extended model so rarely fins a better solution than the basic model is likely because of the short planning horizon. The bins have an expected daily growth rate of 10 units, and a capacity of 100 units, meaning there is in average 10 days from the time a bin is emptied until it has to be emptied again.

7.1.1.2 Solution Time

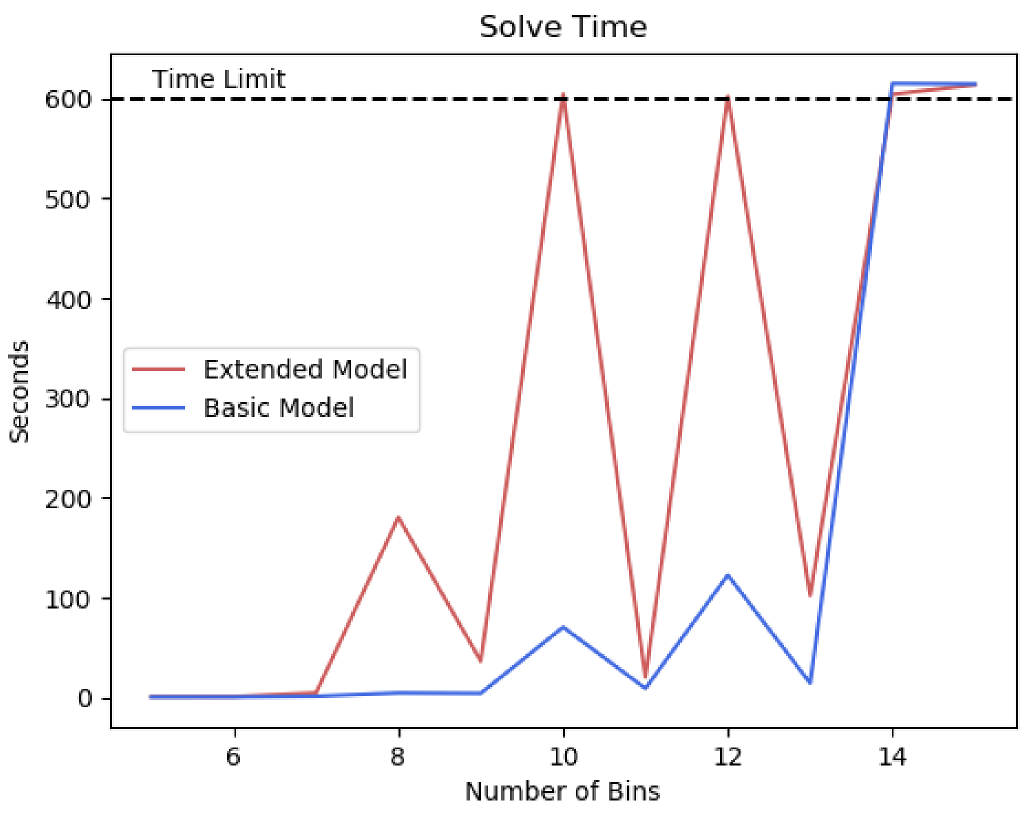

Figure 7: Solution Time

The figure above shows how the time to find the solution. The dotted line shows the ten-minute time limit we set. In general, we can see that it takes less time to solve the basic model than the extended model. When going from 10 to 11 bins, both modes are solved in less time. This might have something to do with the fact that we allowed for one more route per day. There seems to be a point around n=14 where the models can't be solved within the time limit.

The figure below shows the relative MIP gap of the solutions for instances with 10 to 15 bins. There is quite a bit of variation from instance to instance, but again we see that the basic model is doing better than the extended model. The biggest relative MIP gap is just over 40%., meaning the best bound is within 40% of the solution obtained.

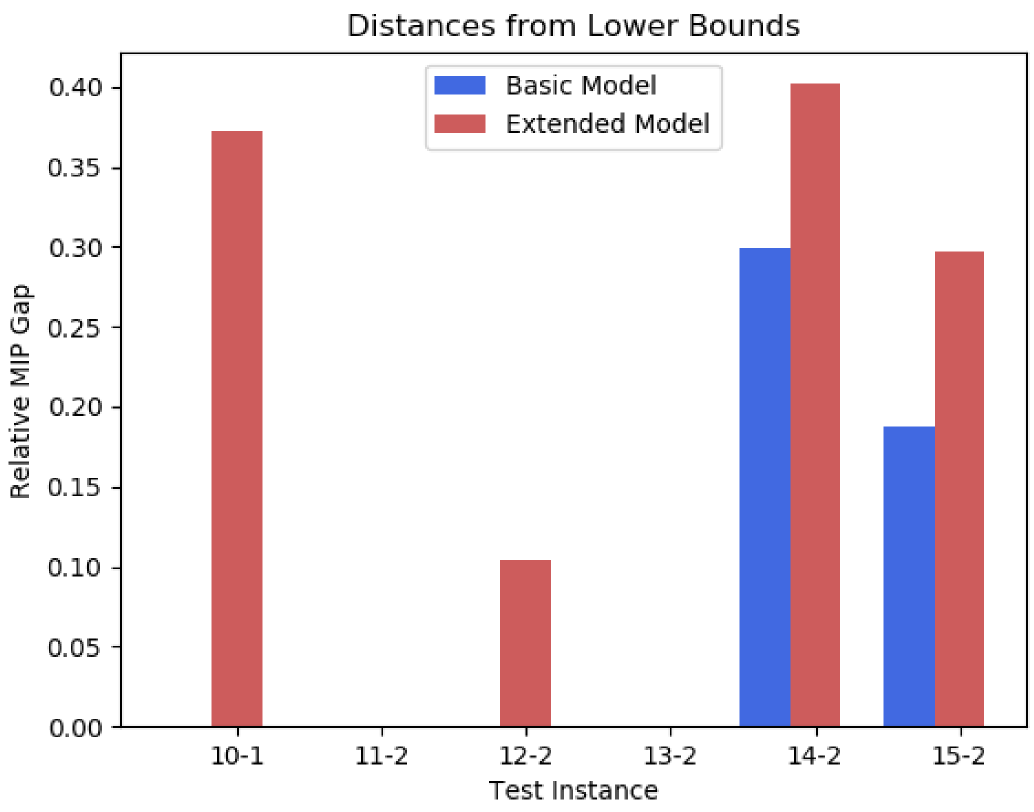

Figure 8: Relative MIP Gap

It is clear that the models aren't given enough time to verify the optimality of the solutions of larger instances, but that doesn't necessarily mean that the solutions are bad. Take the 12-2 instance: both models find the same solution, but the extended model has a relative MIP gap of about 10%, while the basic model's solution is verified as optimal. In addition, there isn't any clear indication that the models always perform worse for larger instances.

For example, the instance 15-2 has a lower relative MIP gap than 14-2. We can't make any conclusions about this, since we only have a single observation for each instance size.

7.1.2 Matheuristic

The matheuristic comprised of the genetic algorithm and the extended model was used to solve all the test instances. The genetic algorithm went through 20 iterations (generations). To evaluate each solution, the chromosome was sent to AMPL, and the y-variable was fixed to the values of the genes. The time limit for CPLEX was set to 10 seconds, which in most cases was sufficient to find the optimal solution, given the tightening done by fixing all y-variables. The parameter deciding how many times in the planning horizon a bin has to be emptied was set to 0. This was done because it shouldn't be necessary to empty the bins that often, when they on average take 10 days to fill up. The bins should rather be empty to avoid the penalty associated with overflow. Because of a technical issue with the population initialization, a random population of size 20 was used. The mutation chance was set to 0.9. In the figure below, the best solution from each generation is plotted.

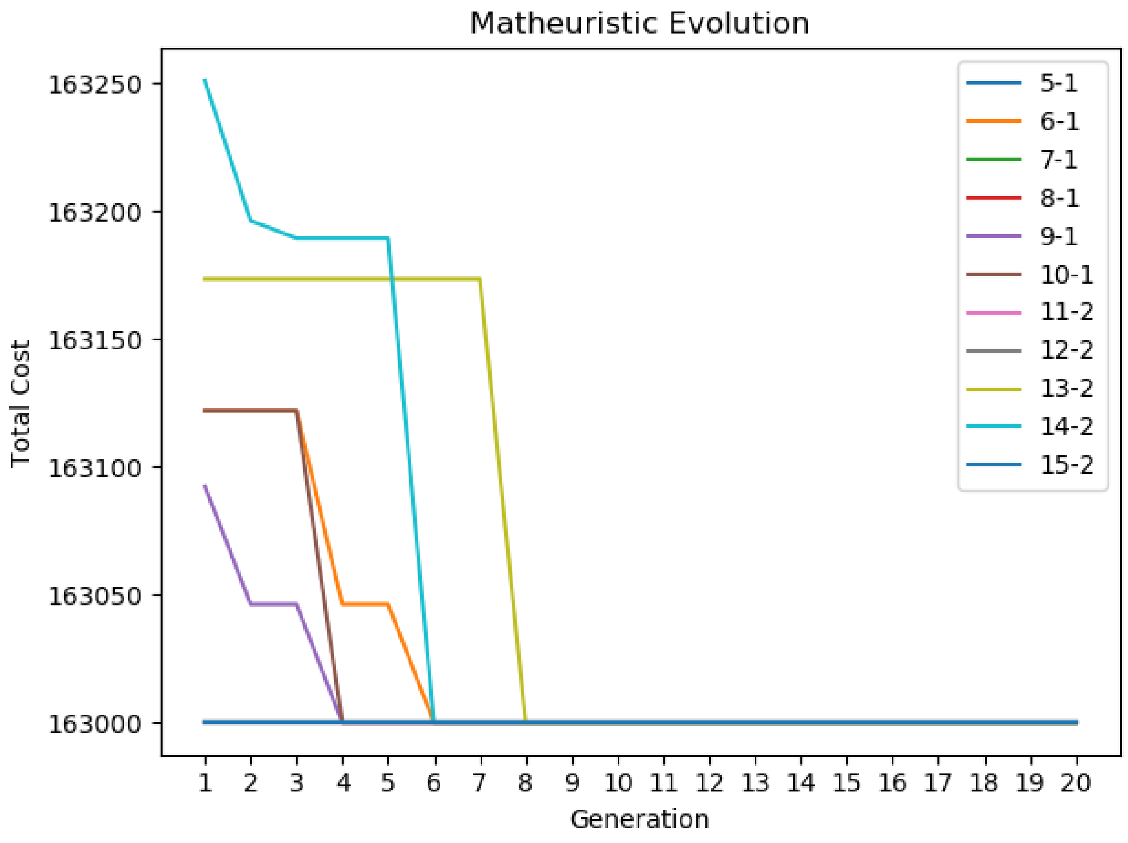

Figure 9: Matheuristic Quality

All instances start with a very high total cost. In most cases a slight improvement is made over the first generations, but they al stagnate before the ninth generation. The total cost is comprised mostly of penalties, and not the actual distance travelled. The total runtime of the metaheuristic is plotted below. There is a slight bit of variation, but it is linearly corelated with the number of bins.

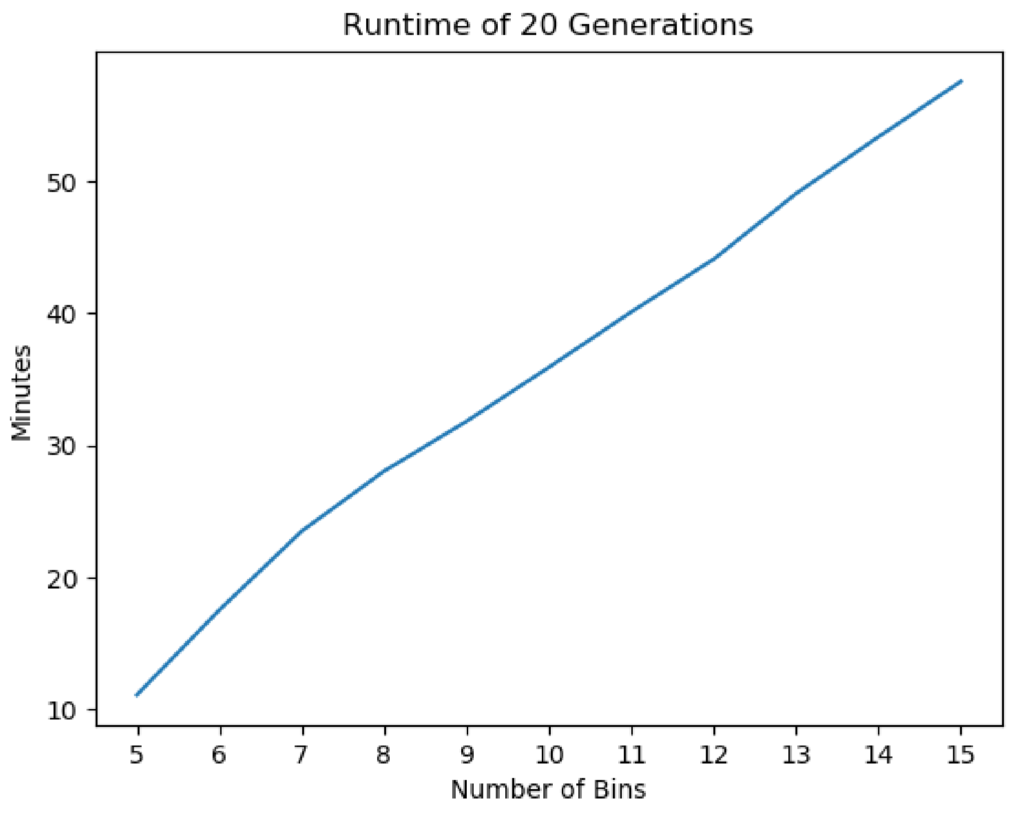

Figure 10: Matheuristic Runtime

The smallest instance (n=5) takes just over 10 minutes, and the largest (n=15) takes just under an hour. Considering that the models found fairly good solution in 10 minutes for all instances, the matheuristic isn't very useful for this problem.

7.2 Simulation

The simulation was done using the extended model, and an instance with 5 nodes. 7 days were used both as the planning horizon in the model and as the number of days simulated. The simulation was done with 10 and 20 as the expected daily production of waste, with standard deviations of 5 and 2.5, respectively.

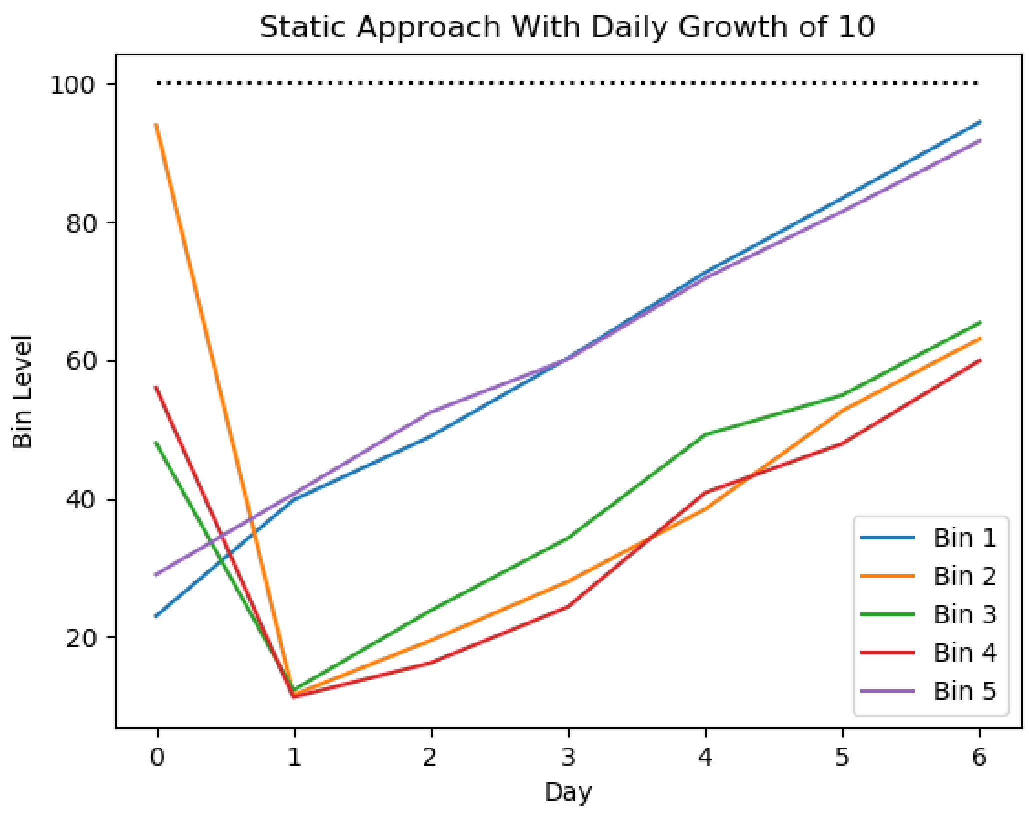

Figure 11: Static approach for the low growth case

In the static approach, the model is only solved one time, and the waste collection plan is executed regardless of how the bin levels deviate from what's expected. In the figure above we see the bin levels for the five bins over the 7-day period. Bins 2, 3 and 4 are emptied on the first day (day 0), while the other bins aren't emptied in the 7-day period. The total distance for the solution is 190.2 and there aren't any overflows.

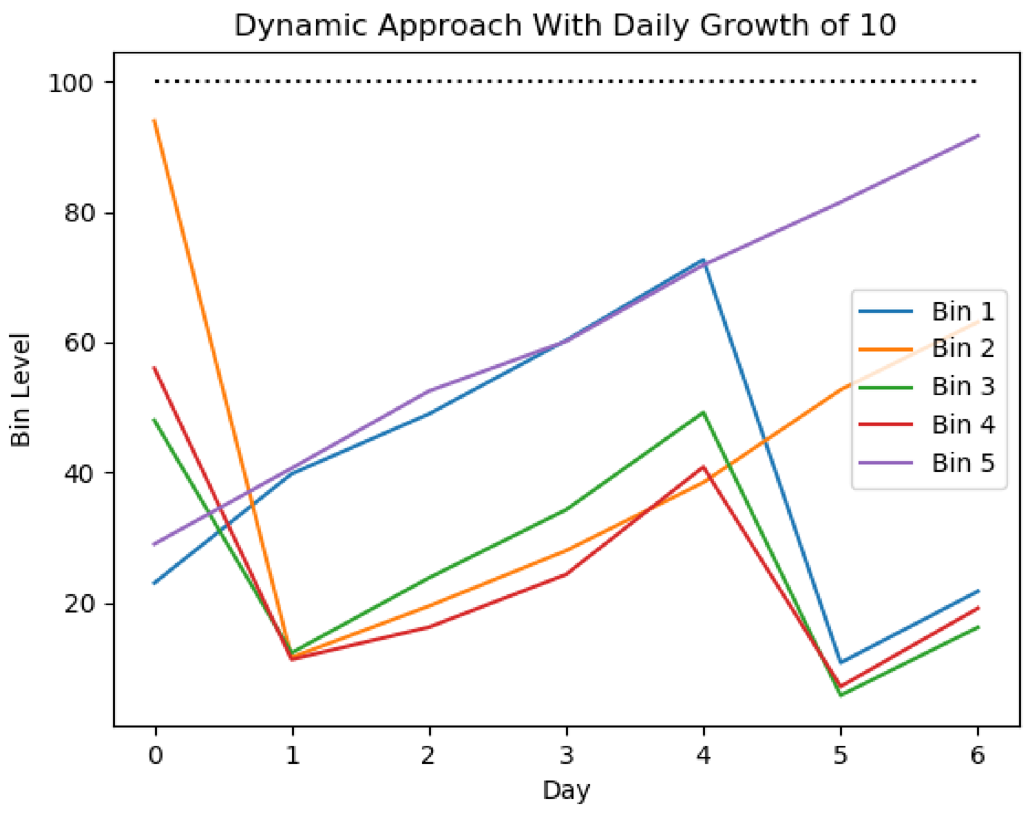

Figure 12: Dynamic approach for the low growth case

In the dynamic approach we solve the model every day, using the updated bin levels. Bins 2, 3 and 4 are emptied on the first day. On day 4 bins 1, 2 and 3 are emptied. Bin 5 isn't emptied, but most likely would be if we simulated one more day. The total distance is 492.8 and there no bins overflow.

With a growth of 10 and only seven day, not all bins need to be emptied, and therefore the method isn't properly tested. Because of this, the expected growth was increased to 20, and the simulation was run again.

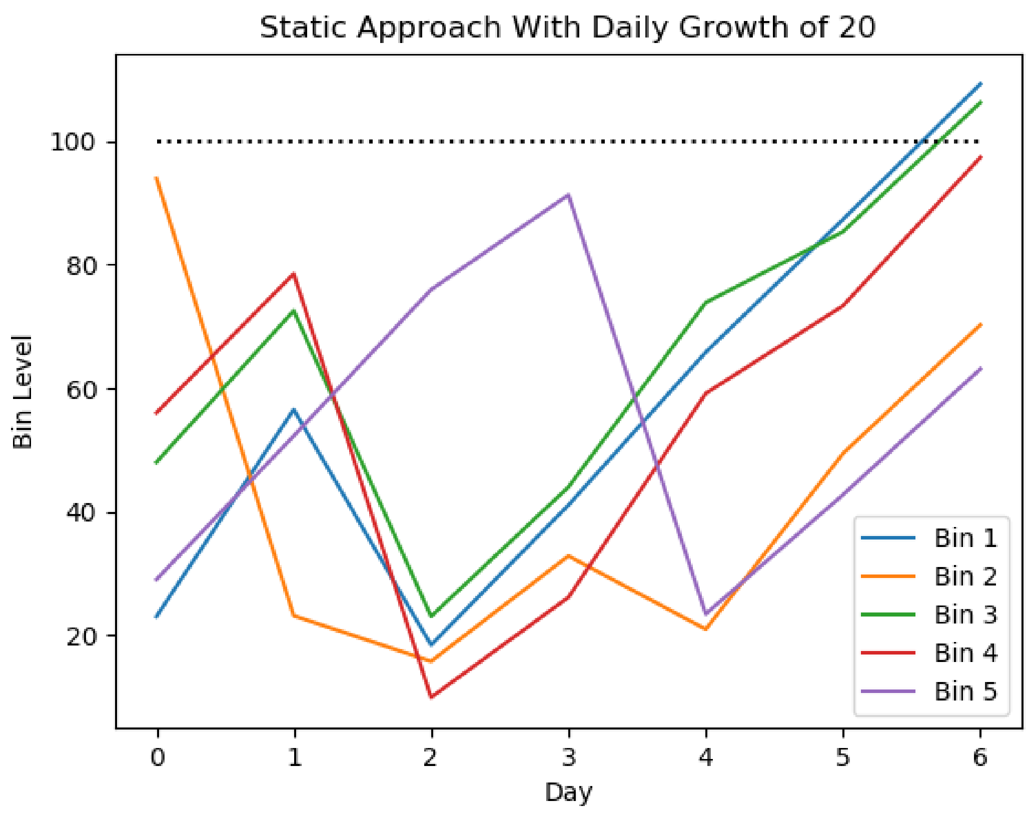

Figure 13: Static approach for the high growth case

With a static approach for the higher growth scenario, there is a lot more activity. All bins are emptied during the 7-day period, and on the last day bins 1 and 3 are overflowing. Bin 4 is also close to overflowing, and most likely will be over its capacity on the next day. The total distance with the static approach is 367.6.

With the dynamic approach, which is plotted below, only one bin overflows during the simulation. The total distance is 678.2.

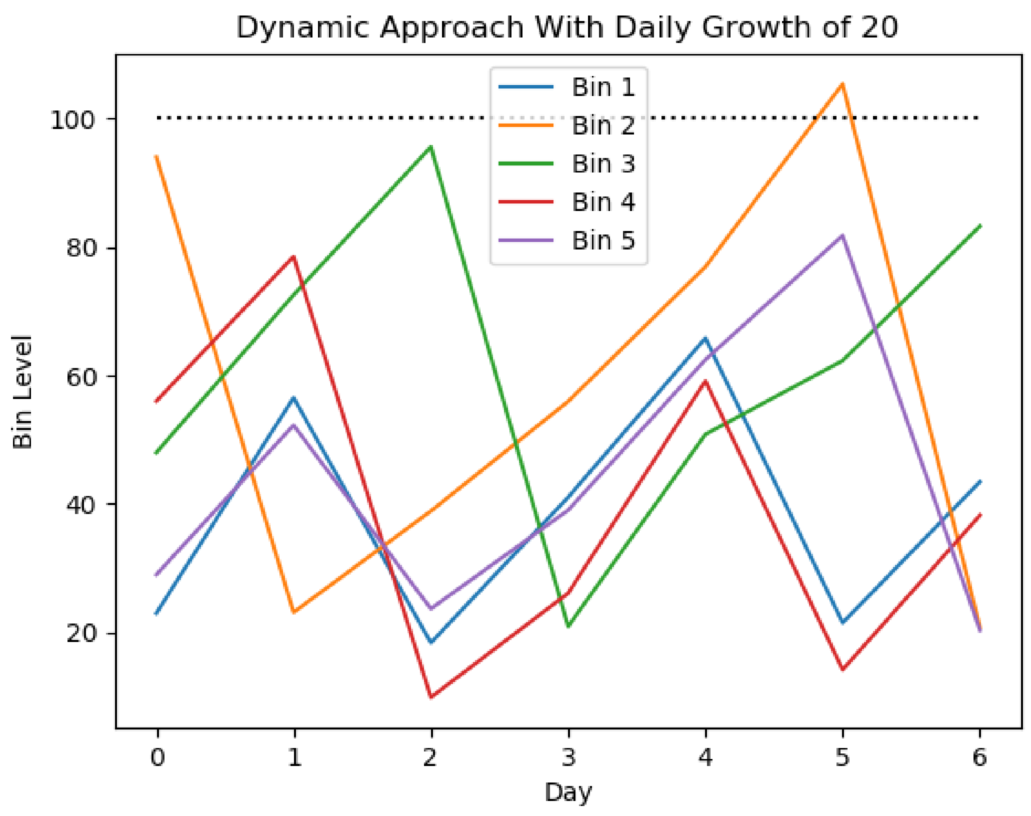

Figure 14: Dynamic approach for the high growth case

For both the high growth and the low growth cases, the static method provides the shortest total distance. When using the static method, there is a buildup towards the end of the simulations, and it might be worth investigating what will happen over a longer horizon.

8.0 Final Remarks

Two models have been developed. One basic, that's quite manageable for linear solvers, and one with more features that is more demanding. The second model was incorporated with a genetic algorithm in a matheuristic. The models outperform the matheuristic, both with regards to the time it takes to find a solution and the quality of the solution. The second model was used in a simulation to analyze how it performs over time. To further develop solution methods for the smart waste management problem, it could be interesting to implement another kind of metaheuristic or to develop a classical heuristic tailored for the problem.

9.0 References

Andersson, H., Hoff, A., Christiansen, M., Hasle, G., & Løkketangen, A. (2010). Industrial aspects and literature survey: Combined inventory management and routing. *Computers & Operations Research*, *37*(9), 1515-1536.

Archetti, C., & Speranza, M. G. (2014). A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, *2*(4), 223-246.

Archetti, C., Feillet, D., Gendreau, M., & Speranza, M. G. (2011). Complexity of the VRP and SDVRP. *Transportation Research Part C: Emerging Technologies*, *19*(5), 741-750.

Baldacci, R., Hadjiconstantinou, E., & Mingozzi, A. (2004). An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations research*, *52*(5), 723-738.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, *35*(3), 268-308.

Boschetti, M. A., Maniezzo, V., Roffilli, M., & Röhler, A. B. (2009, October). Matheuristics: Optimization, simulation and control. In *International Workshop on Hybrid Metaheuristics* (pp. 171-177). Springer, Berlin, Heidelberg.

Buhrkal, K., Larsen, A., & Ropke, S. (2012). The waste collection vehicle routing problem with time windows in a city logistics context. *Procedia-Social and Behavioral Sciences*, *39*, 241-254.

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, *12*(4), 568-581.

Coelho, L. C., Cordeau, J. F., & Laporte, G. (2013). Thirty years of inventory routing. *Transportation Science*, *48*(1), 1-19.

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management science*, *6*(1), 80-91.

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, *2*(4), 393-410.

Fogel, L. J. (1962, January). Toward inductive inference automata. In *Communications of the ACM* (Vol. 5, No. 6, pp. 319-319). 1515 BROADWAY, NEW YORK, NY 10036: ASSOC COMPUTING MACHINERY.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, *13*(5), 533-549.

Holland, J. H. (1975). Adaptation in natural and artificial systems, Univ. of Michigan. *Ann Arbor, MI, 1*, 206.

Hrabec D.,Senland P., Nevrlý V., Popela P., Hoff A., Šomplák R., Pavlas M (2019). Quantity-Predictive Vehicle Routing Problem for Smart Waste Collection. *Chemical Engineering Transactions,* In Press.

Hvattum, Lars Magnus. (2017). "An introduction to heuristics." Molde: Molde University College, December 22.

Karadimas, N. V., Papatzelou, K., & Loumos, V. G. (2007, September). Genetic algorithms for municipal solid waste collection and routing optimization. In *IFIP International Conference on Artificial Intelligence Applications and Innovations* (pp. 223-231). Springer, Boston, MA.

Kim, B. I., Kim, S., & Sahoo, S. (2006). Waste collection vehicle routing problem with time windows. *Computers & Operations Research*, *33*(12), 3624-3642.

Laporte, G., Gendreau, M., Potvin, J. Y., & Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research*, *7*(4-5), 285-300.

Marques, P., Manfroi, D., Deitos, E., Cegoni, J., Castilhos, R., Rochol, J., ... & Kunst, R. (2019). An IoT-based smart cities infrastructure architecture applied to a waste management scenario. *Ad Hoc Networks*, *87*, 200-208.

Mes, M., Schutten, M., & Rivera, A. P. (2014). Inventory routing for dynamic waste collection. *Waste management*, *34*(9), 1564-1576

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

Mole, R. H., & Jameson, S. R. (1976). A sequential route-building algorithm employing a generalised savings criterion. *Journal of the Operational Research Society*, *27*(2), 503-511.

Nicos Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388, Graduate School of Industrial Administration, CMU, 1976.

Rechenberg, I. (1973). Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Frommann–Holzboog.

Ramos, T. R. P., de Morais, C. S., & Barbosa-Povoa, A. P. (2018). The smart waste collection routing problem: Alternative operational management approaches. *Expert Systems with Applications*, *103*, 146-158.

Reeves, C. (2003). Genetic algorithms, Handbook of Metaheuristics. *Springer*, *10,* 0-306.

Sharifyazdim M., Flygansvær, B.M. (2015). DYNAMIC ROUTING IN REVERSE LOGISTICS: The effect of sensors in waste containers on uncertainty in Jæger, B. (2015). NOFOMA 2015: Post Conference Proceedings, Molde, 3-5 June 2015, Nordic Logistics Research Network

Sörensen, K., & Glover, F. W. (2013). Metaheuristics. *Encyclopedia of operations research and management science*, 960-970.

Voß, S., Martello, S., Osman, I. H., & Roucairol, C. (Eds.). (2012). *Meta-heuristics: Advances and trends in local search paradigms for optimization*. Springer Science & Business Media.

Xue, Y., Wen, Z., Bressers, H., & Ai, N. (2019). Can intelligent collection integrate informal sector for urban resource recycling in China?. *Journal of cleaner production*, *208*, 307-315.

10.0 Appendix

10.1 Basic Model AMPL

```
#Sets

set V;
set D;


#Parameters

param c {i in V, j in V: i<>j} >= 0;
param Q >= 0;
param q {V diff {0}, D} >= 0;
param a{V diff {0}, D};
param b;


#Variables

var x {i in V, j in V, D: i<>j} binary;
var u {V diff {0}, D} >= 0;


#Objective function

minimize Total_Distance: sum {i in V, j in V, d in D:i<>j} c[i,j] * x[i,j,d];


#Constraints

subject to maxRoutes {d in D}: sum {j in V:j<>0} x[0,j,d] <= b;
subject to outDegree {i in V diff {0}}: sum {j in V, d in D:i<>j} x[i,j,d] = 1;

subject to xContinuity {i in V, d in D}:
        sum {j in V:i<>j} x[i,j,d] = sum {j in V:i<>j} x[j,i,d];

subject to uContinuity {d in D, i in V diff {0}, j in V diff {0}: i<>j}:
        u[i,d] - u[j,d] >= q[j,d] - Q * (1 - x[i,j,d]);

subject to uUpper {i in V diff {0}, d in D}: u[i,d] <= Q;

subject to serviceDays {j in V diff {0}, d in D}: sum {i in V:i<>j} x[i,j,d] <= a[j,d];
```

10.2 Extended Model

```
# extended model

###

#Sets

set V;
set D;


#Parameters

param c {i in V, j in V: i<>j} >= 0;
param Q >= 0;
param q_sensor {V diff {0}} >= 0;
param q_max {V diff {0}} >= 0;
param g {V diff {0}} >= 0;
param M;
param a {D} >= 0;
param b >= 0;
param f >= 0;
param binover >= 0;
param e >= 0;

#Variables

var x {i in V, j in V, D: i<>j} binary;
var u {V diff {0}, D} >= 0;
var q {V diff {0}, D} >= 0;
#var a {V diff {0}, D} binary;
var y {V diff {0}, D} >= 0;
var w {V diff {0}, D diff {0}} >= 0;
var v {V diff {0}, D diff {0}} >= 0;
var t {D} integer >= 0;

#Objective function

minimize Total_Distance:
        sum {i in V, j in V, d in D:i<>j} c[i,j] * x[i,j,d]
        + sum {i in V diff {0}, d in D diff {0}} w[i,d] * binover
        + sum {d in D} t[d] * e;


#Constraints

subject to maxRoutes {d in D}:
        sum {j in V:j<>0} x[0,j,d] <= b * a[d] + t[d];

subject to outdegree {i in V diff {0}}:
        sum {j in V, d in D:i<>j} x[i,j,d] >= f;

subject to xContinuity {i in V, d in D}:
        sum {j in V:i<>j} x[i,j,d] = sum {j in V:i<>j} x[j,i,d];


subject to uContinuity {d in D, i in V diff {0}, j in V diff {0}: i<>j}:
        u[i,d] - u[j,d] >= q[j,d] - Q * (1 - x[i,j,d]);

subject to uUpper {i in V diff {0}, d in D}:
        u[i,d] <= Q;



s.t. initial_q {i in V diff {0}}:
        q[i,0] = q_sensor[i];
```

```
s.t. linerarized_prod1 {i in V diff {0}, d in D diff {0}}:
        v[i,d] <= M * (1 - y[i,d-1]);

s.t. linerarized_prod2 {i in V diff {0}, d in D diff {0}}:
        v[i,d] <= q[i,d-1];

s.t. linerarized_prod3 {i in V diff {0}, d in D diff {0}}:
        v[i,d] >= q[i,d-1] - (1-(1 - y[i,d-1])) * M;

s.t. update_q {i in V diff {0}, d in D diff {0}}:
        q[i,d] = v[i,d] + g[i];

s.t. update_y {j in V diff {0}, d in D}:
        sum {i in V: i<>j} x[i,j,d] = y[j,d];

s.t. over_flow {i in V diff {0}, d in D diff {0}}:
        w[i,d] >= q[i,d] - q_max[i];
```

10.3 Matheuristic Script

```
import amplpy
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt
import random
import io

starttime = time.time()
#Functions:

def generatePopulation(nDays, nNodes, populationSize):
        population = []
        solsize = nDays * nNodes
        groupsize = populationSize // nDays #størrelsen avhenger av om det gruppene går
opp i  populationSize
        for i in range(1, nDays+1):
                sol = [1] * i * nNodes + [0]*(solsize - (i * nNodes))
                for i in range(groupsize):
                        random.shuffle(sol)
                        population.append(sol)
        return population



def reproduction(population, mutationChance):
        offspring = []
        while len(offspring) < populationSize:
                #choose parents
                parent1 = random.choice(population)
                parent2 = random.choice(population)
                #do crossover to generate offspring
                crossover = random.randint(0, len(parent1)-1)
        child1 = parent1[:crossover] + parent2[crossover:]
        child2 = parent2[:crossover] + parent1[crossover:]
        #by some chance mutate offspring
        if random.random() <= mutationChance:
            gene = random.randint(0, len(child1)-1)
            if child1[gene] == 0:
                child1[gene] = 1
            else:
                child1[gene] = 0
        if random.random() <= mutationChance:
            gene = random.randint(0, len(child2)-1)
            if child2[gene] == 0:
                child2[gene] = 1
            else:
                child2[gene] = 0

        offspring.append(child1)
        offspring.append(child2)
    return offspring



def evalFitness(chromosome):
    ampl.eval('unfix y;')
    index = 0
    for i in range(1,nNodes+1):
        for d in range(nDays):
            varValue = chromosome[index]
            amplcommand = "let y[{},{}] := {};".format(i, d, varValue)
            ampl.eval(amplcommand)
            index += 1

    ampl.eval('fix y;')
    ampl.solve()
    dist = ampl.getObjective('Total_Distance')
```

```
        dist = dist.value()
        return dist

def sumones(solution):
        return sum(solution)

def popFitness(population):
        popfitness = []
        for solution in population:
                #popfitness.append(sumones(solution))
                popfitness.append(evalFitness(solution))
        return popfitness



writefiles = ['5.txt','6.txt','7.txt','8.txt','9.txt','10.txt','11.txt','12.txt','13.txt','14.txt','15.txt']
nodelist = [5, 6, 7, 8, 9, 10, 11, 12 ,13, 14, 15]

#AMPL Setup
ampl = amplpy.AMPL()
modfile = r'M:\ampl\Bergen\extended.mod'
datfile = r'M:\ampl\Bergen\extended_10_5_1.dat'
ampl.read(modfile)
ampl.readData(datfile)
ampl.setOption('solver', 'cplex')
ampl.setOption('cplex_options', 'time=10')

#Setup
random.seed(101)
nDays = 5
nNodes = 5
populationSize = 20 #4 * nNodes
mutationChance = 0.9
bestsols = []
bestfits = []
```

```
#Initiate Population
#population = generatePopulation(nDays, nNodes, populationSize)

for n in range(len(writefiles)):
	chromlen = nDays * nodelist[n]
	population = []
	for i in range(populationSize):
		ones = random.randint(0, chromlen)
		zeros = chromlen - ones
		chrom = [1 for i in range(ones)] + [0 for i in range(zeros)]
		random.shuffle(chrom)
		population.append(chrom)

	outfile = writefiles[n]
	f = open(outfile, 'a')

	for i in range(20):
		#Reproduction
		children = reproduction(population, mutationChance)
		population = population + children
		#Calculate fitness
		popfitness = popFitness(population)
		#sort by fitness
		popfitness, population = (list(t) for t in zip(*sorted(zip(popfitness, 
population))))
		#wipe out worst (for now at least)
		population = population[:populationSize]
		#record best solution
		bestsols.append(population[0])
		bestfits.append(popfitness[0])
		print("End of iteration: ", i)
		for z in popfitness[:5]:
			f.write(' ')
```

```
                f.write(str(z))
                f.write(', ')
        f.write('\n')
print('best of each generation: ',bestfits)
print('best solution: ', bestsols[bestfits.index(min(bestfits))])
print('top five from each generation:')

runtime  = time.time() - starttime

f.write('Runtime: ')
f.write(str(runtime))

f.close()
```

10.4 Simulation Script

```
#Import libraries
import amplpy
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt



#Create the ampl object
ampl = amplpy.AMPL()

#Read model and data for files (Maybe change it to define data in py)
modfile = r'M:\ampl\Bergen\extended.mod'
datfile = r'M:\ampl\Bergen\extended_5_7_1.dat'
ampl.read(modfile)
ampl.readData(datfile)

#Set solver and time limit
ampl.setOption('solver', 'cplex')
ampl.setOption('cplex_options', 'mipgap=0.05')
#ampl.setOption('cplex_options', 'time=40')
###The model can now be solved using the 'ampl.solve()' command###



np.random.seed(101)
simulationDays = 7 #How many times the model is solved
clients = 5
customernodes = [i for i in range(1,clients +1)] #not sure where to put this
exprod = [20 for i in range(clients)]
proddev = [5 for i in range(clients)]
outfile = open(r'M:\ampl\Bergen\simulationoutput.txt', 'a+')
```

```
real_bin_levels = [[] for i in range(clients)]

for simulation in range(simulationDays):

	#Solve the model
	ampl.solve()

	ampl.display('_solve_elapsed_time >> skrivq.txt')
	ampl.display('Total_Distance >> skrivq.txt')
	ampl.display('{i in V, j in V:i<>j} x[i,j,0]*c[i,j] >> skrivq.txt')
	ampl.display('q >> skrivq.txt')
	ampl.display('w >> skrivq.txt')
	ampl.display('t >> skrivq.txt')
	ampl.display('y >> skrivq.txt')



	#Read which nodes are served on the first day 0 of this solution (should read more
data, but this is it for now)
	served = ampl.getVariable('y')
	df = served.getValues()
	dflist = df.toList()
	served_yesterday = []
	for var in dflist:
		if var[1] == 0 and var[2] == 1:
			served_yesterday.append(int(var[0]))



	#write to file, just to see.
	outfile.write('\nNodes served on day ' + str(simulation) + ': ')
	for node in served_yesterday:
		outfile.write(str(node))
```

```
    #Read bin-levels that were used to solve the model
    levelyesterday = ampl.getParameter('q_sensor')
    levelyesterday_df = levelyesterday.getValues()
    levels_yesterday = levelyesterday_df.toList()

    #Store bin levels from day 0. (These are actual levels, not projected ones)
    for i in range(clients):
        real_bin_levels[i].append(levels_yesterday[i][1])
    print(real_bin_levels)

    #Simulate production of waste. All bins have save growth, maybe change this
somehow.
    real_production = [np.random.normal(exprod[i], proddev[i]) for i in range(clients)]

    #Update bin levels:
    levels_today = []
    for dunk in levels_yesterday:
        if dunk[0] in served_yesterday:
            bin_level = real_production[int(dunk[0]-1)]
            levels_today.append(bin_level)
        else:
            bin_level = real_production[int(dunk[0]-1)] + dunk[1]
            levels_today.append(bin_level)



    #Change data using a Pandas DataFrame
    df = pd.DataFrame(
        {'q_sensor': levels_today},
        index=customernodes
        )
    ampl.setData(amplpy.DataFrame.fromPandas(df))

outfile.close()
```

```
#Save realized bin levels to file

import csv

with open('binlevels.csv', 'w+', newline='') as csvfile:
    writer = csv.writer(csvfile, delimiter= ' ', quotechar='|',
quoting=csv.QUOTE_MINIMAL)
    for i in range(len(real_bin_levels)):
        writer.writerow(real_bin_levels[i])
```