Master semester project

Generalization and policy analysis for a rich inventory routing problem

Author:	Prisca Aeby [1]	prisca.aeby@epfl.ch
Supervisors:	Iliya Markov [2]	iliya.markov@epfl.ch
	Yousef Maknoon [2]	yousef.maknoon@epfl.ch
	Michel Bierlaire [2]	michel.bierlaire@epfl.ch

November 10, 2016

[1] Section of Computer Science, Master Semester 3, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

[2] Transport and Mobility Laboratory (TRANSP-OR), School of Architecture, Civil and Environmental Engineering (ENAC), École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, `transp-or.epfl.ch`

Abstract

We start from an existing problem formulation and implementation of a rich stochastic inventory routing problem solved at the Transport and Mobility Laboratory at the Ecole Polytechnique Fédérale de Lausanne. The problem inspired from practice consists of a heterogeneous fleet of vehicles which is used for collecting recycable waste from containers over a finite planning horizon. A forecasting model generated from a history of observations coming from the canton of Geneva is used to estimate the probability of container overflows and route failures. The problem is solved using an adaptative large neighborhood search algorithm integrating the forecasting model. We analyze the algorithmic performance of the current solution implemented in Java and R and we improve it through different methods modifying and extending the code. The computational time is reduced using more efficient algorithms as well as appropriate data structures and caching. The value of the objective function and its stability are ameliorated by improving the the strategy of the search space as well as modifying the uncertainty of the forecasting model.

1 Introduction

The first inventory routing problem (IRP) formulations were introduced as variation of vehicle routing problem models (VRP), taking inventory costs into consideration. The research in this area has been motivated by the introduction of the Vendor Managed Inventory (VMI) technique in supply chain management 30 years ago, which aims at reducing logistics costs and adding business value (Coelho et al., 2014). The general idea consists in coordinating inventory management, vehicle routing and delivery to the customers decisions over the planning horizon. There exist however several formulations of the problem as the tendency is to include specific company functions in the integration process (Bertazzi and Speranza, 2012). In our case, we are interested in analyzing the existing solution proposed for a rich recyclable waste collection problem as it is an important logistic activity within any city which can lead to financial savings as well as pollution reduction (Buhrkal et al., 2012). The solution implementing an adaptive large search neighborhood search (ALNS) has already obtained good performance on IRP benchmarks from the literature as well as IRP instances derived from real data of the canton Geneva, Switzerland (Markov et al., 2016). The problem concists of a heterogeneous fixed fleet of vehicles with different properties which is used for collecting waste over a finite planning horizon. Each tour of a vehicle starts at a depot and is followed by a sequence of collections and disposals at the available dumps. It terminates by a mandatory visit to a dump and it returns at the same depot it started. The depots, containers and dumps have time windows. The containers are equipped with sensors which communicate the waste level at the start of each day and indicate if a container is full. Containers can still serve demand when they are full as people place the waste beside them, however they must be collected within the day with a penalty cost. The solution implements a statistical model with the help of historical data to estimate for each container the point demand forecasts for each day of the planning horizon, as well as an estimate of the forecasting error used to compute the risk of container overflows and route failures. The problem falls under the Stochastic Inventory Routing Problem (SIRP) with no inventory holding costs and unlimited inventory capacity at the dumps. It is solved in two main steps, namely (1) it incorporates probabilistic information about container overflows and route failures impacting the cost, (2) it implements an ALNS algorithm with simulated annealing integrating a demand forecasting model. The ALNS is a type of large neighborhood search (LNS) heuristic which modifies the current solution by a destroy operator removing a number of customers which are reinserted by a repair operator in order to explore neighbors and find better solutions.

Our work consists of improving the performance of the implemented solution in terms of value of the objective function and its stability in mainly two directions. First, we modify the breadth of the search space that is explored in the ALNS algorithm by adding several destroy and repair operators inspired from the VRP literature. Secondly, we modify the uncertainty in the forecasting of future demands by testing artificial ways to improve the forecasting model's ability to fit the observed data.

The remainder of the article is organized as follows. Section 2 positions our work with respect to the relevant IRP literature. Section 3 formulates mathematically our SIRP. Section 4 describes the current ALNS algorithm implementation. Section 5 describes the methodology approaches to improve the value of the objective function. Section 6 presents the numerical experiments. Finally, our concluding remarks are given in section 7.

2 Related IRP Literature

Over the past 30 years many variants of the IRP have been presented, it is therefore complicated to refer to one general formulation of the problem. However, Coelho et al. (2014) conduct an analysis of the basic versions and propose a structural classification scheme according to seven criteria: time horizon, structure, routing, inventory policy, inventory decisions, fleet composition and fleet size. There is a second one classification related to the availability of information on the demand. We first describe the main objective of the general IRP formulation and then we position our problem according to the variants.

Objective of the Problem The main goal is to minimize the whole inventory distribution cost while meeting the demand of each customer. The solution has to respect some constraints, mainly concerning the inventory level and the stockout at each customer and supplier, the capacity of the vehicles and the routing constraints. The solution of the problem determines which customers to serve and when, which vehicles to use, how much to deliver to each visited customer, and the delivery routes.

Structural Variants

- **Time horizon** It can be either *finite* or *infinite.* In our case, the problem is solved in a *finite* rolling horizon framework as it is useful in dealing with uncertainty by making forward-looking decisions. The solution is found for a period of seven to ten days.
- **Structure** It denotes the ordering relation between the number of suppliers and the number of customers. As there are multiple containers that are emptied into multiple dumps, the structure is identified as *many-to-many.*
- **Routing** It is *direct* when there is only one customer per route, *multiple* when there are several customers in the same route, or *continuous* when there is no central depot. Vehicles are taking waste from several containers during one tour so it is classified as *multiple.*
- **Inventory policy** Under the *order-up-to level* (OU) policy any customer has defined a maximum inventory level and when he is served, the delivered quantity is such that the maximum inventory level is reached. Under the *maximum level* (ML) every time a customer is served, the delivered quantity is such that the inventory level at the customer is not greater than the maximum level. Visited containers are always fully emptied, meaning it falls under the OU policy.
- **Inventory decisions** If *back-orderings* are allowed the demand will be served at an other time. If there are no back orders, then the demand is considered as *lost sales.* The problem falls under the *back-ordering* decision as container overflow is served at a penalty and there is a limit on the number of back-order days.
- **Fleet composition and size** the fleet can be *homogeneous* or *heterogeneous*, and the number of vehicles may be *fixed at one, fixed at many*, or *unconstrained.* It is a *heterogeneous multiple fixed* fleet of vehicles.

Availability of Information on Demand

- **Deterministic** The information is fully available at the beginning of the planning horizon.
- **Stochastic** The probability distribution of the information is known.
- **Dynamic** The demand is not fully known in advance but gradually revealed over time.

Our problem falls under the *stochastic* category information-wise and under the *dynamic* category with new container information revealed each day in a rolling horizon fashion.

3 Problem Formulation

In what follows, section 3.1 presents a general mathematical formulation of the SIRP model, section 3.2 explains the derivation of the overflow probabilities, section 3.3 formulates the objective function and section 3.4 describes the constraints.

3.1 SIRP Model

A complete directed graph $G(N, A)$ is given for each day of the planning horizon $T = \{0, ..., \mathrm{u}\}$, with $N = \{\mathrm{o}\} \cup \{\mathrm{d}\} \cup D \cup P$, o representing the depot, d a destination, P a set of containers, D the set of dumps and A the set of arcs between all the points in N. The distance between the points i and j is given by an asymmetric distance matrix π_{ij}. A travel time matrix τ_{ij} indicates the travel time of a vehicle for each arc of the graph. Each point has a time window $[\lambda_\mathrm{i}, \mu_\mathrm{i}]$. If a vehicle arrived before λ_i it has to wait, and he cannot be served after μ_i. There is an expected demand $\mathrm{E}(\rho_{\mathrm{it}})$ for container i on day t. The container capacity is ω_i and a cost χ is charged for an overflowing container. The fleet of vehicle is denoted by K. Each vehicle has a capacity Ω_k, a daily deployment cost γ_k, a unit-distance running cost β_k and a unit-time running cost θ_k. The continuous variable I_{it} represents the expected inventory of container i at the start of day t.

3.2 Derivation of the Overflow Probabilities

There are no inventory holding costs at the containers or dumps. However, if a container is overflowing on day t an emergency collection occurs and empties the container in question. Two costs are added to the objective function: χ has a monetary value which the collector bears and C is a parameter representing the average actual cost of emergency collection. We are interested in the probability of overflow at container i on day t. In order to compute this probability we need to know the expected demand for a container i at day t. This is given by the forecasting model proposed by Markov et al. (2016), which minimizes the sum of squared errors between the observed demand and the expected demand $\mathrm{E}(\rho_{\mathrm{it}})$ over the set of containers P and a period H of data. The error of the forecasting model is represented by a white noise ϵ_{it} (independent and identically distributed random variable following a normal distribution) which is added to the expected demand:

$$\rho_{\mathrm{it}} = \mathrm{E}(\rho_{\mathrm{it}}) + \epsilon_{\mathrm{it}}$$

For the first day of the planning horizon, the probability that a container overflows is simply given by $\mathrm{P}(\mathrm{I}_{\mathrm{i0}} + \rho_{\mathrm{i0}} \geq \omega_\mathrm{i})$. For a container which starts with a zero inventory (it can be either on the first day or at a state of overflow because the inventory is set to zero by the emergency collection) the probability is given by $\mathrm{P}(0 + \rho_{\mathrm{ih}} \geq \omega_{\mathrm{i}})$, $\forall \mathrm{h} \in T$. In the other cases, we need the conditional probability that a container overflows at day h: it is the probability that the initial inventory at day g plus the expected demand between day g and t exceeds the container capacity, knowing that the inventory of the container for the preceding days is smaller than its capacity. It can be formulated as:

$$\mathrm{P}(\mathrm{I}_{\mathrm{ig}} + \sum_{\mathrm{t=g}}^{\mathrm{h}} \rho_{\mathrm{it}} \geq \omega_{\mathrm{i}} | \mathrm{I}_{\mathrm{ig}} + \sum_{\mathrm{t=g}}^{\mathrm{h\text{-}1}} \rho_{\mathrm{it}} \leq \omega_{\mathrm{i}})$$

The evaluation of the necessary conditional and unconditional probabilities can be pre-computed using a statistical package in R.

3.3 Objective Function

The function z which needs to be minimized includes the following costs: the Expected Overflow and Emergency Collection Cost (EOECC), the Routing Cost (RC) and the Expected Route Failure Cost (ERFC):

$$\min(z) = \mathrm{EOECC} + \mathrm{RC} + \mathrm{ERFC}$$

The EOECC is the sum for each point over the planning horizon of the probability of a container overflow times the overflow cost χ and the emergency cost C in case there is no regular collection on that day. The RC is a deterministic expression computing the cost of each vehicle used over the planning horizon, taking into account its daily cost, the unit-distance cost and the service-time cost. The ERFC represents the vehicles' inability due to insufficient capacity to serve the containers on the scheduled depot-to-dump or dump-to-dump trips.

3.4 Constraints

Several constraints make sure the variables can only take values which lead to a feasible solution given our formulation of the problem. It checks that only available vehicles are used and that the ones executing a tour start at the origin and ends with a visit to a dump directly before the destination. Vehicles should no visit inaccessible points and their cumulative quantity capacity should not exceed their limit. Each container has to be visited by at most one vehicle on a given day. Inventory constraints track the container inventories by linking them to the vehicle visits and the pickup quantities. The intra-day temporal constraints make sure ime windows and maximum tour duration are respected.

4 Adaptive Large Neighborhood Search

4.1 Algorithm

ALNS can be combined with any local search framework at the master level. The presented solution uses simulated annealing. The implementation is simple as one solution is sampled in each iteration of the ALNS. A first solution s is found and then the neighbor solution s' generated by the ALNS algorithm is only accepted if f(s') < f(s) or with probability $\mathrm{e}^{\text{-}(\mathrm{f(s')\text{-}f(s)})/T}$, f(s) repsensenting the solution cost and T>0 the current temperature.

The temperature is initialized with a given value T^{start} and is decreased at each iteration by a cooling rate $r \in (0, 1)$. The search stops when T reaches a given threshold T^{end}. The idea of the ALNS algorithm is to remove at each iteration of the search process a number of customers from the current solution by a destroy operator and to reinsert them elsewhere by a repair operator in order to find a neighbor of the current solution. In our problem, not all customers need to be visited every day so they don't need to be all reinserted by the repair operator. The operators to be used in each iteration are chosen using a *roulette wheel* mechanism. An operator i is chosen with probability $\mathrm{W_i}/\sum_{\mathrm{j}\in O}\mathrm{W_j}$. The weight $\mathrm{W_i}$ depends on its past performance and a score. At the beginning the weights and scores are set to 1 and 0 respectively. The entire search is divided into *segments* of F iterations and the weights are updated at the end of each *segment*. The scores are increased in the following situations: the operators find a new best feasible solution, they improve the initial solution or a new solution is accepted. Let $\mathrm{C_i^F}$ denote the score and $\mathrm{N_i^F}$ the number of times operator i has been applied during the *segment*. The new weight $\mathrm{W_i}$ doesn't change if $\mathrm{N_i^F} = 0$ and otherwise is replaced by $\mathrm{(1\text{-}b)W_i + bC_i^F/(m_iN_i^F)}$, $\mathrm{b} \in (0, 1)$ denoting the *reaction factor* and $\mathrm{m_i}$ a *normalization factor* reducing the weights of computationally expensive operators.

The performance and robustness of the overall heuristic is very dependent on the choice of destroy and repair operators. Local search heuristics are often built on neighborhood moves that make small changes to the current solution, leading to difficulties in moving from one local optima of the solution space to another when faced with tightly constrained problems like ours (Ropke and Pisinger, 2006a). The way to avoid those local optima is to allow the search to visit infeasible solutions by relaxing some constraints. A penalty for each type of violation is introduced in the cost of the complete solution f(s) during the search. There are accepted violations for the vehicle and container capacity, the time window, the duration of a tour, the backorder limit and the inaccessible visits.

4.2 Heuristics used in the solution

Destroy operators Remove v containers from a random tour, remove v containers that would lead to the largest savings, empty a random day, empty a random vehicle, remove a random dump, remove the worst dump, remove consecutive visits of the same container and shaw removal which selects a random container i and removes the containers close to it in term of distance.

Repair operators Insert v containers randomly, insert v random containers in the best way, swap v random containers, insert a random dup in a random position, insert a random dump in the best way, swap random dumps, replace a random dump by an other random dump, reorder the dumps in the best way and shaw insertions which inserts containers closed to each other not visited during a random day.

4.3 Heuristics introduced

4.3.1 Insert Best Containers

Idea The function already implemented selects a v random containers from P and insert them in the tour and position that would lead to the smallest increase in the objective function. We modify this insert heuristic by inserting the v best containers. The best position of each container from P and the corresponding cost increase are stored and then the v containers with the smallest cost increase are inserted in their best position.

Code `insertBestRhoContainers()` in the `Schedule` class has been modified. We loop over all the containers and insert each of the container in every possible tour, keeping the best position in a `HashMap`. An `ArrayList` storing `(Point, CostIncrease)` is then sorted first according to `CostIncrease`. We take the first v containers of this list and insert them in their best position (which is cached in the `HashMap`).

4.3.2 Container insertion with regret

Idea Regret heuristics have been used by Potvin and Rousseau (1993) for the VRPTW. It has been implemented as well by Buhrkal et al. (2012) in the context of waste collection vehicle routing problem with time windows in a city logistics context. The principle is to introduce a look-ahead information when selecting the container to insert. Let $\mathrm{R_{ik}} \in \{1,...,\mathrm{m}\}$ be a variable that indicates the route for which inserting the container i has the k$^{\mathrm{th}}$ lowest insertions cost: $\Delta \mathrm{f}_{\mathrm{i,R_{ik}}} \leq \Delta \mathrm{f}_{\mathrm{i,R_{ik'}}}$ for k $\leq$ k'. We define a *regret value* $\mathrm{c}_{\mathrm{i}}^{*} = \Delta \mathrm{f}_{\mathrm{i,\ R_{i1}}} - \Delta \mathrm{f}_{\mathrm{i,R_{ik}}}$, it is the difference in inserting the container in its best route and its kth best route. We choose to insert the container i which maximizes $\mathrm{c}_{\mathrm{i}}^{*}$, in other words the one we will regret most if we don't insert it now. We insert the request i at its minimum cost position. We can run it with different values of k. Ties are broken by inserting the request i in the route with the lowest insertion cost. If a container can't be inserted in different k positions, we compute the regret for the largest_k we find. In order to choose the next container to insert, we first consider the ones with the smallest largest_k. It is repeted v times.

Code `insertRhoContainersWithRegret(int k)` has been added in the `Schedule` class. We loop over all the containers and insert each of the container in every possible tour, keeping the best position in a `HashMap`. An `ArrayList` storing `(Point, (Regret, largest_k))` is then sorted first according to largest_k and then Regret. We take the first v containers of this list and insert them in their best position (which is cached in the `HashMap`).

4.3.3 Shaw removal of related containers

Idea This removal has been inspired by the heuristic presented by Ropke and Pisinger (2006a) in which they solve the pickup and delivery problem with time windows. Instead of computing only the distance between two containers, we introduce a new measure: the relatedness $\mathrm{R_{i,j}}$. The lower $\mathrm{R_{i,j}}$ is, the more related the two containers are. The relatedness chosen for our problem consists of three terms: a distance term, a time term and an overflow probability term. These terms are weighted using the weights a, b and c respectively. The relatedness measure is given by $\mathrm{R(i,j)} = \mathrm{a}(\pi_{\mathrm{i,j}}) + \mathrm{b}(|\lambda_{\mathrm{i}} - \lambda_{\mathrm{j}}| + |\mu_{\mathrm{i}} - \mu_{\mathrm{j}}|) + \mathrm{c}(|\mathrm{o_{i,d}} - \mathrm{o_{j,d}}|)$. The term $\pi_{\mathrm{i,j}}$ denotes the distance between the container i and container j, μ_{i} and λ_{i} the upper and lower time window bound of container i and $\mathrm{o_{i,d}}$ the overflow probability of container i at day d. The three terms are normalized between 0 and 1 by feature scaling $(x - min)/(max - min)$. The algorithm proceeds as follows: it picks a random tour, a random container in this tour and it computes de *Relatedness* of all other containers from the tours scheduled the same day. The *Relatedness* is then normalized again by feature scaling. If the *Relatedness* is under a given *threshold*, it will remove the container.

Code The function `removeAllShawContainersRelatedness(double threshold, double a, double b, double c)` is implemented in the `Schedule` class. When this heuristic is used, we remove the `removeShawContainers()` function from the destroy operators.

4.3.4 Cluster removal of containers using Kruskal algorithm

Idea This clustering technique removal has been taken from the solution implemented by Ropke and Pisinger (2006b) for vehicle routing problems with backhauls. The motivation for the heuristic is to remove large chunks of related requests instead of removing a few requests from each route. The idea is to select a random day in the *Schedule* and to divide the containers visited during this day into k clusters. We choose k to be the number of routes scheduled this day. If there is only one route, we divide the route into 2 clusters. We then choose a random cluster to remove. The size of the cluster has to be smaller than half the number of containers scheduled during the selected random day in order to avoid removing all containers of an entire day.
Kruskal's algorithm works as follows: initially, each container is in its own cluster. Then, it considers each edge (there is an edge between all containers) in turn, order by increasing distance. If an edge (i, j) connects two different clusters, then (i, j) is added to the set of edges of the minimum spanning tree, and two clusters connected by an edge (i, j) are merged into a single cluster. On the other hand, if an edge (i, j) connects two containers in the same cluster, then edge (i, j) is discarded. More formally, for a graph G containing V vertices (containers) and E edges (between all containers), the algorithm is:

```
Data: A graph $G = (V, E)$, number of clusters k
Result: k clusters generated by Kruskal's algorithm
$T = \emptyset$;
for each vertex $v \in V$ do
    MAKE-SET($v$);
end
Sort edges of $E$ in inscresing order by distance;
for each edge $e = (u, v) \in E$ in order do
    if NUMBER-SETS $== k$ then
        break
    end
    if FIND-SET($u$) $\neq$ FIND-SET($v$) then
        $T = T \cup \{e\}$;
        UNION-SET($u, v$);
    end
end
```

Algorithm 1: Kruskal's MST algorithm for clustering

We need *Union − Find* data structure that supports:

- MAKE-SET(v): Create set consisting of v
- UNION-SET(u, v): Unite set containing u and set containing v
- FIND-SET(u): Return unique representative for set containing u

Code Two classes have been added to `alns.schedule`: `KruskalClustering` and `Graph`. `KruskalClustering` has a private class `UnionFind` implementing the desired data structure and `Graph` has a private class `Edge`. The function executing the removal is `removeContainerCluster()` in the class `Schedule`.

5 Numerical Experiments

The ALNS is implemented in Java ... TODO ... Numerical experiments are seperated in two different directions which correspond to the distinct parts of the project: the new heuristics added and the modification of the forecasting model. First, we assess the performance of the new operators implemented in the ALNS on the past results obtained by Markov et al. (2016) on real data instances. Secondly we evaluate how the objective function reacts to different levels of uncertainty in the forcasting model. The comparisons are made for the same running environment. Each instance is solved 10 times, out of which the best and average result are reported. Section 5.1 describes the algorithmic parameters used as well as the instances format. Section 5.2 explains how the parameters of the operators were tuned. Section 5.3 presents the results obtained by the ALNS using the new operators on the instances. Section 5.4 describes the change of the costs composing the objective function react to changes of the forecasting error.

5.1 Running Environment

The same algorithmic parameters are used to solve each instance of the real dataset. For the simulated annealing it uses an initial temperature $\mathrm{T}^{\mathrm{start}}$ = 10,000, a cooling rate r = 0.99998 and a final temperature $\mathrm{T}^{\mathrm{end}}$ = 0.01. For the ALNS-related parameters the segment length is of 2000 and the operators-related parameters are described in section 5.2.

There are 63 different instances collected from canton Geneva, Switzerland. Each of them corresponds to a week (Monday to Sunday) of white glas collections from 2014 to 2016. The time windows are set from 8 a.m. to 12 a.m and each tour has a maximum duration of 4 hours. The heterogeneous fleet of vehicle(s) is composed up to 2 vehicles of capacity around 30,000 liters and weight capacity of 10,000 to 15,000 kg. Vehicles are not available over the week-end. There are on average 41 containers per instance and their volumes range from 1000 to 3000 liters. Their daily deployment cost is 100 CHF, with an additional 40 CHF cost per hour and 2.95 CHF per km. There are 2 dumps located on the outside of the city centre. The overflow cost is set to 100 CHF. The forecast demand for each instance is computed using the information of the past 90 days. The complete objective function with all relevant costs is computed.

5.2 Tuning New Operators

Five instances out of the 63 available ones were randomly chosen in order to select the best parameters of the new operators. The parameters have been tuned one by one, in the order we present them. First, we tested the *container insertion with regret k* parameter. Recall that we can compute the distance between inserting a container in its best position and its k^{th} best position. Best values of the objective function are found with k = 2. For the *Shaw remvoal of related containers* operator, the weights a, b and c of the distance, time and overflow probability terms respectively as well as the threshold t can be tuned. It turns out that optimal solutions were have been found for a=0.7, b=0, c=0.3 and t=0.2. It is not surprising that the time term doesn't have any influence as the time windows are

Table 1: Average over all instances

	Past	New	Improvement
Average	679.54	673.92	0.8%
Minimum	664.76	664.30	0.06%
Maximum	699.50	688.36	1.59%
Max - Min	14.78	9.61	35%
Avg - Min	34.74	24.05	31%

Table 2: Counts for each instance

New minimum found	49%
Better average	78%
Better maximum	79%
Max stability improvement	77%
Avg stability improvement	77%

always the same. We set a normalization factor of 2 for the new operators because they are computationally more expensive than the simple operators.

5.3 Results on Real Data

The 63 instances have been solved adding the new operators to the ALNS framework. We compare the past results with the new results in order to check the two types of improvement: reducing the cost of the objective function as well as improving the stability of the solution. For each instance, there are 10 different cost values as each of them is solved 10 times. The stability is represented as the difference between the maximum and the minimum costs found, as well as the difference between the average cost and the minimum cost found. The Table 1 shows the average of those values over the 63 instances. You can observe that the improvement relies mainly in the stability values.

In addition to the average values, we count how many times those values have been improved for each instance. The results are shown in Table 2.

On average, each instance can be solved in ... minutes. It is ... more than previously, but it is necessary to have a more stable value of the objective function for the analysis of the forecasting model's influence as we will see in the second part of the numerical experiments.

6 Conclusion

References

Bertazzi, L. and Speranza, M. G. (2012). Inventory routing problems: an introduction. *EURO Journal on Transportation and Logistics*, 1(4):307–326.

Buhrkal, K., Larsena, A., and 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.

Coelho, L. C., Cordeau, J. F., and Laporte, G. (2014). Thirty years of inventory routing. *Transportation Science*, 48(1):1–19.

Markov, I., Bierlaire, M., Cordeau, J. F., Maknoon, Y., and Varone, S. (2016). Inventory routing with non-stationary stochastic demands. Technical report, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne.

Potvin, J. Y. and Rousseau, J. M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331 – 340.

Ropke, S. and Pisinger, D. (2006a). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472.

Ropke, S. and Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750 – 775.