A Pickup and Delivery problem with multiple time windows in supply chain industry

Department of Informatics

University of Bergen

Preben Bucher Johannessen

September 18, 2019

abstract

This is a test Shaw (1997). Hopefully it works..

Contents

List of Figures

Chapter 1

Introduction

The automobile industry is one of the most significant industries in Europe today which is facing high cost-reduction pressure from fierce international competition and difficult conditions.4flow AG (2018) The supply chain process in automobile industry requires alot of flexibility in their planning and modelling of transportation which often takes the form some sort of vehicle transportation problem. In planning the most efficient way of organizing your transportation network the classical VRPTW from is likely the first problem that comes to mind.

Parragh et al. (2008) The Vehicle Routing Problem with Time Window Constraints (VRPTW) is defined as the problem of minimizing costs when a fleet of homogeneus vehicles has to destribute goods from a depot to a set of customers satisfying time windows and capacity constraints Cordeau and Groupe d'études et de recherche en analyse des décisions (Montréal(2000). Another classical problem is the PDPTW, Pickup and Delivery Problem with Time Windows, with multiple vehicles is another problem that has been widely researched and solved for example in Nanry and Barnes (2000).

Cordeau and Groupe d'études et de recherche en analyse des décisions (Montréal(2000) Presents a VRPTW Problem as an extension of the Capacitated Vehicle routing problem, with hard and soft timewindows, where soft can be broken agains a cost and hard time-windows cannot. They present a multi-commodity network flow formulation of the problem and use approximation methods to derive upper bounds and use Lagrangean relaxation and column generation to derive lower bounds. To produce integer solutions they use cutting and branching strategies and finally they also present special cases and extensions. Solomon (1987) formalized an algorithm for general PDP Lu and Dessouky (2006) insertion based heuristic Kalantari et al. (1985) An algorithm for the travelling salesman problem with pickup and delivery of customers Dumas et al. (1991) pickup and delivery problem with time windows Fagerholt et al. (2010) speed optimization and reduction of fuel emissions Berbeglia et al. (2010) dynamic and static pickup and delivery problems survey and classification scheme Berbeglia et al. (2007) static Zhou (2013) Inventory management and inbound logistics optimization Savelsbergh and Sol (1995) General pickup and delivery problem. Dror et al. (1989) Properties and solution framework

Christiansen and Fagerholt (2002) present a ship scheduling PDP problem of bulk cargoes with multiple time windows. They also handle specific shipping industry challenges regarding ship idle time, transport risks due to weather and unpredictable service time at ports. They use a set partitioning approach to solve

the problem and found that you could increase the robustness of the schedules on account of costs. Christiansen et al. (2004) Ship routing and scheduling Hemmati et al. (2014) Consider a class of cargo ship routing and scheduling problems from tramp and industrial shipping industry. They provide solutions to a wide range of benchmark instances both optimal for smaller instances, using a commercial mixed-integer programming solver. And they present adaptice large neighbourhood search heuristics to larger instances. They also provide the benchmark instances and an instance generator. Hemmati et al. (2016) A iterative two-phase hybrid metaheuristic for multi-product short sea inventory routing problem Ferreira et al. (2018) variable neighbourhood search for vehicle routing problem with multiple time windows Desrosiers et al. (1995) Time constrained routing and scheduling

In this paper we will propose a mathematical formulation to a multivehicle PDPTW problem, with certain adjustments to satisfy the needs of an automobile manufacturing company. To these type of companies the inbound production side of the supply chain is more in focus and require ajdustments on the classical PDPTW. In these type of industries companies often facing challenges related to the production side of the problem, which is often referred to as looking at the problem from an inbound perspective. Inbound oriented perspective means that you have less focus on the the supplier side of the of the production, and on the means of transportation as long as what is being transported arrives on time for production. This changes our problem away from the classical problems mentioned above. Instead of having a customer oriented view, we must instead shift our view to focus on getting a set of orders from a set of suppliers to a set of factories where the orders are needed for production.

Being more inbound oriented also leads to having more focus on the problems that sometimes occur within the factory and the delivery part of the problem. Some manufacturers might have alot of traffic on their factories, and want to limit that, saying that a vehicle can only visit a certain amount of docks for each visit. This might lead to more ineffective solutions and more use of vehicles in some cases however, for the inbound oriented producer it is a nessecary limitation. To our knowledge dock constraints has not been researched before in PDPTW problems.

Another result from the inbound thinker is that instead of using your own fleet of vehicles, you hire a logistics carrier to do your transports. The carries might have different cost structures and payment methods which you have no influence over. Leading you to need a model that takes that into account. We focus here on dealing with a carrier that offers a varying cost per kilometer. Meaning that the cost parameter of the standard PDP will change depending on how far a vehicle is travelling. Using a carrier also changes the problem to that you dont care so much where the vehicle is travelling from to its first pickup. The vehicle rather starts at its first pickup point because the cost of the car manufacturer starts only from the time of the first pickup. Before pickup and after delivery the carrier has the costs. To our knowledge very little research has been made into using a logistics carrier as vehicle fleet.

In car manufacturing business you might have production periods that go over several days with strict opening times on delivery. This leads to the problem of multiple time windows. The manufacturer might only want the parts delivered in the morning on one of three days but have a break in the middle of each time window (lunch). There might also be advantageous to move the delivery of one order to the

next day to bundle orders together and we want our model to be able to handle such problems.

In Favaretto et al. (2007) they propose a time window model to handle multiple time windows and multiple visits. We are using a modified version of this model to handle the multiple time windows in this paper to handle the time window constraints. However as they focus on satisfying a set of customers instead of production companies our model differs greatly when it comes to the rest of the constraints.

To make a model that is as realistic as possible it is important to integrate each of the above aspects and take them all into account. One aspect of the model could greatly influence the decision of another so it is important to build a model that satisfies all the aspects above. Research often forcuses on solving single issues, like the multiple time windows, to handle this specific problem for a certain industry. However handeling all of the above mentioned aspects have to our knowledge never been researched before and makes it therefore an important problem to solve.

The goal of this paper is to present a realistic mathematical model that can be further developed into a realistic model solving pickup and delivery problems on a daily/weekly basis for manufacturing companies. The goal is also to have a mathematical basis to be used in a solver such as AMPL to solve a small instance, and finally to make a heuristic model that can solve the problem efficiently and be used by companies in similar sort of industries.

test after file

Chapter 2

Model Description

Like mentioned above to take the perspective of a vehicle manufacturing company, we see that we have a certain set of factories where the products are produced, each with a set of demanded parts/orders for production. The parts, or orders (typically named transport orders but we refer to orders here), can be delivered from a set of suppliers. At a given point in time (could be delivery for one day or a week) we consider a planning problem with a demand for orders that should be satisfied with the given suppliers using vehicles provided by different logistic carriers.

v	-	vehicle
i	-	node
f	-	factory
p	-	timewindow
s	-	stop location
α	-	distance interval in cost structure
β	-	weight interval in cost structure

Table 2.1: Indices

The vehicles have different capacities(kg and volume), cost structures, incompatabilities and start at the first pickup location at the first pickup time (ie. costs to get from start to first pickup are not relevant to the manufacturer). The cost structure depend on the total distance a vehicle is driving, the maximum weight transported by that vehicle, and a fix cost. There is also a cost for stopping at a node.

When a delivery is assigned to a vehicle, the vehicle must load the delivery from the supplier, and deliver the delivery at the factory dock. The docks (each reperesented by a different node) in each factory can differ according to which order is being delivered and there is a limit to how many docks each vehicle can unload at per visit to the factory.

Each delivery/pickup can have several time windows, lapsing sometimes over several days. If a car arrives before a time window it has to wait.

The mathematical formulation of the problem will now follow. The problem can be viewed as a graph $G(E, N)$ where $N = \{0, ..., 2n\}$ are the verticies and n is the number of orders in the problem, and $E = \{(i, j) : i, j \in N, i \neq j\}$ represent the edges in the graph.

	Sets	
N	-	nodes $\{1, .., 2n\}$ where n is number of orders
V	-	vehicles
E	-	edges
E_v	-	edges visitable by vehicle v
N_v	-	nodes visitable by vehicle v
N^P	-	pickup nodes
N^D	-	delivery Nodes
F	-	factories
N_f	-	delivery nodes for factory f
A_v	-	index of elements in the cost where α goes from $(1, .., \gamma_v)$ and β from $(1, .., \mu_v)$
P_i	-	set of time windows at node i, $\{1, .., \pi_i\}$
T_i	-	set of time parameters $[\underline{T_{ip}}, \overline{T_{ip}}]$ at node i where $p \in P_i$
S	-	set of stops indicating a pickup/delivery location
L_s	-	Sets of nodes sharing a stop location $s \in S$

The set of vehicles used is denoted by V and weight capacity of each vehicle $v \in V$ is denoted by K_v^{kg} and volume capacity is denoted K_v^{vol}. The set of Edges that each vehicle can traverse is represented by E_v. Since n is the number of orders in the problem, then if i is a specific pickup-node then $i+n$ corresponds to the delivery node for the same order. The set of pickup Nodes (supplier docks) we denote using N^P and each delivery node (Factory dock) is denoted by N^D. All nodes are therefore equivalent to $N = N^P \cup N^D$. Each Factory, $f \in F$, also has a set of Nodes belonging to the same factory wich we denote N_f. Since all factories are delivery nodes these sets only include delivery nodes. Each vehicle has a set of Nodes it can travel to represented by N_v. This set also includes an origin node, $o(v)$ and a destination node $d(v)$ which is a fictive start and ending point unique to each vehicle v. The distance and costs from here to the first pickup is zero. The factory docking limit is denoted by H_f.

Parameters

n	-	amount of orders
K_v^{kg}	-	weight capacity of vehicle $v \in V$
K_v^{vol}	-	volume capacity of vehicle $v \in V$
$o(v)$	-	starting node of vehicle v
$d(v)$	-	ending node of vehicle v
Q_i^{kg}	-	weight of order at node $i \in N$
Q_i^{vol}	-	volume of order at node $i \in N$
H_f	-	docking limit at factory $f \in F$
T_{ijv}	-	travel time for vehicle $v \in V$ over edge $(i,j) \in E_v$
π_i	-	amount of time windows at node $i \in N$
$\overline{T_{ip}}$	-	upper bound time of time window $p \in P_i$ at node $i \in N$
$\underline{T_{ip}}$	-	lower bound time of time window $p \in P_i$ at node $i \in N$
γ_v	-	amount of distance intervals for vehicle v
μ_v	-	amount of weight intervals for vehicle v
$C_{v\alpha\beta}^{km}$	-	cost per distance unit (km) in cost matrix element $(\alpha,\beta) \in A_v$ for vehicle v
$C_{v\alpha\beta}^{kg}$	-	cost per weight unit (kg) in cost matrix element $(\alpha,\beta) \in A_v$ for vehicle v
$C_{v\alpha\beta}^{fix}$	-	fixed cost in index $(\alpha,\beta) \in A_v$ for vehicle v
C_i^{stop}	-	costs of making a stop at node i
C_i	-	cost of not transporting order $i \in N^P$
D_{ij}	-	distance between node $i \in N$ and $j \in N$
B_α	-	distance for interval α in cost matrix A_v column index
Z_β	-	weight for interval β in cost matrix A_v row index

Each delivery node has a variable h_i indicating how many docks have been visited including the node i.

Each pickup node has a weight Q_i^{kg} and a volume Q_i^{vol} parameter indicating the weight and volume of the order at that node. Each node has a set T_i of time windows represented by $[\underline{T_{ip}}, \overline{T_{ip}}] \in [0,T]$ where $p \in P_i = \{0,1,...,\pi_i\}$ and all nodes should be picked up and delivered within given timewindows. Each node has a current time based on when its being served, denoted by t_i and where $i \in N$. The distance from node i to node j is denoted by D_{ij} and the time for each vehicle v to travel between them is represented by T_{ijv}.

Each time a vehicle v makes a stop and a node i there will be a stop cost represented by C_i^{stop}. The costs of vehicle v depends on the total distance of that vehicle and the maximum weight transported. Each possible interval of weight and distance is represented by an index pair (α,β), where α is the distance interval index ranging $(1..\gamma_v$ and β is the weight interval ranging from $(1..\mu_v)$. Together these pairs make a matrix we refer to in this paper as a cost matrix. Each type of cost has a matrix, including distance, weight, and fix costs and the cost in a certain interval (α,β) is represented by $C_{v\alpha\beta}^{cost-type}$. The total distance travelled by vehicle v will be denoted by the variables $d_{v\alpha\beta}$ for each $(\alpha,\beta) \in A_v$, where only one variable per vehicle will have the value equal to the total distance of that vehicle. The maximum weight transported by a vehicle is represented by $l_{v\alpha\beta}$ and also only one of these variables per vehicle will have a the corresponding value. Which $d_{v\alpha\beta}$ and $l_{v\alpha\beta}$ has a value will be determined by the binary variable $b_{v\alpha\beta}$ and the distance insterval parameter B_α and the weight interval parameter Z_β.

Variables

x_{ijv}	-	binary indicating travel from node $i \in N$ to $j \in N$ of vehicle $v \in V$
y_i	-	binary indicating that an order $i \in N^P$ is not picked up
l_{iv}^{kg}	-	weight of vehicle v after visiting node i
l_{iv}^{vol}	-	volume of vehicle v after visiting node i
h_i	-	docking times in factory after visiting node $i \in N_f$
t_i	-	time after visiting node $i \in N$
u_{ip}	-	binary indicating usage of time window $p \in P_i$ at node i
$d_{v\alpha\beta}$	-	total distance travelled of vehicle $v \in V$ if it fits in interval $(\alpha, \beta) \in A_v$
$b_{v\alpha\beta}$	-	binary indicating interval $(\alpha, \beta) \in A_v$ for vehicle $v \in V$
$l_{v\alpha\beta}$	-	the highest weight trasported by vehicle $v \in V$ for interval $(\alpha, \beta) \in A_v$

l_{iv}^{kg} is the weight and l_{iv}^{vol} is the volume on the vehicle v leaving node i. x_{ijv} is a binary vairable indicating if vehicle v is travelling between i and j node. The cost of not transporting an order will be represented by C_i for each node i, with a corresponding binary variable y_i, indicating that an order is not picked up.

$$\min \sum_{v \in V} \sum_{(\alpha,\beta) \in A_v} (C_{v\alpha\beta}^{km} d_{v\alpha\beta} + C_{v\alpha\beta}^{kg} l_{v\alpha\beta} + C_{v\alpha\beta}^{fix} b_{v\alpha\beta}) + \sum_{v \in V} \sum_{s \in S} \sum_{\substack{i \in L_s \\ j \in N_v \notin L_s}} C_i^{stop} x_{ijv} + \sum_{i \in N^P} C_i y_i \tag{2.1}$$

subject to:

$$\sum_{v \in V} \sum_{j \in N_v} x_{ijv} + y_i = 1, \qquad i \in N^P \tag{2.2}$$

$$\sum_{j \in N_v} x_{ijv} - \sum_{j \in N_v} x_{jiv} = 0, \qquad v \in V, i \in N_v \notin \{o(v), d(v)\} \tag{2.3}$$

$$\sum_{j \in N_v} x_{o(v)jv} = 1, \qquad v \in V \tag{2.4}$$

$$\sum_{j \in N_v} x_{jd(v)v} = 1, \qquad v \in V \tag{2.5}$$

$$\sum_{j \in N_v} x_{ijv} - \sum_{j \in N_v} x_{(i+n)jv} = 0, \qquad v \in V, i \in N_v^P \tag{2.6}$$

$$l_{iv}^{kg} + Q_j^{kg} - l_{jv}^{kg} \leq K_v^{kg}(1 - x_{ijv}), \qquad v \in V, j \in N_v^P, (i,j) \in E_v \tag{2.7}$$

$$l_{iv}^{kg} - Q_j^{kg} - l_{(j+n)v}^{kg} \leq K_v^{kg}(1 - x_{i(j+n)v}), \qquad v \in V, j \in N_v^P, (i, n+j) \in E_v \tag{2.8}$$

$$0 \leq l_{iv}^{kg} \leq K_v^{kg}, \qquad v \in V, i \in N_v^P \tag{2.9}$$

$$l_{iv}^{vol} + Q_j^{vol} - l_{jv}^{vol} \leq K_v^{vol}(1 - x_{ijv}), \qquad v \in V, j \in N_v^P, (i,j) \in E_v \tag{2.10}$$

$$l_{iv}^{vol} - Q_j^{vol} - l_{(j+n)v}^{vol} \leq K_v^{vol}(1 - x_{i(j+n)v}), \qquad v \in V, j \in N_v^P, (i, n+j) \in E_v \tag{2.11}$$

$$0 \leq l_{iv}^{vol} \leq K_v^{vol}, \qquad v \in V, i \in N_v^P \tag{2.12}$$

$$h_i + 1 - h_j \leq (H_f + 1)(1 - x_{ijv}), \qquad v \in V, f \in F, i \in N_f, j \in N_f, j \neq i \tag{2.13}$$

$$h_j \leq H_f, \qquad v \in V, f \in F, j \in N_f, \tag{2.14}$$

$$h_j \geq \sum_{\substack{i \in N_v \\ i \notin N_f}} (x_{ijv}) \qquad v \in V, j \in N_f \tag{2.15}$$

$$\sum_{p \in P_i} u_{ip} = 1, \qquad i \in N \tag{2.16}$$

$$\sum_{p \in P_i} u_{ip} \underline{T_{ip}} \leq t_i, \qquad i \in N \tag{2.17}$$

$$\sum_{p \in P_i} u_{ip} \overline{T_{ip}} \geq t_i, \qquad i \in N \tag{2.18}$$

$$t_i + T_{ijv} - t_j \leq (\overline{T_{i\pi_i}} + T_{ijv})(1 - x_{ijv}), \qquad v \in V, (i,j) \in E_v \tag{2.19}$$

$$t_i + T_{i(i+n)v} - t_{(i+n)} \leq 0, \qquad v \in V, i \in N_v^P \tag{2.20}$$

$$\sum_{(\alpha,\beta) \in A_v} d_{v\alpha\beta} = \sum_{(i,j) \in E_v} x_{ijv} D_{ij}, \qquad v \in V \tag{2.21}$$

$$\sum_{(\alpha,\beta) \in A_v} l_{v\alpha\beta} \geq l_{iv}^{kg} \qquad v \in V, i \in N_v \tag{2.22}$$

$$B_{(\alpha-1)} b_{v\alpha\beta} \leq d_{v\alpha\beta} \leq B_\alpha b_{v\alpha\beta}, \qquad v \in V, (\alpha,\beta) \in A_v \tag{2.23}$$

$$Z_{(\beta-1)} b_{v\alpha\beta} \leq l_{v\alpha\beta} \leq Z_\beta b_{v\alpha\beta}, \qquad v \in V, (\alpha,\beta) \in A_v \tag{2.24}$$

$$\sum_{(\alpha,\beta) \in A_v} b_{v\alpha\beta} \leq \sum_{j \in N_v} x_{o(v)jv}, \qquad v \in V \tag{2.25}$$

$$h_i, t_i \geq 0, \qquad i \in N \tag{2.26}$$

$$u_{ip} \in \{0,1\}, \qquad i \in N, p \in P_i \tag{2.27}$$

$$b_{v\alpha\beta} \in \{0,1\}, \qquad v \in V, (\alpha,\beta) \in A_v \tag{2.28}$$

$$d_{v\alpha\beta}, l_{v\alpha\beta} \geq 0 \qquad v \in V, (\alpha,\beta) \in A_v \tag{2.29}$$

$$y_i \in \{0,1\}, \qquad i \in N^P \tag{2.30}$$

$$x_{ijv} \in \{0,1\}, \qquad v \in V, (i,j) \in E_v \tag{2.31}$$

The objective function (2.1) sums up to the cost of all vehicles given corresponding costs from their cost matrix. Costs could be variable per distance, weight, fixed and/or related to stops made. Loads not transported will be penilized with costs and the aim is to minimize the sum of all these costs. Constraint 2.2 secures that a load is picked up once and only by one vehicle or not picked up at all. 2.3 makes certain that if a load at a node j is travelled to it is also left. This is not the case for

origin and final destination node. Then 2.4 makes sure that origin is only left once by each vehicle, and 2.5 ensures that desination is only arrived to once. Finally 2.6 is there to say if a load is picked up it must also be delivered.

Regarding weight, constraint 2.7 ensures that if a load at node j is picked up, the weight before plus the weight of the load at node j minus the weight leaving the node is equal to zero. If not the summation cannot exceed the capacity of the vehicle. Also 2.8 makes certain that at delivery of node j the weight before delivery minus the weight of the load at j minus the weight after, is equal to zero, ie. the weight of the vehicle after is lighter than before by exactly the weight of the load at j. The final weight constraint 2.9 says that weight leaving node i always has to be between 0 and the capacity of the vehicle.

The next constraints 2.10-2.12, ensures the same as for the weight but for volume, that each load is increased by the volume of the order, that the volume is decreased once delivered and that the volume at any node is between 0 and the volume capacity of that vehicle.

It follows from constrain 2.13 that within a certain factory, if you travel between two nodes, the amount of stops you have made should always be iterated by one. If you are not travelling between them, the summation cannot be bigger than one plus the limitation corresponding to that factory. The next constraint on factories 2.14, ensures that any node visited in a factory cannot exceed the docking limit. Then 2.15 makes sure that when a vehicle enters a factory from outside, the docking amount gets an initial value of 1, and if one is not travelling from i to j the value has to be greater than or equal to 0.

Constraint 2.16 ensures only one timewindow is used per node, and 2.17 says that the time a node i is visited has to exceed a lower timewindow limit. 2.18 ensures the same upper bound timewindow is used. Then constraint 2.19 ensures that the travel from one node to the next is appropriately increased by the travel time between them and for any other two nodes the values cannot exceed a large constant. Finally 2.20 ensures that the time a vehicle visits the delivery of an order must always be after the pickup of that order.

From 2.21 we have that for each vehicle, the sum of the total distance variables has to be equal to the total travel distance of that vehicle. Then regarding weight 2.22 ensures that the sum of the max weight of a vehicle v is greater than or equal to the weight at all nodes visited by that vehicle. Constraint 2.23 ensures that for each vehicle v the total distance variable can only exist in the appropriate distance interval. The same is the case in 2.24 for maximum weight in the appropriate weight interval. Finally 2.25 says that if a vehicle is not leaving its origin node, there cannot be a cost interval binary for that vehicle, which in turn ensures that we dont calculate the fixed costs of said vehicle.

Constraints 2.26 to 2.31 are ensuring positive and binary variables where appropriate.

Chapter 3

Solution Method

Describe Local search algorithms and refer some papers. These algorithms can have difficulties moving from one solution space to another, even when this is cosidered in the metaheuristics.

A way to deal with this could be to, refer to some paper that tries to implement some form of heuristic to search larger neighbourhoods.

3.1 Local Neighbourhood Search

3.2 Simmulated Annealing

Chapter 4

An Adaptive Large Neighbourhood Search Aproach

This chapter explains how we have applied ALNS to the PDPMTW. The model in this paper differs to models from papers like Ropke and Pisinger (2006) in the following way:

1. We use only one removal and one insertion heuristic in each search iteration in what we call an operator. We describe each of these operators and their heuristics in 4.2.

4.1 Initial Solution

Many different algorithms have been made for finding an initial solution. In our paper we have chosen to simply start with an initial solution where no orders are assigned to any vehicle which is very efficient and will not lead us to getting This is a test Shaw (1997). Hopefully it works..

4.2 Operators

This section presents the operators or heuristics used by our ALNS algorithm. The first three operators 4.2.1, 4.2.2, 4.2.3, we call shuffeling operators. They try to quickly schuffle a given solution around to find new solutions in the same neighbourhood. The last four operators, 4.2.4, **??**, 4.2.6, 4.2.6, are using removal and reinsertion heuristics. These are used partly to move from one neighbourhood to another, and partly for local search depending on the amount of solution elements being reinserted.

Removal and reinsertion heuristics are well reserached tools used in solving PDP. We have implemented and adapted the most succesful heuristics and created some new ones to help solve our specific problem.

4.2.1 Swap first fit

This operator simply swaps the pickup and delivery of two orders first time it fits. This operator is very efficient and jumps randomly around the solution space.

4.2.2 3-exchange

4.2.3 2-opt

The 2-opt operator used in this paper is based on the 2-opt-L operator from ref: Carcasse2006. It selects a random vehicle with more than 2 orders (it breaks out after not finding any fitting vehicle after o tires, where o is the amount of orders in the problem). For the selected vehicle it divides up the route of the vehicle in 3 parts. All orders up until the index i of the vehicle route are inserted normally. Orders from the index j+1 until the end of the route are also inserted normally. then finally orders from the index i+1 until index j are inserted in inverse order. If the new route has a smaller cost than the original route the route the route is remembered as the new best route. When all reverses have been performed on the current route, the best route is selected as the new route. This operation is continued until no improvement can be made ie. the best possible schedule for the selected vehicle has been found.

4.2.4 Remove random and insert first fit

Removes randomly between 2 orders and 10 percent of the amount of orders and reinserts them randomly in their first possible position. This operator is used to jump from one neighbourhood to another and is trying to search for possible solutions regardless of the cost they produce.

4.2.5 Remove non-clustered and insert clustered

This heuristic tries to remove orders that are bundled together but belong to different clusters. It then tries to bundle orders together in the best possible way. The orders pickup and delivery locations are all assorted into clusters based on the distance between them. We use a hirarchial single linkage clustering algorithm to cluster locations together into k clusters. (TODO: make algorithm table for herarchial single linkage algorithm) Then we compare each set of k cluster to eachother using the siluette coefficient, and keep the best one. (TODO: make algorithm table for siluette coefficient)

To choose which order to remove we have ranked the orders based on how many clusters are shared within a vehicles schedule. If an orders pickup and delivery shares no cluster with any other pickup or delivery nodes the rank is 0. If the order shares the same cluster with both pickup and delivery node as all other pickup and delivery nodes within a vehicle it is given a rank 1. This leads us to quite easily differenciate between a well clustered order and a not well clustered one. The orders are sorted based on their rank and we choose the order to pickup that has the lowest rank, using some randomization, based on the value p. We choose the order on the position k^p in the rank where k is a random number. We have chosen $p = 4$ in this paper.

To reinsert the orders we have chosen to try and maximize the cluster rank we introduced above. This means we find the vehicle where the rank is the highest and insert the order in the best possible position in the chosen vehicle.

4.2.6 Remove Worst and Insert Greedy

Removing orders in the most costful positions and reinserting it in its cheapest (greedy) position seems to be a reasonable way of moving towards a better solution. We therefore propose a heuristic that removes the orders with the highest cost C_i. The C_i is calclated as the increase in a vehicles schedule cost with the chosen order. We remove orders again based on the same randomness factor explained above. We do this by first ranking the orders based on cost and choose the order in the k^p position.

The reinsertion is using a greedy algorithm where it simply checks to find the best possible possition for the order and inserts it there.

Remove Similar and Regret Insert

The removal heuristic used in this operator is based on Shaw (1997) with slight modifications based on our PDPMTWF problem. It removes orders that share specific similar qualities, as the basic idea is that by replacing these orders by eachother we find new, hopefully better, solutions. Another good reason to remove similar orders is that it could be advantagous to reinsert these orders togther on the same vehicle since they share alot of the same properties in regards to distance, time etc. We define a relatedness factor r_{ij} which represents how much the order i is related to the order j. The lower the value of r_{ij} the more the two orders i and j are related. The relatedness of two orders we base here on the following properties: a distance property, a weight property, an overlapping timewindow property, a property indicating if the same vehicles can be used to serve each request, and finally if the orders belong to the same factory.

The relatedness factor is given by the following equation:

$$r_{ij} = \psi(D_{ij}+D_{(i+n)(j+n)})+\omega|Q_i-Q_j|+\phi(1-\frac{|V_i \cap V_j|}{max(|V_i|,|V_j|)})+\tau G_{ij}+\chi(U_{ij}+U_{(i+n)(j+n)}) \tag{4.1}$$

D_{ij}, Q_i, are defined in the problem formulation section and these values have been normalised to result in a value between [0..1]. V_i is the set of vehicles that can serve order i. The parameter G_{ij} is 1 if i belong to another factory than j and 0 if they belong to the same factory. U_{ij} is the timewindows at the pickup and delivery location, equal to 0 if the two orders have identical time windows and 1 if not. It corresponds to the sum of overlapping time windows divided by the overlapping span of the two time window sets. It can be formulated as follows

$$U_{ij} = 1-\frac{\sum\limits_{\substack{p\in\pi_i \\ o\in\pi_j \\ \underline{T_{ip}}\le\overline{T_{jo}} \\ \underline{T_{jo}}\le\overline{T_{ip}}}} (\min(\overline{T_{ip}},\overline{T_{jo}}) - \max(\underline{T_{ip}},\underline{T_{jo}}))}{\max\left(\max\limits_{p\in\pi_i}\overline{T_{ip}},\max\limits_{o\in\pi_j}\overline{T_{jo}}\right) - \min\left(\min\limits_{p\in\pi_i}\underline{T_{ip}},\min\limits_{o\in\pi_j}\underline{T_{jo}}\right) - \sum\limits_{\substack{p\in\pi_i \\ o\in\pi:q_j \\ \underline{T_{ip}}\ge\overline{T_{j(o-1)}} \\ \underline{T_{jo}}\ge\overline{T_{i(p-1)}}}} (\min(\underline{T_{ip}},\underline{T_{jo}}) - \max(\overline{T_{i(p-1)}},\overline{T_j} \tag{4.2}$$

Here $\overline{T_{ip}}$ and T_{ip} are the upper and lower time windows defined in the problem formulation. Thus the relatedness measure is given a value $0 \leq r_{ij} \leq 2\psi+\omega+\phi+\tau+\chi$. We have chosen the following values in this paper $\psi = 0.7$, $\omega = 1.0$, $\phi = 0.8$, $\tau = 0.3$, $\chi = 0.3$.

The insertion part of this operator tries to improve on the insert greedy by calculating a regret value, c_i^*, which represents how much is lost by inserting this order in its second best position.

4.2.7 Wild operator

4.3 Choosing an operator

4.4 Adaptive weight adjustment

4.5 Acceptance criteria

4.6 Radical jump away from neighbourhood

ALNS algorithms are known to be good at searching locally aswell as globally. However they do sometimes get stuck in one neighbourhood, and it is important that our algorithm is able to adapt in these situations. If we for 500 iterations find no improvement in the solution we are assuming that we are stuck. In this case we need to do something to get unstuck. We have designed our algorithm so that in the case when no improvement in found, we run 100 iterations apart from the normal iterations, where we accept any new solution found regardless of the objective value. In these iterations we use what we call a wild operator. The wild operator is basically a collection of the most unspecific targeting operators from 4.2. We use the operators from section 4.2.4, 4.2.2 and 4.2.1 as these operators are not trying to improve and select targeted solutions but rather returns the first feasible solution they find. We also increase the amount of elements the operators are working on to effectively move further then the normal search iterations.

Chapter 5

Computational Results

This chapter will present the results from the following experiments:

- Wild operator evaluation
- Operators evaluations
- ALNS evaluation

Before going into the experimental results we will describe the experimental setup. This icludes the techical side aswell as the setup of the analytical experiments themselves. After that we will explain the generation ofthe instances used in the final result section.

5.1 Experimental Setup

In this section we describe the technical setup of the experiments aswell as the Analytical setup of our eperiments.

5.1.1 Technical Setup

The computational experiments in this paper is run on two different computers. The more demanding experiments are run on a 64-bit Windows 8 computer with a 2.7 Ghz i7-8500 processor and 16GB RAM. We will shorten the name of this computer to "Windows computer". The less demanding experiments are run on a 64-bit Ubuntu 18.04 computer with a 1.8 Ghz quad core i7-8550u processor and 16GB RAM. We call this computer for short "Ubuntu computer".

The instance generator described in the next section was implemented in Java (version number). The mathematical model from section 2 is setup in AMPL (version number), using the Gurobi solver. All AMPL experiments were run on the Windows computer. The ALNS heuristics are implemented in Java (verison number). These experiments were run partly on the Ubuntu and partly on the Windows computer. The statistical experiments are performed in Matlab (version number) on the Ubuntu computer.

5.1.2 Analytics Setup

In section 3 we described seven operators, aswell as a wild operator, some invented in this paper and others based on known ALNS heuristics. For our testing we generated five instance sets of each five instances, totally 25 instance. Each instance set has 5 instances with sizes ranging from $4 - 150$ orders. The sizes used in this paper are 4, 12, 35, 80 and 150 orders. While testing the wild operator, we used 5 of these instances of each of the sizes. The test was run 10 times using all of the seven described operators from section 3. The results from these To analyse the performance of each of the seven operators, 5 reasonably sized instances was solved 10 times using each of the $2^7 = 128$ combinations of operators. To determine which of the operators influence the result we have performed several statistical tests, including ANOVA and regressional analysis, and t-tests. After the analysis of the operators, 5 heuristics are chosen for further testing. For these tests we compare the results toward the best known solution from AMPL and ..... TODO. We use a 95% confidence level for all statistical experiments in this paper.

5.2 Instances

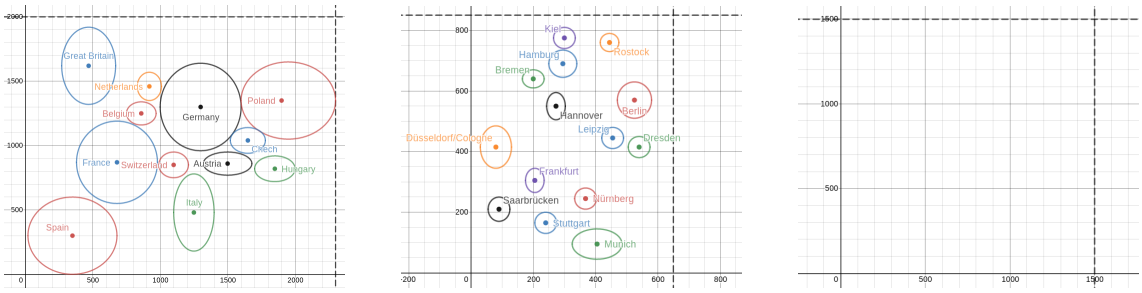
The instance generator was created based on real data from an anonumous costomer of 4flow. Following, we will describe how we designed the generator and how we generated the instances used in the analytical part of the paper.

5.2.1 Generate Instances based on real data

The 4flow data gives information about the number of orders $|N|$, locations $|L|$, factories $|F|$. The amount of vehicles $|V|$ are kept large compared to the amount of orders to make sure there are enough but not too many vehicles. A solution that uses all but one vehicle is the ideal here and we found $|N|/2 \leq |V| \leq 2/3|N|$ to be fitting. $|N|, |V|, |L|, |F|$ are given as input to the generator. In addition the data from 4flow gave us information about the size of a vehicle and its compatabilities. Some vehicles might have cooling possibilities amd some speical equipment required for transport of special goods or equipment required at the pickup or delivery location etc.). Other information aquired by the data was travel distances, cost structure etc.

To keep the instances feasible but still as realistic as possbile it makes sense to limit the data to different possibilities. Our data was generated with the following properties:

- Orders are assigned to pickup and delivery locations randomly. Orders assigned to the same location are given the same stop L_s
- Each delivery location is assigned to a factory at random N_f.
- Each location aswell as each order is assigned a special property with 5% probability. This will decide which vehicle can pickup which order.
- We let the vehicles types be split up in 3 different vehicle types, small, medium, large, each with extected capacitied and capabilities.



(a) Europe (b) Germany (c) Uniform

Figure 5.1: Area of random point generation

  - Large vehicle: slower but compatible with all locations and orders, with $Q_v^{kg} = 24'$ and $Q_v^{vol} = 102$
  - Medium vehicle: medium fast and compatible with all locations but not orders with special compatability, with $Q_v^{kg} = 18'$ and $Q_v^{vol} = 71$
  - Small vehicle: fastest but not compatible with special locations and special orders, with $Q_v^{kg} = 12'$ and $Q_v^{vol} = 55$

- The distances were calculated using pytagoras on the randomly generated points described in the next section
- The time T_{ijv} were scaled with 60% of the travel distance, added with a random variation of $+/-10\%$ of the travel distance, multiplied by the speed of the vehicle (slow: $* = 105\%$, medium: $* = 102.5\%$).
- The cost matrices $C_{v\alpha\beta}^{km}$, $C_{v\alpha\beta}^{kg}$, $C_{v\alpha\beta}^{fix}$ were based on a real 4flow cost matrix, scaled to the size of the instance and to the size of the vehicle.
- The cost of no transport was set to a minimum lower bound scaled based on the weight/volume/distance of the order.
- Time windows $[\underline{T_{ip}}, \overline{T_{ip}}]$ were generated randomly based on typical factory opening hours. $\overline{1\text{-}2}$ timewindows per day, and 3-7 days per week based on the instance size.

Random locations based on real georaphical data

Most of 4flow's customers are based either in Germany or in Europe. To make the instance generator as realistic as possible we have decided to split the instances into 3 geographical types; European, German and uniform geographically distributed locations. We made 2 maps based on real scale approximations of geographical data from National Geographics, in km. To simplify we have sticked to geographical points with an eliptic uniformly distributed area surrounding the point to represent a country or a city. 5.1 illustrates the areas of possible locations used in the generator. Larger elipses are more likely to be selected by the generator than smaller elipses.

For the selected elipse a point was selected within the elipse at random with a uniform distribution. For 5.1c points were generated at random within the limits shown. From our 5 instance sets, two were generated using 5.1a, two with 5.1b and one with the uniform distribution from 5.1c. If a point belong in the same factory as

a previously generated point, that point was generated within a reasonable radius of three kilometer.

5.2.2 Generated Instances

TODO: Describe the instances I have generated and how I refer to them in the paper. Finish the presentation of the results first to, know how I do this.

5.3 Results

In the first two parts of this section we will present the data from our experiments, that leads to the final composition of our model. The results for the final algorithm are presented in the third part of this section.

5.3.1 Evaluation of a stuck algorithm

To help our algorithm our of a stuck position we designed a wild operator described in section 4. To

5.3.2 Evaluation of operators

5.3.3 Evaluation of algorithm

5.4 Observations

This section will focus on the observation made from the results in the previous section.

5.4.1 Choosing the final composition

5.4.2 Performance of final composition

Chapter 6

Conclusions

This is a test Shaw (1997). Hopefully it works..

Appendix A

Appendix

This is a test Shaw (1997). Hopefully it works..

Bibliography

4flow AG. Industries & references - automotive manufacturers, www.4flow.com. 2018.

G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, 2007.

G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European journal of operational research*, 202(1):8–15, 2010.

M. Christiansen and K. Fagerholt. Robust ship scheduling with multiple time windows. *Naval Research Logistics (NRL)*, 49(6):611–625, 2002.

M. Christiansen, K. Fagerholt, and D. Ronen. Ship routing and scheduling: Status and perspectives. *Transportation science*, 38(1):1–18, 2004.

J.-F. Cordeau and Q. Groupe d'études et de recherche en analyse des décisions (Montréal. *The VRP with time windows*. Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8: 35–139, 1995.

M. Dror, G. Laporte, and P. Trudeau. Vehicle routing with stochastic demands: Properties and solution frameworks. *Transportation science*, 23(3):166–176, 1989.

Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European journal of operational research*, 54(1):7–22, 1991.

K. Fagerholt, G. Laporte, and I. Norstad. Reducing fuel emissions by optimizing speed on shipping routes. *Journal of the Operational Research Society*, 61(3): 523–529, 2010.

D. Favaretto, E. Moretti, and P. Pellegrini. Ant colony system for a vrp with multiple time windows and multiple visits. *Journal of Interdisciplinary Mathematics*, 10 (2):263–284, 2007.

H. S. Ferreira, E. T. Bogue, T. F. Noronha, S. Belhaiza, and C. Prins. Variable neighborhood search for vehicle routing problem with multiple time windows. *Electronic Notes in Discrete Mathematics*, 66:207–214, 2018.

A. Hemmati, L. M. Hvattum, K. Fagerholt, and I. Norstad. Benchmark suite for industrial and tramp ship routing and scheduling problems. *INFOR: Information Systems and Operational Research*, 52(1):28–38, 2014.

A. Hemmati, L. M. Hvattum, M. Christiansen, and G. Laporte. An iterative two-phase hybrid matheuristic for a multi-product short sea inventory-routing problem. *European Journal of Operational Research*, 252(3):775–788, 2016.

B. Kalantari, A. V. Hill, and S. R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22(3):377–386, 1985.

Q. Lu and M. M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175(2):672–687, 2006.

W. P. Nanry and J. W. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, 34(2):107–121, 2000.

S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery models part ii: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

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

M. W. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995.

P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 1997.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

H. Zhou. *Inventory Management and Inbound Logistics Optimization for a Food Processing Company*. PhD thesis, University of Cincinnati, 2013.