

Chapter 1

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 includes the technical side as well as the setup of the analytical experiments themselves. After that we will explain the generation of the instances used in the final result section.

1.1 Experimental Setup

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

1.1.1 Technical Setup

The computational experiments in this paper are 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 (version 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.

1.1.2 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 regression 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.

1.2 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.

1.2.1 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, N_v^P and N_v^D
- We let the vehicles types be split up in 3 different vehicle types, small, medium, large, each with extected capacited and capabilities.

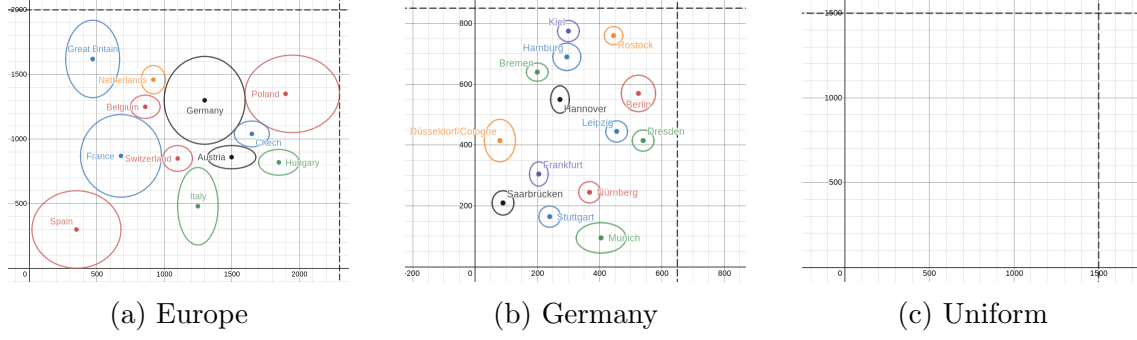


Figure 1.1: Area of random point generation

- 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 pythagoras 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: $\ast = 105\%$, medium: $\ast = 102.5\%$).
- 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 $[T_{ip}, \overline{T_{ip}}]$ were generated randomly based on typical factory opening hours. 1-2 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 stuck to geographical points with an eliptic uniformly distributed area surrounding the point to represent a country or a city. 1.1 illustrates the areas of possible locations used in the generator. Larger ellipses are more likely to be selected by the generator than smaller ellipses.

For the selected ellipse a point was selected within the ellipse at random with a uniform distribution. For 1.1c points were generated at random within the limits shown. From our 5 instance sets, two were generated using 1.1a, two with 1.1b and one with the uniform distribution from 1.1c. If a point belong in the same factory as

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

1.2.2 Generated Instances

For testing our algorithm in this paper, 5 instance sets of each 5 instances of varying sizes were generated. Among the 5 instance sets, 2 were generated based on 1.1a, two based on 1.1b and one based on 1.1c. Each instance set contains the following sizes: 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.

1.3 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.

1.3.1 Evaluation of a stuck algorithm

TODO: work on title of this section. To help our algorithm out of a stuck position we designed a wild operator described in section ??. To evaluate the implementaion of this operator we have decided to

1.3.2 Evaluation of operators

1.3.3 Evaluation of algorithm

1.4 Observations

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

1.4.1 Choosing the final composition

1.4.2 Performance of final composition