Towards a sustainable last mile deliveries and city logistics

A Multi-Actor Multi-Criteria Approach with a modelling implementation, aimed at identifying the most sustainable alternatives for city logistics and model their effects for all stakeholders combined

J.J. Daleman

4514742

Master Thesis

TU Delft

postnl

Towards sustainable last mile deliveries and city logistics

A Multi-Actor Multi-Criteria Approach with a modelling implementation, aimed at identifying the most sustainable alternatives for city logistics and model their long-term effects for all stakeholders combined

By

Jasper Daleman

27th of August, 2018

In partial fulfilment of the requirements for the degree of

Master of Science
In Engineering and Policy Analysis

At the Delft University of Technology,
To be defended publicly on Monday 10 September, 2018 at 14:00h

Graduation Committee

Thesis Committee:	Dr. J.H.R. (Ron) van Duin	(TU Delft; TBM)
	Dr. ir. B. (Bert) Enserink	(TU Delft; TBM)
External Supervisor:	Mr. M. (Marien) Vaandrager	(PostNL)

TU Delft

Preface

This report is the product of my final research project for my master programme Engineering and Policy Analysis (EPA) of the TU Delft. After completing my specialisation into the direction of Supply Chain Management in Gothenburg, Sweden, I got inspired by the so-called last mile problem of Supply Chains. I started my graduation project at PostNL into the direction of the same issue, where I have tried to apply my knowledge at best to generate useful outcomes and insights that inspire successors of my study.

I would not have been able to conduct this study without the help of some important people to me, so I would like to thank them in advance of the report. Firstly, I would like to thank my graduation committee, consisting of dr. ir. B. Enserink, dr. J.H.R. van Duin and mr. M. Vaandrager of PostNL. Their constructive feedback helped me creating a conclusive case as presented in this report. Despite calling me (too) ambitious right from the start, I think we ended up with some nice work.

Secondly, I would like to thank PostNL as a company for providing the opportunity to conduct this study. I wrote them myself with the idea, but without the support, facilities, support and commitment they offered this result would have been possible. I hope I have delivered something they can use in their future operations.

Finally, I would like to thank my family, friends and fellow students for all of the support throughout the project, the interesting discussions and inspiring questions about what exactly it was what I was studying. It all helped me thinking through what to write down to get my message across.

I wish you lots of fun reading this report. Thank you all.

Jasper Daleman

Leiderdorp, 2018

Executive Summary

The GHG-emissions of the Dutch transport sector are ever increasing. This trend is accompanied by the growth of the e-commerce sector, leading to more transport movements on the Dutch road network. In order to mitigate the externalities of the e-commerce related parcel delivery market and try to make it more sustainable, the following research question has been drafted:

How could the last mile delivery process become more sustainable, i.e. minimising traffic impacts and emissions, while maintaining the social and economic benefits of e-commerce and home deliveries?

To answer the research question, this study follows a Multi-Actor Multi-Criteria Approach (MAMCA), which is especially defined for large transport-related projects that require high stakeholder involvement. Based on a stakeholder analysis and an assessment of their points of view, a sustainability framework has been defined. This framework consists of a set of criteria along which several 'more sustainable' last mile alternatives have been analysed. The most important criteria are the reduction of GHG emissions, delivery time and costs and no decrease of customer satisfaction.

In explicit, this study assesses the costs and benefits of the implementation of cargo bikes, electric vans, Urban Consolidation Centres (UCCs), crowdsourcing systems, evening and night deliveries. First, a Simple Multi-Attribute Rating Technique (SMART) method is applied to identify the alternative(s) that offer the highest utility (most benefits). According to the SMART analysis, parcel lockers, UCCs (with electric transport) and night delivery are the most beneficial alternatives for a sustainable last mile in all different cases (best-, middle- and worst-cases).

After implementing these alternatives in a Discrete-Event Simulation (DES) model and conducting carefully designed experiments with it, the conclusion can be drawn that implementing or expanding the parcel locker infrastructure significantly enhances the operational efficiency the best. Furthermore, these lockers can easily be replenished by night, which reduces the traffic impact of parcel delivery even further. UCCs or city hubs with a focus on smaller vehicles are significantly less efficient than the current system, as more

kilometres and more time are needed to transport the same demand throughout the city. However, due to the electric transport, significant reductions in GHG-emissions can be obtained.

It is perceived unfeasible to implement a city hub for the Amsterdam parcel delivery market in the traditional sense. However, the city hub's effect gets stronger when more small transporting companies are consolidated at the city hub and are being transhipped to the dense networks of big transporting companies. A sketch of this process is shown in Figure 7-4. Besides, it remains unclear what the willingness to walk of customers is to a parcel locker. This should be further investigated to further optimise the parcel locker infrastructure and think of elegant solutions to not disrupt the street image with big walls with parcel lockers.

Thus, a sustainable last mile delivery process consists of a widely used and publicly available parcel locker infrastructure that is replenished by night. Furthermore, the system contains a city hub that focusses more on special deliveries with smaller vehicles and the consolidation of the shipments of smaller transporting companies. The bigger transporting companies then have to start shifting to electrified transport in cities in order to stay competitive in the future.

Table of Contents

List of Figures

List of Tables

List of abbreviations

3BL	–	Triple Bottom Line
B2B	–	Business to Business
B2C	–	Business to Consumer
C2C	–	Consumer to Consumer
CDP	–	Collection/Drop-off Point
CO_2	–	Carbon di-oxide
DC	–	Distribution Centre
DES	–	Discrete-Event Simulation
EC	–	European Commission
EU	–	European Union
GDZES	–	Green Deal Zero Emissions Stadslogistiek
GHG	–	Green House Gases
GRI	–	Global Reporting Initiative
IT	–	Information Technology
MAMCA	–	Multi-Actor Multi-Criteria Analysis
MCDA	–	Multi-Criteria Decision Analysis
SC	–	Supply Chain
SCM	–	Supply Chain Management
SDG	–	Sustainable Development Goals
SMART	–	Simple Multi-Attribute Rating Technique
SSCM	–	Sustainable Supply Chain Management
UCC	–	Urban Consolidation Centre
UN	–	United Nations

1 | Introduction

The emission of Green House Gases (GHG) has been a problem to the environment for many years now. The global transport sector is one of the largest contributors, as it accounts for 23% of the global emissions (PBL, 2016). According to the EIA (2017), transportation demands are expected to rise even more towards 2050. This expectation is already noticable in recent numbers from the Netherlands, stating that the increase in (freight) transport movements is causing the emissions from the transport sector to increase (CBS, 2017a; CBS, 2017b; CBS, 2017c; CBS, 2018).

The increase in transport movements does not only have a bad effect on the emission of GHG, but also on the amount of congestion. According to the ANWB (2017), there is still a growing number of traffic jams. Furthermore, the growing congestion is causing the congestion impact (measured in kmmin) to increase by 38% in 2021 (KiM, 2016). Hence, the Dutch *"Kennisinstituut Mobiliteitsbeleid"* (KiM) is raising awareness for the fact that, if no further action is taken, cities become clogged.

Furthermore, traffic jams have an additive effect on GHG-emissions (Ligterink, van Zyl, & Heijne, 2016). Ligterink et al. (2016) note that heavy-duty vehicles, like trailer-trucks, can emit up to 100% more CO_2 when caught in a traffic jam, compared to their cruising speed of 80 km/h. Traffic has less influence on light-duty vehicles like delivery vans, but they can still emit up to 60% more CO_2 compared to when driving 80 km/h (Ligterink, van Zyl, & Heijne, 2016). So, by the increasing number of traffic movements, the resulting increase in traffic impact and the additive effect this congestion has on vehicles' emissions, it is easy to see that the problem is growing.

1.1 The influence of e-commerce

Consumers are increasingly ordering online, which could well be a reason why the number of freight transport movements is increasing. This is expressed by the fact that the number of web shops has doubled in the period 2010 till 2015 and that the total value of the Dutch e-commerce market has increased by 13% compared to the foregoing year (CBS, 2016; Thuiswinkel waarborg, 2017). Furthermore, PostNL is noting a growth rate of 17% over 2017 due to the growing volume of parcels (PostNL, 2018).

However, it is not only the growing market that is worrying. Most e-commerce related parcel deliveries are conducted in city areas (Cárdenas, Beckers, & Vanelslander, 2017). It is currently assumed that many workers have their parcels being delivered at the office in the city. Taking the current trends of urbanisation into account (by 2020 about 80% of the European citizens is expected to live in cities), even more pressure will be exerted on the urban road network (Cárdenas, Beckers, & Vanelslander, 2017; European Union, 2013).

1.2 The field of City Logistics

City logistics is defined in multiple ways, of which one of them is *"the last leg in the supply chain to the customer location in the city, or the first leg from a customer location in a city back into the supply chain"* (Boer, Kok, Ploos van Amstel, Quak, & Wagter, 2017, p. 13). Furthermore, city logistics is seen as a multi-disciplinary bridge between urban freight transport and city sustainability (Çaliskan, Kalkan, & Ozturkoglu, 2017). It aims at optimising the logistics and transport activities of the last leg, while also considering the traffic environment, traffic congestion and energy consumption (De Marco, Mangano, & Zenezini, 2018). Hence, city logistics literature may provide useful insights in how to mitigate the negative effects of goods distribution in the city.

Today, an increasing number of governments and municipalities are acknowledging the upcoming challenges regarding the distribution of freight through the cities (PostNL, 2018). Therefore, they are looking for new innovative ways for freight distribution in cities. According to the analysis of the municipal elections that was conducted for PostNL (2018), these municipalities are looking to cargo bikes and Urban Consolidation Centres (UCCs) to optimise the parcel deliveries in the cities. PostNL is trying to be a part of these developments, and already offers alternative delivery methods like delivery by bike, parcel lockers and/or pick-up at retail locations.

1.3 Research gap and questions

Despite the efforts of developing new delivery options, there is still a lack of knowledge in research about their sustainable performance. Furthermore, there is not much insight in the criteria for a sustainable home delivery mechanism. Besides, much of the alternatives that have been studied in literature have not been analysed on the same factors of interest. Therefore, Cárdenas et al. (2017) raised the question of comparing the most promising 'last mile delivery' alternatives on the same 'sustainability' factors. Hence, the following research question is raised:

How could the last mile delivery process become more sustainable, i.e. minimising traffic impacts and emissions, while maintaining the social and economic benefits of e-commerce and home deliveries?

There is not one straightforward answer to this main research question. Therefore, several sub-questions are defined to divide the research into smaller parts. The answers to the sub-questions together provide the desired answer to the main research question. The sub-questions can be found in Table 1-1 at the next page.

Table 1-1: Sub-questions

Number	Question
1.	How is a sustainable last mile delivery defined?
2.	What are the most important stakeholders in the PostNL case? And, what are their points of view regarding a more sustainable last mile delivery process?
3.	What are the current policies steering the sector in a sustainable direction? And, are they effective?
4.	What are the most promising last mile delivery alternatives, given the sustainability requirements and perceptions of the stakeholders?
5.	What benefits can be obtained by implementing one of these alternatives?
6.	How could this alternative be implemented?

1.4 Research approach and -method

To reach a definitive and satisfying answer to the main research question, an appropriate research approach has to be applied. A typical and widely used approach for choosing between different alternatives based on a set of both quantitative and qualitative criteria, is the Multi-Criteria Decision Analysis (MCDA) approach (Ampe & Macharis, 2008; Vincke, 1992). However, it is not a given that all stakeholders' opinions are included in a MCDA and therefore, the Multi-Actor Multi-Criteria Analysis (MAMCA) approach was developed (Hadavi, Macharis, & Van Raemdonck, 2018).

By including more stakeholders in the early stages of the MCDA, in explicit, with the problem definition and defining the criteria, the MAMCA approach claims to aid more sustainable decision-making in large and complex transport projects (Macharis, Turcksin, & Lebeau, 2012). Since the last mile problem in cities is a problem that concerns many different stakeholders with different points of view, as can be derived from the introduction, the MAMCA approach seems to be a better fit for this research than regular MCDA approaches. Besides, the MAMCA approach not only aids the decision-making, but also covers implementation, which is a totally different stage in these kinds of projects. Hence, it is more appealing from a sustainability perspective and is therefore applied in this study.

The MAMCA approach starts with defining the problem and the alternative measures (Macharis, Turcksin, & Lebeau, 2012). Secondly, the relevant stakeholders have to be identified, followed by assessing their key objectives and the relative importance of the objectives. This is covered by the second sub-question. Then, indicators for the criteria can be defined, in explicit, the criteria's units of measurement. The next step is the actual evaluation of the alternatives with regard to the criteria, resulting in a ranked outcome of the alternatives revealing their strengths and weaknesses in the sixth step. Hereby, the fourth sub-question is answered. The final step is to implement the chosen alternative.

Hence, it can be stated that the MAMCA approach supports the complete project cycle for large scale transport projects, which is what makes it attractive for the kind of problems this study also tries to solve. However, the choice for the last mile delivery alternative is based on data with high uncertainty, thus implementing a completely new logistics system throughout PostNL would be too far-fetched. Furthermore, full-scale implementation of one of the alternatives would require high investments of finance and time. Instead, modelling and simulation provides a safe environment to experiment with different scenarios and explore the new system's behaviour. So, that is where the application of the MAMCA-approach in this study differs from the original definition.

Simulation is about modelling a real process or proposed system to conduct numerical experiments, providing better understanding of the system's behaviour for a given set of conditions (Kelton, Sadowski, & Sturrock, 2003, p. 7). By firstly defining the simulation model, and secondly the set of conditions for which the proposed alternative works as desired, estimates about proper implementation can be made. Hence, both the fifth and sixth sub-question can be answered.

By answering all the sub-questions, it should be possible to derive the answer to the main research question. However, it is not yet made clear what exactly the sub-questions contribute to the definitive answer. Therefore, the research methods that will be applied to answer each of the sub-questions, are described in the following paragraphs.

1.4.1 Sustainable Last Mile

The first sub-question aims at defining a framework of requirements, or criteria, for a sustainable last mile delivery process by means of a literature study. A large body of literature is available that discusses the sustainability issues, both within companies (corporate sustainability) and within supply chains (sustainable supply chain management (SSCM)) (Ahi & Searcy, 2013; Seuring & Müller, 2008). First, sustainability in general is defined. This is then projected on the last mile problem theory, which can also be found within the SSCM literature. Hence, a definition of a sustainable last mile is obtained, which defines the objectives for the new system to fulfil.

1.4.2 Stakeholders and their perspectives

As mentioned before, the identification of stakeholders and their key objectives is an important aspect of the MAMCA-approach (Macharis, Turcksin, & Lebeau, 2012). A literature study has been conducted to answer the second sub-question, which is defined so that the stakeholders of the last mile and city logistics problem can be identified. Furthermore, based on PostNL documents, the key objectives of the stakeholders are identified, which are used to extend the sustainability framework.

1.4.3 Identify active policies

The issues around city logistics have played an important role in the municipal elections last March (PostNL, 2018). Since many different opinions about the design of city logistics have been raised, it is important to get more insight in both the active policies and intended policies regarding city

logistics. Thereby, opportunities (subsidies) and constraints (restrictions) become clear, finalising the sustainability framework. The insights have been obtained by literature studies and interviews within PostNL.

1.4.4 Selecting the best alternatives

The next step of the MAMCA-approach is the selection of the best alternative for city logistics, which is determined by the fourth sub-question. The selection is based on the performance of the alternatives on the criteria in the sustainability framework and is executed by means of a Multi Criteria Decision Analysis (MCDA) method, in explicit, the SMART-method (Edwards, 1977).

The MAMCA-approach overlaps the SMART-method on the first steps, as the first steps of the SMART-method are the identification of stakeholders, the alternatives (decisions) to choose from and the definition of the dimensions of value to analyse the alternatives (Edwards, 1977). The next step is to assign weights to the criteria, or dimensions. The weights can simply be based on a ranking, or a ranking combined with assigning a relative importance. In this study, the weights will be determined based on only the ranking of the criteria, based on the perceptions of different stakeholders. The obtained ranking is already uncertain, since representatives from the stakeholder group will be consulted, without consulting more individuals from this group. Hence, by assigning a relative weight to the obtained ranking, uncertainty would only increase and possibly mitigate trends in the rankings, as one is more conservative with assigning importance than others. Finally, the alternatives have to be measured along the dimensions, followed by a calculation of their total utility.

The purpose of the fourth sub-question is to make a quick selection from the alternatives that will be implemented in a simulation model, based on fuzzy and uncertain data. The SMART-method is aimed at the busy decision-maker that wants a quick method to analyse decisions (Edwards, 1977). According to Edwards (1977), the SMART-method yields very close approximations of the weighted averages compared to much more complicated non-linear approaches. Furthermore, the SMART-method is easy to compute and understand, and therefore fulfils the requirements for implementation in this study.

1.4.5 Costs and benefits of the selected alternatives

A logistics operation in cities is subject to many uncertainties: what is the traffic situation in the city, for how many red lights or pedestrians does the delivery van have to stop, how long does it take for the customer to open the door and sign for reception, does the delivery man have to walk up a four-story building before he can drop-off the package? By using a simulation model, and in particular a discrete-event model, all these uncertainties can be sampled from an uncertainty space (Walker, Marchau, & Kwakkel, 2013). In contrast, a spreadsheet is a deterministic model using statistical averages in most of the aforementioned cases, leading to less reliable results.

As already mentioned, full-scale implementation of one of the alternatives would require high investments of finance and time. Since the process of parcel delivery can easily be broken down to a set of events that follow up in time (which is covered in more detail in the fifth chapter) and discrete-event simulation (DES) is widely used among logistical problems, the best alternative(s) will be implemented in DES-model (Behiri, Belmokhtar-Berraf, & Chu, 2018; Simoni & Claudel, 2018; White & Ingalls, 2009).

The DES-model can then be used to conduct experiments with. To reduce the required simulation time, not all possible combinations of input variables will be simulated. Instead, a Design Of Experiments (DOE) method can be applied (Kleijnen, 2001), depending on the complexity of the simulation model. Thereby, the focus of the experiments will be more on the inputs and outputs of interest, in explicit, the variables that provide the desired results. By conducting experiments with different settings for the (uncertain) variables, outcomes with regard to the costs and benefits of the alternative(s) can be obtained. Hence, the fifth sub-question can be answered.

1.4.6 Implementation

The sixth sub-question is defined to provide PostNL and the other stakeholders with a descriptive policy advice that makes it more attractive to implement the new last mile delivery process. This will be obtained by conducting desk research based on the optimal outcomes of the simulation experiments. This will be focussed on solutions applied in other (city) logistics sectors to assess opportunities for similar design solutions for the last mile delivery alternatives. Hence, the sixth sub-question can be answered.

1.5 Research Flow and report structure

The study as proposed in this chapter is translated into a Research Flow Diagram, which is shown in Figure 1-1. The Research Flow Diagram shows the order of activities or topics that will be executed during this study in the same order as they are covered in this thesis.

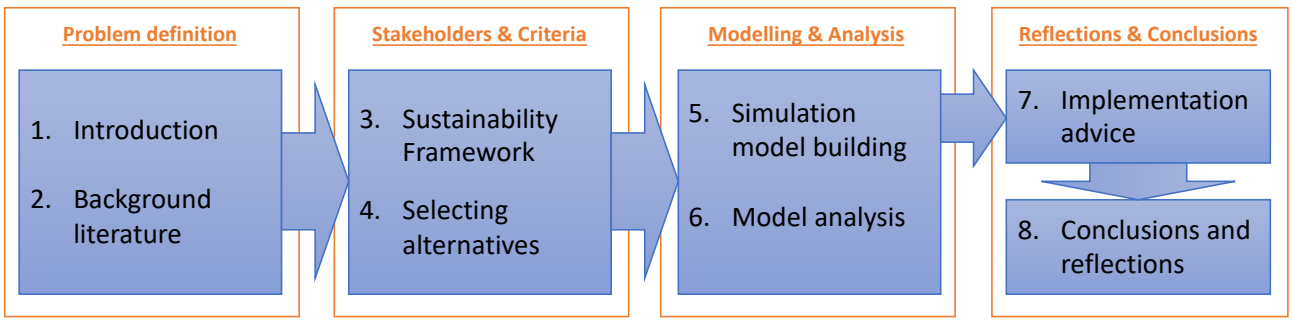


Figure 1-1: Research Flow Diagram

The first block defines the problem dealt with in this study. The problem definition consists of the introduction of this chapter and the background literature as presented in chapter 2. Therein, the applied definitions of the last mile problem and city logistics are presented, together with the alternative last mile processes. After that, the *'Stakeholders and Criteria'* block is covered, which consists of the stakeholder analysis and the definition of the sustainable last mile in chapter 3 and the criteria analysis and MCDA in chapter 4. Then, the *'Modelling and Analysis'* block covers the creation of the initial simulation model of the sustainable last mile in chapter 5. The results of the experiments are then analysed in chapter 6, Finally, the *'Reflections and Conclusions'* block provides the reader with a descriptive advice for the implementation of the alternative system in chapter 7. The conclusions and reflections are written in chapter 8 and conclude this report.

2 | Identification of Last Mile alternatives

Since the research question indicates the need for a more sustainable way of transporting goods along the last mile, it is interesting to see what has already been written in literature. To structure the search for last mile alternatives, key words like *last mile problem, sustainable last mile, last mile efficiency* and others have been applied in the Scopus search engine. Based on the results, the basic definitions around the last mile are explained first, followed by the last mile alternatives.

2.1 What is the last mile?

Before products reach the consumers' hands, they flow through a chain of organisations that try to add value to the product. This is captured in the Supply Chain Management (SCM) theory, in which a supply chain (SC) is defined as: *"…a network of organizations that are involved, through upstream and downstream linkages, in the different processes and activities that produce value in forms of products and services in the hands of the ultimate consumer."* (Christopher, 1998).

Within the SCM theory, the coordination of material-, financial- and information flows is seen as very important (Stadtler, 2004). As the developments in Information Technology (IT) are advancing, big increases in information flows and links between all tiers in the SC can be noticed. Moreover, retailers are increasingly interacting with their customers through the internet, which has opened a new market, called *"e-commerce"* (Khan, Varshney, & Quadeer, 2011; Stadtler, 2004). While e-commerce was initially focussed on the Business-to-Consumer (B2C) market, it is increasingly becoming active in the Business-to-Business (B2B) market.

Since most products cannot be utilised by only online interactions, they have to be transported to the recipient's location. This last step in the SC is called *"The Last Mile"* in SCM theory (Brown & Guiffrida, 2014; Edwards, McKinnon, & Cullinane, 2010). Since the Netherlands have a very high urbanisation degree, a Dutch last mile is closely related to the city logistics field, which was already defined as *"the last leg in the supply chain to the customer location in a city, or the first leg from a customer location in a city back into the supply chain"* (Boer, Kok, Ploos van Amstel, Quak, & Wagter, 2017, p. 13). Hence, the last mile alternatives are seen as possible solutions for city logistics as well.

However, the Last Mile process has still been treated as a black box. Hence, the Last Mile process under investigation could either be very broad (SCM perspective), or very narrow (transporters perspective). Furthermore, the definition of Boer et al. (2017) could be interpreted from both perspectives, despite their definition being closely related to the definition of Gevaers et al. (2009), which is defined as *'the last stretch of a business to consumer (B2C) parcel delivery to the final consignee (consumer) who has to take reception of the goods at home or at a cluster/collection point'* (Gevaers, Van de Voorde, & Vanelslander, 2009). Gevaers et al.'s definition however is more

focussed on the actual parcel delivery part of the supply chain, where Boer et al.'s definition is defined in a broader sense. To clarify the last mile process under study, Figure 2-1 is adopted.

Figure 2-1: Regular delivery process (Gevaers et al., 2009)

According to Gevaers et al. (2009), the regular operation starts at picking up a parcel at the shipper's location and transport the parcel to a nearby sorting centre. From there, a line haul continues the journey to a sorting centre close to the customer's location. Then, the *'last mile'* parcel delivery is conducted, where the parcel is being delivered to the customer's location (highlighted in red in Figure 2-1). This last, highlighted part of the last mile will also be the further focus of this study, as that is the part of the last mile where companies can really differentiate and innovate.

2.2 Alternative last mile delivery processes

Regarding last mile 'delivery', Edwards et al. (2010) make a distinction between *"home deliveries"* and *"personal shopping"* and state that personal trips to shopping centres can be more energy-consuming than the entire upstream supply chain. Instead they conclude that home deliveries are likely to produce less CO_2, even after including failed deliveries in the analysis. Hence, home deliveries can still be seen as a legit sustainable option of last mile delivery, providing that the current externalities like carbon emissions are mitigated.

2.2.1 Collection and Drop-off Points and parcel lockers

The alternative of Collection/Drop-off Points (CDPs) has already been introduced by companies like PostNL. According to Smit (2018), PostNL uses classic retail locations like Albert Heijn stores as a CDP, where PostNL collects the items one or a few times a day. Furthermore, PostNL has operationalised the use of parcel lockers in crowded public areas, where people can collect their parcels 24/7 at a moment that fits their agendas best. Besides, *"Click & Collect"* or *"Customer Pick-Up"* services have been introduced by Albert Heijn and bol.com, Walmart, Amazon and Tesco, where products ordered online are directly delivered to the retail location nearby. According to McKinsey and Company (2017), parcel lockers (and presumably CDPs) have the ability to cut both labour costs and emissions drastically. However, this highly depends on recipient's mode of transport.

2.2.2 Urban Consolidation Centres

Another alternative that is increasingly getting attention in literature is the implementation of Urban Consolidation Centres (UCCs). UCCs are transhipment points just outside the city boundaries that can be used to consolidate shipments with the same destinations and switch to greener transport

modes (Clausen, Geiger, & Poting, 2016). This can result in less traffic in the cities, reduce emissions, enhance liveability and reduce costs (Gogas & Nathanail, 2017). However, the business case is depending on a lot of uncertain variables, which makes it unattractive for parties to invest at first and thereby sensitive for subsidies (Janjevic & Ndiaye, 2017).

2.2.3 Crowdsourcing logistics services

However, due to other concerns like the traffic impact of delivery vans and the corresponding emissions, there is a growing need to change. The first alternative to discuss is one that is receiving an increasing amount of attention in literature, namely crowdsourcing the logistics services from a pool of workers (Wang, Zhang, Liu, Shen, & Hay Lee, 2016). Wang et al. (2016) propose a model based on pick-own-parcel (pop)-stations as used by Singapore Post. After a parcel arrives at the pop-station, the delivery job is outsourced as a crowd-task by means of an app. Wang et al. (2016) state that operational and environmental benefits can be obtained due to reduced labour and handling costs. However, they excluded the amount of extra transport the crowd will make for each delivery.

Kafle et al. (2017) propose a somewhat similar model to Wang et al. (2016), but they replaced the pop-station by a conventional delivery van as pickup point for crowd-workers to pick-up parcels at so-called relay points. Kafle et al. (2017) assume that crowd-workers primarily walk or cycle when delivering parcels, which has a positive effect on the reduction of traffic movements. This is in contrast to Wang et al. (2016), whom assume that crowd-workers only use cars for their delivery tasks. Both studies conclude that the proposed last mile solutions are more environmental and economically friendly. However, the conclusion only reflects on the (operational) costs and lacks decent estimates of the traffic impact.

2.2.4 Drone Delivery

On the side, drone delivery may be an ambitious alternative for a little more into the future. Drones are an attractive alternative for the conventional delivery van, since they do not have to use roads to travel (Lohn, 2017). Furthermore, drones are green vehicles due to the use of electronic engines in most occasions. However, a lot still has to be done before drones can actually be used for parcel delivery, like changing regulations and apply changes to public spaces (Lohn, 2017).

2.2.5 Summary

Most of the alternatives mentioned above are, among many other alternative systems for last mile delivery, analysed by McKinsey and Company (2017) to provide insight in the most promising alternatives for the future. The six most promising alternatives according to the outlook are: UCCs, parcel lockers, load pooling, night delivery, electric vehicles and unmanned automated vehicle lockers. However, multiple solutions have to be combined to achieve the most success and a conclusive combination on what would be best is lacking. Besides, the results are primarily assumption based, thus mainly without quantified results.

2.3 Alternatives implemented by PostNL

Some of the aforementioned alternative last mile options have already been implemented by PostNL as an additional service. These alternatives for the last mile are the result from PostNL signing the Green Deal Zero Emission 'Stadslogistiek' (GDZES) agreement, which is aimed to make city logistics free of emissions (Green Deal ZES, n.d.). PostNL started research projects into sustainability and city logistics and as a result, the department *"city logistics"* has been established to manage these projects (Tuinhout, 2018).

One of the services is called "*Pakje Gemak"*, enabling the customer to define the desired pickup location (PostNL, n.d.). Beside the existing postal office network, PostNL has extended the service to partner retailers like Albert Heijn and eventually placed parcel lockers at busy and populated areas like train stations. This is still a popular alternative and operational at a big scale (Smit, 2018). Furthermore, Smit (2018) states that both the parcel lockers and retail locations options cause customer satisfaction to increase, while the number of kilometres driven is decreasing. Hence, operational emissions are expected to decrease, but there is no clear insight therein.

Next to the *"Pakje Gemak"* service, PostNL recently started to offer evening delivery as a premium service (Hünteler, 2018; van den Berg, 2018). Thereby, PostNL offers to deliver the parcel between 18:00h and 22:00h, so there is a higher probability of the recipient being home. The recipient still has to physically accept the parcel, and currently there are no plans inside PostNL to combine evening delivery with parcel lockers. This is in contrast to the night delivery option of McKinsey and Company (2017), which uses (personal) parcel lockers in the central mail hall of apartment-buildings, or next to the front door, so that customers do not have to be home to accept the parcel. Thereby, a strong decrease in operational costs, kilometres driven and the related emissions can be obtained. However, according to Hünteler (2018), evening delivery is more expensive than conventional day delivery and less efficient. That is why evening and night delivery have to be analysed separately.

PostNL has recently conducted a study into the possibilities of making the parcel lockers part of an unmanned vehicle network (Tuinhout, 2018). According to Tuinhout (2018), vehicles then drive to certain places in the city closer to the customer, offering more convenience, reducing operational costs, reducing emissions and reducing traffic in the city. At the moment, this project is awaiting a business case to be tested on a bigger scale. Also, several regulatory changes and technical advancements have to be made before this project can be executed.

Lastly, the city logistics department is running a UCC project in cooperation with several companies within the Green Deal ZES agreement. The UCC is located in Duivendrecht (Deudekom), where products for the Universities of Amsterdam and the municipality of Amsterdam are collected,

consolidated and transhipped onto greener transport modes like electric vans, *"goupils"* and/or cargo bikes (Tuinhout, 2018). This is one of the first projects to be executed on this scale (statement derived from internal presentation). However, there is not a lot of insight in the effects and benefits of this alternative. Therefore, more research is needed in order to make the business case even stronger (Tuinhout, 2018).

2.4 Conclusion

So, it can be noted that there are many different alternatives for last mile delivery. However, what options are worth considering when opting for sustainable city logistics? The aforementioned report of McKinsey and Company (2017) lists six promising alternatives, but also notes that a combination between these six may result in even better results. Beside the McKinsey and Company (2017) report, the alternatives *crowdsourcing, CDPs and lockers, UCCs, drones* and *evening delivery* have been discussed. Of these, the following will be taken into consideration in the remainder of the report:

Table 2-1: The alternatives taken into consideration

Alternative
Electrification of vehicle fleet
Cargo bikes
Evening and Night delivery
UCCs
CDPs and parcel lockers
Crowdsourcing logistics services

Drones have not been taken into consideration in the analysis, as the opportunities to implement drones for parcel delivery are very limited. Furthermore, drones have to be further developed and a lot of regulations still have to be made to make it possible. Hence, the implementation of drones is a plan for far in the future, and not for now. The load pooling option from the McKinsey and Company (2017) report is combined with the crowdsourcing option, as they seem very similar in nature (McKinsey & Company, 2017). The unmanned vehicles from the same report have not been taken into consideration either, since the same issues apply for these as for drones.

With these conclusions, none of the sub-questions can yet be answered, despite the first block of the research flow diagram being completed by now. Furthermore, the problem definition should be clarified by these first two chapters. So, it can be stated that there are many alternative systems and solutions for the last mile parcel delivery that could make the process more sustainable (i.e. enhance efficiency and reduce the GHG-emissions), but a lack of insight remains in the costs and benefits of each of these alternatives with regard to the same set of criteria. Therefore, the next block focusses on defining the sustainability criteria framework for assessing the last mile alternatives.

3 | Defining the sustainable last mile

Sustainability is getting an increasing amount of attention in both research and practice, which in the early days was defined by the so-called Brundtland commission as utilising resources to meet the needs for the present without compromising the needs for future generations (WCED, 1987). Problems like environmental pollution, child labour, hunger and extreme poverty are phenomena that should not be present today. Therefore, countries belonging to the United Nations (UN) defined 17 *"Sustainable Development Goals"* (SDGs) to tackle these problems for good (United Nations, 2015).

To be able to contribute to the SDGs, the Dutch government has adopted certain policies to enforce commitment to these SDGs by big companies (Ploumen, 2016). Furthermore, due to the 2014/95/EU directive from the European Union (EU), big companies like PostNL are forced to become more transparent about their non-financial bookkeeping. In response to the directive, PostNL is implementing the Global Reporting Initiative's (GRI) standards method for environmental reporting in order to transparently report on their contributions on reaching the SDGs (GRI, 2017).

As more companies and supply chains are concerned about their sustainable performance, research is trying to define the terms and requirements of sustainability. Hence, research responded with the terms *"corporate sustainability"* and *"sustainable supply chain management"* (Ahi & Searcy, 2013). What this exactly entails, is covered in 3.1 and 3.2. Furthermore, based on the process of PostNL, a basic last mile delivery process has been drawn to provide more insight in the stakeholders of the last mile problem. This is described in 3.3, followed by an assessment of their points of view in the fourth paragraph. Finally, a short summary is provided in 3.4.

3.1 Corporate sustainability

In the beginning, sustainability definitions primarily focussed on the environmental effects of companies. However, businesses are increasingly involving the social aspects of their operations as well (Ahi & Searcy, 2013). Many scholars have defined corporate sustainability in relation with the triple bottom line approach (Elkington, 2002), with one option being "the creation of resilient organisations through integrated economic, social and environmental systems" (Bansal, 2010).

For the success of a sustainability strategy in a company, leadership and the commitment of management is perceived as important (Székely & Knirsch, 2005). Furthermore, companies should be flexible to adapt to changes regarding sustainability and the strategy should be well aligned with the core business processes (Engert & Baumgartner, 2015). Lastly, stakeholder engagement is necessary to develop shared understanding of approaches and expectations (Székely & Knirsch, 2005).

Based on the aforementioned success factors, Székely and Knirsch (2005) have identified several methods and tools for determining the sustainable performance of companies. Among these, sustainability indicators as raised by the GRI may be used, which are distinguished in *productivity/efficiency ratios, intensity ratios,* and *general percentages*. These categories include ratios like the labour productivity, emissions intensity and return on investments ratios and hence, include indicators from all directions of the sustainability spectrum (Székely & Knirsch, 2005). Therefore, the indicators defined by Székely and Knirsch (2005) will be used as a first input to the sustainability framework.

3.2 Sustainable Supply Chain Management (SSCM)

Though, there is a growing recognition that becoming sustainable is not achievable by one company on its own. As more companies are adopting SCM strategies, there is also a growing interest in SSCM research (Seuring & Müller, 2008). Beside the first big literature study of Seuring and Müller (2008) into SSCM research and definitions, Ahi & Searcy (2013) conducted a literature review 5 years later to see if the definition could be extended. By combining corporate sustainability and SSCM, they came up with the following definition:

"The creation of coordinated supply chains through the voluntary integration of economic, environmental, and social considerations with key inter-organizational business systems designed to efficiently and effectively manage the material, information, and capital flows associated with the procurement, production, and distribution of products or services in order to meet stakeholder requirements and improve the profitability, competitiveness, and resilience of the organization over the short- and long-term. (Ahi & Searcy, 2013, p. 339)"

By adopting SSCM strategies, companies and SCs can improve their environmental performance (Esfahbodi, Zhang, Watson, & Zhang, 2017). Furthermore, Esfahbodi et al. (2017) state that by exerting the right pressure, governments can successfully motivate companies to adopt SSCM strategies. Besides, companies with a sustainable reputation can gain a competitive advantage over competitors, as SSCM practices result in more favourable consumer brand evaluations and more purchase intentions (Gillespie & Rogers, 2016).

It is also indicated that inter-firm collaboration in SCs in a sustainability context can enhance sustainable performance (Niesten et al., 2017). Sustainable performance can be further improved by maintaining relationships and trust with customers, as normative pressures from customers and markets are key motivations for the adoption of SSCM-practices (Zhu, Feng, & Choi, 2017). Moreover, collaboration that results in alliances that stimulate the use of sustainable technologies

can result in enhanced legitimacy for the involved firms (Kishna, Niesten, Negro, & Hekkert, 2017). Hence, collaboration between firms, markets and customers is seen as an important factor for sustainable performance throughout this research.

However, most of the studies cited above have also indicated the need for research into indicators and decision variables for environmental and social performance, as clear examples are lacking. According to Seuring (2013), *"carbon emissions"* is the only environmental decision variable that is regularly discussed in SSCM literature.

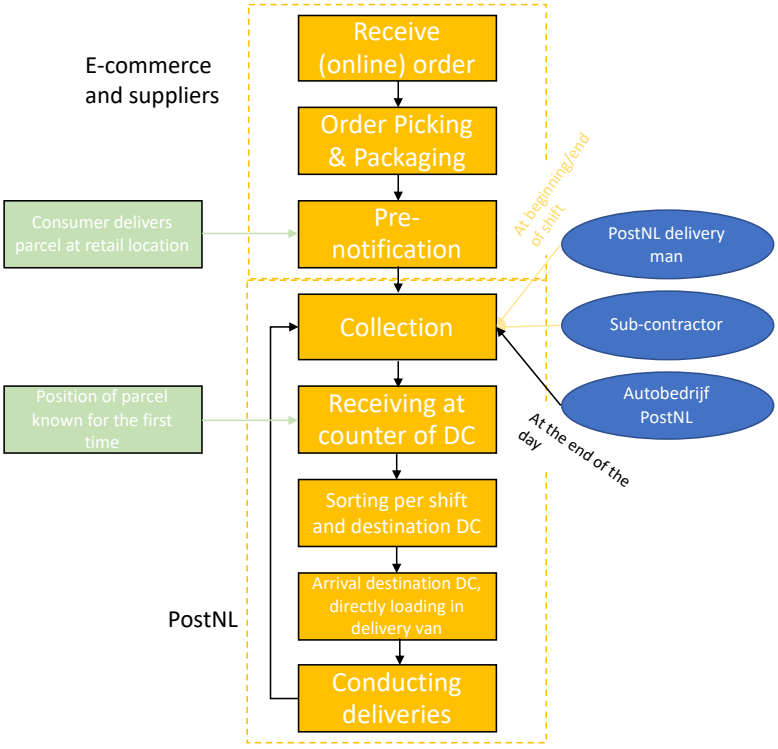
3.3 The comprehensive last mile delivery process

From the foregoing paragraphs it can be stated that research still has to make some steps in order to properly define corporate and SC sustainability. As mentioned before, the last mile delivery problem is part of the SCM literature. However, from SSCM literature it cannot be made clear what requirements a sustainable last mile has to fulfil. Hence, this chapter proceeds following Kishna et al.'s (2017) statement that collaboration between firms, markets and customers is required to enhance sustainable performance.

By firstly defining the last mile delivery process in more detail, important stakeholders can be identified. In the next paragraph, the stakeholders' points of view have been assessed. This provides more insight in the criteria for a sustainable last mile and hence, it becomes clear what aspects of the last mile are likely to enhance support from stakeholders.

3.3.1 The (PostNL) process

The process can be separated in two parts: one for the e-commerce and general suppliers, and a part for PostNL (or any other logistics service provider) (Bakr, 2018). Many web shops do not have their own transport services. Therefore, transport services are outsourced to so-called third-party logistics (3PL) providers. PostNL is one of these 3PL providers, since they can offer the complete logistics backbone to transport parcels to every address in the country (Vaandrager, 2018).



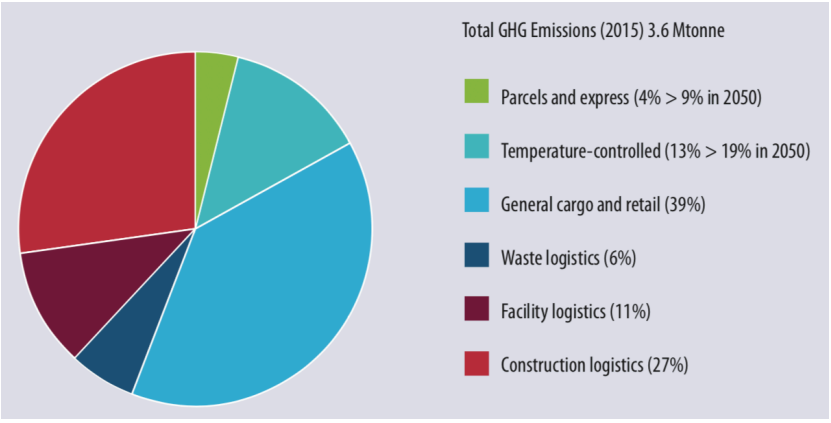
It all starts with a retailer or consumer placing an order at their supplier or an e-commerce company. The order is picked and packed before it is being *"Pre-notified"*. This means it is notified to PostNL that the parcel is ready for pickup. Besides, there is a possibility for consumers to deliver a parcel at a retail location. The retail location then takes the responsibility for the *"Pre-notification"* procedure. Hence, retailers, e-commerce companies and suppliers have done enough for sending their parcels.

The next step in the process is *"Collection"* by PostNL, which is another word for picking up the parcels at their respective locations (Bakr, 2018). There are three options (shown as blue ovals in Figure 3-1), namely by either the PostNL delivery man or a sub-contractor that picks up the parcels at the beginning/end of his/her shift, or by the *"Autobedrijf PostNL"*, which picks up the parcels at the end of the day and delivers them to the nearest DC to be sorted.

After receiving the parcels at the counter of the respective DC, their position in the chain of PostNL is finally known (Bakr, 2018). Parcels are sorted per destination and shift by means of big sorter machines. The sorted containers are then sent (by night) to the next DC, where the delivery van is waiting for its containers to load. During the remaining day, eight to ten different shifts will depart from the DC that conduct the deliveries. Besides, as mentioned before, the delivery man and sub-contractor are free to choose when (at the beginning or end of their shift) they want to pick-up parcels at retail locations.

3.3.2 The stakeholders

From the aforementioned process, three different roles are easy to identify: there is always a *"sender"*, a *"recipient"* and (almost always) a *"logistics provider"* involved. These roles are similar among most of the different segments of city logistics, which have been defined in Figure 3-2 (Boer, Kok, Ploos van Amstel, Quak, & Wagter, 2017). From Figure 3-2 it can be noticed that the distribution of parcels and express goods are only responsible for 4% of the GHG emissions. All others are concerning B2B deliveries of, for instance, construction sites, maintenance of buildings, or retail. Hence, the priority of changing the parcel delivery system to reduce congestion could be questioned. However, as can be noticed from 3.4, stakeholders still perceive this sector as an important source of the congestion problems. Therefore, all efforts to make it more sustainable could have a positive



effect on the stakeholders' perspectives of the problem. Furthermore, solutions found by this study may also be applicable and interesting for other sectors in Figure 3-2.

Since all of the city logistics segments consist of senders, recipients and logistics providers, these roles will be considered as three different stakeholder groups in the city logistics context Furthermore, these roles can be found in all B2B, B2C and C2C deliveries, so all of these markets will be considered. Hence, a sender could be an e-commerce company that delivers products to consumers, a supplier to a restaurant or a consumer to another consumer for instance. Recipients can be then be seen as consumers or businesses in the city. The logistics providers can then be seen as companies like PostNL or DHL, distributing goods between suppliers and recipients.

Besides the three obvious roles, governmental organisations are important to consider as stakeholders. National governments and municipalities in particular are affected by the externalities of city logistics and try to find new ways for mitigating these effects therefore. Municipalities are introducing new regulations (see for instance the "Uitvoeringsagenda Stedelijke Logistiek Amsterdam"), while the Dutch national government has implemented national policies to cope with rising emissions (Ploumen, 2016).

There are also several organisations that represent the 3PL branch in the governmental discussions. Two examples are *"Transport en Logistiek Nederland"* (TLN), and *"Evofenedex"*, both fighting for unambiguous policies regarding city logistics (TLN, 2018). They state that tightening the rules regarding environmental zones (Milieuzones) in some city centres will cause 3PL providers to conduct unnecessary investments in new material for these areas. Furthermore, the depreciation on this new material will be way higher than necessary, increasing the risk that many 3PL providers to go bankrupt (TLN, 2018).

Lastly, *"thuiswinkel.org"* represents the Dutch e-commerce businesses and offers a quality mark for its members. Businesses connected to thuiswinkel.org and have the label are guaranteed to offer the rights services to the consumer (Thuiswinkel.org, n.d.). Furthermore, giants like bol.com and Coolblue are connected and have started a campaign together to make the consumers more aware of their ordering behaviour, called *"Bewust Bezorgd"* (Thuiswinkel.org, 2018).

3.3.3 Shortlist

Concluding this paragraph, a shortlist is created in Table 3-1 to provide a quick overview of what has been discussed and the stakeholders considered during the analysis.

Table 3-1: Stakeholder shortlist Parcels and Express

	Parcels and Express
Supplier	E-commerce, general retail and/or consumers
Logistics provider	PostNL or 3PL
Customer	Consumers and/or other companies
Government	Local and domestic
Other	TLN and Evofenedex for transport Thuiswinkel.org for e-commerce

3.4 Assessing stakeholders' points of view

So, what exactly concerns the previously defined stakeholders regarding the sustainability of their last mile through the city? To answer that question, each stakeholder group as defined in Table 3-1 (the first column) has been analysed based on research, news articles, social media and interviews within PostNL. Each stakeholder is covered in a separate paragraph.

3.4.1 Consumers and business recipients

According to a survey from the European Commission (2015), the main concern of consumers is the delivery price. Home delivery has to be free of charge (or a low fee), fast and reliable track and trace. Some of these findings are underpinned by other research, stating that *"delivery costs"* is a dissatisfying factor by many consumers (Lowe & Rigby, 2013). Furthermore, home deliveries are still highly preferred above other delivery options like *"Click and Collect"* or *"Collection Points"* (Lowe & Rigby, 2013).

There is not much research into the preferences of B2B deliveries. It is assumed that companies that order their products online or have them delivered, prefer to have them delivered at the company's address. Furthermore, delivery costs are assumed to be an important factor of interest as well, as this may have a negative effect on the end-product's potential profit.

3.4.2 E-commerce and suppliers

Suppliers are, in this document, referred to as the suppliers of the different segments as defined in Figure 2-1, or as the e-commerce companies supplying the end-consumers. A definition that covers both of types of suppliers and will be used throughout this document is the party that ships or hires a 3PL provider to ship products to their customers.

On behalf of PostNL, McKinsey and Company recently conducted a survey to identify important stakeholders of PostNL parcel delivery's operations and discover their most important factors of interest (van Spronsen & Middelburg, 2018). Most respondents are a supplier or business partner of PostNL. They stated that they find it important for PostNL to focus on carbon free delivery, the well-being of employees in a good working climate and customer satisfaction. From their own perspective, carbon free delivery is ranked even higher than from the PostNL perspective, together with customer

satisfaction and well-being of employees (van Spronsen & Middelburg, 2018). Hence, it can be stated that sustainability issues are high on the stakeholders' agendas.

3.4.3 Logistics providers

The 3PL providers are increasingly trying to become sustainable. PostNL writes that they have the ambition to become fully carbon neutral (PostNL, 2017). Therefore, PostNL states that more flexibility in operations and better cooperation with customers is needed. Furthermore, they express the need for innovative IT projects to support the sustainable developments.

Beside PostNL's position paper, DHL has expressed their ambitions for the future in a position paper. DHL is currently investing heavily in cleaner and electric vehicles to mitigate emissions (van Benten, 2017). Furthermore, DHL expresses their concerns about the labour force that is active in the sector, as there are a lot of different constructions among the companies. Therefore, they express the need for ambiguous agreements within the sector that support the increased need for flexibility.

3.4.4 Governmental organisations

As already mentioned in the introduction, city municipalities and the Dutch national government are increasingly experiencing the problem of clogging cities. Hence, according to a study that WKPA conducted on behalf of PostNL, clogging cities and sustainable city logistics were reoccurring themes in the municipal elections in the Netherlands (PostNL, 2018).

Several parties in the municipal board have expressed their interests in implementing or expanding the so-called *"milieuzones"* (PostNL, 2018). Besides, some parties promote to create a car-free city and make the city better accessible by public transport or bicycle. Regarding parcel delivery in the city centre, most of the parties in the big cities agree upon the implementation of Urban Consolidation Centres, where the streams of parcels meant for the city are consolidated outside the city and delivered by one operator (PostNL, 2018).

In August 2009 the Dutch cooperation for air quality improvements (NSL) was formed to achieve compliance with the 2008/50/EG7 directive of the EU (Rijkswaterstaat, n.d.). Recently, this cooperation has been prolonged, since some inner-city roads exceeded the limits for NO_2 and PM_{10} instances (Ministerie van Infrastructuur en Waterstaat, 2018). Therefore, new measures have been identified to aid the achievement of the targets set by the EU directive.

Lastly, many local governments, provinces and municipalities agreed upon making the city distribution of goods emissions-free by having signed the Green Deal ZES agreement (Green Deal ZES, n.d.). They will try to achieve this by cooperating with the branch organisations, public-private partnerships and the transport sector with the aim of achieving a zero-emission city distribution by 2025.

3.4.5 Branch-organisations

As can be noted from Table 3-1, the *other* stakeholders are divided in two categories: the logistics branch organisations and the e-commerce branch organisations. Firstly, TLN and evofenedex are involved on behalf of the logistics branch, both fighting for ambiguous policies and regulations domestically (TLN, 2018). Furthermore, they are opponents to the ideas of municipalities to extent regulations regarding the "milieuzones", as this provokes wrong investing behaviour at companies. Besides, TLN and evofenedex are working together towards zero-emission city distribution in 2025, as they have both signed the Green Deal ZES agreement (Green Deal ZES, n.d.).

On the other hand, thuiswinkel.org is involved on behalf of the e-commerce sector. As aforementioned, thuiswinkel.org has recently started a campaign to raise awareness of consumers' online shopping behaviour and have set the goal to make e-commerce and home deliveries more sustainable (Thuiswinkel.org, 2018).

3.5 Conclusion

Concluding this section, all the opinions and points of view towards city logistics and the last mile delivery problem have been summarised in Table 3-2.

Table 3-2: Summarising points of view stakeholders

Stakeholder	Points of view
End-Consumer and Customers	As cheap and fast as possible delivery at the doorstep, with the highest possible flexibility and security.
3PL providers	Become carbon free as fast as possible, increased cooperation between parties and consumers, maintain good working environment
Suppliers	Carbon free delivery, good working conditions and highest possible customer satisfaction with the service.
Governmental organisations	Decrease the negative effect of traffic movements in city, while maintaining economic growth and benefits of B2B and B2C deliveries. The implementation of UCCs could be a solution for big cities.
Others	Firstly, the logistics branch organisations are fighting for ambiguous policies and regulations among municipalities, while also taking the GDZES seriously. Secondly, thuiswinkel.org tries to make consumers more aware of their online shopping behaviour and make them behave more sustainable.

Furthermore, based on Table 3-2, the first three sub-questions of this study can be answered. Firstly, the *sustainable last mile* is defined as one that considers and improves performance on all three pillars of the Triple Bottom Line (Elkington, 2002). Hence, it is seen as important that the last mile complies with the requirements and expectations of all stakeholders from Table 3-2. Thus, the sustainable last mile should be as cheap, fast, flexible and secure as possible, while coming as close as possible to the consumers' doorsteps. The sustainable last mile should therefore enhance the

efficiency of the process, reduce the GHG-emissions and maintain a good working environment for employees. Finally, the sustainable last mile is insusceptible for changing policies and regulations in different cities or areas.

The second sub-question can also be answered based on Table 3-2, which was defined as: *What are the most important stakeholders in PostNL's case?* The most important stakeholders in the process of PostNL are defined as different groups. Firstly, PostNL itself is a 3PL provider in the general process, so other 3PL providers are stakeholders as they are competitors in the field. Then, the consumers and suppliers are important as PostNL is providing them a service to pick-up and deliver a parcel. Since PostNL mainly operates in cities, municipalities and other governmental organisations are important stakeholders. Lastly, PostNL is active in the transport sector, so transport sector branch organisations are important to consider when analysing different alternative last mile solutions.

The third sub-question was defined as: *What are the current policies steering the sector in a sustainable direction?* Based on the foregoing chapter, one of the most present policies is the implementation of a *'Milieuzone'* in certain municipalities. This zone in the city centres tries to fend off older diesel-powered vehicles in order to improve the air quality and traffic flow in the city. Hence, transporting companies with many clients in these areas are forced to look for sustainable transport alternatives to reach their clients in the city. However, as mentioned before, this policy is also getting some critique for the fact that it is not yet the right time for companies to invest in other modes of transport. Besides, there are some opportunities for getting subsidies for the investments in new, electric vehicles. However, these are mainly focussed on the consumers markets.

4 | Analysing the alternatives' sustainable performance

Since there is not only one actor that decides about the optimal city logistics system, the perceptions of all stakeholders have to be included in a set of criteria (Edwards, 1977). Moreover, this is likely to increase the support of stakeholders for the proposed solutions. Besides, including more stakeholders is according to the SMART method, which is applied for the first analysis of the alternatives' sustainable performance. The SMART method is a simple, yet robust, decision analysis tool that results in a ranking of the alternatives based on how good they perform on the set of criteria (Edwards, 1977).

The simplicity of the SMART method is what makes it so attractive for this study, as the first analysis is aimed to indicate promising alternatives. Furthermore, there is a high uncertainty about the performance of the alternatives, which would make the robustness of other more intensive methods questionable. Besides, the linear average method has proven to yield very close approximations to much more complicated non-linear methods (Edwards, 1977). For the purpose of just providing a little insight in promising alternatives, the SMART method is expected to be sufficiently accurate.

First, the criteria have been defined in 4.1, which is according to fourth step of the SMART method (Edwards, 1977). The criteria are mainly based on the outcomes of Table 3-2. This is followed by the fifth step, defining the weights of the criteria based on a ranking, which is covered in 4.2. The sixth step is not applied for the sake of simplicity and the already high nature of uncertainty in the analysis, so step seven is covered in 4.3. Therein, the performance of each of the alternative delivery methods will be assessed along the criteria. The results will be used as input for the analysis, following the ninth step of SMART, which is described in 4.4. Finally, some conclusions have been drawn in 4.5, providing the focus of the remaining chapters.

4.1 Defining the criteria

As already mentioned in 1.4.4, the SMART-method partly overlaps the MAMCA approach with the first few steps. Moreover, the first three steps of the SMART method have already been conducted by the foregoing chapters, as the important stakeholders and the alternatives to choose from (issues) have been identified. However, the fourth step is to identify the 'dimensions' (criteria) along which the alternatives have to be assessed (Edwards, 1977).

To come up with a framework of criteria that are important for all stakeholders, it is important to first have insight in the issues they value individually. Therefore, Table 3-2 is consulted and based on the reoccurring issues among stakeholders, criteria can be derived. Each of the stakeholders is assessed separately to provide a structured overview.

4.1.1 End-consumers and recipients

End-consumers and recipients value a fast and cheap delivery at the doorstep, while also flexibility in the delivery options and product safety and security matter to them (see Table 3-2). From this sentence, some criteria could be derived. These are summarised in Table 4-1 below:

Table 4-1: Criteria from end-consumers

Criterion	Part of the sentence that describes it
Delivery Time	They value fast delivery
Delivery Cost	They also value cheap delivery, as cheap as possible and preferably free of charge.
Product Safety and Security	As can be noticed from chapter 3.4.1., consumers especially are concerned with the track and trace of their product. They are more likely to order at a website that seems reliable (Lowe & Rigby, 2013).
Flexibility	They want to have multiple delivery options to choose from.

4.1.2 3PL Providers

The 3PL providers want to become carbon free as fast as possible, increase the cooperation between parties and consumers while maintaining a good working environment for their employees. Besides, they want to offer their customers reliable delivery times for the lowest possible costs (Tuinhout, 2018). Hence, the criteria that are important to the 3PL providers can be derived and are shown in Table 4-2.

Table 4-2: Criteria 3PL providers

Criterion	Part of the sentence that describes it
Delivery Time	They value fast and reliable delivery times
Delivery Cost	They also value cheap delivery, as cheap as possible
Emissions	Becoming carbon free as fast as possible
Safe working environment	They want to maintain a good and safe working environment for their employees.
Safety and responsibility	They would like to increase the cooperation between parties to realise a more responsible supply chain, providing more safety to the customers' products as a result.
Customer satisfaction	When offering a service, it is the usual matter that the customer has to be satisfied with the service, otherwise they will choose for another provider. Therefore, this is seen as an important factor for the 3PL providers, although it is not specifically mentioned.

4.1.3 Suppliers

The suppliers value carbon free delivery, good working conditions and the highest possible customer satisfaction with the service. The latter is assumed to be achieved by offering flexibility in delivery options to the customer, combined with reliability of delivery times. Furthermore, suppliers are assumed to always look for the most economical offer (cheap and reliable). Hence, the criteria can be derived and they are shown in Table 4-3 below.

Table 4-3: Criteria suppliers

Criterion	Part of the sentence that describes it
Delivery Time	They value fast and reliable delivery times
Delivery Cost	They also value cheap delivery, but reliable at the same time
Emissions	They value carbon free delivery of the 3PL
Safe working environment	They want to maintain a good and safe working environment for their employees. Furthermore, they value the working conditions at the 3PL too.
Customer satisfaction	When offering a service, it is the usual matter that the customer has to be satisfied with the service, otherwise they will choose for another supplier. Therefore, this is seen as an important factor.

4.1.4 Governmental organisations

Governmental organisations are concerned by the number of traffic movements in the city, which have to decrease. Therefore, they prefer the option of a UCC, which is increasingly involved in their discussions. However, there is no interest in the big investments that cities now have to make to make UCCs feasible. Besides, governmental organisations are trying to mitigate the carbon emissions of cities. Hence, the criteria from Table 4-4 can be derived.

Table 4-4: Criteria governmental organisations

Criterion	Part of the sentence that describes it
Emissions	They value carbon free delivery, while also cars are banned in city centres.
Safe working environment	They want to maintain a good and safe working environment, and also a safe city.
Traffic impact	Trying to decrease the number of traffic movements
Investments in Infrastructure	They would like UCCs, but without having to make huge investments.

4.1.5 Branch organisations

Finally, logistics branch organisations are fighting for ambiguous policies and regulations among municipalities, while also taking the GDZES seriously. Furthermore, branch organisations stand for a safe working environment while they can also support sector-wide investments. Besides, thuiswinkel.org tries to make consumers more aware of their online shopping behaviour and make them behave more sustainable. Hence, the following criteria from branch organisations can be derived:

Table 4-5: Criteria branch organisations

Criterion	Part of the sentence that describes it
Emissions	Their commitment to GDZES describes their involvement in decreasing carbon emissions.
Safe working environment	They want to maintain a good and safe working environment.
Investments in Infrastructure	They can support the sector with investing in sustainable infrastructure solutions.
Policy sensitivity	They fight for ambiguous policies, which could also mean that they value a solution that does not have to deal with these policies at all (e.g. is not sensitive to current policies or proposed policies)

4.1.6 Final criteria

All stakeholders have to be valued equally to provide a sustainable solution in the sense that it suits everyone's desires. Therefore, most of the aforementioned criteria will be considered during the analysis. However, some of these criteria are partly overlapping each other, or have a direct effect on each other. Therefore, some collective criteria have been defined to cover these overlaps. This has resulted in the following list of criteria with an explanation for each criterion, which is shown in Table 4-6.

Table 4-6: Final criteria

Criterion	Explanation
Delivery Cost	Means either the costs for the actual delivery (3PL perspective) or the costs that customers have to pay. This can be measured in a percental increase or decrease.
Delivery Time	Means both the time it takes to deliver the parcel, but also refers to the options that the delivery time could be specified. This can be measured in a percental increase or decrease.
Emissions	Refers to the possibilities of reducing the GHG emissions. This can be measured in a percental increase or decrease.
Customer Satisfaction	Refers to the effects on the overall satisfaction of customers, in explicit, does it increase or decrease.
Safe Working Environment	Refers to the extent to which the alternatives mitigate risks at the working environment or increases risks for accidents.
Security and Responsibility	Refers to the perceived security of personal records and the product itself. Furthermore, the extent to which the responsibility in the chain of companies has to be redefined and the value that is attached to the responsibility is measured on an increase or decrease scale.
Traffic Impact	Refers to the extent to which the alternative is capable of reducing traffic in the city. This is done on a percental scale and refers to the number of traffic movements.
Investments in Infrastructure	Refers to the need for investments, which can either be in infrastructure or software. These are subjective estimates.
Policy Sensitivity	Refers to the extent to which current and intended policies can affect a certain alternative. If it is affected, then it is perceived as sensitive.

This list of criteria has been verified by the sustainability officer of PostNL as factors of interest to PostNL as well. One thing the sustainability officer mentioned though, is that the options for the circular economy are becoming increasingly important to the stakeholders. However, the alternatives assessed are not yet defined to handle the circular economy, and the analyses of the municipal elections do not show any interest in implementing circular economies either. Therefore, circularity is left out of the analysis, which is accepted by PostNL's sustainability officer.

4.2 Determining the weights of the criteria

Not all criteria are valued equally. Their importance in the analysis can differ, which can be noticed from the fifth step in the SMART method. This step tries to obtain a ranking that shows the relative importance that stakeholders attach to the criteria (Edwards, 1977). In this study, representatives of the stakeholder groups have been carefully selected, and have been asked to fill in a form. In this form, they were asked to rank the criteria with regard to the importance they thought all the stakeholder groups would attach. An example form is shown in Table 4-7.

Table 4-7: Stakeholder ranking form

	Delivery Cost	Delivery Time	Emissions reduction	Customer satisfaction	Safe working environment	Security and Responsibility	Traffic impact	Investments in infrastructure	Policy sensitivity
Consumers/recipients									
Suppliers									
Logistics providers									
Governments									
Branche organisations									

Within the company of PostNL there are plenty of people that work with one or more of the stakeholder groups on a daily basis. Some of them are the managers of the pilot projects (with regard to logistics), others are sales managers (for suppliers and consumers knowledge), public affairs managers (governments, branch organisations and consumers concerns) and sustainability managers (governments, branch organisations and consumers). Since the analysis is mainly meant to provide a focus for the simulation model, their knowledge seems sufficient to draw conclusions about the different criteria to take in consideration for the stakeholders.

Furthermore, since people from many different departments and thus different backgrounds have been asked to participate, many different perceptions have been included in the ranking procedure. By linking their opinions and perceptions to the researcher's own interpretation (based on the foregoing chapters), more accurate estimates on the trends can be made. Hence, the obtained overview of rankings provides a good representation of the situation and the fifth step of the SMART method has been completed (Edwards, 1977).

The normal procedure is to proceed with step six, where the stakeholders assign a relative importance to the previously obtained ranking. However, due to the high uncertainty in the ranking itself, assigning a relative importance would increase the differences even more, making it harder to distinguish any subtle differences. Hence, this study proceeds with step seven, where the weights get assigned directly by calculating the averages over the different stakeholders (Edwards, 1977).

By means of a python (jupyter) notebook, all the different perceptions have been combined. The notebook can be found in Appendix I. First, the so-called reciprocal values have been calculated, which is sophisticated English for *1/rank*. Then, the sum of the reciprocal values is taken, which in the case of Figure 4-1 is the sum over the columns. This is followed by calculating the normalised values for each criterion. If no rank had been assigned, the non-value was replaced by a zero. Finally, the average normalised value for each criterion/stakeholder combination was calculated. An example of the calculation procedure is shown in Table 4-8.

Table 4-8: Weights calculation example

	Delivery Cost	Delivery time	Emissions reduction	Customer satisfaction	Safe working environment	Security and responsibility	Traffic impact	Investments in infrastructure	Policy sensitivity	Sum
Consumers rank	1	1	4	2		3	4			
Consumers reciprocal	1	1	0,25	0,5		0,33333333	0,25			3,33333333
Consumers normalised	0,3	0,3	0,075	0,15	0	0,1	0,075	0	0	1

Based on the table that was obtained by the previous step of averaging the normalised values, the average weight for each criterion was calculated. Furthermore, the standard deviation was calculated to be able to generate different cases that take the standard deviation into account. Hence, the robustness of the alternatives can be measured among the different cases. First, the standard deviation of an alternative was subtracted from the average and the difference was added equally to the other criteria. Second, the standard deviation of an alternative was added to its average and the difference was added equally to the other criteria. That resulted in a table with many different cases with different weights for the criteria.

4.3 Assessing performance of the alternatives

The eighth step in the SMART method is about measuring the performance of each of the measures (or alternatives) regarding all the identified criteria (Edwards, 1977). The performances are gathered from both literature and expert interviews within PostNL, as not all dimensions were covered in literature. The expert interviews result in a higher uncertainty in the obtained performances and hence, the final analysis. However, as already mentioned, this analysis is only meant to structure the choice of delivery model that promises the most benefits, despite the high uncertainties. The final

model mitigates these uncertainties by calculating in more detail the effects the chosen alternative last mile delivery model has compared to the current delivery model.

This paragraph covers the performance of each alternative last mile delivery model as presented in Table 2-1 on page 17 separately, providing an overview of literature and statements from the interviews that support the numbers listed. Finally, a table with the numeric values that will be used in the analysis of the next paragraph, will be provided.

4.3.1 Electrification of the Vehicle Fleet

What would be the effect if, from tomorrow onwards, all vehicles will be replaced by electric equivalents? That is the question that was raised before analysing the performance of this alternative. This has not been done yet, so there is also not a lot of information available about this rigorous alternative. Hence, the analysis contains a lot of assumptions, based on educated guesses and have been well discussed.

Firstly, the assumption is that the *delivery costs* for a parcel will remain similar to what it is now. However, that is under the assumption that the investment costs for the EVs are not directly passed on to the customers. Furthermore, it is assumed that EVs have a similar performance to the conventional diesel vans and hence, *delivery time* will remain comparable.

EVs are sold with the message that they do not emit any CO_2, but that is only partly true. Depending on the way electricity is generated, the *carbon emissions* can be reduced drastically, noting a 30% decrease in emissions if grey electricity (coal) is used and 70% if a green source is used (Verbeek, Bolech, van Gijlswijk, & Spreen, 2015). Furthermore, this is based on a life cycle of 220.000 km, due to the emissions-intensive manufacturing process of the batteries.

Since there are no structural changes in the system of delivering the parcels, it is assumed that the *safety of the working environment* and the *perceived security and responsibility* are not changed either. Furthermore, the service remains the same, so it is assumed that the *customer satisfaction* will remain similar too. It may be that the customer satisfaction increases slightly though, since a (assumed) small share of customers values the efforts of a company to become sustainable.

The *traffic impact* is assumed to be similar to the current system with conventional vans. However, the range and capacity of the EVs should be similar, or fast charging facilities should be available at the distribution centre. If either capacity or range is smaller than the current vans, more vehicles will be needed to deliver the same amount of goods, leading to more traffic movements and hence, more traffic in the city.

The main drawback of switching to EVs is the need for *high investments* in both vehicles and in charging infrastructure. Not only the new vehicles have to be purchased, but the old fleet has to be written off, which is going to cost the companies a huge amount of money. It is assumed, that these costs will not be included in the price customers pay. Besides, there are subsidies for the purchase of vehicles or charging infrastructure in some big cities that will make it a little bit cheaper to change[1].

The main benefit of switching to EVs is that they are not sensitive for the differing policies among municipalities. The so-called *"Milieuzones"* do not apply for EVs, enabling PostNL for instance to still get into the city centres without hurdles. By combining all the foregoing statements, Table 4-9 is created for EVs:

Table 4-9: EVs Performance

Criterion	Description or score	Source
Delivery costs	Similar, meaning a 0% increase or decrease	Assumption
Delivery time	Similar, meaning a 0% increase or decrease	Assumption
Emissions reduction	Depending on generation of electricity, 30-70% decrease	(Verbeek, Bolech, van Gijlswijk, & Spreen, 2015)
Safe working environment	Similar, due to the lack of changes in the system	Assumption
Security and responsibility	Similar	Assumption
Customer satisfaction	Similar or a slight increase, due to sustainable image	Assumption
Traffic impact reduction	Similar or a slight increase, depending on capacity and range of the EVs	Assumption
Investments in infrastructure	Very high	Assumption and footnote
Policy sensitivity	Not sensitive, but depends on available subsidies	Assumption

4.3.2 (E-)Cargo bikes

(E-)Cargo bikes (further referred to as cargo bikes) are increasingly being used by 3PL providers like PostNL and DHL to deliver parcels in busy cities (PostNL, 2017). According to PostNL (2017), cargo bikes are able to reduce *delivery costs* by 8-10% (Nijhuis, 2018). Moreover, according to other research, the total costs could be reduced by 40% (Arnold, Cardenas, Sörensen, & Dewulf, 2018).

[1] See https://www.amsterdam.nl/parkeren-verkeer/amsterdam-elektrisch/subsidie/ and https://www.laadkabelwinkel.nl/over-ons/subsidie-laadpaal-thuis

The *delivery time* is highly dependent on the location of the delivery points in the city, but Nijhuis provided an estimation that delivery time could be reduced by 50% in heavily congested city centres. However, delivering parcels by cargo bikes that depart from the current depots is unfeasible, since the depots are too far from the city centres in most cases (Nijhuis, 2018). Therefore, Maes and Vanelslander (2012) chose to deliver the parcels from small depots in the city that are replenished by delivery vans. The last mile of the last mile is then conducted by cargo bike. This alternative seems feasible but is assumed to only have a slight impact on the *emissions*. Nijhuis provided an estimation of 10%, but that is highly dependent on the share of cargo bike deliveries and the amount of delivery points in the city.

Furthermore, cargo bikes are presumed to be more vulnerable than vans for accidents. Besides, accidents are more likely to lead to bigger injuries as a result. Hence, the *safety of the working environment* is assumed to decrease. Since the deliveries are still conducted by the same company and only the modality has changed, the perceived *product security and responsibility* is assumed to be similar.

According to Nijhuis, *customer satisfaction* increases by implementing cargo bikes as a delivery option. Customers respond surprised but satisfied by the efforts of PostNL to provide more sustainable delivery options. Furthermore, the reliability of the delivery time and the flexibility in time slots increases, which has a positive effect on customer satisfaction as well (Nijhuis, 2018).

Depending on the point of view one takes at *traffic reduction*, traffic could be drastically reduced, or traffic could increase. By applying cargo bikes, traffic on the roads for cars reduce However, for each delivery van two bicycles are needed, which also depends on the density of delivery points in the city (Arnold, Cardenas, Sörensen, & Dewulf, 2018). Hence, bicycle paths are likely to be flooded by cargo bikes. However, the focus in this study is the reduction of traffic on roads for cars, so this effect is left out of consideration in this part of the analysis. Nijhuis assumed that in the best case, 50% reduction of car traffic is achievable.

Compared to the other alternatives, investment costs are relatively low. Cargo bikes only require smaller depots (but a higher number of depots), and their price is way less than an electric van. Furthermore, cargo bikes are insensitive for policies in the city, for instance the "*milieuzones*" do not apply for cargo bikes. Hence, Table 4-10 can be created, summarising the performance of cargo bikes.

Table 4-10: Performance cargo bikes

Criterion	Description	Source
Delivery costs	10-40% reduction possible, depending on implementation	Arnold et al. (2018), Nijhuis (2018)
Delivery time	In the best case, 50% reduction is feasible	Maes and Vanelslander (2012) Nijhuis (2018)
Emissions reduction	Depending on the implementation, approximately 10%	Maes and Vanelslander (2012) Nijhuis (2018)
Safe working environment	Decreases due to the increased risk of harmful accidents	Assumption, Nijhuis (2018)
Security and responsibility	Perceived similar, as the same parties conduct deliveries	Assumption, Nijhuis (2018)
Customer satisfaction	Increases due to increased flexibility	Maes and Vanelslander (2012) Nijhuis (2018)
Traffic impact reduction	Strongly decreases, depending on the point of view. Assumed to be 50% reduced	Assumption, Nijhuis (2018)
Investments in infrastructure	Relatively low	Assumption, Nijhuis (2018)
Policy sensitivity	Insensitive	Assumption, Nijhuis (2018)

4.3.3 Evening- and Night delivery

Evening and night delivery are still mentioned in one sentence, as they are quite alike. However, according to Hünteler (2018), there are structural differences if the performance of the two systems are analysed. Evening delivery is conducted in less available time, since PostNL only conducts deliveries between 18:00h and 22:00h. In contrast, night delivery is designed to be operational all night, where parcels are being delivered in personal parcel lockers next to the front door. Hence, recipients do not have to be awake to receive the parcel, and more benefits can be obtained. Therefore, the two options have now been analysed apart, which has resulted in two separate tables.

Firstly, evening delivery as it is currently done by PostNL is 50-100% more expensive than their operations at daylight (Hünteler, 2018). This is mainly due to the lower volume and the premium price that has to be paid for the service (Hünteler, 2018). *Delivery time* could be reduced by approximately 10-25%, depending on the used modality and the traffic situation in the city. In contrast, according to McKinsey and Company (2017), night delivery is capable of cutting *delivery costs* by 40% and *delivery time* by 50%.

Due to the increased efficiency in both the evening and night deliveries, *emissions* could in potential be reduced by 70% (McKinsey & Company, 2017). PostNL admits not to have correct numbers for the possible emission reductions, but they state that 70% seems a little high for the current implementation of evening delivery. Hence, a possible reduction of 0-50% is assumed.

Since traffic is a lot less intense in the evenings compared to daytime, it is assumed that the risks for accidents decrease. However, this is depending on the modality used to conduct the evening and night deliveries. Hence, it is assumed that the *safety of the working environment* slightly increases by applying either night or evening deliveries.

Since the customer still has to physically accept the parcel with the current implementation of evening delivery, it is assumed that the *security and responsibility* factor is perceived similar compared to the current system. However, it is assumed that for night delivery, customers act a little more reserved and require more safety from their parcel locker at home. Hence, the security and responsibility factor is assumed to decrease for the night delivery alternative.

As these alternatives can both be linked to an increase in flexibility regarding delivery options, it is assumed that *customer satisfaction* will increase. Furthermore, the first hit rate is likely to increase, which has a positive effect on the traffic impact. It is assumed that night delivery is capable of reducing traffic in the city by 50%. However, evening delivery is not that efficient and has not the same amount of time. Hence, completely shifting towards evening delivery is expected to have a catastrophic effect on the traffic system (Hünteler, 2018).

The *investments in infrastructure* that are needed differ among the two alternatives. If night delivery is the way to go, huge investments have to made into parcel lockers for the customers that are secure enough. In contrast, the investments for evening delivery highly depend on the scale of implementation. If the scale is increased, more depots and vehicles are needed to be capable of delivering the demand within the four-hour time window. Therefore, a wide range of options is considered for both the alternatives, from small investments to very high investments.

Lastly, both the alternatives are assumed to be *insensitive* for the current *policies* in the cities. The only aspect that could cause trouble in the future is the emission of sound by the current diesel-powered vans. As they drive through the streets by night, their sound levels become more noticeable, people can start complaining and problems may arise. However, that is of future concern.

Table 4-11: Performance evening delivery

Criterion	Description	Source
Delivery costs	Increase by 50-100%	Hünteler (2018)
Delivery time	Decrease by 10-25%	Hünteler (2018)
Emissions reduction	Could possibly decrease by 0-50%	Assumption
Safe working environment	Slight increase, but it depends on the modality	Assumption, Hünteler (2018)
Security and responsibility	Similar	Assumption, Hünteler (2018)
Customer satisfaction	Slightly increases, due to the increase in flexibility	Hünteler (2018)
Traffic impact reduction	Depends on the scale, but it will only increase	Hünteler (2018)
Investments in infrastructure	Small to very big investments, depending on scale	Assumption, Hünteler (2018)
Policy sensitivity	Insensitive, until sound emissions become a problem	Assumption

Table 4-12: Performance night delivery

Criterion	Description	Source
Delivery costs	40% reduction possible	McKinsey and Company (2017)
Delivery time	50% reduction	McKinsey and Company (2017)
Emissions reduction	Could reduce 70% of the emissions in the best case	McKinsey and Company (2017), Hünteler (2018)
Safe working environment	Could slightly increase	Assumption
Security and responsibility	Decreases, as it may feel strange not to accept the parcel in person	Assumption
Customer satisfaction	Could either increase or decrease, depending on the perceived security	Assumption
Traffic impact reduction	Could reduce by 50% in the best case	Assumption, Hünteler (2018)
Investments in infrastructure	High investments needed for lockers	Assumption, Hünteler (2018)
Policy sensitivity	Insensitive	Assumption

4.3.4 UCCs

As already mentioned, UCCs have become a popular city logistics alternative to implement by municipalities. Furthermore, during the Dutch municipal elections of 2018, UCCs were a major point of discussion in the big cities. According to McKinsey and Company (2017), UCCs could cut *delivery costs* by 25%. *Delivery time* is assumed to be similar, or in a good case slightly decreased compared to the current system, due to the extra step in the chain. However, due to more efficient modes of transport in the city, it has the potential to decrease the delivery time considerably (Bakr, 2018). Hence, numbers between 0-25% have been applied.

Since it is such a popular alternative, this was one of the only systems that has been studied a few times, mostly on the *reduction of emissions.* If only conventional vehicles are applied, emissions could be cut by 20% (Clausen, Geiger, & Poting, 2016). However, the definition of a UCC is that it transhipps goods to green transport modes and hence, if only green transport modes are applied, emissions could be cut by 100%. Therefore, a range of 20-100% is applied, to find out at what reduction the UCC becomes an attractive alternative.

The safety of the working environment is slightly compromised, as there is an extra step in the process. The risk of damaging products during transhipments increases and since the volumes handled by a UCC are presumed to be high and the available space is limited (Bakr, 2018). Hence, it is assumed that the safety of the working environment slightly decreases. *Security and responsibility* is also an issue, as the UCC is meant to consolidate shipments from multiple transporters as well. Hence, the question about which party is responsible for the parcel in what step of the chain remains an open end in the discussion. As this is still work in progress and not yet defined for all UCCs run by PostNL, it is left in the middle. So, it neither increases or decreases.

By implementing a UCC, more flexible delivery options can be offered, as transport from this UCC is better equipped for city delivery. Hence, it is assumed that customer satisfaction increases and the company image improves (Bakr, 2018). Furthermore, as green transportation modes are applied to transport goods from the UCC, the traffic impact is likely to decrease. According to the report of McKinsey and Company (2017), about 45% of the kilometres driven in the city could be reduced.

However, a UCC comes with a cost. New facilities are needed at the expensive ground near the city centre. Furthermore, *investments* in IT-systems that enable all the companies to work together are needed, as this is not a common thing yet. Therefore, it is assumed that the investment costs are high for the involved companies. The height of the investment costs is highly dependent on the policies and subsidies of cities. As mentioned before, many municipalities express their interest in UCCs and are willing to cooperate. If they are also willing to invest, the business case for a 3PL may

become stronger. Hence, it is assumed that a UCC is highly *sensitive to policies*. Based on the information above, Table 4-13 is constructed.

Table 4-13: Performance of UCC

Criterion	Description	Source
Delivery costs	Reduced by 25%	McKinsey and Company (2017)
Delivery time	Reduced by 0-25%	Assumption, Bakr (2018)
Emissions reduction	Could be reduced by 20-100%	Clausen, Geiger & Pöting (2016), Bakr (2018)
Safe working environment	Similar to slightly decreasing	Assumption, Bakr (2018)
Security and responsibility	Similar, due to the remaining discussions	Assumption, Bakr (2018)
Customer satisfaction	Increases due to the better street image	Assumption, Bakr (2018)
Traffic impact reduction	Cut by 45%	McKinsey and Company (2017)
Investments in infrastructure	High to very high investments, depending on who invests	Assumption, Bakr (2018)
Policy sensitivity	Highly sensitive	Assumption, Bakr (2018)

4.3.5 CDPs and Parcel Lockers

Customers are increasingly demanding more flexible delivery options. One of the alternatives that definitely offers more flexibility, is a CDP or a parcel locker. There is one important difference though, CDPs of PostNL are retail location and customers are bound to opening hours to pick up or drop off their parcel (Smit, 2018). However, a parcel locker is located at a public place that can be used 24/7. The logistics concept could be combined however, enabling the 3PL to deliver the parcels with a (almost) 100% hit rate at whatever time they like.

Firstly, *delivery costs* could be reduced by 35% (McKinsey & Company, 2017). This is under the assumption that the investment costs are not directly passed on to the customer. Besides, the *delivery time* and the time a delivery man is handling the package can be reduced by 70% (McKinsey & Company, 2017). However, the customer has to pick-up the parcel himself, which also costs some time. This is however left out of the delivery time.

By implementing the CDPs and parcel lockers as a more convenient delivery option, there is aen *emissions reduction* potential of 70% (McKinsey & Company, 2017). However, this is only seen through the 3PL provider's perspective and does not consider the movement of the customer to the parcel locker or retail location. The net reduction could therefore be a little lower, but this is out of the scope of PostNL (Smit, 2018).

Since the operator saves a lot of time in the city, there is less time available for hazardous situations. Hence, it is assumed that implementing CDPs and parcel lockers has a positive effect on the *safety of the working environment*. However, the *perceived security and responsibility* at the parcel locker location could be reduced, which highly depends on the location of the parcel locker. Furthermore, it is still a question if customers trust the parcel locker system. According to Smit (2018), customers are still concerned about criminals breaking in to the lockers. Who is then responsible?

Despite the reduced perceived security, *customer satisfaction* is increasing (Smit, 2018). There is however no insight in the effects of completely changing the delivery options to only CDPs and parcel lockers. Since 89% of the customers still chooses for home delivery, while only 16% chooses (or sometimes chooses) for parcel lockers, some resistance is to be expected (Lowe & Rigby, 2013). Hence, customer satisfaction is assumed to slightly increase, only if customers are able to choose for their option.

The *traffic impact* of this alternative is estimated at a 40% reduction of kilometres driven in the city centres. However, the activities of the customer are left out of the assumption. Hence, the traffic impact is highly depending on the modality the customer uses to pick-up his parcel.

The *investments in infrastructure* that are needed are highly dependent on what is needed, as CDPs require way less money than placing parcel lockers throughout the city. Hence, a range between low till high investments has been applied in the analysis of 4.4. Furthermore, the same reasoning applies for the *policy sensitivity*, as CDPs are not influenced by policies, while parcel lockers that are placed in public areas have to comply with certain regulations (Smit, 2018).

Table 4-14: Performance CDPs and parcel lockers

Criterion	Description	Source
Delivery costs	Reduced by 35%	McKinsey and Company (2017)
Delivery time	Cut by 70%	McKinsey and Company (2017)
Emissions reduction	Potentially reduced by 70%	McKinsey and Company (2017)
Safe working environment	Increased safety of the working environment	Assumption, Smit (2018)
Security and responsibility	Could slightly reduce	Assumption, Smit (2018)
Customer satisfaction	Could either slightly decrease or increase	Assumption, Smit (2018)
Traffic impact reduction	40% reduction	McKinsey & Company (2017)
Investments in infrastructure	Can both be low and high investments, depending on implementation	Assumption, Smit (2018)
Policy sensitivity	Depending on implementation, low or high sensitivity	Assumption, Smit (2018)

4.3.6 Crowdsourcing logistics services

Crowdsourcing is getting an increased amount of attention in both research and in business, with companies like Uber and Airbnb it is more visible than ever. Also, in logistics, crowdsourcing is developing itself as a promising alternative for the current last mile problem. In theory, crowdsourcing logistics services for the last mile could lead to a 25% reduction in *delivery costs* (McKinsey & Company, 2017). However, depending on the applied system, delivery costs could increase if the fine system of Kafle et al. (2017) is used. Therefore, a range between a 0 till 25% reduction is applied in the analysis.

Kafle et al. (2017) furthermore state that the *delivery time* could be comparable, or even increase, compared to the conventional home deliveries. Hence, no points are assigned for a possible reduction in delivery time. However, depending on the provider of the service, the used modality and the chosen crowdsourcing system, emissions could be cut by approximately 30% (McKinsey & Company, 2017).

The *working environment* with regards to the normal delivery man increases in safety, as approximately less kilometres are needed in the city and the risk of accidents decreases. However, *product security and responsibility* is perceived worse, since it is unclear which party is responsible for the product after it is handed over to the crowd-worker. Furthermore, the risk of damages increases. Furthermore, if this model is forced into operation and customers are obliged to choose for delivery by a crowd-worker, *customer satisfaction* is assumed to decrease.

The *traffic impact* depends, as most of the other criteria for this alternative, on the implementation of the system and the modality that is used by the crowd-worker. There are possibilities to reduce the traffic impact by 24-30% (McKinsey & Company, 2017; Wang, Zhang, Liu, Shen, & Hay Lee, 2016).

Besides, the assumption is made that implementing a crowdsourcing system requires a moderate investment in a new IT-platform, since no changes in vehicle fleet or other infrastructure are needed. Since the sharing economy is currently under the policy-making microscope, it is furthermore assumed that the crowdsourcing alternative is currently highly sensitive for the coming policies. Hence, a crowdsourcing alternative should be well overthought before actually implementing it. Its overall performance has been used to construct Table 4-15.

Table 4-15: Performance crowdsourcing logistics services

Criterion	Description	Source
Delivery costs	Could reduce by 0-25%	McKinsey and Company (2017), Kafle et al. (2016), Wang et al. (2016)
Delivery time	Should be similar, or may take slightly longer, depending on crowd-worker	Kafle et al. (2016)
Emissions reduction	Possible reduction of 30%	McKinsey and Company (2017)
Safe working environment	Slightly increased safety due to less kilometres in the city	Assumption
Security and responsibility	Decreases due to risks of damage by crowd-workers	Assumption
Customer satisfaction	Could decrease	Assumption
Traffic impact reduction	About 24-30%	Wang et al. (2016), McKinsey and Company (2017
Investments in infrastructure	Moderate investments in IT	Assumption
Policy sensitivity	Highly sensitive	Assumption

4.3.7 Constructing the inputs table

As the performance of all the alternatives is known, a numeric table has to be constructed in order to conduct the analysis. For this table, a few remarks have to made. Firstly, for some criteria a numeric scale has already been used in the sense of the reduction percentage. This means that the higher the percentage, the better the score. The percentages are translated to numbers between 0 and 1. Secondly, some criteria are very rough estimates, like that the safety will increase. For the decrease-increase scale there is also a numeric scale that is used. If, for instance, the objective is

to have the lowest possible investments, then "no investments" scores 1, while "very high investments" scores 0 points. Furthermore, the scores will be divided over a five-points scale. This is shown in more detail in Table 4-16.

Table 4-16: 5-points scale scores

Target:					
As low as possible	Nothing 1	Small 0.75	Moderate 0.5	High 0.25	Very high 0
Should decrease	Strongly decrease 1	Slightly decrease 0.75	Similar 0.5	Slightly increase 0.25	Strongly increase 0
Should increase	Strongly increase 1	Slightly increase 0.75	Similar 0.5	Slightly decrease 0.25	Strongly decrease 0

As many scores from the previously defined alternatives fall within certain ranges, it is hard to assign one specific number for the test. Therefore, the Best-Worst method is applied, which defines worst, average and best-case scenarios for each of the alternatives (Rezaei, 2015). By means of this method, more consistent and reliable results can be obtained, while also respecting the uncertainty ranges from the previous chapters. The cases are shown in Table 4-17. How and why the different cases are defined the way they are has been further explained in Appendix II.

Table 4-17: Performances input table

Alternative / Criterion	EVs	Cargo bikes	CDPs and lockers	UCCs	Evening delivery	Night Delivery	Crowd-sourcing
Costs	0; 0; 0	0.1; 0.1; 0.1	0.35; 0.35; 0.35	0.25; 0.25; 0.25	0; 0; 0	0.4; 0.4; 0.4	0; 0; 0
Time	0; 0; 0	0.2; 0.35; 0.5	0.7; 0.7; 0.7	0; 0.125; 0.25	0.1; 0.175; 0.25	0.5; 0.5; 0.5	0; 0; 0
Emissions	0.3; 0.5; 0.7	0.1; 0.2; 0.3	0.4; 0.55; 0.6	0.2; 0.6; 1	0.25; 0.375; 0.5	0.7; 0.7; 0.7	0; 0.15; 0.3
Customer satisfaction	0.5; 0.5; 0.5	1; 1; 1	0.75; 0.75; 0.75	1; 1; 1	0.5; 0.625; 0.75	0.5; 0.5; 0.5	0.25; 0.25; 0.25
Safety at work	0.5; 0.5; 0.5	0; 0; 0	0.5; 0.5; 0.5	0.25; 0.25; 0.25	0.75; 0.75; 0.75	0.5; 0.625; 0.75	0.75; 0.75; 0.75
Security & responsibility	0.5; 0.5; 0.5	0; 0; 0	0; 0.125; 0.25	0.5; 0.5; 0.5	0.75; 0.75; 0.75	0; 0; 0	0; 0; 0
Traffic impact	0; 0; 0	0; 0.25; 0.5	0.2; 0.3; 0.4	0.45; 0.45; 0.45	0; 0; 0	0.25; 0.375; 0.5	0.2; 0.25; 0.3
Investments infrastructure	0; 0; 0	0.75; 0.75; 0.75	0.25; 0.5; 0.75	0; 0.125; 0.25	0; 0; 0	0; 0; 0	0.25; 0.375; 0.5
Policy sensitivity	0.5; 0.75; 1	1; 1	0.5; 0.625; 0.75	0.25; 0.375; 0.5	1; 1; 1	0.25; 0.25; 0.25	0; 0; 0

4.4 Conducting the analysis

Now, the analysis can be conducted, which is according to the ninth step of the SMART method (Edwards, 1977). The so-called utility of each alternative has been calculated and the alternatives have been ranked accordingly. The higher the utility, the better the alternative fits the criteria of the stakeholders. The corresponding Python notebook can be found in Appendix III.

The notebook takes the separate Excel-sheets of the best-, intermediate- worst case as inputs, together with the sheet that contains the cases with different weights for the criteria. By multiplying the score with the weight, a utility is calculated. The total utility of an alternative is the sum of the utilities for every criterion. This calculation is done for every case separately and hence, a dictionary with outcomes for all cases is returned.

The outcomes of each individual case have then been used as input for calculating the average scores of the alternatives and their standard deviations. For all different kind of weights for the criteria, it now becomes clear which alternatives perform better than others. The outcomes of the averages and standard deviations can easily be plotted by Python. The code yields three different plots: one for

the best scenarios, one for the intermediate scenario and one for the worst scenario. These plots are shown in Figure 4-3, Figure 4-4 and Figure 4-5.

Figure 4-1: Analyse the average utilities

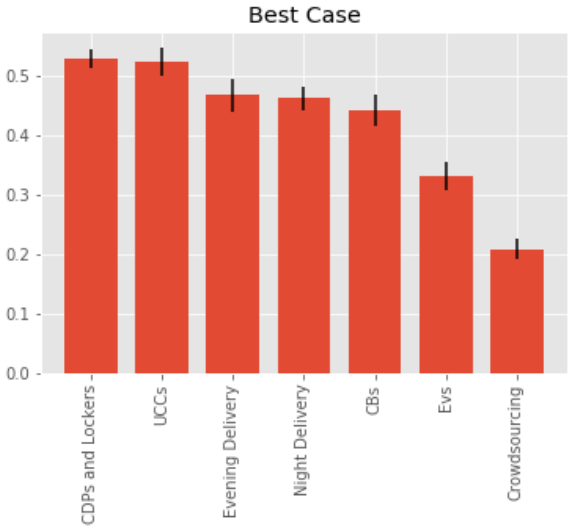


Figure 4-2: Best case scores

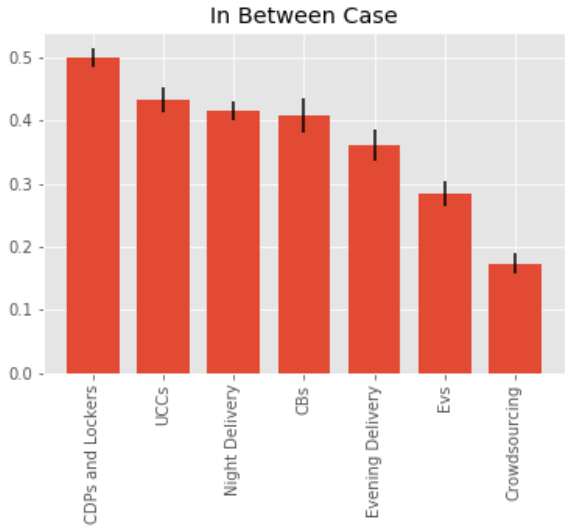


Figure 4-3: Intermediate case scores

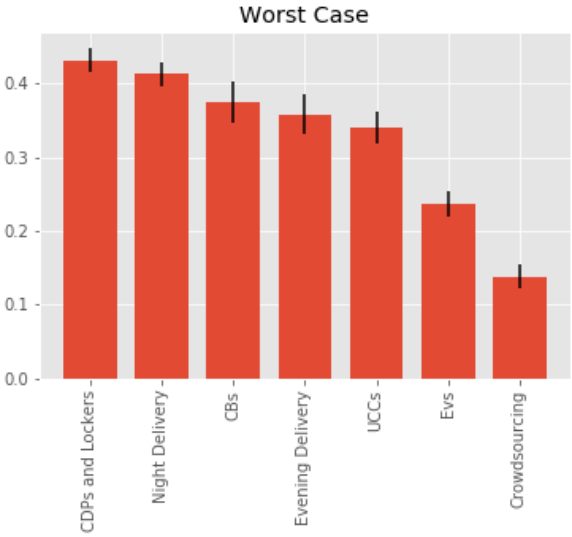


Figure 4-4: Worst case scores

The outcome is that CDPs and parcel lockers are the most promising alternative in all possible cases. Another alternative that is performing very well, is the UCC. However, there are large differences between the best- and worst case regarding UCCs. This is due to the large range that is applied for the emissions reduction. The combination of a large range and a high weight cause the total utility to differ quite a lot. Lastly, night delivery performs steady among the different cases, which is mainly due to a lack of applied ranges in the analysis. Furthermore, it cannot be seen as a dominant alternative, since cargo bikes and evening delivery have a very comparable score for all cases.

4.5 Conclusions

From Figure 4-3, Figure 4-4 and Figure 4-5 it can be noticed that the CDPs and parcel lockers are dominant in all defined scenarios. However, looking more closely to the numbers, it has to be pointed out that it is only dominating by a very small margin. Hence, a combination between CDPs and parcel lockers, UCCs and night deliveries however, seems to provide a more solid base for sustainable city logistics. This is following the statement of McKinsey and Company (2017) that a combination of alternatives is probably leading to more benefits than drawbacks.

UCCs perform bad in comparison to the other alternatives in the worst case. The worst-case data is coming from studies that have analysed the performance of UCCs that applied conventional vehicles for their deliveries. The 20% reduction in emissions is not enough to compete with the other alternatives. Therefore, the UCC in the remainder of this study has to apply sustainable vehicles like cargo bikes, stints and electric vans for all deliveries. However, as can be noticed from the foregoing analysis, especially EVs (vans) are not competitive and their application should be limited to a minimum.

Looking back to the fourth sub-question that is being answered in this chapter: *"What are the most promising last mile delivery alternatives, given the sustainability requirements and perceptions of the stakeholders?"*, the answer is that a combination between home deliveries and deliveries to a CDP or parcel locker from a UCC is taken as base for the remainder of this study. Besides, night deliveries can be combined with the replenishment of parcel lockers, since McKinsey and Company's night delivery alternative is not yet feasible for large scale adoption.

5 | Modelling sustainable city logistics

So, a more sustainable last mile delivery process at least contains a more robust and present parcel locker infrastructure, a city hub for consolidating the shipments of multiple transporters and offers options for night deliveries. However, insight is still lacking how these alternatives all work together, and what would be the combined benefits if they are applied for the parcel delivery sector? That is why a simulation model has been created, to get more insight in the effects of these combinations for companies like PostNL.

According to White and Ingalls (2009), a model is a simplified representation of reality for systems of interest. These models can be used for representing systems that only exist in concept, are expensive to implement and test for the outcomes. Hence, simulation models provide opportunities to aid decision-makers with insights in variables that affect the modelled system, by generating outcomes for different settings of the input variables. The simulation model of this study is created following the Sargent modelling cycle (Sargent, 2010). The aim of the Sargent cycle is to create more valid simulation model, in explicit, models that actually answer the questions decision-makers have. The cycle consists of the following steps: Conceptualisation, Specification, Model Building, Verification, Validation, Experimentation.

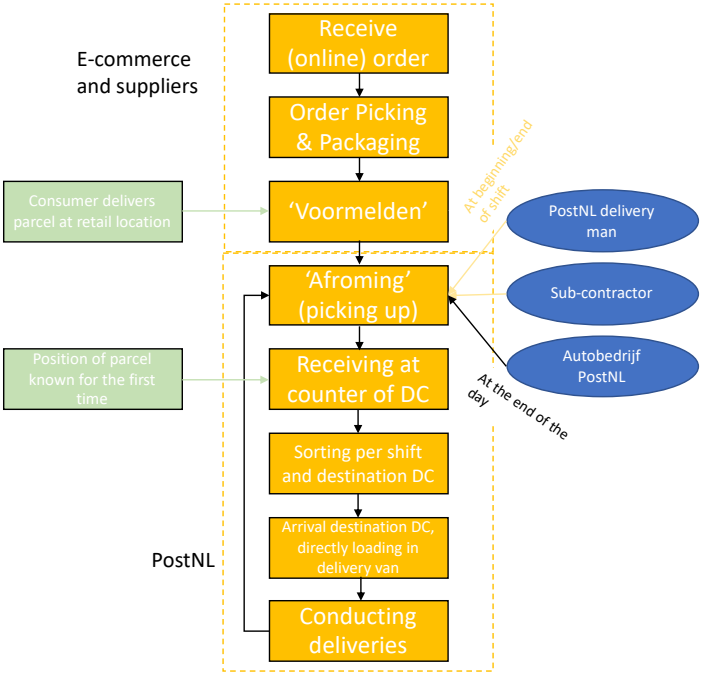
Conceptualisation is all about trying to model the problem situation and the variables that affect this situation. *Specification* is a detailed software description of how the simulation software is going to work. After this, the *Simulation Model* can be built. This building phase is then followed by the *Verification* of the model, which is an iterative approach of testing whether or not the model works right and according to the rules. If the model finally works properly and the complete specification has been incorporated, the model can be *Validated.* In explicit, does the model do the right things according to its stakeholders? Finally, *Experiments* can be conducted to investigate the model behaviour for different settings of the input variables.

Each of the phases of Sargent's (2010) modelling cycle will be covered in the following paragraphs, in the same order as they are explained above.

5.1 Conceptualisation

The simulation model that is created in this chapter has to provide more insight in the performance of the parcel delivery system in the city of Amsterdam when multiple new alternative systems are being combined. To get these insights, certain parts of the original process as shown in Figure 5-1 have to be modelled. The purpose of this paragraph is to determine a scope for the model to prevent the simulation model to become overcomplicated.

The process as shown in Figure 5-1 incorporates all actions conducted by both e-commerce and other suppliers and PostNL. Since the alternative systems only describe the logistical processes, the actions related to e-commerce can be left out of the simulation. The simulated process therefore only has to describe the actions as taken by PostNL or any other transporting company.



Furthermore, the processes of collection and sorting are different depending on the transporting company under investigation. Therefore, the simulation model excludes these processes and starts with the parcels arriving at the sorting centre close to the city under study.

Normally at PostNL, parcels are sorted per shift and drivers have the freedom to decide what parcels to include in their route. When it is really busy and a lot of parcels have to be delivered, PostNL also offers the option to sort the parcels per route. Despite it being unusual to sort the parcels per route, the simulation model has to calculate a route for the parcels in advance of the simulation, since it has to be known if the route fits within all the limits. Moreover, there is no driver that can make this calculation in the simulation, so the system has to decide. How the route planning exactly works, is explained in further detail in 5.2.

Besides, PostNL drivers have the freedom to decide when to collect parcels at the collection points. There are several of these points within their routes, but whether they collect the parcels at the beginning or the end of the route is up to them. However, most of the drivers choose to collect the parcels at the end of the route, since their vehicles are empty and most of the parcels will fit. Similar to the delivery of parcels, the simulation does not contain drivers that can make these decisions and therefore, collection is included in the route planning. Furthermore, the collection is added to the end of each route so there will always be enough space in the vehicle to store the planned parcels. Hence, the process to model can be summarised as in Figure 5-2.

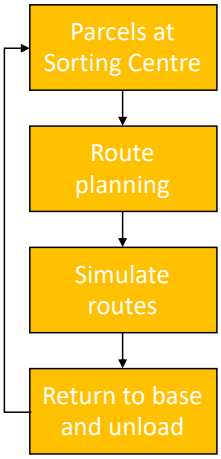


Figure 5-2 provides more insight in the process to translate into a simulation model. However, little insight is yet being provided in the variables that influence this process and therefore have to be included in the simulation. The simulation model is further being presented as a *black box* model and the remainder of this paragraph focusses on the information going in and the information going out of this black box. Besides, the user can have certain control over the behaviour of the model by means of settings, which will be defined in this paragraph too. The black box model is presented in Figure 5-3.

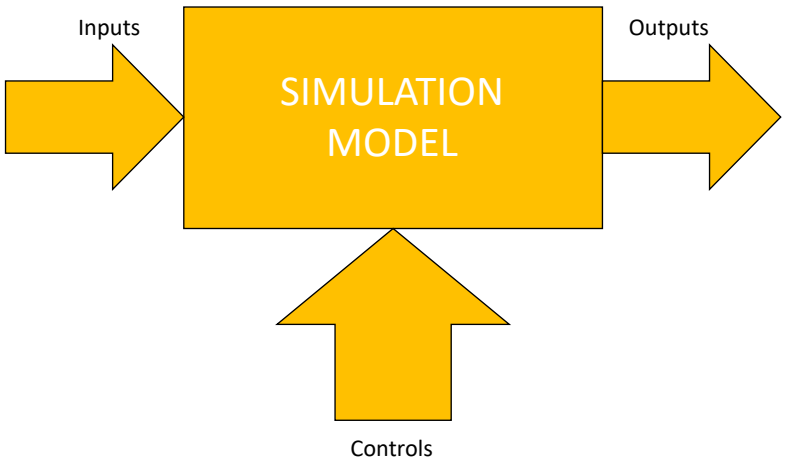


Figure 5-3: Black box representation simulation model

5.1.1 Determining the simulation model inputs

Before the simulation model can actually simulate the parcel delivery process in the city of Amsterdam, it requires some information in advance. What this information entails, is further explained in this paragraph. Beginning with the most obvious data that the simulation model requires: the geographical data of Amsterdam. These data could be postal codes, postal code areas or road networks for instance. It is decided to only use the postal code areas (4 digits). A rough level of detail for individual parcel deliveries, but it is expected that otherwise the model will become too extensive to conduct experiments with. Thus, the simulation model requires shapefiles of the city of Amsterdam, dividing the city into the postal code areas[2].

Next, the model requires demographic data of Amsterdam's population. At first it was assumed that the distribution of inhabitants among the postal code areas was a good predictor for the demand of parcels to deliver. This assumption has been confirmed by analysing PostNL data of parcel deliveries conducted in Amsterdam, indicating that the distribution of delivered parcels is significantly similar to the distribution of inhabitants (see Appendix IV for detailed analysis). Therefore, the distribution of inhabitants is still being used as a predictor of parcel delivery demand and treated as input for the simulation model.

[2] Retrieved from: http://geoplaza.vu.nl/data/dataset/postcode

In order to determine the starting point of the vehicles from where the deliveries can be conducted, the geographical locations of the transporters' sorting centres have to be known. PostNL's location of its sorting centre near Amsterdam is known within the company, but for the other transporting companies Google Maps has been used to find out where their sorting centres near Amsterdam are. It is assumed that their distribution centres near Amsterdam are also responsible for the deliveries in Amsterdam, but more research is needed to really be sure from where they conduct deliveries. For now, these locations are good to provide insight in the effect of the city hub and are therefore being useful, despite the uncertainty if they are chosen right.

Following up on locations, the simulation model needs the locations of the parcel lockers in the city if these are being used. The locker locations are fixed and can only be changed outside of the simulation model to ensure that always the same locations are being used. However, if the user desires to change the number of lockers, it could be achieved by changing some of the control variables. This is explained in more detail in 5.1.2.

Following up on the number of transporting companies to include in the analysis, it is important to know their market share. In explicit: for what share of the generated demand is the respective company responsible to deliver? Based on a report from the Dutch Authority for Consumer and Market (ACM), Table 5-1 is created with the market shares for each of the parcel delivery companies.

Table 5-1: Market shares of transporters (ACM, 2016)

Transporter	Market Share
PostNL	65%
DHL	25%
DPD	5%
GLS	4%
UPS	1% (raw estimate)

However, before the demand can be split up among the five companies defined in Table 5-1, an indication for the average demand the system has to deliver should be provided. By analysing the PostNL data about parcel delivery in Amsterdam, the model will be tuned for a demand between 25,000 and 50,000 parcels per day (see Appendix IV for detailed analysis).

Finally, the model takes the vehicle data from the different companies as input. This can be divided into two different categories: the vehicle fleet and the individual vehicle data. In explicit, the vehicle fleet provides the model with information about how many vehicles a certain company has, while the individual data about the vehicle provides data about emissions, range and capacity for instance.

Based on the provided inputs for the simulation model, Figure 5-3 can be expanded as presented in Figure 5-4. In Figure 5-4, the inputs are incorporated in the black box model.

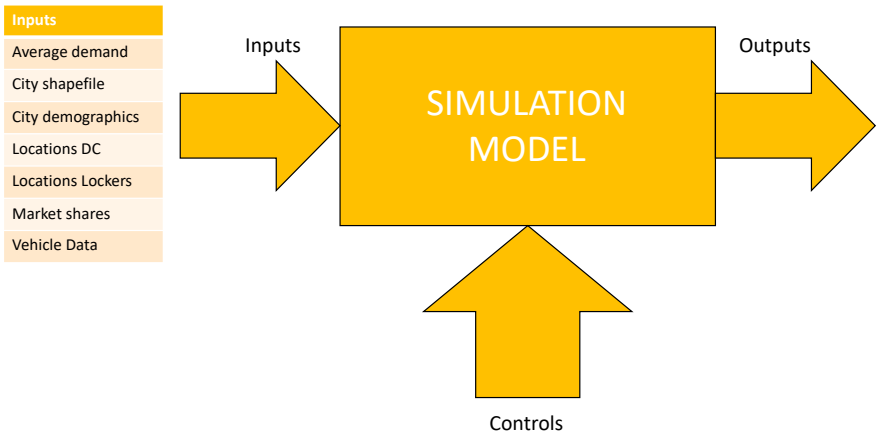


Figure 5-4: Extended black box model inputs

5.1.2 Determining the simulation model outputs

Before determining the model outputs, it is important to reflect on the purpose of the simulation model. As mentioned in 1.4.5, the simulation model is created to provide more insight in the sustainable performance of the alternative last mile delivery systems combined. Therefore, the simulation model outputs have to be related to the sustainability criteria as defined in 4.1. From the list of criteria, the traffic impact, transport costs and transport time per parcel are perceived as most important for stakeholders. Furthermore, these criteria are easy to calculate numerically, so what indicators could be used to reflect on these criteria?

Firstly, the distance travelled in total and the number of kilometres per parcel are seen as important outcomes of the simulation model, since the traffic impact of parcel delivery was one of the main motivations to conduct this study. Secondly, the total time needed in sense of vehicle use is taken as outcome of interest, as each vehicle needs an employee to drive it. Moreover, employee-related cost is one of the most important factors of the delivery costs, so this outcome could support the reflection on delivery costs. Lastly, GHG emissions of the system are perceived as an outcome of interest, since GHG-emissions reduction is a good indicator for environmental sustainability. Hence, Figure 5-4 can be expanded with the model outputs as shown in Figure 5-5.

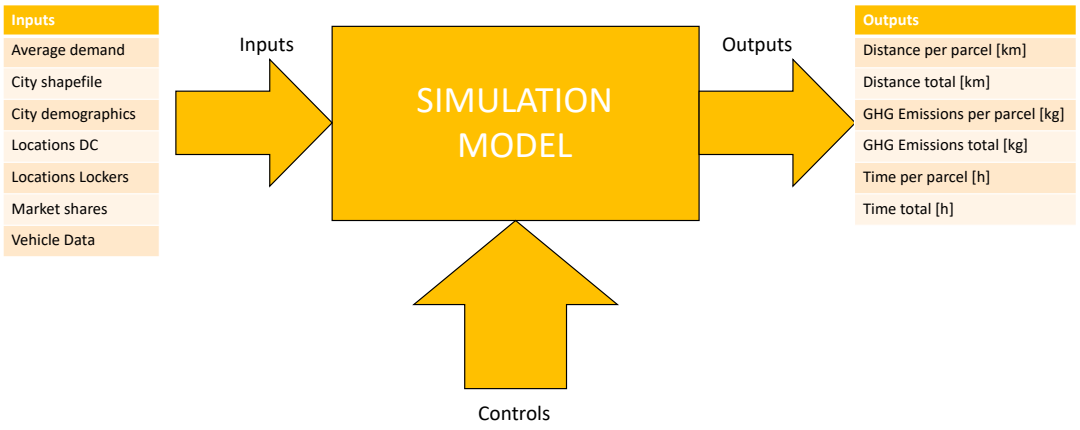


Figure 5-5: Extended black box model outputs

5.1.3 Determining the simulation model controls

Since the simulation model is used to compare different alternative last mile delivery systems, it is important to enable the user to choose between different configurations. Hence, the simulation model is provided with certain control variables that are controllable by the user before the simulation run is initiated. For instance, the user has the freedom to decide the time horizon for which the simulation will be run by means of the control variable *'Simulation Time Limit'*. By default, it is set to 24 hours.

Furthermore, the configurations can be changed by flipping a switch, so the existing system can be combined with the city hub, or only with the parcel lockers infrastructure, or both the city hub and parcel lockers infrastructure, it is up to the user. Besides, the user has the freedom to choose between two city hub configurations: one that only delivers parcels inside the *'Milieuzone'* of the city, or in all of the city. Lastly, the user has the possibility to change the number of transporting companies included in the simulation, to see whether or not the number of companies affects the outcomes. Hence, Figure 5-5 can be expanded by including these controls in the figure, shown in Figure 5-6.

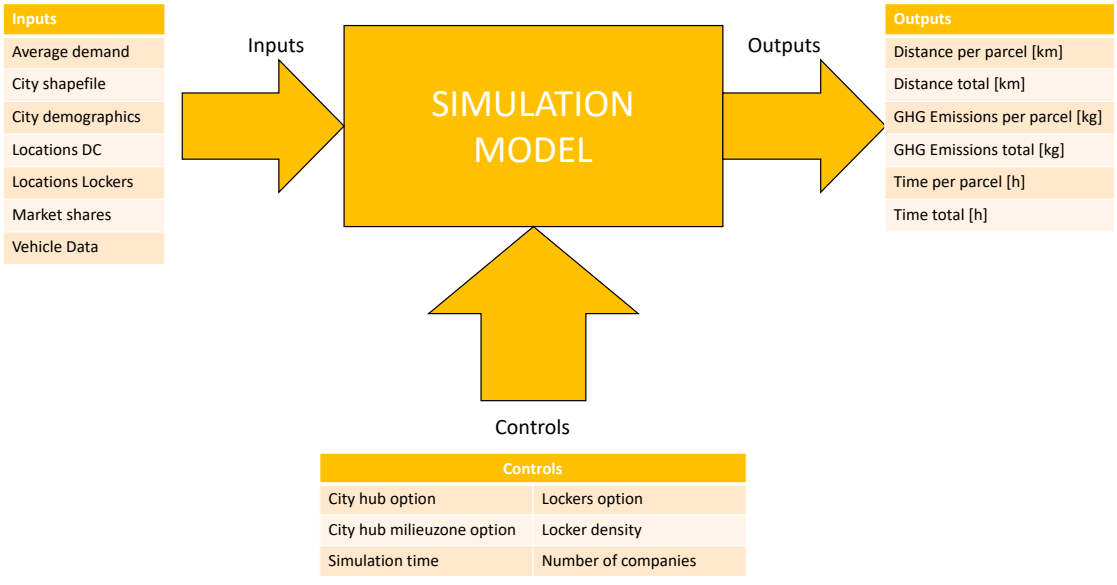


Figure 5-6: Expanded black box model controls

5.2 Specification

As already decided in 1.4.5, the model will be developed according to the Discrete-Event Simulation (DES) modelling 'language' (Behiri, Belmokhtar-Berraf, & Chu, 2018; Simoni & Claudel, 2018; White & Ingalls, 2009). DES describes a system's behaviour over time by means of a series of events that occur (Allen, et al., 2015; Tako & Robinson, 2012). These events are triggered by entities that are flowing through the system. All these entities in the system are defined and assessed individually and all have unique properties (Tako & Robinson, 2012). Besides, discrete event models use statistical distributions to generate randomness in the events, which makes it an interesting technique for processes with high uncertainties regarding the variables.

This paragraph discusses the specifications of the simulation with regard to the DES modelling language. Furthermore, the black box as presented in Figure 5-6 is now opened to see how the model is going to work underneath. There is a wide variety of software packages to conduct DES experiments, but in 5.2.1 it is explained why the simulation model is written entirely in Python. Then, the classes needed to conduct the simulation are defined in 5.2.2. Finally, some more comprehensive diagrams of the processes and activities the simulation model will simulate have been described in 5.2.3.

5.2.1 DES in Python

Of course, the market offers software solutions to companies, universities and enthusiasts that like to create DES models from scratch. Examples are Simio[3] and Arena[4], both extensive and expensive software packages. The advantage of such a software package is that it often offers a clear graphical user interface, which usually makes it more pleasant and easier for the user to work with. Furthermore, in Simio for instance, it is possible to create nice visualisations while running the model, which makes it easy to spot mistakes and give more meaning to the results.

However, visualisations require more computing power than plain programming. Hence, conducting experiments with these models can be time consuming. Furthermore, one has to completely understand the syntax (the details of how to use the software, which buttons to use) before being able to create robust models. This itself is not the problem, but these software companies exist due to the provided courses on how to use their software. Hardly any information is for free. Hence, plain programming in any other language seems way more attractive, as this is often open-source.

The world of programming languages consists of many different and unique options. Probably the most used, or most well-known languages are C, C++, Java, Python and R, of which each of them has their own advantages and disadvantages. All of the programming languages have in common that

[3] https://www.simio.com/index.php
[4] https://www.arenasimulation.com/

they are cross-platform (so can be used on Mac, Windows and Linux most of the time), open-source (information is freely available) and offer almost unlimited freedom.

Instead of making a comparison between all of the available programming languages and list their advantages and disadvantages, the motivation of why to choose for the Python programming language is provided. Python is arguably the fastest growing major programming language out there (Robinson, 2017), which makes it almost trending on forums like StackOverflow, a website where programmers meet to solve each other's problems. Hence, it is easy to find solutions for problems encountered when programming in Python. Furthermore, Python is known as a very easy language to understand, easy to read syntax and a lot of libraries that can-do amazing stuff. All these libraries are easy to find and install with Python's own package manager (pip).

As Python is one of the most used programming languages taught in the curriculum of the Engineering and Policy Analysis master, and the skills have already been mastered before the start of the thesis project, the choice was easy. Besides, in order to make a comprehensive simulation model as intended by this study, a heavy-duty computer is needed to run the experiments in a considerable time when using Simio (which was the software package taught in the curriculum). A similar model in plain Python is expected to run considerably faster, which makes it of course more attractive to run for multiple experiments. One argument against the use of Python would again be 'speed', as it is considerably slower than other languages, according to various internet sources. However, these other languages are way harder to read, understand and program from scratch. Hence, Python is the language of choice to program the DES-model for this study.

5.2.2 Class diagrams of Python model

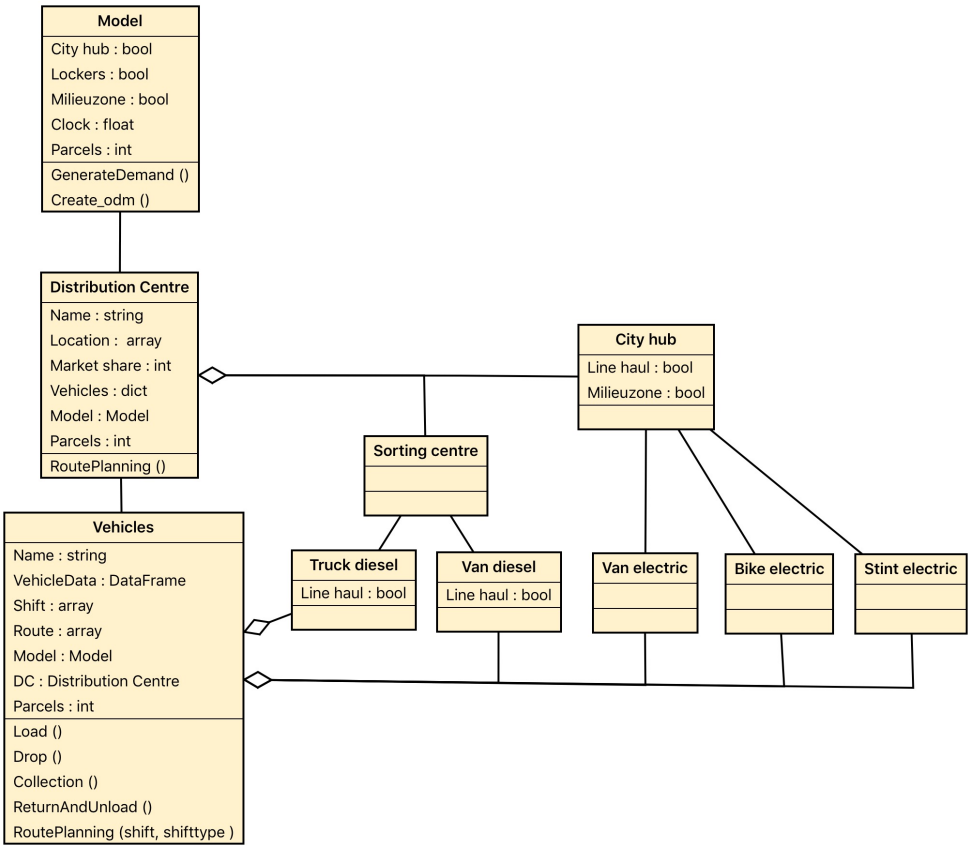
The simulation model is defined as a *Class* in Python, which can be instantiated when intended to be used. Within Python classes, variables can be defined as a part of the class, making them accessible by all the different functions within and outside the class. For simplicity, the simulation model class is called *Model* in Figure 5-7 and only the most important properties are shown. The properties of the class are mostly the settings for the configurations, like the *City hub*, *Lockers* and the *Milieuzone.* Furthermore, the Model class stores the *Clock* variable that keeps track of the simulation time and the *Parcels* variable that provides insight in all the parcels that are currently available in the system.

The model also has certain operations, or Python *Functions*, attached to it. Firstly, the Model class generates the demand of parcels to be delivered on a daily basis, with the *GenerateDemand()* operation. This will be generated for each of the *Distribution Centre* instances individually. If a city hub configuration is used, all demand is consolidated at the city hub and transported by a line haul to the city hub. An Origin-Destination Matrix (ODM) is generated for each *Distribution Centre* instance by executing the *Create_odm()* operation (Ekowicaksono, Bukhari, & Aman, 2016).

As mentioned above, the Model consists of (multiple) instances of the *Distribution Centre* class. Each distribution centre instance is unique and is defined based on the number of transporting companies to include in the simulation. The most important properties of this class are the *Name* of the instance, its *Location*, its *Market share*, the *Vehicles* in the fleet, a reference to the *Model* class and the *Parcels* to be delivered. Then, after demand has been generated, the distribution centre instance initiates the *RoutePlanning()* operation, which calls the route planning operation for each vehicle in its fleet.

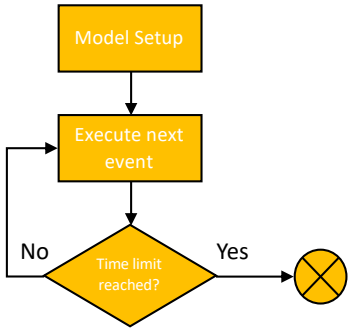
As already mentioned, the Distribution Centre class contains (multiple) instances of the *Vehicles* class. Vehicles take care of the transportation of parcels between their distribution centre and the destinations and there are five different types of vehicles defined. Important properties of the *Vehicles* class are the *Name*, the unique *Vehicle data*, the *Shift* that the vehicle is planned to operate, a list of *Routes*, a reference to the *Model* and *Distribution Centre*, and the *Parcels* loaded onto the vehicle. Furthermore, the *Truck diesel* and the *Van diesel* instances can be used for a line haul service, if a city hub configuration is used.

The *Vehicle* class has the most operations from all classes, as they are executing all the work. First, the *Load()* operation, where the parcels for the planned route are loaded onto the vehicle. The following event is *Drop()*, where the parcels are dropped at the next destination. *Collection()* takes care of picking up the parcels, while *ReturnAndUnload()* handles the picked up parcels and unloads them at the *Distribution Centre.* Besides, following the *Distribution Centre* class, the vehicle has a *RoutePlanning()* operation that generates the routes within the limits of the vehicle. The complete process is explained in more detail in 5.2.3.

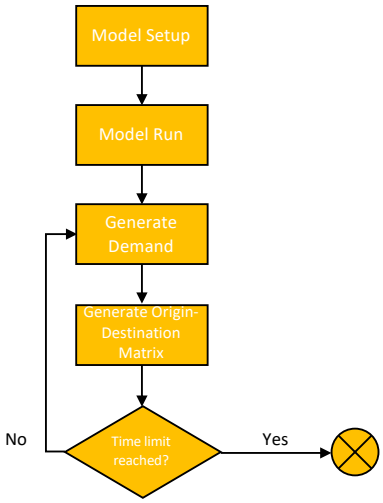


5.2.3 Process diagrams of Python model

The processes executed during the simulation have been summarised in so-called flowcharts. These processes use the classes' operations as defined in Figure 5-7 in a certain way. Starting with the model class, the general process that is being executed is shown in Figure 5-8. In explicit, all events of the different instances are stored centrally and the model executes these sequentially. So, as long as the simulation time limit has not been exceeded, the model looks for the next scheduled event and executes the respective operation. Looking at the operations that are only available to the Model class itself, Figure 5-9 can be created. As long as the simulation time limit has not been exceeded, the model generates demand and the ODM at the scheduled times. The Python code can be found in Appendix V.



The operations of the Distribution Centres class are even simpler than the model, as there is only one operation inside this class: *RoutePlanning()*. As mentioned before, the route planning gets called right after the demand has been generated and the ODM has been created and will continue regardless of the simulation time being exceeded. Hence, the flowchart is defined as shown in Figure 5-10. The Python code can be found in Appendix VI.



Finally, the process of the *Vehicle* class is explained in more detail, the process that mainly steers the simulation. The route planning operation gets called by the route planning operation of the distribution centre the vehicle belongs to. The route planning is based on the parcels that have to be delivered, so as long as there are parcels that are not included in a route and no vehicle limit has been exceeded, the route planning operation is executed. Then, vehicles are loaded with the planned number of parcels for the route and the parcels are dropped sequentially, as long as there are parcels left to drop. After dropping all parcels, the parcel to collect can be collected. Finally, if the route has been completed, the vehicle returns to the distribution centre and unloads the collected parcels. This sequence of steps is repeated as long as the vehicle has routes planned. If all routes are completed, the shift of the vehicle is updated to the next day. This process is summarised in the flowchart of Figure 5-11. The Python code can be found in Appendix VII.

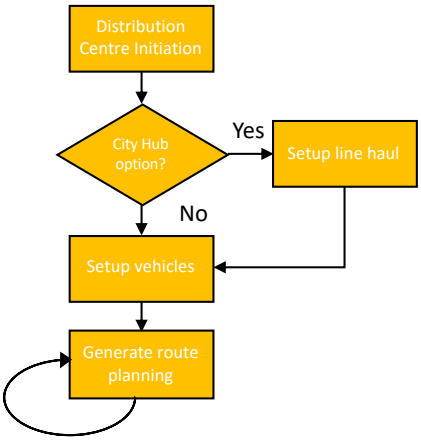
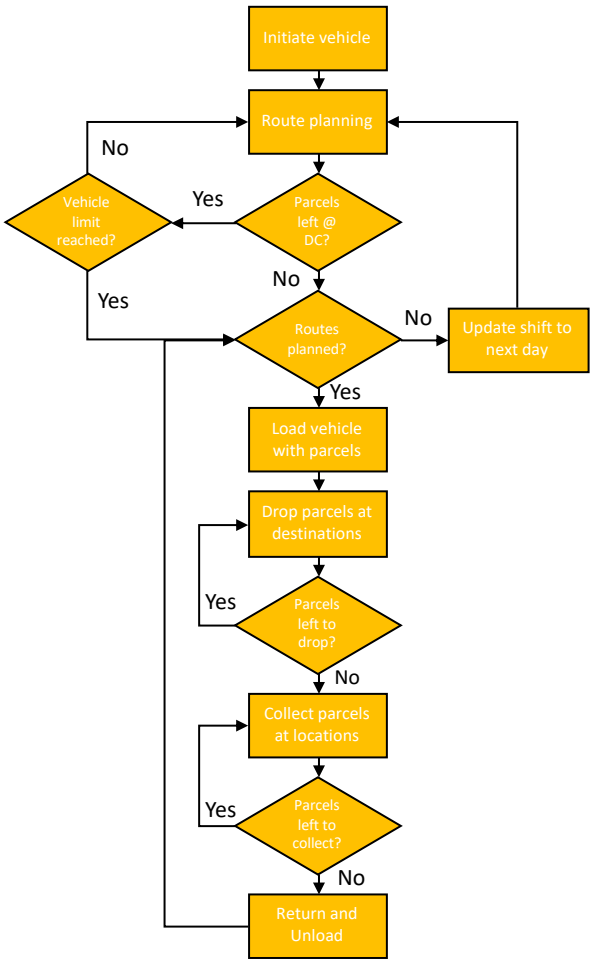


Figure 5-8

5.2.4 Calculating the output variables

As mentioned in the conceptualisation of 5.1, the simulation should yield at least six different outcomes. In this paragraph, the calculations of each of these outcomes are explained in further detail.



Firstly, the route planning already provides insight in the expected number of kilometres that will be driven. However, due to unforeseen events, this actual distance travelled might differ after the simulation (a detour for instance). Therefore, the travelled distance during the simulation will be measured for each route from the time the parcels get loaded onto the vehicle. When the vehicle returns, the number of kilometres the vehicle has driven gets updated by the distance of the route. After the simulation, the total distance travelled gets updated by assessing all of the vehicles. In order to calculate the distance per parcel, the total number of kilometres is divided by the total number of parcels delivered. The total number of parcels delivered also includes the collected parcels that are dropped off at the sorting centre.

Secondly, the GHG emissions. The vehicle data as entered in the Model class contain at least the number for the average emission of CO_2 per kilometre travelled for each type of vehicle and the average emission of NO_x/PM_{10} per kilometre travelled. The emissions are updated similarly to the kilometres travelled per route and can thereby be easily calculated by multiplying the distance travelled by the average emissions per kilometre. Furthermore, after the simulation time limit has been exceeded, the total emissions are assessed from all vehicles. To calculate the emissions per parcel, the total emissions are divided by the total number of parcels delivered.

Lastly, the time consumed for the delivery of parcels is calculated by keeping track of the starting time and ending time of a particular route. Hence, only the time a vehicle is operative is considered in the analysis. Then, after the simulation time limit has been exceeded, the total time is calculated by assessing the time operative for each individual vehicle. Furthermore, the operative time per parcel is calculated by dividing the total time by the total number of parcels delivered.

5.3 Simulation model

As mentioned in the foregoing paragraphs, the simulation model is written entirely in Python and only uses some external files like the shapefiles for instance. However, this paragraph tries to explain how the model is stored and setup among different files and scripts, so a follow-up study can easily start from this point onwards. First, the parcel locker locations are set-up by a separate script, which is covered in 5.3.1. Then, the model setup script is discussed in 5.3.2, followed by a brief description of the model usage in 5.3.3. Finally, a brief explanation of the model dashboard is provided in 5.3.4.

5.3.1 Parcel locker setup

Initially, the parcel locker locations are distributed randomly across the city of Amsterdam, based on the values entered in the model dashboard (see 5.3.4) regarding the willingness to walk a certain time to a locker and the average walking speed. Hence, the average number of parcel lockers to yield a desired average distance in-between these lockers can be calculated and can be randomly distributed among the postal code areas. The algorithm used for the distribution of lockers among the postal code areas is further explained in Appendix VIII and can be found in the *setup_lockers.py* file.

The algorithm for distributing the parcel lockers as mentioned above takes quite some calculation time. Therefore, setting up these parcel locker locations is done in advance of setting up the simulation model. Furthermore, in order to make a reliable comparison between different configurations, it is important that the parcel lockers' locations remain the same, so all configurations use the same parcel locker infrastructure. To see the effects of more or less parcel lockers, the setup script has to run again to store the new parcel locker locations. The parcel locker locations are stored in a file called *'LockerLocations.xlsx'*, and is called each time a Model-instance is created.

5.3.2 Simulation model setup

The model setup script is stored in a separate file than the model script itself, so the code of the model script stays clean and tidy. The model setup script is called each time a Model instance is created. The setup script then reads the required files into memory and stores them in the right formats. More explicitly, the setup script processes and stores the shapefile, the data-file about Amsterdam, the parcel locker locations, the *'Milieuzone'*-area and the dashboard file (see 5.3.4), and returns this to the model instance. The files can be found in the *Data*-directory attached to the *Model*-directory. The Python code can be found in Appendix IX and in the file called *setup.py*.

5.3.3 Simulation model script

The simulation model script is created to be used to instantiate and run the simulation model and can be found in the *Model.py* file. Furthermore, settings can be changed, so different experiments can be executed (see 5.6 for further explanation). The simulation model script calls the setup script in the initiation step, resulting in a defined set of variables that have been processed in advance. Then, based on the dashboard, the configuration of the model is determined, and the model is ready to run.

The model can be run by calling the *Model.run()* function. The run function executes the *Model.next_event()* function as long as the simulation time limit has not been exceeded (see Figure 5-8). Afterwards, the model returns the results of the outcomes of interest: *Parcels Delivered*, *Distance Travelled*, *Operative Time* and *Emissions*. The outcomes can be both printed on the screen, or stored in an Excel file for instance, for further analysis.

5.3.4 Model dashboard

The model dashboard has been referred to a few times before in this report. The model dashboard is an Excel file that contains all the input variables and controls as defined in 5.1 to influence the simulation model's behaviour. Moreover, the dashboard can be seen as the user interface of the simulation model, the easiest way to change the settings and variables in the Model. However, changes have to be made before the model is instantiated, otherwise it should be done via a Python script.

The model dashboard is stored in the *Data*-directory as *Dashboard.xlsx*. It contains sheets with *Basic Information*, *Vehicle Information*, *Vehicle Fleets*, *Locations*, and *Settings*. The basic information sheet is mainly filled with data about the simulation time, demand, maximum route times and shift times. Also, the share of demand to deliver in parcel lockers and in the evening can be set in this sheet. Vehicle information contains all info about the individual vehicles, like their range, capacity and average stop time for instance. Vehicle fleets lists all vehicles a certain location as defined in the Locations sheet owns. Lastly, the Settings sheet can be used to change the configuration of the model. In explicit, the city hub and parcel locker configurations can be switched on or off.

5.4 Verification

Creating the simulation model is one thing, checking whether or not the model works correctly is second. Hence, verification is defined as *'the evaluation whether or not the product or service complies with regulations, requirements and specifications'* (IEEE, 2013). In contrast to validation, verification looks to the inside of the simulation model to check the correctness. Hence, a few different approaches have been discussed in this paragraph.

First, one parcel is traced through the system, with different configurations, to check whether or not the parcel has been delivered and no vehicle got stuck. Second, an extreme number of parcels is entered in the dashboard, to see whether or not vehicles get stuck transporting the amount, or that parcels just keep waiting until they are being transported. Lastly, a sensitivity test with extreme values for other simulation variables has been conducted to see whether or not the simulation models' behaviour or outcomes change extraordinary.

5.4.1 Single parcel trace

The single parcel trace verification test is conducted twice, one time without the city hub implemented, and one time with the city hub implementation. From these two tests, it can be concluded that the parcel was delivered correctly two times and that the outcomes match the expectations. Moreover, the outcomes only show the results of one vehicle being utilised to deliver one parcel to one destination. All other vehicles were on standby, ready to be used the next day. The results of both tests are shown in Figure 5-12 and Figure 5-13.

```
---------------> RESULTS OF THE SIMULATION <---------------
---> HUB option is False
---> LOCKER option is False
---> LOCKER percentage is 10
---> EVENING percentage is 10
---> Included forwarders: ['PostNL', 'DHL', 'DPD']
Total simulated time [h]:       24.00
Parcels in system [pcs]:        1.00
Parcels delivered [pcs]:        1.00
Total distance driven [km]:     43.94
Distance per parcel [km]:       43.94
Total time operative [h]:       1.43
Average time/parcel [h]:        1.43
Total emissions CO2 [kg]:       5493.10
CO2 emissions/parcel [kg]:      5493.10
Total emissions NOx [kg]:       23.95
NOx emissions/parcel [kg]:      23.95
```

Figure 5-12: 1 parcel trace, no city hub

```
---------------> RESULTS OF THE SIMULATION <---------------
---> HUB option is True
---> LOCKER option is False
---> LOCKER percentage is 10
---> EVENING percentage is 10
---> Included forwarders: ['PostNL', 'DHL', 'DPD']
Total simulated time [h]:       24.00
Parcels in system [pcs]:        1.00
Parcels delivered [pcs]:        1.00
Total distance driven [km]:     55.93
Distance per parcel [km]:       55.93
Total time operative [h]:       97.81
Average time/parcel [h]:        97.81
Total emissions CO2 [kg]:       12273.56
CO2 emissions/parcel [kg]:      12273.56
Total emissions NOx [kg]:       136.87
NOx emissions/parcel [kg]:      136.87
```

Figure 5-13: 1 parcel trace, 1 city hub

NOTE: The unit for the emissions in the figure above should be grams, not kilograms.

5.4.2 Extreme number of parcels

The extreme number of parcels test is also conducted twice, the first time without the city hub implemented and the second time with a city hub implemented. Just like the single parcel trace test, the extreme number of parcels test yields promising results, as all vehicles are fully utilised in both the day and evening shifts and conduct all possible deliveries. Parcels that did not fit in the planning, as the current capacity is too low, are still in the system waiting for the next day to be delivered. The only not so realistic part of this simulation is that in the normal configuration (without city hub), the diesel trucks are also planned to deliver parcels. This almost never happens in the real world.

```
---------------> RESULTS OF THE SIMULATION <---------------
---> HUB option is False
---> LOCKER option is False
---> LOCKER percentage is 10
---> EVENING percentage is 10
---> Included forwarders: ['PostNL', 'DHL', 'DPD']
Total simulated time [h]:       24.00
Parcels in system [pcs]:        91773.00
Parcels delivered [pcs]:        33627.00
Total distance driven [km]:     10081.13
Distance per parcel [km]:       0.30
Total time operative [h]:       1969.46
Average time/parcel [h]:        0.06
Total emissions CO2 [kg]:       1823972.78
CO2 emissions/parcel [kg]:      54.24
Total emissions NOx [kg]:       9543.09
NOx emissions/parcel [kg]:      0.28
```

Figure 5-14: Extreme numbers, no city hub

```
---------------> RESULTS OF THE SIMULATION <---------------
---> HUB option is True
---> LOCKER option is False
---> LOCKER percentage is 10
---> EVENING percentage is 10
---> Included forwarders: ['PostNL', 'DHL', 'DPD']
Total simulated time [h]:       24.00
Parcels in system [pcs]:        68982.00
Parcels delivered [pcs]:        56418.00
Total distance driven [km]:     20401.26
Distance per parcel [km]:       0.36
Total time operative [h]:       3499.25
Average time/parcel [h]:        0.06
Total emissions CO2 [kg]:       1880630.26
CO2 emissions/parcel [kg]:      33.33
Total emissions NOx [kg]:       11013.69
NOx emissions/parcel [kg]:      0.20
```

Figure 5-15: Extreme numbers, city hub

NOTE: If the number of parcels is too large, there is a risk that the computer returns a memory error.

5.4.3 Extreme values for simulation variables

The extensive sensitivity test of the model can be found in Appendix X. This paragraph only discusses the most outstanding results from this analysis. Firstly, it turned out that changing the number of parcels to deliver per day heavily influences the outcomes of the simulation. By an approximate 90% decrease in the number of parcels, the distance and time per parcel increase by about 350% (estimated), indicating that the results are sensitive to the number of parcels entered. Furthermore, the higher the number of parcels entered in the dashboard, the longer the simulation takes to complete.

The next variable that showed surprising effects on the outcomes, was the capacity of the vehicles. By increasing the capacity with 50%, the distance per parcel can be reduced by 16% and the time per parcel by approximately 2.5%. Besides, decreasing the capacity by 50% leads to a 21% increase in distance per parcel and about 5% increase in time. However, this is only true for the system including a city hub. For the small vehicles used, the capacity is usually the limiting factor and an increase mitigates these drawbacks. However, in the normal system, neither increasing nor decreasing the capacity heavily influences the distance and time per parcel.

Lastly, the number of transporters included in the analysis can heavily affect the simulation outcomes. For all the foregoing analyses, three transporting companies have been included (PostNL, DHL and DPD). The first effect that can be noticed after including more transporting companies into the analysis, is that the total number of kilometres decreases when the city hub gets implemented. Moreover, this effect gets stronger when multiple smaller companies with only a few destinations are included. For the transporters available in the dashboard file however, the maximum benefit that can be achieved by implementing a city hub is only about 2% distance and time per parcel.

5.4.4 Conclusion verification

Concluding the paragraph, it can be stated that the model is verified for further use in the analysis. The model shows predictable behaviour under extreme conditions without failures. Tracing one parcel was successful in multiple configurations, while an extremely high number of parcels resulted in a large number of parcels that could not be included in a route planning. These parcels were stored to be delivered the next day. Under all other extreme value tests, the model also showed predictable behaviour and hence, the model is verified for further use in this study.

5.5 Validation

Since the model is built upon many assumptions and based on a Euclidian ‘road’ network, the simulation results can only be indicative. Analysing real-world data from PostNL to validate the simulation model would yield high inaccuracies and therefore, the model is validated by conducting expert interviews. The interviewees were experts from PostNL.

The conceptual model has been validated by interviewing the experts from PostNL. They all confirmed the correctness of the process, despite some assumptions that make it easier to model. The simulation model provides PostNL sufficient insight in the performance of the alternative delivery systems and thereby, the model is valid to use for further experimentation to reach the conclusions for this study.

The only noted drawbacks of this simulation model are the night delivery not being implemented yet and neither is the obligation for PostNL to pick-up a parcel the same day is not an obligation in the simulation model. Both these issues would over-complicate the simulation model in its current state, which is the main reason why they have not been implemented. However, these issues do not have a big effect on the outcomes, as can be noticed from the sensitivity analysis in Appendix X.

5.6 Experimentation

The last step in Sargent's modelling cycle is to set-up experiments to conduct with the simulation model. These experiments have been defined with focus on the last mile delivery alternatives. In explicit, by conducting the experiments this study aims at providing more insight in how the chosen alternatives combined yield the most promising outcomes. This is done by changing One Factor At the Time (OFAT), since it is expected that this results in sufficient insights with less time needed for the design of experiments (Kleijnen, 2001). Hence, Table 5-2 is constructed, containing all different cases.

Table 5-2: Simulation experiments

Case name	City Hub option	Lockers option - Lockers percentage	Evening percentage
Case 0	False	False	10
Case 1	False	True - 10	10
Case 2	False	True - 30	10
Case 3	False	True - 50	10
Case 4	False	True - 70	10
Case 5	False	True - 10	30
Case 6	False	True - 10	50
Case 7	False	True - 40	50
Case 8	True	False	10
Case 9	True	True - 10	10
Case 10	True	True - 30	10
Case 11	True	True - 50	10
Case 12	True	True - 70	10
Case 13	True	True - 10	30
Case 14	True	True - 10	50
Case 15	True	True - 40	50

5.6.1 Calculate the required number of experiments

The base case (Case 0) has been executed 10 times in advance of the analysis. With the outcomes of interest, an estimation about the preferred number of replications can be made (van Soest, 1992). For the total distance, total time operative and the emissions, van Soest's formula is applied:

$$n = \frac{n_{test\ run} * \sigma}{0.5 * \max(test\ run) * 0.05}$$

In the formula, n is the desired number of replications, $n_{test\ run}$ is the number of replications done for the test run, σ equals the half width confidence interval of the set and *max (test run)* represents the maximum value of the variable in the set. Van Soest's formula has been applied for the total distance, total time operative and the total CO_2-emissions of the outcomes of the base case test runs. Based on the maximum CO_2-emissions and its standard deviation from the average in the set, it can be stated that the following experiment need at least 5 experiments to yield the desired accuracy (van Soest, 1992). The calculation is shown below, the complete procedure can be found in Appendix XI.

$$n = \frac{10 * 25{,}789}{0.5 * 2{,}179{,}608 * 0{,}05} = 4.7$$

5.6.2 Some further notifications on the configurations

Despite the cases as defined in Table 5-2 being fairly straightforward, some further notifications about some assumptions have to be made. Firstly, the city hub configuration is based on the requirement to shift to smaller vehicles. Therefore, the focus of the city hub is on cargo bikes and stints, and electric vans are used when all other vehicles are utilised. Secondly, all the cases with a parcel locker infrastructure use the same reference file that contains the locations. Furthermore, in the file all lockers are placed within 500 metres of distance between each other. It is assumed that this is a reasonable distance, based on the willingness to walk and the average walking speed. The calculation and distribution of lockers is further explained in Appendix VIII. Lastly, the experiments will all use the same daily demand of 27000 parcels.

5.7 Conclusion

So, it is possible to create a DES-model for the parcel delivery system in Amsterdam that works according to the processes of PostNL. Furthermore, the simulation model provides reliable, yet indicative, data about the performance costs and benefits that can be obtained by implementing several of the last mile delivery alternatives as chosen in the fourth chapter. The outcomes of the experiments will show the real costs and benefits however, which will be discussed in the next chapter. Hence, it is yet impossible to answer the fifth sub-question of this study: *What benefits can be obtained by implementing one of these alternatives?* The following chapter will provide the answer.

6 | Analysing simulation results

The experiments as defined in Table 5-2 have been conducted. Furthermore, based on the outcome of van Soest's formula, each case has been run at least five times to ensure the desired accuracy of the outcomes (van Soest, 1992). The first paragraph analyses the results to find out which of the alternatives influences the system's behaviour most beneficial. Then, the settings used can be analysed with regard to the sustainability factors as defined in paragraph 4.1. Finally, some conclusions have been drawn and the fifth sub-question can be answered.

6.1 Analysis

As mentioned in 5.6, the first case (Case 0) is defined as the base case. In explicit, the base case is not using a city hub nor parcel lockers and about 10% of the daily demand is delivered in the evening. The performance of all the other cases is compared with the base case to see whether or not performance gains can be obtained by implementing the delivery alternatives in the defined way. In order to easily notice the differences in performance, the percental differences are shown in Figure 6-1. Furthermore, positive outcomes, or percental decreases, are highlighted in green, while negative outcomes, or percental increases, are highlighted in red. The analysis can be found in Appendix XIII.

Table 6-1: Percental outcomes of the experiments

Variable	Total distance [km]	Distance/ parcel [km]	Total time [h]	Time / parcel [h]	Emissions CO2 [g]	Emissions CO2/parcel [g]
Case 0	100	100	100	100	100	100
Case 1	95	95	95	94	92	92
Case 2	87	85	82	81	97	96
Case 3	86	84	68	67	88	86
Case 4	81	79	55	53	82	80
Case 5	111	110	98	98	100	100
Case 6	91	100	69	76	87	95
Case 7	100	112	89	99	92	102
Case 8	135	142	117	123	49	51
Case 9	136	142	116	121	48	50
Case 10	130	134	144	149	48	50
Case 11	109	110	154	156	45	46
Case 12	108	109	154	156	45	46
Case 13	126	130	189	196	54	56
Case 14	127	131	190	196	55	57
Case 15	126	131	189	197	55	57

To make it a little simpler to read the table, the following division could be used: first, the share of parcel lockers is increased (Case 1-4), then the share of evening deliveries (Case 5-6) and ending with a combined case (Case 7). From Case 8, the same division is used together with a city hub implementation. So, with this division in mind, it can be stated that the parcel lockers implementation has the largest influence on decreasing the travelled distances and the operative times when no city hub is being used. Hence, the GHG-emissions reduce as a result of the number of kilometres decreasing.

In contrast, the high share of smaller vehicles that is being used by the city hub, large reductions in GHG-emissions can be obtained at the costs of more kilometres and operative time. Furthermore, the positive effect lockers have on decreasing the time operative in the regular case, operative time only seems to increase for an increasing share of parcel locker use. This is caused by the fact that the parcel lockers on average have more parcels to store than the capacity of the smaller vehicles. Hence, more vehicles are being utilised to replenish only one parcel locker unit. Evening delivery causes operative time to increase even more, as more vehicles are needed due to the loss of efficiency as mentioned in 4.3.3.

So, from the basic implementation of the alternatives as presented in Table 6-1, it can be concluded that only the parcel lockers alternative provides solid performance gains when its use is enhanced. This statement can be motivated by Case 4, which applies a 70% share of parcel locker deliveries, 10% evening deliveries and no city hub. Thereby, a 20% reduction in both travelled distance and its related GHG-emissions can be obtained, while also reducing about 45% of the operative time can be realised. The standard deviations of these variables remain within a 1% range and thereby, the outcomes provide an accurate estimation of the benefits

The implementation of a city hub in combination with a 70% share of parcel locker deliveries could reduce GHG-emissions by 55%, while increasing the travelled distance and operative time 10% and 55% respectively. Since smaller vehicles are being used that can travel via bicycle lanes, it is assumed that most of these travelled kilometres are having less impact on the traffic system in the city compared to the application of (electric) delivery vans. However, the division of these kilometres should be further investigated before more conclusive insights can be provided.

6.2 Reflecting upon the sustainability factors

The simulation model does not yield outcomes for all the sustainability factors as defined in 4.1. Therefore, the outcomes of the experiments from Table 6-1 will be used to reflect upon the remaining sustainability factors to see whether or not the case can be called sustainable. Furthermore, the reflection is based on the outcomes of Case 4, since it is able to enhance the performance of all outcomes of interest, in contrast to the best city hub configuration.

Firstly, the implementation and enhancement of parcel locker use is expected to decrease both the traffic impact, delivery time and delivery costs massively. Furthermore, due to the decrease in distance travelled per parcel, the GHG emissions in the city are expected to decrease as well, as shown in Table 6-1. Hence, if sold appropriately to the public, the customer satisfaction in the city may increase due to the increased flexibility the services offer.

Furthermore, as vehicles spend less time and distance on the road, the safety of the working environment may increase, due to the mitigated risks for accidents on the road. Security and responsibility are then perceived similar as what it was before, since no extra transhipment points are implemented in the system (like a city hub). Moreover, parcels are still being transported by the assigned companies, so no transfer between companies is required, mitigating legal obstacles.

However, Case 4 as it is presented in Table 5-2 is not sustainable yet. It is highly policy sensitive with regard to the so-called 'Milieuzones', since polluting diesel vans are still being used. As mentioned earlier in this report, these vans are about to be abandoned from the city centres, so companies are encouraged to electrify their vehicle fleet. Hence, it can be concluded that Case 4 is not yet sustainable and some changes in the setup have to be applied before it is.

Besides, the size of parcel locker growth requires large investments in the infrastructure. Since parcel lockers are assumed to be publicly owned, it remains unclear which party (or parties) have to cover for these expenses. Furthermore, the electrification of the vehicle fleet could be an expensive change as well, since electric delivery vans are not widely produced and thus used either. One way to handle this hurdle is to gradually electrify the vehicle fleet by not buying new diesel vans, but replacing old vans with new electric variants when the lease contracts are exceeded. However, this seems only feasible for big transporting companies with dense city networks like PostNL, which still leaves the question how smaller transporting companies could reach their clients in the city centre?

6.3 Conclusions

From the three alternatives implemented in the simulation model (parcel lockers, a city hub and evening delivery), parcel lockers seem to provide the most reductions regarding the average distance per parcel. By cutting the distance per parcel, emissions per parcel thus also decrease. Furthermore, parcel lockers lead to less time per parcel to be delivered, cutting on the employment costs as well. Hence, implementing a parcel lockers infrastructure is expected to yield the best results.

In contrast, the city hub option focussing on transhipment to smaller vehicles yields a less efficient system, though more environmentally friendly due to the electrification of city centre deliveries. However, concerns about this configuration are the large increase in expected delivery costs and the shift from the road to the bicycle lanes. Will there be enough space for this number of small vehicles?

Lastly, it can be stated that evening delivery yields even less efficient results in both configurations (with and without city hub). This conclusion is following Hünteler (2018), stating that the distance to the city plays a bigger part in the evening delivery due to the less available time to conduct the deliveries. Since the available time for evening delivery is one of the only reasons causing the inefficiency, the further development of night delivery can be seen as a solution to also gain more reductions.

This chapter is aimed at answering the fifth sub-question of this study: *What benefits can be obtained by implementing one of these alternatives?* Based on the foregoing conclusions, implementing and enhancing parcel locker deliveries yields the largest benefits in terms of efficiency. Numerically speaking, 20% of the kilometres in the city and the GHG emissions and 45% of the time in operation could be spared. However, parcel locker deliveries cannot be sustainable if not being conducted by electric vehicles. By implementing a city hub, smaller, electric vehicles are applied for the deliveries, yielding large benefits with regard to reducing the GHG-emissions (about 55% reduction) at the costs of increasing the travelled distance and time in operation (about 10% and 55% respectively). Lastly, evening delivery only reduces the system's efficiency, but due to the fact that cities are less congested in the evening, the perceived traffic impact is assumed to may decrease too.

So, each of the alternatives has some positive and negative sides to them. In order to make the parcel delivery more sustainable and thereby providing a satisfying answer to the main research question, more reflection on the implementation of each of the alternatives is required. This is done in the following chapter.

7 | Reflecting on real-life implementation

So, a more sustainable last mile delivery process is defined as one that reduces the number of kilometres driven in the city, drastically reduces the GHG-emissions whilst also be profitable for the involving companies. According to the analysis in the foregoing chapter(s), this process consists of a more present and denser parcel locker infrastructure, optionally a city hub for transhipment on smaller, electric vehicles that also enables delivery by night. But, practically speaking, what would this system be like? What are the findings that make it (more) sustainable?

7.1 Parcel lockers

Firstly, the use of parcel lockers has to be expanded. However, before implementing the parcel locker infrastructure, a good analysis has to be conducted on the optimal distance between each locker. In explicit, what is the willingness to walk of consumers that optimise the use of lockers, and what capacity do the lockers need to have? Furthermore, as the capacity per locker increases, companies have to consider a re-design of the devices, as very big walls of lockers seem undesirable. They have to disappear from the streets and therefore, empty stores or garage-boxes are proposed as alternative locations to place big walls full of lockers. Or, following the garbage-disposal market and/or bicycle storage in Japan (see Figure 7-1 and Figure 7-2), the lockers may be placed underground as a sort of underground warehouse. However, this last option may make the infrastructure very expensive.

Figure 7-1: Underground bicycle storage[5]

Figure 7-2: Underground disposal containers[6]

Besides, the more parcel lockers are being used, the more interesting it becomes to replenish them by night, as no one has to stay awake to accept their parcel. Since most of the inhabitants are asleep, it is assumed that the impact on the city reduces, so does the risk of hitting a traffic jam, while the route efficiency increases. However, in order to mitigate the noise emissions of diesel vans driving through the streets at night, night delivery is only perceived acceptable if electric vehicles are to be used.

[5] Retrieved from: https://www.outdoordesign.com.au/news-info/automated-bicycle-parking-system/5392.htm
[6] Retrieved from: http://www.prommenz.nl/locatiestudie-ondergrondse-containers-en-clusterplaatsen

7.2 City Hub

That is where the city hub comes into play. Since the city hub is located so near the city, the range of battery powered vehicles becomes much less of an issue. Besides, as more deliveries are intended for parcel lockers and the replenishment of these parcel lockers take place at night, the number of parcels to deliver by day reduce. During the night, electric vans can be used to consolidate the parcels destined for lockers, as these vans have no impact on city traffic at this time of day. In contrast, day deliveries can be divided among the electric vans and smaller EVs, like cargo bikes and stints, to reduce the overall traffic impact by day.

However, this research has already shown that the city hub option is not efficient for big transporting companies in the current parcel delivery market. The number of parcels to be delivered is too big to make the transhipment at the city hub interesting. Instead, one of the propositions is that these parties facilitate the city hub for smaller transporters. That is, smaller transporters can drop their parcels at the city hub and the big transporters pick them up, as there is a high probability that the big transporters are passing by the destination of the small transporter anyway. This can be arranged in two different ways, as shown in Figure 7-3 and Figure 7-4.

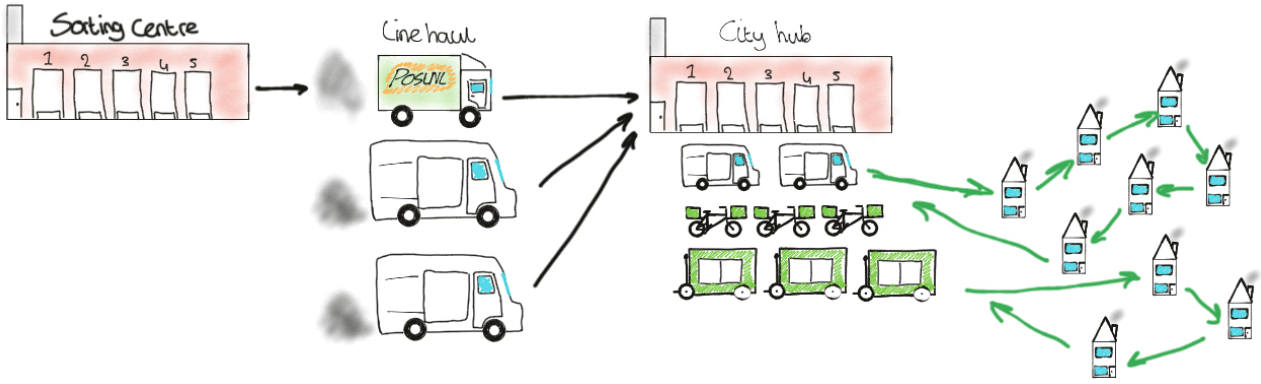


Figure 7-3: City hub with line haul

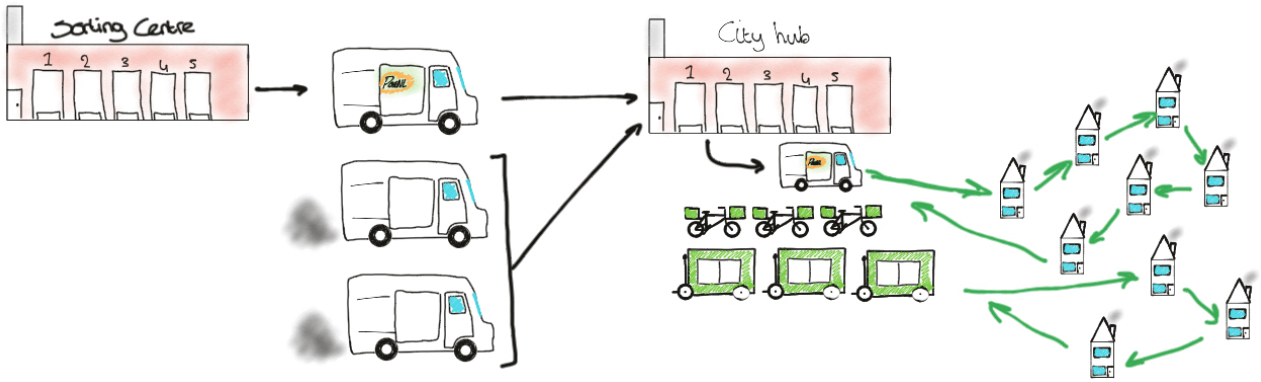


Figure 7-4: City hub as pick-up point

Firstly, the city hub can be used as intended: PostNL and other parties deliver their parcels by means of a line haul to the city hub. The city hub is facilitated by PostNL and PostNL tranships the parcels to electric and/or smaller vehicles (see Figure 4-3). This structure also offers opportunities for other big transporters, as other big transporters are also able to facilitate their own city hubs and enter the competition to compete for clients.

On the other hand, the city hub can be used by PostNL as a sort of pick-up point, while the city hub is facilitated by a third party (see Figure 4-4). The smaller transporters still deliver their parcels at the city hub, but PostNL also includes them in their route planning. Hence, PostNL's loading process can be split up between their sorting centre and the city hub. It is assumed that this does not take as long as loading a van twice. Besides, the city hub can provide the smaller vehicles for unforeseen or custom shipments, while the regular shipments are transported by PostNL's electric vans.

The latter is particularly interesting for the bigger transporting companies as they themselves do not have to own and operate the city hub, since this is done by a third party. They only have to focus on the electrification of their own fleet. Furthermore, the big transport companies could use the same facility together, which makes the use of separate city hubs unnecessary and saves space. A third party that could be interested in operating the city hub, could be a local government, a municipality or an entrepreneur with a very bright business idea.

7.3 Conclusions

With the proposed solutions in this chapter, the best of the alternative last mile systems can be combined. One of the most important success factors of the city hub now is to include more smaller transporting companies and engage them in the dense delivery networks of the bigger companies. Furthermore, by making the parcel locker infrastructure public property, all delivery companies can use them, making it a more interesting delivery alternative to invest in. Hence, both efficiency and sustainability of the parcel delivery sector in cities enhance.

This chapter aims at reflecting on the last sub-question: *How could the alternatives be implemented?* Based on the success of underground garbage containers, underground parcel lockers seem to provide a solid starting point for large-scale implementation without disrupting the beloved street image. Furthermore, combining the parcel locker infrastructure with a city hub that combines the dense networks of big transporting companies with smaller ones offers incentives for the big companies to electrify their fleet. Obliging all companies to consolidate their shipments at a city hub is unfeasible, due to the enormous scale of parcel delivery these days. Lastly, a city hub with small electric vehicles offers a good base for the (further) development of night-replenishment of parcel lockers. However, that is a plan for the future, as there is currently little insight in its effects.

8 | Conclusions and reflection

In order to provide an answer to the main research question, the sub-questions as defined in Table 1-1 have to be answered. Moreover, these have already been answered in the foregoing chapters of this report. The objective of this study was to identify sustainable alternatives for the last mile parcel delivery and implement these in a simulation model to see their costs and benefits. However, it turned out that sustainable parcel delivery was not yet properly defined. Besides, the last mile alternatives did not report on the same factors of interest. Therefore, the following sub-questions were defined:

Table 8-1: Research sub-questions

Number	Question
1.	How is a sustainable last mile delivery defined?
2.	What are the most important stakeholders in the PostNL case? And, what are their points of view regarding a more sustainable last mile delivery process?
3.	What are the current policies steering the sector in a sustainable direction? And, are they effective?
4.	What are the most promising last mile delivery alternatives, given the sustainability requirements and perceptions of the stakeholders?
5.	What benefits can be obtained by implementing one of these alternatives?
6.	How could this alternative be implemented?

8.1 Answering the sub-questions

Firstly, sustainable last mile delivery was defined by analysing research on sustainability, corporate sustainability and sustainable supply chain management. The reoccurring theme among this line of research is the so-called Triple Bottom-Line (3BL) theory, stating that a sustainable future considers both environmental, social and economic aspects (Elkington, 2002). If a product or project gains on one of these aspects by losing on another, it cannot be called sustainable. Based on this principle, popular themes and issues among stakeholders have been assessed. This led to a shortlist of criteria that are important to at least one of the stakeholders. The (more) sustainable last mile has to score better than the current process on as many criteria as possible. The shortlist is shown in Table 8-2.

Table 8-2: Shortlist of criteria

Criterion:	
Delivery cost	Emissions (CO_2, NO_x and/or PM_{10})
Delivery time	Customer satisfaction
Safe working environment	Security and responsibility
Traffic impact	Investments in infrastructure
Policy sensitivity	

Secondly, the most important stakeholders have been defined as a part of defining the sustainable last mile. Since the last mile distribution of parcels can be closely linked to city logistics, the governmental organisations are included in the analysis (Boer et al., 2018). Furthermore, a transportation assignment involves at least a sender and recipient and often a logistics provider, which can be of any kind (consumer, local store, companies, restaurants, etc.). Besides, branch organisations try to influence domestic policy to improve the transport sector. Examples that have been included are *'Transport en Logistiek Nederland'* (TLN) and *'Evofenedex'*.

Summarising their points of view, a visible trend could be noticed from the ranking sheets the representatives had to fill in (see Table 4-1 for an example). Delivery cost and delivery time are perceived as very important by the consumers or other recipients, suppliers and the transporting companies, whereas GHG-emissions, traffic impact and safe working environment are ranked high at the governmental organisations and the branch organisations. Besides, suppliers and logistics providers value their customers' satisfaction highly as well. All other factors are somewhere in-between.

The third sub-question can also be answered by the same stakeholder analysis. As a part of their point of view regarding parcel delivery in cities, municipalities announced their intended policies as part of their run for the elections. From PostNL's analyses of these election programmes it could be noticed that the municipalities of big cities are looking at ways to either implement a so-called 'Milieuzone', or to extend it. This means that (older) diesel vans and trucks are not allowed to enter the inner city, so logistics providers are incentivised to change to electric vehicles. Furthermore, to make electric transport more interesting to invest in, many governmental organisations express their interest in a so-called city hub. How to implement the city hub is still an open end to the discussion. Lastly, (local) governments also provide subsidies and/or tax arrangements for electric vehicles to make their purchase even more attractive. However, these policies differ among municipalities.

PostNL is already experimenting with new methods for last mile parcel delivery on a small scale. For instance, they have trial projects for a parcel locker infrastructure, evening delivery and cargo bike deliveries. However, PostNL had little insight in how these projects perform compared to each other, especially on the sustainability factors as defined by the first sub-question. Therefore, interviews have been conducted with the project managers, so more insight in the performance was obtained. Based on their individual scores in best-worst-case scenarios, a SMART-analysis has been conducted. This revealed that CDPs and parcel lockers perform best under all scenarios, followed by night deliveries and the city hub. Regarding the city hub, it is important to note that they only provide good results when electric vehicles are used. Otherwise the impact is nil. Hence, the optimal and more sustainable last mile parcel delivery system consists of these three options.

The three alternatives have been implemented in a discrete-event simulation (DES) model. The DES-model is written in the Python programming language, based on the processes as explained by Bakr (2018). The outcomes of the experiments show that again parcel lockers provide the best results with regard to reducing the number of kilometres driven per parcel in the city. By implementing a city hub this effect can be obtained too, but it will cost more time per parcel to conduct the deliveries. This is mainly due to the fact that smaller vehicles are being used, thus more vehicles and personnel are needed to deliver the same number of parcels. Furthermore, a city hub for B2C parcel delivery is unfeasible, as the volumes are too big to consolidate. However, the main benefit of the city hub is that the GHG emissions reduce drastically due to the use of electric vehicles. Evening delivery shows to be less efficient, following the comments of Hünteler (2018), due to the less available time.

Lastly, an optimal sustainable last mile delivery combines the simulated alternatives, with evening delivery being extended to night delivery in the future. Parcel lockers and pick-up points have to be present throughout the city, without disrupting the street-image. In explicit, nobody is expected to accept big walls of lockers at the corner of their street. Instead, lockers may be placed underground, or in empty garage boxes and buildings. Besides, the city hub will be implemented near big cities to consolidate shipments destined within the 'Milieuzone'. However, companies still have to investigate the best business case for their implementation of the city hub (either Figure 7-3 or Figure 7-4), as either of them has advantages and disadvantages. Furthermore, as lockers will increasingly be used and night delivery will develop further, replenishment of the lockers can take place at night to further decrease the traffic impact.

8.2 Answering the main research question

So, with all the answers on the sub-questions, an answer to the main research question of this study can be provided. The main research question was:

How could the last mile delivery process become more sustainable, i.e. minimising traffic impacts and emissions, while maintaining the social and economic benefits of e-commerce and home deliveries?

The last mile delivery process of parcel delivery can become more sustainable by implementing or expanding multiple alternative systems. Especially the implementation of parcel lockers reduces the total distance travelled in the city and time needed for delivery and thereby, reduces the traffic impact and personnel related costs of parcel deliveries. Furthermore, parcel lockers can easily be replenished by night, providing a more solid base for night delivery to develop and reduce the traffic impact even more.

The UCC-option, or city hub as called in this study, turned out to be less sustainable for the parcel delivery market than perceived in advance by most of the stakeholders. Despite the large potential of reducing GHG-emissions, the city hub causes the total time and distance travelled to increase significantly due to the use of smaller vehicles that have to deliver multiple routes instead of one. Hence, the traditional idea of a city hub is found not feasible in combination with the parcel delivery market. Instead, the design of Figure 7-4 is proposed for parcel deliveries in cities.

With the proposed city hub design, big transporting companies are responsible for the electrification of their vehicle fleet that conducts deliveries in cities. The city hub can then be used to consolidate the shipments of smaller transporters and tranship them onto the dense routes of the big transporters. The effect of the city hub on the distance driven per parcel becomes stronger, the more small transporting companies are using the city hub. Thereby, traffic, time and especially GHG-emissions can be reduced significantly. Moreover, the city hub could then also provide smaller vehicles for special deliveries, like cargo bikes and stints, to enhance instant delivery services caused by unknowing companies for instance.

By implementing the three proposed options, performance can be enhanced on most of the sustainability criteria. The system offers more flexibility for customers and thereby customer satisfaction is expected to increase. By the application of electric vehicles, the system is less sensitive to the 'Milieuzone' policies of the municipalities. Furthermore, the implementation of the city hub and the reduction of kilometres in the city reduces the traffic impact in the city significantly. However, serious negotiations have to be done in order to align the responsibilities of stakeholders with regard to the high investments have to be made to create the infrastructures.

8.3 Scientific reflection

So yes, the proposed system is more sustainable than the current way parcel delivery is conducted. But, have these conclusions been based on solid research, and could the assumptions have any effect on the outcomes of this research? What could have been done differently? What if other data becomes available? Is the answer to the main research question really the answer to all problems, or, what is needed extra to provide a more solid answer? All these questions are part of the reflection that is presented in this paragraph.

First of all, the research got an early focus on the city hub option, since the city hub is one of the projects PostNL was working on with other companies. Furthermore, the city hub is a reoccurring theme among several municipal election programmes, while it is presumed that the municipalities do not even know the implications for the parcel delivery market. In the first intension, the city hub will only be used for deliveries inside the 'Milieuzone', whereas other areas of the city remain being

delivered by diesel vans. This has been included in the simulation and hence, the city hub might be capable of reducing the emissions even further than simulated.

However, a city hub that can handle more than 30,000 parcels a day on average (average of PostNL in Amsterdam plus the remainder of the market) requires a lot of space, man-power and automation to make it work. Moreover, the city hub looks more like a new sorting centre, with different parcels from different companies. Hence, the intension of combining these streams is unfeasible, and other options of designing the processes have to be considered.

Another assumption, or way of implementation, that could have affected the outcomes of this study, is the ignorance of so-called route factors. The simulation model is built upon the Euclidian distances between destinations, whereas in real-life normal roads would have been used. These routes are always longer than the Euclidian distance, which is sometimes defined by a certain factor for certain cities. Furthermore, there are reasons to assume that this factor smaller for bicycles compared to cars, since bicycles are assumed to be able to drive through small alleys and shorter bicycle-lanes. Hence, using a cargo bike might as well be faster in the city than currently defined in the simulation model. Thereby, the case for a city hub focussed on these small electric vehicles might become even stronger than shown by this research.

Furthermore, this study does not yet really provide insight in the impact on city congestion. Yes, by implementing the alternative systems and combining them in an optimal way, the time and distance parcel delivery vans drive in the city decrease and thereby, their impact on the traffic system reduces. However, the parcel delivery and express market is only accountable for 4% of the GHG-emissions in cities. The remaining GHG-emissions are from other sectors and the inhabitants themselves. Hence, the question is raised whether or not reducing the distance and time parcel delivery vehicles drive in cities helps solving the congestion problems. At least the sector is becoming more sustainable and the focus may have to shift to another sector, or personal mobility, to mitigate city congestion.

Lastly, the simulation model is based on regular delivery assignments that look mostly like the normal home delivery (small number of parcels per stop, average sized parcels). This implies that special deliveries are left out of the simulation, in explicit, deliveries with 4+ parcels or deliveries that contain washing machines and have to be connected. Hence, the simulation only provides insight in the average situation and not in exceptions. If parcel dimensions are considered while creating the demand, a more accurate estimation of the vehicle capacity can be made. This may lead to benefits for electric vans in the process, as they are less sensitive for large-sized parcels. In explicit, the bigger and heavier the parcel, the lower the probability that it fits in a cargo bike. Thereby, the applicability of cargo bikes for the complete process of parcel deliveries becomes questionable.

8.4 Recommendations for further research

To conclude this chapter, and this thesis, some recommendations for further research are proposed. First, following the two different propositions made in 7.2, different operational structures for the city hub should be studied in more detail. In explicit, it should be defined what the responsibilities are for each of the stakeholders, so an organisational structure can be designed around it. Besides, this clarifies the requirements for new IT-systems that could be implemented.

Second, it is advised to further investigate the traffic flows in cities in relation to city logistics. How do they develop over time, and how are they related to (personal) mobility? Thereby, the current simulation model can be integrated with the traffic model to see the effects on traffic and congestion by changing different professional sectors. Furthermore, bicycle networks should be included in the analysis, as city logistics alternatives are moving towards these vehicles for transportation. Despite the benefits, bicycles can also cause congestion, thus getting more insight therein is important.

Third, related to the second recommendation, it is advised to study the route factors of cities in the Netherlands for both (electric) vans and bicycles. Thereby, more accurate results of the analyses as conducted in this study can be obtained. To obtain these factors, the road network in relation to the locations of physical addresses has to be analysed. Then, the route factor is defined as the average factor of the physical route distance (by road) being longer than the Euclidian distance between all locations in the area. As already mentioned, it is expected that the route factor of bicycles is smaller than the route factor for (electric) vans, which could strengthen the case for city hubs with bicycles.

Fourth, logistics service providers and municipalities should investigate how to design a universal locker infrastructure that can be used by multiple parties. The assumption is that many lockers will be placed in public areas. So, just like the garbage collection points in Dutch cities, which is managed by the municipality itself, (underground) lockers could be located near these points. Hence, the investment in lockers becomes a public investment and publicly owned as a solution to a mainly public problem. The main aspects to study are still the design and IT-system behind these lockers, so that all different parties have access to these lockers.

PostNL

Lastly, with regard to PostNL, the company supporting this research, some follow-up actions have been proposed. Namely, company has to investigate the investments to be made before they would be able to establish the sustainable last mile system as proposed in this study. What is possible by themselves, and what is needed from other stakeholder in order to achieve it all? Also, PostNL is advised to think of ways to facilitate the city hub. Important to consider is the importance of remaining the front-runner in the subject. If the feasible organisation has been found, it is key to spread the system among the big cities in order to comply to the Green Deal ZES (Green Deal ZES, n.d.).

References

Çaliskan, A., Kalkan, M., & Ozturkoglu, Y. (2017). City logistics: Problems and recovery proposals. *International Journal of Logistics Systems and Management, 26*(2), 145-162. doi:DOI: 10.1504/IJLSM.2017.081497

ACM. (2016). *Post- en pakkettenmonitor 2016.*

Ahi, P., & Searcy, C. (2013). A comparative literature analysis of definitions for green and sustainable supply chain management. *Journal of Cleaner Production, 52*, 329-341.

Ahold Delhaize. (n.d.). *Bol.com pick-up points expand to almost all Albert Heijn stores*. Retrieved 03 06, 2018, from Ahold Delhaize: https://www.aholddelhaize.com/en/media/media-releases/bolcom-pick-up-points-expand-to-almost-all-albert-heijn-stores/

Allen, M., Spencer, A., Gibson, A., Matthews, J., Allwood, A., Prosser, S., & Pitt, M. (2015). Right cot, right place, right time: improving the design and organisation of neonatal care networks - A computer simulation study. *Health Services and Delivery Research, 3*(20).

Ampe, J., & Macharis, C. (2008). The use of multi-criteria decision analysis (MCDA) for the evaluation of transport projects: a review. *Journal of Multi-Criteria Decision Analysis*, xx.

ANWB. (2017, December 27). *Files nauwelijks toegenomen in 2017*. Retrieved March 02, 2018, from ANWB: https://www.anwb.nl/verkeer/nieuws/nederland/2017/december/files-nauwelijks-toegenomen-in-2017

Arnold, F., Cardenas, I., Sörensen, K., & Dewulf, W. (2018). Simulation of B2C e-commerce distribution in Antwerp using cargo bikes and delivery points. *European Transport Research Review, 10*(2), 1-13.

Bakr, S. (2018, 04 19). Interview UCCs. (J. Daleman, Interviewer)

Bansal, T. (2010). *Network for Business Sustainability.* Retrieved March 30, 2018, from http://nbs.net/wp-content/ uploads/Primer_Business_Sustainability.pdf

Behiri, W., Belmokhtar-Berraf, S., & Chu, C. (2018). Urban freight transport using passenger rail network: Scientific issues and quantitative analysis. *Transportation Research Part E, 115*, 227-245.

Boer, E. d., Kok, R., Ploos van Amstel, W., Quak, H., & Wagter, H. (2017). *Outlook City Logistics 2017.* Topsector Logistics.

Brown, J., & Guiffrida, A. (2014). Carbon emissions comparison of last mile delivery versus customer pickup. *International Journal of Logistics Research and App[lications, 17*(6), 503-521.

Bryant, B. P., & Lempert, R. J. (2010). Thinking inside the Box: a participatory computer-assisted approach to scenario discovery. *Technology Forecast for Societal Change*, 34-49.

Cárdenas, I., Beckers, J., & Vanelslander, T. (2017). E-commerce last-mile in Belgium: Developing an external cost delivery index. *Research in Transportation Business & Management*, In Press.

References

CBS. (2016). *Meer Nederlanders shoppen online*. Retrieved March 02, 2018, from CBS: https://www.cbs.nl/nl-nl/nieuws/2016/24/meer-nederlanders-shoppen-online

CBS. (2017a, May 16). *Hogere CO2-uitstoot in het eerste kwartaal 2017*. Retrieved March 02, 2018, from CBS: https://www.cbs.nl/nl-nl/nieuws/2017/20/hogere-co2-uitstoot-in-het-eerste-kwartaal-2017

CBS. (2017b, August 16). *Lagere CO2-uitstoot in het tweede kwartaal 2017*. Retrieved March 02, 2018, from CBS: https://www.cbs.nl/nl-nl/nieuws/2017/33/lagere-co2-uitstoot-in-het-tweede-kwartaal-2017

CBS. (2017c, November 14). *CO2-uitstoot vrijwel even hoog in derde kwartaal 2017*. Retrieved March 02, 2018, from CBS: https://www.cbs.nl/nl-nl/nieuws/2017/46/co2-uitstoot-vrijwel-even-hoog-in-derde-kwartaal-2017

CBS. (2018, February 14). *Lagere CO2-uitstoot in het vierde kwartaal 2017*. Retrieved March 03, 2018, from CBS: https://www.cbs.nl/nl-nl/nieuws/2018/07/lagere-co2-uitstoot-in-het-vierde-kwartaal-2017

CBS. (2018b, March 09). *Kerncijfers wijken en buurten 2017.* Retrieved 07 25, 2018, from Statline: https://opendata.cbs.nl/statline/#/CBS/nl/dataset/83765NED/table?ts=1529053570930

Christopher, M. (1998). In *Logistics and Supply Chain Management – Strategies for reducing costs and improving services* (2nd Edition ed., p. p. 15). London.

Clausen, U., Geiger, C., & Poting, M. (2016). Hands-on testing of last mile concepts. *Transportation Research Procedia, 14*(2016), 1533-1544.

De Marco, A., Mangano, G., & Zenezini, G. (2018). Classification and benchmark of City Logistics measures: An empirical analysis. *International Journal of Logistics: Research and Applications, 21*(1), 1-19.

Edwards, J., McKinnon, A., & Cullinane, S. (2010). Comparative analysis of the carbon footprints of conventional and online retailing: A "last mile" perspective. *International Journal of Physical Distribution and Logistics, 40*(1/2), 103-123.

Edwards, W. (1977). How to use multiattribute utility measurement for social decision making. *IEEE Transaction on Systems, Management and Cybernetics, 7*(5), 326-340.

EIA. (2017). *International Energy Outlook 2017*. Retrieved September 30, 2017, from US Environmental Information Administration: https://www.eia.gov/outlooks/ieo/pdf/0484(2017).pdf

Ekowicaksono, I., Bukhari, F., & Aman, A. (2016). Estimating Origin-Destination Matrix of Bogor City Using Gravity Model. *IOP. Conf. Series: Earth and Environmental Science, 31*, 1-5.

Elkington, J. (2002). *Cannibals with forks: the triple bottom line of 21st century business.* Oxford: Capstone.

Engert, S., & Baumgartner, R. (2015). Corporate sustainability strategy – bridging the gap between formulation and implementation. *Journal of Cleaner Production, 113*(2016), 822-834.

References

Esfahbodi, A., Zhang, Y., Watson, G., & Zhang, T. (2017). Governance pressures and performance outcomes of sustainable supply chain managment – An empirical analysis of UK manufacturing industry. *Journal of Cleaner Production, 155*, 66-78.

European Commission. (2015). *Summary of Responses to the European Commission's 2015 Public Consultation on Cross-border Parcel Delivery.* European Union.

European Union. (2013). *General Union Environment Action Programme to 2020: Living well, within the limits of our planet.* Brussels: European Union.

Gevaers, R., Van de Voorde, E., & Vanelslander, T. (2009). *Characteristics of innovations in last mile logistics.* University of Antwerp, Department of Transport and Regional Economics, Antwerp.

Gillespie, B., & Rogers, M. M. (2016). Sustainable Supply Chain Management and the End User: Understanding the Impact of Socially and Environmentally Responsible Firm Behaviors on Consumers' Brand Evaluations and Purchase Intentions. *Journal of Marketing Channels, 23*(1-2), 34-46.

Gogas, M., & Nathanail, E. (2017). Evaluation of Urban Consolidation Centers: A Methodological Framework. *Procedia Engineering, 178*(2017), 461-471.

Green Deal ZES. (n.d.). *Doel*. Retrieved March 05, 2018, from Zero Emission Stadslogistiek: https://greendealzes.connekt.nl/doel/

GRI. (2017). *Reporting on the SDGs*. Retrieved March 02, 2017, from Global Reporting: https://www.globalreporting.org/information/SDGs/Pages/Reporting-on-the-SDGs.aspx

Hünteler, R. (2018). Informatie-uitwisseling avondbelevering. (J. Daleman, Interviewer)

Hadavi, S., Macharis, C., & Van Raemdonck, K. (2018). The Multi-Actor Multi-Criteria Analysis (MAMCA) Tool: Methodological Adaptations and Visualizations. *Advances in Intelligent Systems and Computing, 572*, 39-53.

Hedges, A. (2002). *Finding distances based on Latitude and Longitude*. Retrieved 07 30, 2018, from Andrew Hedges: https://andrew.hedges.name/experiments/haversine/

IEEE. (2013). IEEE Guide – Adoption of the Project Management Institute (PMI) Standard A Guide to the Project Management Body of Knowledge (PMBOK) Guide.

Janjevic, M., & Ndiaye, A. (2017). Investigating the financial viability of urban consolidation centre projects. *Research in Transportation Business & Management, 24*(2017), 101-113.

Johnson, D., & McGeoch, L. (1995). The Travelling Salesman Problem: A Case Study in Local Optimization. In E. Aarts, & J. Lenstra, *Local Search in Combinatorial Optimization* (pp. 215-310). London: John Wiley and Sons.

Kafle, N., Zou, B., & Lin, J. (2017). Design and modeling of a crowdsource-enabled system for urban parcel relay and delivery. *Transportation Research Part B, 99*, 66-82.

Kelton, D., Sadowski, R., & Sturrock, D. (2003). Simulation with Arena.

References

Kenniscentrum InfoMil. (n.d.). *Shapefile en csv-bestand*. Retrieved 07 25, 2018, from Kenniscentrum InfoMail: https://www.infomil.nl/onderwerpen/lucht-water/luchtkwaliteit/slag/monitoren-nsl/handleiding/monitoringstool/exporteren/shapefile-csv/

Khan, D., Varshney, P., & Quadeer, M. (2011). E-commerce: from shopping carts to credit cards. *IEEE International Conference on Communication Software and Networks (ICCSN).* Xi'an: IEEE.

KiM. (2016). *Mobiliteitsbeeld 2016.* Den Haag. Retrieved March 02, 2018, from http://web.minienm.nl/mob2016/index.html

Kishna, M., Niesten, E., Negro, S., & Hekkert, M. (2017). The role of alliances in creating legitimacy of sustainable technologies: A study on the field of bio-plastics. *Journal of Cleaner Production, 155*(1), 7-16.

Kleijnen, J. (2001). *Experimental Design for Sensitivity Analysis of Simulation Models.* Tilburg.

Kwakkel, J. H. (2017). The Exploratory Modeling Workbench: An open source toolkit for exploratory modeling, scenario discovery and (multi-objective) robust decision making. *Environmental Modelling & Software, 96*(2017), 239-250.

Lee, S., Avanchi, A., & Abourisk, S. (2010). Modeling Framework and Architecture for Hybrid System Dynamics and Discrete Event Simulation for Construction. *Computer-Aided Civil And Infrastructure Engineering, 26*(2), 77-91.

Ligterink, N., van Zyl, P., & Heijne, V. (2016). *Dutch CO2 emissions factors for road vehicles.* Earth, Life and Social sciences. Utrecht: TNO.

Lohn, A. J. (2017). *What's the Buzz? City-scale Impacts of Drone Delivery.* Santa Monica: RAND Corporation.

Lowe, R., & Rigby, M. (2013). *The Last Mile - Exploring the online purchasing and delivery journey.* Conlumino.

Macharis, C., Turcksin, L., & Lebeau, K. (2012). Multi actor multi criteria analysis (MAMCA) as a tool to support sustainable decisions: State of use. *Decision Support Systems, 54*(2012), 610-620.

Maes, J., & Vanelslander, T. (2012). The use of bicycle messengers in the logistics chain, concepts further revised. *Procedia - Social and Behavioral Sciences, 39*, 409-423.

McKinsey & Company. (2017). *An integrated perspective on the future of mobility.* Center for Business and Environment. McKinsey & Company.

Ministerie van Infrastructuur en Waterstaat. (2018). *Aanpassing Nationaal Samenwerkingsprogramma Luchtkwaliteit (NSL) 2018.* Den Haag.

Nijhuis, M. (2018, 04 24). Prestaties fietsnetwerk. (J. Daleman, Interviewer)

Page, J. (n.d.). *Centroid of a triangle (Coordinate geometry).* Retrieved 07 25, 2018, from Math open reference: https://www.mathopenref.com/coordcentroid.html

PBL. (2016). *Trends in global CO2 emissions: 2016 report.* Den Haag: PBL.

References

Ploumen, E. M. (2016, September 30). Brief van de Minister voor Buitenlandse Handel en Ontwikkelingssamenwerking [Kamerbrief]. Den Haag.

PostNL. (2017). *Position Paper Pakketten.* PostNL.

PostNL. (2017, 05 17). *PostNL replaces 100 car rides with e-cargo bicycles in Amsterdam*. Retrieved 05 31, 2018, from PostNL: https://www.postnl.nl/en/about-postnl/press-news/news/2017/postnl-replaces-100-car-rides-with-e-cargo-bicycles-in-amsterdam.html

PostNL. (2018). *Accelerating transformation: Annual report 2017.* Den Haag: PostNL.

PostNL. (2018). *Analyse PostNL gemeenteraadsverkiezingen 2018.* Den Haag.

PostNL. (n.d.). *Uw pakket ophalen bij een PostNL-locatie*. Retrieved 03 07, 2018, from PostNL: https://www.postnl.nl/ontvangen/pakket-ontvangen/ophalen-op-een-postnl-locatie/

Pruyt, E. (2013). Small Systems Dynamics Models for Big Issues. Delft.

Rezaei, J. (2015). Best-worst multi-criteria decision-making method. *Omega, 53*, 49-57.

Rijkswaterstaat. (n.d.). *NSL*. Retrieved March 27, 2018, from InfoMil: https://www.infomil.nl/onderwerpen/lucht-water/luchtkwaliteit/regelgeving/wet-milieubeheer/nsl/

Robinson, D. (2017, 09 06). *Incredible growth python.* Retrieved 07 25, 2018, from StackOverflow: https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Sargent, R. G. (2010). Verification and Validation of Simulation Models. *Proceedings of the 2010 Winter Simulation Conference* (pp. 166-183). IEEE.

Seuring, S. (2013). A review of modeling approaches for sustainable supply chain management. *Decision Support Systems, 54*, 1513-1520.

Seuring, S., & Müller, M. (2008). From a literature review to a conceptual framework for sustainable supply chain management. *Journal of Cleaner Production, 16*, 1699-1710.

Simoni, M., & Claudel, C. (2018). A simulation framework for modeling urban freight operations impacts on traffic networks. *Simulation Modelling Practice and Theory, 86*, 36-54.

Smit, C. (2018). Uitleg Lockers. (J. Daleman, Interviewer)

Stadtler, H. (2004). 1 Supply Chain Management - An Overview. In H. Stadler, & C. Kilger, *Supply Chain Management and Advanced Planning* (Third Edition ed.). Berlin: Springer.

Stein, D. M. (1978). An asymptotic, probablistic analysis of a routing problem. *Mathematics of Operations Research, 3*(2), 89-101.

Székely, F., & Knirsch, M. (2005). Responsible Leadership and Corporate Social Responsibility: Metrics for sustainable performance. *European Management Journal, 23*(6), 628-647.

Tako, A., & Robinson, S. (2012). The application of discrete event simulation and system dynamics in the logistics and supply chain management context. *Decision Support Systems, 52*, 802-815.

References

Thuiswinkel waarborg. (2017, September 01). *Online bestedingen stijgen in de eerste zes maanden van 2017 met 13% naar €10,66 miljard*. Retrieved March 02, 2018, from Thuiswinkel: https://www.thuiswinkel.org/nieuws/3532/online-bestedingen-stijgen-in-eerste-zes-maanden-van-2017-met-13-naar-10-66-miljard

Thuiswinkel.org. (2018). *Bewust Bezorgd*. Retrieved from Thuiswinkelwaarborg: https://bewustbezorgd.thuiswinkel.org/wat-is-bewust-bezorgd/

Thuiswinkel.org. (n.d.). *Wat is Thuiswinkelwaarborg?* Retrieved March 27, 2018, from Thuiswinkelwaarborg: https://www.thuiswinkel.org/consumenten/wat-is-thuiswinkel-waarborg

TLN. (2018, March 20). *Harde afspraken milieuzones voor vrachtverkeer*. Retrieved March 26, 2018, from TLN: https://www.tln.nl/actueel/nieuws/Paginas/Harde-afspraken-milieuzones-voor-vrachtverkeer.aspx

TLN. (2018, March 20). *Harde afspraken milieuzones voor vrachtverkeer*. Retrieved March 27, 2018, from TLN: https://www.tln.nl/actueel/nieuws/Paginas/Harde-afspraken-milieuzones-voor-vrachtverkeer.aspx

Tuinhout, L. (2018, March 05). Interview City Logistics. (J. Daleman, Interviewer)

United Nations. (2015). *Sustainable development goals*. Retrieved March 20, 2018, from United Nations: http://www.un.org/sustainabledevelopment/sustainable-development-goals/

Vaandrager, M. (2018, 02 19). Rondleiding in Houten. (J. Daleman, Interviewer)

Vaandrager, M. (2018b, August 02). Update gesprek model. (J. Daleman, Interviewer)

van Benten, W. (2017). *Position Paper DHL.* DHL Nederland.

van den Berg, B. (2018, 04 26). Avondbezorging gesprek. (J. Daleman, Interviewer)

van Soest, J. (1992). Elementaire statistiek. Delft: VSSD.

van Spronsen, M., & Meijer, E. (2018, August 01). Bespreken uitkomsten model. (J. Daleman, Interviewer)

van Spronsen, M., & Middelburg, D. (2018, Febuary 02). Material topics, SDG's en jaarverslag. Den Haag.

Verbeek, R., Bolech, M., van Gijlswijk, R., & Spreen, J. (2015). *Energie- en milieu-aspecten van elektrische personenvoertuigen.* Delft: TNO.

Vincke, P. (1992). Multicriteria Decision-aid. New York: John Wiley & Sons.

Vrije Universiteit van Amsterdam. (2017). *Postcode map of the Netherlands.* Retrieved 07 25, 2018, from Geoplaza: http://geoplaza.vu.nl/data/dataset/postcode

Vyt, D., Jara, M., & Cliquet, G. (2017). Grocery pickup creation of value: Customers' benefits vs. spatial dimension. *Journal of Retailing and Consumer Services, 39*(2017), 145-153. doi:DOI: http://dx.doi.org/10.1016/j.jretconser.2017.08.004

References

Walker, W. E., Marchau, V., & Kwakkel, J. (2013). Uncertainty in the Framework of Policy Analysis. In W. Thissen, & W. Walker, *Public Policy Analysis* (pp. 215-261). New York: Springer Science+Business Media.

Wang, Y., Zhang, D., Liu, Q., Shen, F., & Hay Lee, L. (2016). Towards enhancing the last-mile delivery: An effective crowd-tasking model with scalable solutions. *Transportation Research Part E, 93*, 279-293.

WCED. (1987). *Our Common Future.* Oxford: Oxford University Press.

White, K., & Ingalls, R. (2009). Introduction To Simulation. *Proceedings of the 2009 Winter Simulation Conference* (pp. 12-23). IEEE.

Wiegerinck, A. (2018, July 31). Model validatie Jasper. (J. Daleman, Interviewer)

Zhu, Q., Feng, Y., & Choi, S.-B. (2017). The role of customer relational governance in environmental and economic performance improvement through green supply chain management. *Journal of Cleaner Production, 155*(2), 46-53.

Appendices

APPENDIX I: Determining Weights of the criteria

Determining the weights of the criteria

The weights of the criteria are based on the individual assessments of experts' opinions. The experts have been asked to fill in a form and rank the criteria for each of the stakeholders. Thereby, several perceptions on the same problem have been gathered and have been analysed in this notebook.

Reading the Excel-sheets

Each of the experts' opinions have been stored in a separate sheet in an Excel file. Each sheet is read into memory separately and stored in a Pandas DataFrame to make the next steps easier.

In [1]:

```
# Imports of necessary libraries
import numpy as np
import pandas as pd
```

In [2]:

```
# Names of the experts that have been interviewed
names = ['Marielle', 'Marien', 'Sam', 'Valerie', 'Mark', 'Rogier', 'Ruben']

# Dictionary to store the DataFrames
AllRankings = {}

# By iterating through the names defined above, the Excel-sheets are opened.
for name in names:
    df = pd.read_excel('Criteria%s.xlsx' %name) # Open Excel-sheet
    df.name = name # Assigning the name to the DataFrame
    AllRankings[name] = df # Store DataFrame in previously defined dictionary
```

In [3]:

```
# Showing one example DataFrame
AllRankings['Marielle']
```

Out[3]:

	Delivery Cost	Delivery Time	Emissions reduction	Customer satisfaction	Safe working environment	Se Res
Consumers/recipients	2.0	1.0	NaN	3.0	NaN	NaN
Suppliers	2.0	2.0	3.0	1.0	3.0	2.0
Logistics providers	2.0	2.0	3.0	2.0	3.0	1.0
Governments	NaN	NaN	1.0	NaN	1.0	NaN
Branche organisations	NaN	NaN	1.0	NaN	NaN	NaN

The universal weight function

The weight function contains three different parts:

1. Calculating the reciprocal values
2. Calculate the normalised values of the cells
3. Determine the means of each column

Reciprocal Values
The reciprocal values are equal to 1/rank in the corresponding cell.

Normalised Values
First, the sum of the rows is calculated, followed by dividing the reciprocal value by the row-total

Means speak for itself.

In [4]:

```
def Weighing(df):
    # Calculating the Reciprocal values
    reciprocal = 1/df
    reciprocal.replace(np.nan, 0, inplace=True)
    # Summing the rows
    totals = reciprocal.sum(axis=1)
    # Calculating the normalised DataFrame
    normalised = reciprocal.copy()
    for index, row in reciprocal.iterrows():
        temp = row/totals[index]
        normalised.loc[index] = temp
    # Calculating the averages
    avg = normalised.mean(axis=0, skipna=False)
    avg.name = 'Averages'
    # Make one DataFrame that will be returned
    normalised = normalised.append(avg)

    return normalised
```

In [5]:

```
# Applying the function for each DataFrame in the dictionary
NormalisedMeans = {}

for name in names:
    temp = AllRankings[name]
    temp = Weighing(temp)
    NormalisedMeans[name] = temp
```

In [6]:

```
# Show example
NormalisedMeans['Marielle']
```

Out[6]:

	Delivery Cost	Delivery Time	Emissions reduction	Customer satisfaction	Safe working environment	Se Resp
Consumers/recipients	0.240000	0.480000	0.000000	0.160000	0.000000	0.00
Suppliers	0.146341	0.146341	0.097561	0.292683	0.097561	0.14
Logistics providers	0.126582	0.126582	0.084388	0.126582	0.084388	0.25
Governments	0.000000	0.000000	0.285714	0.000000	0.285714	0.00
Branche organisations	0.000000	0.000000	0.333333	0.000000	0.000000	0.00
Averages	0.102585	0.150585	0.160199	0.115853	0.093533	0.07

Determine the mean of all the average weights

Now, the mean of all the average weights can be determined. Hence, it becomes clear what criterion is valued higher on average than others among the different experts' perceptions.

In [7]:

```
avgdf = pd.DataFrame(columns=NormalisedMeans['Marielle'].columns)

for name in names:
    df = NormalisedMeans[name]
    avg = df.loc['Averages']
    avg.name = 'Averages_'+name
    avgdf = avgdf.append(avg)

avgdf
```

Out[7]:

	Delivery Cost	Delivery Time	Emissions reduction	Customer satisfaction	Safe working environment	Securi Respons
Averages_Marielle	0.102585	0.150585	0.160199	0.115853	0.093533	0.079901
Averages_Marien	0.141394	0.081501	0.228049	0.168403	0.048134	0.039401
Averages_Sam	0.142129	0.137584	0.117343	0.142129	0.103057	0.096500
Averages_Valerie	0.109145	0.125037	0.160574	0.159899	0.065499	0.109999
Averages_Mark	0.261030	0.105353	0.111692	0.152169	0.062221	0.068068
Averages_Rogier	0.181379	0.212149	0.138602	0.090690	0.080030	0.080030
Averages_Ruben	0.226718	0.109061	0.085294	0.147345	0.156577	0.119310

In [8]:

```
average_criteria = avgdf.mean(axis=0)
average_criteria
```

Out[8]:

```
Delivery Cost                    0.166340
Delivery Time                    0.131610
Emissions reduction              0.143108
Customer satisfaction            0.139498
Safe working environment         0.087007
Security and Responsibility      0.084744
Traffic impact                   0.107025
Investments in infrastructure    0.065548
Policy sensitivity               0.075120
dtype: float64
```

In [9]:

```
stdev_criteria = avgdf.std(axis=0)
stdev_criteria
```

Out[9]:

```
Delivery Cost                    0.059697
Delivery Time                    0.042100
Emissions reduction              0.046234
Customer satisfaction            0.027148
Safe working environment         0.036002
Security and Responsibility      0.026928
Traffic impact                   0.037619
Investments in infrastructure    0.017458
Policy sensitivity               0.024613
dtype: float64
```

Approach 2

Now, a different approach is applied to see whether that makes a difference in the average weights of the criteria that the experts filled in. Instead of taking the average of all average scores, the comparison will now take the average ranking into account. Hence, maybe a more accurate estimation of the weights can be obtained.

First, a new function to determine the average rank is defined. This uses the reciprocal values of each criterion and determines an average.

In [10]:

```
# Takes the Normalised dictionary as base

def avg_rank(dic):
    new_dict = {}
    lst = []
    for key in dic['Marielle'].columns:
        new_dict[key] = pd.DataFrame(index=dic['Marielle'].index)

    for key in dic.keys():
        df = dic[key]
        for crit in df.columns:
            newdf = new_dict[crit]
            ser = df[crit]
            ser.name = crit+'_'+key
            newdf = pd.concat([newdf, ser], axis=1)
            new_dict[crit] = newdf

    # Determine average over the columns
    avg = {}
    for key in new_dict.keys():
        averages = new_dict[key].mean(axis=1)
        stdev = new_dict[key].std(axis=1)
        averages = averages.drop('Averages')
        stdev = stdev.drop('Averages')
        averages['Average'] = averages.mean()
        stdev['Average'] = stdev.mean()
        averages.name = 'Average'
        stdev.name = 'Standard Deviation'
        avg[key] = pd.concat([averages, stdev], axis=1)

    return avg
```

In [11]:

```
avg = avg_rank(NormalisedMeans)
avg['Delivery Cost']
```

Out[11]:

	Average	Standard Deviation
Consumers/recipients	0.227952	0.143048
Suppliers	0.251559	0.077309
Logistics providers	0.231229	0.087464
Governments	0.026700	0.034560
Branche organisations	0.094260	0.126737
Average	0.166340	0.093823

As can be noticed, the average of all the stakeholders taken together is exactly the same as the approach 1. However, since the mean of the standard deviation of each individual actor is taken, the standard deviation of the second approach is slightly higher than before. So, it can be concluded that there is slightly more variation in the results than the former approach reflected. Hence, the results from the second analysis will be used throughout the remainder of the analysis.

Create different outcomes

Now it is time to create different outcomes for the average weights of the criteria, since there is a large deviation in the data. Hence, alternatives will be weighed according to different schemes that will be created by the following code.
The assumption is made that if one criterion is changed according to its standard deviation, this difference is equally compansated among the other criteria.

In [113]:

```
def definitive_weights(dic):
    keys = dic.keys()
    df = pd.DataFrame(columns=['Average', 'Standard Deviation'])
    for key in keys:
        crit = dic[key]
        avgs = crit.loc['Average']
        avgs.name = key
        df = df.append(avgs)

    base_case = df['Average']
    # If weight decreases by stdev, the other criteria's weights increase
    # First the lowering scenarios
    lowered = pd.DataFrame(columns=base_case.index)
    l = len(df) - 1 # Divider for remaining criteria

    for key in df.index:
        temp = df.copy()
        lower = temp.loc[key, 'Average'] - temp.loc[key, 'Standard Deviation']
        temp.loc[key, 'Average'] = lower
        remaining = df.loc[key, 'Standard Deviation'] / l

        temp2 = temp.loc[temp.index != key, 'Average'] + remaining

        for key in temp2.index:
            temp.loc[key, 'Average'] = temp2.loc[key]

        lowered = lowered.append(temp['Average'])


    increased = pd.DataFrame(columns=base_case.index)

    for key in df.index:
        temp = df.copy()
        higher = temp.loc[key, 'Average'] + temp.loc[key, 'Standard Deviation']
        temp.loc[key, 'Average'] = higher
        remaining = df.loc[key, 'Standard Deviation'] / l

        temp2 = temp.loc[temp.index != key, 'Average'] - remaining

        for key in temp2.index:
            temp.loc[key, 'Average'] = temp2.loc[key]

        increased = increased.append(temp['Average'])

    all_cases = pd.concat([lowered, increased], axis=0)
    all_cases = all_cases.append(base_case)
    all_cases.reset_index(drop=True, inplace=True)

    return all_cases
```

In [114]:

```
all_cases = definitive_weights(avg)
all_cases
```

Out[114]:

	Delivery Cost	Delivery Time	Emissions reduction	Customer satisfaction	Safe working environment	Security and Responsibility	Traffic impact
0	0.072517	0.143338	0.154836	0.151226	0.098735	0.096472	0.118753
1	0.174658	0.065069	0.151425	0.147816	0.095325	0.093062	0.115343
2	0.175260	0.140530	0.071747	0.148418	0.095927	0.093664	0.115945
3	0.175136	0.140406	0.151904	0.069129	0.095804	0.093540	0.115821
4	0.173285	0.138554	0.150052	0.146443	0.031449	0.091689	0.113970
5	0.173771	0.139041	0.150539	0.146930	0.094439	0.025294	0.114456
6	0.174284	0.139554	0.151052	0.147442	0.094951	0.092688	0.043474
7	0.171550	0.136819	0.148317	0.144708	0.092217	0.089954	0.112235
8	0.172664	0.137934	0.149432	0.145823	0.093332	0.091068	0.113349
9	0.260164	0.119882	0.131380	0.127770	0.075279	0.073016	0.095297
10	0.158022	0.198151	0.134790	0.131181	0.078690	0.076426	0.098707
11	0.157420	0.122690	0.214468	0.130578	0.078087	0.075824	0.098105
12	0.157544	0.122814	0.134312	0.209868	0.078211	0.075948	0.098229
13	0.159395	0.124665	0.136163	0.132554	0.142565	0.077799	0.100080
14	0.158909	0.124178	0.135676	0.132067	0.079576	0.144194	0.099594
15	0.158396	0.123666	0.135164	0.131554	0.079063	0.076800	0.170576
16	0.161130	0.126400	0.137898	0.134289	0.081798	0.079534	0.101815
17	0.160016	0.125285	0.136783	0.133174	0.080683	0.078420	0.100701
18	0.166340	0.131610	0.143108	0.139498	0.087007	0.084744	0.107025

In [115]:

```
# Store them in a new Excel file to be accessed later
all_cases.to_excel('Weights_cases.xlsx')
```

APPENDIX II: Determine the Best-Worst case scenarios of the alternatives' performance

In this appendix, the procedure of setting up the best-, middle- and worst-case scenarios for the MAMCA-analysis has been further explained. First, a short recap to the foregoing research is provided, introducing the alternative last mile delivery systems to choose from and the stakeholders' criteria. Based on this recap, the best-, middle- and worst-case scenarios can be determined.

Research recap:

In advance of the scenario development, this study has identified several interesting alternative systems to conduct last mile deliveries. These alternatives are summarised in Table 0-1.

Table 0-1: Alternative last mile systems

Alternative
Electrification of vehicle fleet
Cargo bikes
Evening and Night delivery
UCCs
CDPs and parcel lockers
Crowdsourcing logistics services

As the last mile delivery includes many different stakeholders, a stakeholder analysis has been conducted. Based on this analysis, an assessment of their different points of view with regard to the last mile problem has been conducted. These points of view have been translated into a set of criteria by which each of the alternative systems as defined in Table 0-1 can be analysed.

Before the analysis can be executed, the individual performance of each of the alternatives has to be known. The performance numbers have been obtained by both desk research and interviews to clarify the findings. Hence, each of the alternatives had an own Table in chapter 4, describing the individual performance with regard to the critiria.

However, some of these performance numbers were defined as a range. These numbers were uncertain and could only be treated as rough estimates. Hence, only conducting one analysis based on numbers with high uncertainty would yield uncertain and unreliable outcomes. Therefore, it was determined to apply a Best-Worst method with regard to the performance numbers, so that in both scenarios it would become clear what would be an optimal alternative (Rezaei, 2015).

APPENDIX II: Determine the Best-Worst case scenarios of the alternatives' performance

Determining the Best-Worst scenarios:

As mentioned before, the Best-Worst method was adopted to cover the uncertain variables along their complete range. Hence, the best case is defined as the case where the performance on the respective criterion is highest, while the worst case is defined as the case where the performance on the respective criterion is the lowest. Furthermore, a medium scenario has been defined to provide some more insight in the utilities if the ranges are averaged out.

Another important fact to mention is how the qualitative performance numbers are handled. Some of the performances could only be described by a qualitative assessment of the expert that was interviewed. However, these qualitative assessments also had to be translated to numerical values for the SMART analysis. Therefore, the following Table 0-2 was constructed:

Table 0-2: Qualitative values translation

Target:					
As low as possible	Nothing 1	Small 0.75	Moderate 0.5	High 0.25	Very high 0
Should decrease	Strongly decrease 1	Slightly decrease 0.75	Similar 0.5	Slightly increase 0.25	Strongly increase 0
Should increase	Strongly increase 1	Slightly increase 0.75	Similar 0.5	Slightly decrease 0.25	Strongly decrease 0

Based on the numerical performance and Table 0-2 for the qualitative performances, Table 0-3 was constructed:

APPENDIX II: Determine the Best-Worst case scenarios of the alternatives' performance

Table 0-3: Performances input table

Alternative / Criterion	EVs	Cargo bikes	CDPs and lockers	UCCs	Evening delivery	Night Delivery	Crowd-sourcing
Costs	0; 0; 0	0.1; 0.1; 0.1	0.35; 0.35; 0.35	0.25; 0.25; 0.25	0; 0; 0	0.4; 0.4; 0.4	0; 0; 0
Time	0; 0; 0	0.2; 0.35; 0.5	0.7; 0.7; 0.7	0; 0.125; 0.25	0.1; 0.175; 0.25	0.5; 0.5; 0.5	0; 0; 0
Emissions	0.3; 0.5; 0.7	0.1; 0.2; 0.3	0.4; 0.55; 0.6	0.2; 0.6; 1	0.25; 0.375; 0.5	0.7; 0.7; 0.7	0; 0.15; 0.3
Customer satisfaction	0.5; 0.5; 0.5	1; 1; 1	0.75; 0.75; 0.75	1; 1; 1	0.5; 0.625; 0.75	0.5; 0.5; 0.5	0.25; 0.25; 0.25
Safety at work	0.5; 0.5; 0.5	0; 0; 0	0.5; 0.5; 0.5	0.25; 0.25; 0.25	0.75; 0.75; 0.75	0.5; 0.625; 0.75	0.75; 0.75; 0.75
Security & responsibility	0.5; 0.5; 0.5	0; 0; 0	0; 0.125; 0.25	0.5; 0.5; 0.5	0.75; 0.75; 0.75	0; 0; 0	0; 0; 0
Traffic impact	0; 0; 0	0; 0.25; 0.5	0.2; 0.3; 0.4	0.45; 0.45; 0.45	0; 0; 0	0.25; 0.375; 0.5	0.2; 0.25; 0.3
Investments infrastructure	0; 0; 0	0.75; 0.75; 0.75	0.25; 0.5; 0.75	0; 0.125; 0.25	0; 0; 0	0; 0; 0	0.25; 0.375; 0.5
Policy sensitivity	0.5; 0.75; 1	1; 1	0.5; 0.625; 0.75	0.25; 0.375; 0.5	1; 1; 1	0.25; 0.25; 0.25	0; 0; 0

What can be noticed from Table 0-3 is that for some alternatives on certain criteria only one value is defined for all three the cases (best-, middle- and worst-case). This is due to the fact that these numbers were obtained by the desk research and have a proper reference in literature. Hence, there was no reason to presume these estimates were false. Neither was there an indication for the margin to consider. Therefore, only the value found in literature has been applied in Table 0-3. For all other rough estimates, the best-worst cases have been defined as previously explained.

APPENDIX II: Determine the Best-Worst case scenarios of the alternatives' performance

APPENDIX III: MAMCA-Analysis

Conducting the analysis

Based on the different cases generated in the foregoing notebook, this notebook tries to generate the results for all the different cases and tries to identify the most robust performing alternatives.

Importing libraries

In [1]:

```
import pandas as pd
import numpy as np
```

Reading the cases

In [2]:

```
WorstCase = pd.read_excel('Performance.xlsx', sheet_name='WorstCase')
BestCase = pd.read_excel('Performance.xlsx', sheet_name='BestCase')
InBetween = pd.read_excel('Performance.xlsx', sheet_name='InBetween')
```

In [3]:

```
WorstCase
```

Out[3]:

	Evs	CBs	CDPs and Lockers	UCCs	Evening Delivery	Night Delivery	Crowdsourcing
Delivery Cost	0.0	0.10	0.35	0.25	0.00	0.40	0.00
Delivery Time	0.0	0.20	0.70	0.00	0.10	0.50	0.00
Emissions reduction	0.3	0.10	0.40	0.20	0.25	0.70	0.00
Customer satisfaction	0.5	1.00	0.75	1.00	0.75	0.50	0.25
Safe working environment	0.5	0.00	0.50	0.25	0.75	0.75	0.75
Security and Responsibility	0.5	0.00	0.00	0.50	0.75	0.00	0.00
Traffic impact	0.0	0.50	0.20	0.45	0.00	0.25	0.20
Investments in infrastructure	0.0	0.75	0.25	0.00	0.00	0.00	0.25
Policy sensitivity	0.5	1.00	0.50	0.25	1.00	0.25	0.00

In [4]:

```
BestCase
```

Out[4]:

	Evs	CBs	CDPs and Lockers	UCCs	Evening Delivery	Night Delivery	Crowdsourcing
Delivery Cost	0.0	0.10	0.35	0.25	0.00	0.40	0.00
Delivery Time	0.0	0.50	0.70	0.25	0.25	0.50	0.00
Emissions reduction	0.7	0.30	0.70	1.00	0.50	0.70	0.30
Customer satisfaction	0.5	1.00	0.75	1.00	0.75	0.50	0.25
Safe working environment	0.5	0.00	0.50	0.25	1.00	1.00	0.75
Security and Responsibility	0.5	0.00	0.00	0.50	0.75	0.00	0.00
Traffic impact	0.0	0.50	0.40	0.45	0.00	0.50	0.30
Investments in infrastructure	0.0	0.75	0.75	0.25	0.50	0.00	0.50
Policy sensitivity	1.0	1.00	0.50	0.50	1.00	0.25	0.00

In [5]:

```
InBetween
```

Out[5]:

	Evs	CBs	CDPs and Lockers	UCCs	Evening Delivery	Night Delivery	Crowdsourcing
Delivery Cost	0.00	0.10	0.350	0.250	0.000	0.400	0.000
Delivery Time	0.00	0.35	0.700	0.125	0.175	0.500	0.000
Emissions reduction	0.50	0.20	0.550	0.600	0.375	0.700	0.150
Customer satisfaction	0.50	1.00	0.750	1.000	0.500	0.500	0.250
Safe working environment	0.50	0.00	0.500	0.250	0.875	0.625	0.750
Security and Responsibility	0.50	0.00	0.125	0.500	0.750	0.000	0.000
Traffic impact	0.00	0.50	0.300	0.450	0.000	0.375	0.250
Investments in infrastructure	0.00	0.75	0.500	0.125	0.000	0.000	0.375
Policy sensitivity	0.75	1.00	0.625	0.375	1.000	0.250	0.000

Beside these scorecards, the different cases with weights for the criteria have to be loaded

In [6]:

```
Cases = pd.read_excel('Weights_Cases.xlsx')
Cases
```

Out[6]:

	Delivery Cost	Delivery Time	Emissions reduction	Customer satisfaction	Safe working environment	Security and Responsibility	Traffic impact
0	0.072517	0.143338	0.154836	0.151226	0.098735	0.096472	0.118753
1	0.174658	0.065069	0.151425	0.147816	0.095325	0.093062	0.115343
2	0.175260	0.140530	0.071747	0.148418	0.095927	0.093664	0.115945
3	0.175136	0.140406	0.151904	0.069129	0.095804	0.093540	0.115821
4	0.173285	0.138554	0.150052	0.146443	0.031449	0.091689	0.113970
5	0.173771	0.139041	0.150539	0.146930	0.094439	0.025294	0.114456
6	0.174284	0.139554	0.151052	0.147442	0.094951	0.092688	0.043474
7	0.171550	0.136819	0.148317	0.144708	0.092217	0.089954	0.112235
8	0.172664	0.137934	0.149432	0.145823	0.093332	0.091068	0.113349
9	0.260164	0.119882	0.131380	0.127770	0.075279	0.073016	0.095297
10	0.158022	0.198151	0.134790	0.131181	0.078690	0.076426	0.098707
11	0.157420	0.122690	0.214468	0.130578	0.078087	0.075824	0.098105
12	0.157544	0.122814	0.134312	0.209868	0.078211	0.075948	0.098229
13	0.159395	0.124665	0.136163	0.132554	0.142565	0.077799	0.100080
14	0.158909	0.124178	0.135676	0.132067	0.079576	0.144194	0.099594
15	0.158396	0.123666	0.135164	0.131554	0.079063	0.076800	0.170576
16	0.161130	0.126400	0.137898	0.134289	0.081798	0.079534	0.101815
17	0.160016	0.125285	0.136783	0.133174	0.080683	0.078420	0.100701
18	0.166340	0.131610	0.143108	0.139498	0.087007	0.084744	0.107025

1. Iterating through weights' cases

The first step in the analysis is to generate the utilities of the alternatives for each of the different cases of the weights. Thereby, for each performance case (Best, Medium, Worst), 18 different outcomes are calculated. These are analysed further in step 2.

Anyway, step 1 first defines the *analysis* function. This function does the same calculations for all the three performance cases. First, it defines an empty dictionary in which the calculated outcomes are stored. Then, the function iterates through the cases with the weights for the criteria, and the utilities of each alternative are calculated for each case. These outcomes are finally stored in the dictionary and returned for later reference.

In [7]:

```
# Small preparation
WorstCase.index = BestCase.index = InBetween.index = Cases.columns
```

In [8]:

```
def analysis(performances, weights):
    outcomes_dict = {}
    for idx, row in weights.iterrows():
        outcomes = performances.multiply(row, axis=0)
        scores = outcomes.sum(axis=0)
        scores.name = 'Total Score'
        outcomes = outcomes.append(scores)
        outcomes_dict['Case%s' %idx] = outcomes

    return outcomes_dict
```

In [9]:

```
BC_out = analysis(BestCase, Cases)
WC_out = analysis(WorstCase, Cases)
IB_out = analysis(InBetween, Cases)
```

Analyse the outcomes

So, for all of the alternatives the utilities for all different cases have been calculated and stored in the corresponding dictionaries. However, these outcomes have to be analysed a second time. Moreover, this second step of the analysis calculates the mean and the standard deviation of the alternatives.

The function is shown below.

In [10]:

```
def analyse_scores(outcomes):
    scoredf = pd.DataFrame(columns=outcomes['Case0'].columns)
    for case in outcomes.keys():
        df = outcomes[case]
        scores = df.loc['Total Score']
        scoredf = scoredf.append(scores)
    avg_score = scoredf.mean(axis=0)
    stdev = scoredf.std(axis=0)

    return avg_score, stdev
```

In [11]:

```
BC_scores, BC_dev  = analyse_scores(BC_out)
WC_scores, WC_dev = analyse_scores(WC_out)
IB_scores, IB_dev = analyse_scores(IB_out)
```

In [12]:

```
BC_scores
```

Out[12]:

```
Evs                 0.330921
CBs                 0.442663
CDPs and Lockers    0.528179
UCCs                0.523326
Evening Delivery    0.467540
Night Delivery      0.461565
Crowdsourcing       0.207944
dtype: float64
```

In [13]:

```
BC_dev
```

Out[13]:

```
Evs                 0.024623
CBs                 0.026588
CDPs and Lockers    0.016469
UCCs                0.022668
Evening Delivery    0.028068
Night Delivery      0.018789
Crowdsourcing       0.016944
dtype: float64
```

Lets show all the results in separate figures to see the dominant alternatives

In [14]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

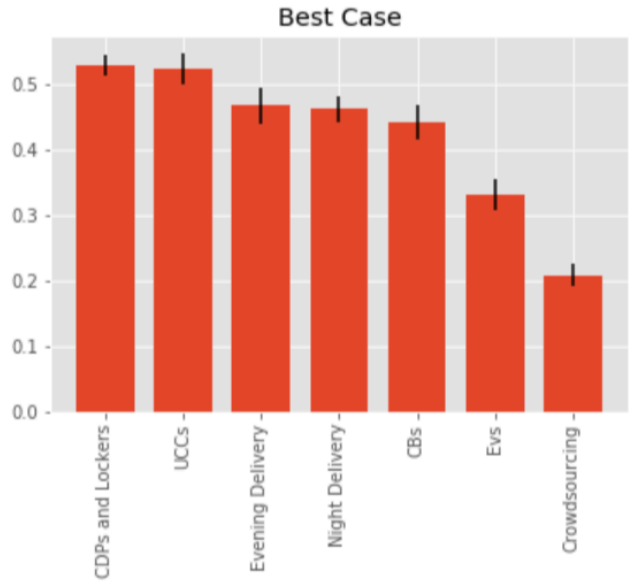
Plot the outcomes

In [15]:

```
def plot(scores, devs):
    df = pd.concat([scores, devs], axis=1)
    df.columns = ['Scores', 'Standard Deviations']
    df.index = scores.index

    df = df.sort_values(by='Scores', ascending=False)

    x = np.arange(0, len(df)*2, 2)

    plt.style.use('ggplot')
    plt.xticks(rotation=90)
    plt.title(scores.name)
    plt.bar(x, df['Scores'].values, tick_label=df.index, width=1.5, yerr=df['Sta
ndard Deviations'].values)
```

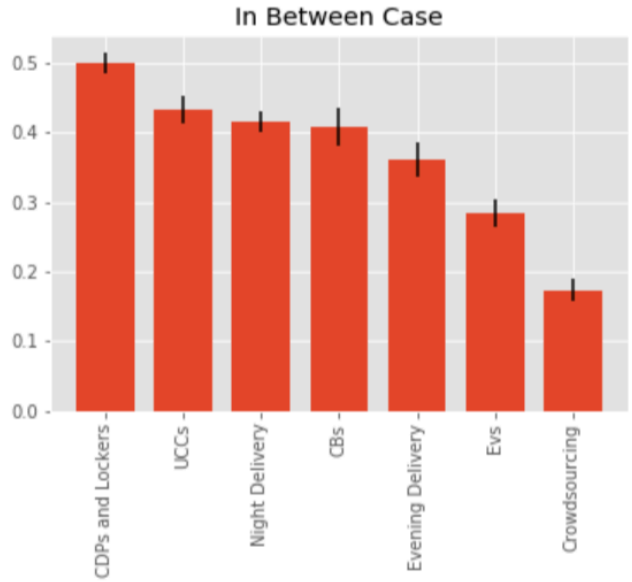
In [16]:

```
BC_scores.name = 'Best Case'
plot(BC_scores, BC_dev)
```



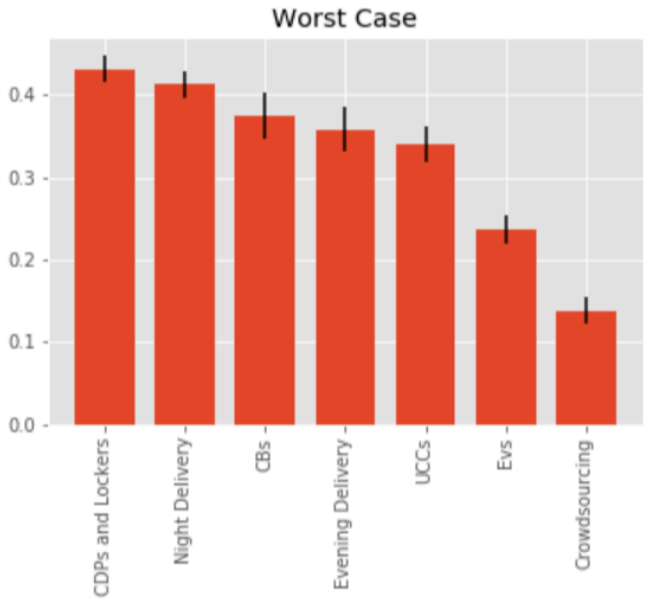
In [17]:

```
IB_scores.name = 'In Between Case'
plot(IB_scores, IB_dev)
```



In [18]:

```
WC_scores.name = 'Worst Case'
plot(WC_scores, WC_dev)
```



APPENDIX IV: Analysing Amsterdam-PostNL Data

Analysing parcel delivery in Amsterdam

In this notebook, the file with data provided by PostNL is analysed to see what a 'normal' day of parcel delivery in Amsterdam looks like. Furthermore, it is investigated where most of the parcel are delivered and if this distribution matches the distribution of inhabitants as found in the Amsterdam population data. After reading this Notebook it becomes clear how many parcels are being delivered on a daily basis in each postal code area in Amsterdam, and how these parcels are distributed among the postal code areas.

Importing the necessary libraries

In [1]:

```
import numpy as np
import pandas as pd
```

Import and process the file

In [2]:

```
# Define the file name to open
fn = '../data/postnl_data_improved.csv'
```

In [3]:

```
# Store the file in a pandas DataFrame
df = pd.read_csv(fn, index_col=0)
```

```
/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshel
l.py:2698: DtypeWarning: Columns (1) have mixed types. Specify dtype
option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
/anaconda3/lib/python3.6/site-packages/numpy/lib/arraysetops.py:472:
FutureWarning: elementwise comparison failed; returning scalar inste
ad, but in the future will perform elementwise comparison
  mask |= (ar1 == a)
```

In [4]:

```
# Reset the columns names to readible names
df.columns = ['year sequence', 'year iso', 'month sequence', 'month iso',
              'day sequence', 'date', 'landcode', 'postcode', 'woonplaats',
              'straatnaam', 'huisnummer', 'huisnummer toevoeging', 'aantal coll
i',
              'gemiddeld gewicht', 'gemiddelde lengte', 'gemiddelde breedte',
              'gemiddelde hoogte', 'gemiddeld volume']

# Extract only the necessary ones for the analysis
df = df[['date', 'postcode', 'woonplaats', 'straatnaam', 'huisnummer',
         'aantal colli', 'gemiddeld gewicht', 'gemiddelde lengte',
         'gemiddelde breedte', 'gemiddelde hoogte', 'gemiddeld volume']]
```

Calculating obvious statistics

For this research, we are mainly interested in the average values regarding colli per address (number of parcels), parcels per day and the average dimensions. Therefore, these statistics are calculated first, as these are the most straightforward based on the data in the DataFrame.

In [5]:

```
# Colli per address
max_colli = df['aantal colli'].max()
min_colli = df['aantal colli'].min()
avg_colli = df['aantal colli'].mean()
std_colli = df['aantal colli'].std()
```

In [6]:

```
# Dimensions per address
avg_lengte = df['gemiddelde lengte'].mean()
std_lengte = df['gemiddelde lengte'].std()
avg_breedte = df['gemiddelde breedte'].mean()
std_breedte = df['gemiddelde breedte'].std()
avg_hoogte = df['gemiddelde hoogte'].mean()
std_hoogte = df['gemiddelde hoogte'].std()
```

In [7]:

```
# Weight and volume
avg_weight = df['gemiddeld gewicht'].mean()
std_weight = df['gemiddeld gewicht'].std()
avg_volume = df['gemiddeld volume'].mean()
std_volume = df['gemiddeld volume'].std()
```

Determine macro statistics

In explicit: What is the average number of parcels that is being delivered in the city of Amsterdam each day? Furthermore, what is the distribution of parcels among the postal code areas? To get more insight in the demand for postal code areas, the postal codes will be transformed to *pc4*-format, as this matches the simulation model.

First, the data has to be sorted by date. Then, pandas can group the data by date and sum all the parcels per date. Hence, the average number of parcels per day can be calculated.

In [8]:

```
df = df.sort_values(by='date')
```

In [9]:

```
colli_per_day = df.groupby('date')['aantal colli'].sum()
avg_colli_per_day = round(colli_per_day.mean())
std_colli_per_day = round(colli_per_day.std())
max_colli_per_day = colli_per_day.max()
min_colli_per_day = colli_per_day.min()
```

In [10]:

```
# Check the minimum and maximum values for outliers
print('Minimum:', min_colli_per_day)
print('Maximum:', max_colli_per_day)
print('Average:', avg_colli_per_day)
```

```
Minimum: 0.0
Maximum: 60189.0
Average: 24429
```

More than 60.000 parcels per day seems a little bit too much, but still reasonable considering that the average is already about 24.000. This may be a special celebration day or something, like christmas, and therefore this number is still reasonable. However, a minimum of 0 could just as well be excluded from the sample. Moreover, there may be more days that 0 parcels were registered to be delivered. These will be excluded from the analysis. Other low values may also be special deliveries which can be taken into account.

In [11]:

```
todrop = colli_per_day[colli_per_day == min_colli_per_day].index.values
idx = df[df['date'].isin(todrop)].index

df.drop(idx, inplace=True)
```

In [12]:

```
# Recalculate the average
colli_per_day = df.groupby('date')['aantal colli'].sum()
avg_colli_per_day = round(colli_per_day.mean())
std_colli_per_day = round(colli_per_day.std())
max_colli_per_day = colli_per_day.max()
min_colli_per_day = colli_per_day.min()
```

In [13]:

```
# Check the minimum and maximum values for outliers
print('Minimum:', min_colli_per_day)
print('Maximum:', max_colli_per_day)
print('Average:', avg_colli_per_day)
```

```
Minimum: 35.0
Maximum: 60189.0
Average: 27121
```

In [14]:

```
print('Average colli per day:', avg_colli_per_day)
print('Standard deviation:', std_colli_per_day)
```

```
Average colli per day: 27121
Standard deviation: 14573
```

Furthermore, since this is a large enough dataset, we can check whether or not the data is normally distributed. We can use the *scipy* library for that, since it includes the shapiro test.

If the data is normally distributed, the model will be applied with a normal distribution to sample the daily demand from. If not, the demand will be sampled from a uniform distribution instead.

In [15]:

```
# Check if sample is normally distributed
from scipy.stats import shapiro

stats, p = shapiro(colli_per_day.values)
alpha = 0.05
if p > alpha:
    print('Data set is likely to be normally distributed', p)
else:
    print('Data set is likely not to be normally distributed', p)
```

```
Data set is likely not to be normally distributed 4.309885917120129e
-14
```

So now, the maxima and minima of the obvious variables have to be determined again.

In [16]:

```
# Colli per address
max_colli = df['aantal colli'].max()
min_colli = df['aantal colli'].min()
avg_colli = df['aantal colli'].mean()
std_colli = df['aantal colli'].std()

# Dimensions per address
avg_lengte = df['gemiddelde lengte'].mean()
std_lengte = df['gemiddelde lengte'].std()
avg_breedte = df['gemiddelde breedte'].mean()
std_breedte = df['gemiddelde breedte'].std()
avg_hoogte = df['gemiddelde hoogte'].mean()
std_hoogte = df['gemiddelde hoogte'].std()

# Weight and volume
avg_weight = df['gemiddeld gewicht'].mean()
std_weight = df['gemiddeld gewicht'].std()
avg_volume = df['gemiddeld volume'].mean()
std_volume = df['gemiddeld volume'].std()
```

Transforming the postal codes to 4pp

In [17]:

```
cp = df['postcode']
cp = cp.str.slice(0, 4)
```

In [18]:

```
df['postcode'] = cp
df_pcs = df.groupby('postcode')['aantal colli'].sum()
```

Now, a distribution can be calculated, which is equal to normalising the data

In [19]:

```
distribution = df_pcs/df_pcs.sum()
```

In [20]:

```
# Combine them together (normal and normalised) in a DataFrame
df_pcs.name = 'absolute numbers'
distribution.name = 'normalised'

df_pcs_distributed = pd.concat([df_pcs, distribution], axis=1)
```

Compare the distribution to distribution of population

Setting up this part of the analysis is pretty similar to the start of the notebook. Opening the specified file and filter the right data out of it. That is it.

In [21]:

```
# First, read the original file into memory
# And copy the analysis of the model setup
adam_file = '../data/bevolkingscijfers_amsterdam.xls'
adamdf = pd.read_excel(adam_file, header=[0,1])
adamdf.columns = adamdf.columns.droplevel(-1)
```

In [22]:

```
# Filter for the right informatino in the df
adamdf = adamdf[adamdf['Soort regio'] == 'Buurt']

# Assess all the available postal code
pcs = adamdf['']Meest voorkomende postcode'].unique()
# Postal codes with missing information will be deleted
pcs = pcs[np.where(pcs != '.')]
adamdf = adamdf.replace('.', np.nan)
```

In [23]:

```
# Preparing the df for the calculated variables
variables = ['Population', 'Population density', 'Horeca',
             'Horeca density', 'Companies', 'Company density']
# Define the empty df
newdf = pd.DataFrame(columns=variables)

for pc in pcs:
    recs = adamdf[adamdf['||Meest voorkomende postcode'] == pc]
    inw = recs['||Aantal inwoners'].sum()
    hor = recs['|||G+I Handel en horeca'].sum()
    ovb = (recs['||Bedrijfsvestigingen totaal'] - recs['|||G+I Handel en horeca'
]).sum()

    area = recs['Oppervlakte land'].sum()

    inw_dens = inw/area
    hor_dens = hor/area
    ovb_dens = ovb/area

    s = pd.Series([inw, inw_dens, hor, hor_dens, ovb, ovb_dens],
                  index=variables, name=pc)
    newdf = newdf.append(s)

newdf.sort_index(inplace=True)

# Calculate the distribution of inhabitants
for var in ['Population', 'Horeca', 'Companies']:
    newdf[var+' distribution'] = newdf[var] / newdf[var].sum()

# Change the order of columns for correctness
cols = ['Population', 'Population density', 'Population distribution',
        'Horeca', 'Horeca density', 'Horeca distribution',
        'Companies', 'Company density', 'Companies distribution']
newdf = newdf[cols]
```

Select the same postal codes as the PostNL file

In [24]:

```
# The postal codes of the Amsterdam file have already been filtered properly.
# Therefore, these postal codes will be taken as base
pcs_to_select = newdf.index.values
pcs_to_select = pcs_to_select.astype(str)
newdf.index = pcs_to_select

df_pcs_distributed = df_pcs_distributed.loc[pcs_to_select]
```

In [25]:

```
df_pcs_distributed = pd.concat([df_pcs_distributed,
                                newdf['Population distribution']], axis=1)
```

In [26]:

```
df_pcs_distributed['distribution differences'] = (df_pcs_distributed['normalise
d'] - df_pcs_distributed['Population distribution']).abs()
```

The *Scipy* library also offers a function to check the similarities between the distribution of two samples. This is shown below.

In [28]:

```
from scipy.stats import ks_2samp

res = ks_2samp(df_pcs_distributed['normalised'].values,
               df_pcs_distributed['Population distribution'])
res
```

Out[28]:

```
Ks_2sampResult(statistic=0.1375, pvalue=0.40877808861325815)
```

From these results, it turns out that both the distributions are statistically significant the same. Therefore, it is right to use either one of the distributions in the model to predict the demand in certain areas. Hence, the model remains unchanged in regard to the distribution of parcels.

APPENDIX V: Model class

Defining the Model class and its operations

The *Model* class is the most high level class in the simulation model and also represents the simulation model in Python. The *Model* class basically controls the instances and their events as they are defined within the model. This Notebook explains in detail how the *Model* class is defined, what information is needed and what operations and events can occur during the simulation.

Normally, a class is defined in only one block of code. This Notebook divides the class and its functions however among multiple code blocks to emphasize more on the functions in the class. Hence, this Notebook is for educational purposes only and cannot be used to run simulations on its own.

Import the necessary libraries

In [ ]:

```
import numpy as np
import pandas as pd
import geopandas as gpd
from shapely.geometry import Polygon, MultiPolygon, Point

import setup # The setup script, covered in another Notebook.
```

As can be noticed from the imported libraries, one of the imports is the *setup* script. This script has been created especially for this simulation model to keep the initiation function clean and tidy. The *setup* script can be found in the file *setup.py*, which is provided in the *Model* directory of this study. Furthermore, the *setup* script is covered in another Notebook within this study.

Initialising the *Model* class

Anyway, the *setup* script provides the simulation model with the necessary data to conduct the simulation experiments. These are the following variables, dictionaries or DataFrames:

- The Dashboard
- The Shapefile
- The Population data
- Postal codes and their locations
- Locker locations
- The eventual 'Milieuzone'

Based on these variables, different settings with regard to the configuration of the simulation model will be set in the initiation function. This is covered in the code block below:

In [ ]:

```
class Model:
    def __init__(self, *args): # The args are provided with the experiments
        # First, store the outcomes of the setup in memory
        self.db, self.shapes, self.data, self.pcs,
        self.locker_locs, self.milieuzone = setup.main()

        # This should actually be executed in the setup, but anyway
        # the format of the postal codes has to be changed
        self.pcs = np.array(self.pcs)
        self.pcs = self.pcs.astype(str)

        # Extract information from the dashboard, depending on the args
        # provided
        if len(args) == 2:
            self.BasicInfo = args[0]
            self.Settings = args[1]
            self.VehicleInfo = self.db['VehicleInfo']
        elif len(args) == 3:
            self.BasicInfo = args[0]
            self.Settings = args[1]
            self.VehicleInfo = args[2]
        else:
            self.BasicInfo = self.db['BasicInfo']
            self.Settings = self.db['Settings']
            self.VehicleInfo = self.db['VehicleInfo']

        # Storing the locations of Sorting Centres and City Hub
        self.sc_locs = self.db['Locations']

        # Set the switches for City Hub, Parcel Lockers and Milieuzone
        if self.Settings['Setting'][self.Settings['Var']\
           == 'City Hub'].values[0] == 'True':
            self.switch_CH = True
        else: self.switch_CH = False

        if self.Settings['Setting'][self.Settings['Var']\
           == 'Parcel Lockers'].values[0] == 'True':
            self.switch_PL = True
        else: self.switch_PL = False

        if self.Settings['Setting'][self.Settings['Var']\
           == 'Milieuzone'].values[0] == 'True':
            self.switch_MZ = True
        else: self.switch_MZ = False

        # Call the function to create the Distribution Centre instances
        self.create_dcs()

        # Define the different shifts throughout the day
        self.dayshift = np.array([self.BasicInfo['Value'][self.BasicInfo['Variable'
                                  == 'Start day shift'].values[0],
                                  self.BasicInfo['Value'][self.BasicInfo['Variable'
                                  == 'End day shift']])
        self.eveningshift = np.array([self.BasicInfo['Value'][self.BasicInfo['Variab
                                      == 'Start evening shift'].values[0],
                                      self.BasicInfo['Value'][self.BasicInfo['Variab
                                      == 'End evening shift']])
        self.lineshift = np.array([self.BasicInfo['Value'][self.BasicInfo['Variable
                                   == 'Start line shift'].values[0],
```

```
                                    self.BasicInfo['Value'][self.BasicInfo['Variable
                                    == 'End line shift']])

        # Define the model variables
        self.ParcelsInSystem = 0
        self.ParcelsDelivered = 0

        self.clock = 0

        # Define timing variables for the Model's events
        self.t_demand = 0
        self.t_event = 0

        # Setup the model events
        self.setup_events()
```

Functions defined for initialisation

As can be noticed from the foregoing **init** function, initialising the *Model* instance requires two other functions. Firstly, the *create_dcs()* function, which defines the Sorting Centre instances and the optional City Hub. Secondly, the *setup_events()* function, which assesses all the events from the different instances (either *Distribution Centre* or *Vehicle* class) and stores them inside the model instance. Both of these functions are explained in the code blocks below.

In [ ]:

```
    def create_dcs(self):
        # First, assess the number of transporters to include
        n_trans = self.Settings['Setting'][self.Settings['Var'] \
                                           == 'Transporters'].values[0]
        scs = self.sc_locs.iloc[:n_trans]

        # Check if a City Hub has to be implemented
        if self.switch_CH:
            ch = self.sc_locs[self.sc_locs['Location'].str.contains('City Hub')]

        # Define empty dictionaries for further reference
        self.SortingCentres = {}
        self.CityHubs = {}

        for i in range(len(scs)):
            # Get the correct row in the DataFrame
            sc = scs.iloc[i]
            # Create the DC instance
            SC = DistributionCentre(sc['Location'], sc[['Lat', 'Lon']],
                                    sc['Market share'], self)
            # Store a reference to the instance in the dictionary
            self.SortingCentres[SC.name] = SC

        if self.switch_CH:
            # Create the City Hub instance
            CH = DistributionCentre(ch['Location'], ch[['Lat', 'Lon']].values[0],
                                    ch['Market share'], self)
            # Store a reference to the instance in the dictionary
            self.CityHubs[CH.name] = CH
```

In [ ]:

```
    def setup_events(self):
        # First, define the Model's own events
        self.events = [self.GenerateDemand]
        # Add Sorting Centres' events to the list
        for sc in self.SortingCentres.values():
            self.events += sc.events
            # Add the events of the corresponding vehicles
            for v in sc.vehicles.values():
                self.events += v.events

        # Add the City Hub's events to the list
        for ch in self.CityHubs.values():
            self.events += ch.events
            # Add the events of the corresponding vehicles
            for v in ch.vehicles.values():
                self.events += v.events
```

Defining the Model's events

The initialisation functions of the *Model* class have been properly addressed. So now, it is time to define the simulation events of the *Model* class. To make it easy, there is only one event that occurs within the *Model* class itself: The *GenerateDemand()* event.

This event generates the demand for one day of parcel deliveries, based on the data PostNL provided about their deliveries in Amsterdam. The dashboard offers a lower limit and an upper limit of demand to fill in by the user. The simulation model then draws a number from a uniform distribution between the lower and upper limit. This number represents the total demand for the day. The demand is then split among the different transporters that are included in the simulation. The demand per transporting company is then split further in *day demand, evening demand, locker demand* and *collection demand*.

The demand for lockers is then distributed among the defined lockers based on the probability distribution (based on the distribution of the population) of the parcel having to be delivered in the respective postal code area the parcel locker is placed in.

The demand for day deliveries, evening deliveries and collection is distributed randomly among the postal code areas, based on the same (population) distribution. For simplicity, either 1, 2 or 3 parcels will be delivered per destination, since the average number of parcels per address according to PostNL is about 1.4. Finally, each destination gets a unique name and location for further reference.

Besides, if a City Hub is used, the demand for each of the Sorting Centres will be consolidated at the City Hub. In explicit, all the individual demand DataFrames will be combined. However, if the City Hub is only to be used to deliver inside the '*Milieuzone*', then only the demand within this zone is combined in one demand DataFrame. This can be found at the bottom of the following code block.

In [ ]:

```
    def GenerateDemand(self):
        # Assess the lower and upper limit for demand from the dashboard
        lower_limit = self.BasicInfo['Value'][self.BasicInfo['Variable']\
                      == 'Minimum parcels'].values[0]
        upper_limit = self.BasicInfo['Value'][self.BasicInfo['Variable']\
                      == 'Maximum parcels'].values[0]

        # Draw the sample
        amount = np.random.randint(lower_limit, upper_limit)
        # Define the probability distribution
        probs = self.data['Population distribution'].values

        # Define the percentages of demand to split
        collection_perc = self.BasicInfo['Value'][self.BasicInfo['Variable']\
                          == 'Percentage afroming demand'].values[0]
        collection_amount = int(round(collection_perc /100 * amount))
        evening_perc = self.BasicInfo['Value'][self.BasicInfo['Variable']\
                       == 'Percentage evening deliveries'].values[0]

        if self.switch_PL:
            locker_perc = self.BasicInfo['Value'][self.BasicInfo['Variable']\
                          == 'Percentage lockers demand'].values[0]
        else: locker_perc = 0

        # Generate the demand for each individual Sorting Centre
        for sc in self.SortingCentres.values():
            # Determine the resulting amounts per type/shift
            d = int(round(sc.marketshare / 100 * amount))
            dloc = int(round(locker_perc / 100 * d))
            deve = int(round(evening_perc / 100 * d))
            dday = d - dloc - deve
            dcol = int(round(sc.marketshare / 100 * collection_amount))

            # Update the Parcels in stock in the SC
            sc.ParcelsInStock += (dloc + dday + deve)
            # Update Parcels in system of whole model
            self.ParcelsInSystem += (dloc + dday + deve + dcol)

            # Generate locker demand, sampling postal codes
            pcloc = np.random.choice(self.pcs, size=dloc, p=probs)
            tv, c = np.unique(pcloc, return_counts=True)
            demand_lockers = dict(zip(tv, c))

            # Divide locker demand among the lockers
            for pc in demand_lockers.keys():
                n = demand_lockers[pc]
                # Filter the lockers in the selected postal code area
                lockers = self.locker_locs[self.locker_locs['Postcode'] = int(pc)]
                # Randomly choose the lockers in this area
                lockers = np.random.choice(lockers['Name'].values, size=n)
                tv, c = np.unique(lockers, return_counts=True)

                # Create the DataFrame with Locker Demand
                for i, t in enumerate(tv):
                    name = '{} {}'.format(sc.name.split(' ')[-1], t)
                    if name in sc.DayDestinations.index:
                        sc.DayDestinations['Amount'].loc[name] = sc.DayDestinations|
                    else:
                        locs = self.locker_locs[['LAT', 'LON']][self.locker_locs['Na
```

```
                s = pd.Series([locs[0], locs[1], pc, c[i]],
                              index=['LAT', 'LON', 'Postal Code', 'Amount'],
                              name=name)
                sc.DayDestinations = sc.DayDestinations.append(s)

    # Generate day demand
    pcday = np.random.choice(self.pcs, size=dday, p=probs)
    tv, c = np.unique(pcday, return_counts=True)
    demand_day = dict(zip(tv, c))

    for pc in demand_day.keys():
        n = demand_day[pc]
        # Assess the polygon from shapefile, generate random points inside
        poly = self.shapes['geometry'][self.shapes['pc4txt'] == pc]
        # If it is a Polygon object points can immediately be generated
        # For a MultiPolygon, these actions have to be executed
        # multiple times
        if isinstance(poly, Polygon):
            # Extract the outer coordinates of the Polygon
            xmin, ymin, xmax, ymax = poly.bounds
            i = 0
            while n > 0: # i.e. as long as there's demand left
                c = np.random.choice([1,2,3], p=[0.8, 0.15, 0.05])
                if c > n:
                    # Last parcels
                    c = n
                # Generate random point
                point = [np.random.uniform(xmin, xmax),
                         np.random.uniform(ymin, ymax)]
                p = Point(point)
                # Check if the point is within the Polygon
                if p.within(poly):
                    n -= c
                    lon = point[0]
                    lat = point[1]
                    name = '{} Destination {} day {}'.format(sc.name.split(
                                                             pc, i)
                    # Prevent same names from entering the DataFrame
                    if name in sc.DayDestinations.index:
                        i += 1
                        name = '{} Destination {} day {}'.format(sc.name.sp
                                                                 pc, i)
                    s = pd.Series([lat, lon, pc, c],
                                  index=['LAT', 'LON', 'Postal code', 'Amou
                                  name=name)
                    sc.DayDestinations = sc.DayDestinations.append(s)
                    i += 1
        else: # Instance is MultiPolygon
            i = 0
            for part in poly:
                xmin, ymin, xmax, ymax = poly.bounds
                n2 = int(round(n/len(poly)))
                while n2 > 0:
                    c = np.random.choice([1,2,3], p=[0.8, 0.15, 0.05])
                    if c > n2:
                        # Last parcels
                        c = n2
                    # Generate random point
                    point = [np.random.uniform(xmin, xmax),
                             np.random.uniform(ymin, ymax)]
                    p = Point(point)
```

```
                        # Check if the point is within the Polygon
                        if p.within(poly):
                            n2 -= c
                            lon = point[0]
                            lat = point[1]
                            name = '{} Destination {} day {}'.format(sc.name.sp
                                                                     pc, i)
                            # Prevent same names from entering the DataFrame
                            if name in sc.DayDestinations.index:
                                i += 1
                                name = '{} Destination {} day {}'.format(sc.name
                                                                         pc, i)
                            s = pd.Series([lat, lon, pc, c],
                                          index=['LAT', 'LON', 'Postal code', '
                                          name=name)
                            sc.DayDestinations = sc.DayDestinations.append(s)
                            i += 1
        """
        The same algorithm is repeated twice more to generate
        the evening and collection demand

        The evening demand is stored in the EveningDestinations DataFrame
        The collection demand is appended to DayDestinations

        After that, the names in the DataFrames are made unique
        once again by the following lines of code.
        """

        sc.DayDestinations.index = self.DayDestinations.index \
                                 + sc.DayDestinations.groupby(level=0).cumcount
        sc.EveningDestinations.index = self.EveningDestinations.index\
                                     + self.EveningDestinations.groupby(level=0

        sc.linehaulremaining = 0

    # Check if the City Hub option is switched on
    if self.switch_CH:
        ch = self.CityHubs['City Hub']
        for sc in self.SortingCentres.values():
            # Distinguish the Milieuzone cases
            if self.switch_MZ:
                day_ta = sc.DayDestinations[sc.DayDestinations.index.map(lambda
                eve_ta = sc.EveningDestinations[sc.EveningDestinations.index.map
                # Calculate line haul
                sum1 = day_ta['Amount'].sum()
                sum1 += eve_ta['Amount'].sum()

                sc.linehaulremaining = sum1

                sc.DayDestinations.drop(day_ta.index, inplace=True)
                sc.EveningDestinations.drop(eve_ta.index, inplace=True)

                ch.DayDestinations = ch.DayDestinations.append(day_ta)
                ch.EveningDestinations = ch.EveningDestinations.append(eve_ta)

            else:
                ch.DayDestinations = ch.DayDestinations.append(sc.DayDestinatio
                ch.EveningDestinations = ch.EveningDestinations.append(sc.Eveni

                sc.linehaulremaining = sc.ParcelsInStock
                # Reset SC demand
```

```
                dfcols = sc.DayDestinations.columns
                sc.DayDestinations = pd.DataFrame(columns=dfcols)
                sc.EveningDestinations = sc.DayDestinations.copy()

            # Make names unique
            ch.DayDestinations.index = ch.DayDestinations.index \
                                     + ch.DayDestinations.groupby(level=0).cumcount
            ch.EveningDestinations.index = ch.EveningDestinations.index \
                                         + ch.EveningDestinations.groupby(level=0).c


        # Then the Origin-Destination matrices can be generated
        # This is covered in the following code block
        self.create_odm()

        # Update the timing for the event
        self.t_demand += 24
```

As mentioned in the *GenerateDemand()* event, the operation is not yet completely defined. A so-called origin-destination matrix (ODM) has to be created that defines all the distances between all destinations/locations in the current system. Since the *create_odm()* function is part of the *GenerateDemand()* event, the function is explained in the code block below.

However, before showing the code of the *create_odm()* function, some important notifications have to be made. Firstly, the distances are based on the Euclidian distances between the locations. Thus, no road network is taken into account in the simulation. The Euclidian distances can be calculated with the **Haversine** formula, which calculates the distance between two (Latitude, Longitude) coordinates on the earth, compensating for the spherical shape of the earth (Hedges, 2002). This function has been processed into the simulation model as well and the code is shown first.

In [ ]:

```
    def distances(self, arr):
        """The function gets a 4xn matrix with the Latitudes/Longitudes of
        the destinations and the Latitude/Longitude of the origin.
        It calculates the distances between the origin and all destinations"""
        r = 6371 # Radius of the earth

        a = (np.sin(arr[:,4]/2))**2 + np.cos(arr[:,2]) * np.cos(arr[:,0])\
            * (np.sin(arr[:,5])/2)**2
        c = 2 * np.arcsin(np.sqrt(a))
        d = r * c

        return d
```

In [ ]:

```
    def create_odm(self):
        # Check for the city hub option
        if self.switch_CH:
            ch = self.CityHubs['City Hub']
            ch.dayremaining = list(ch.DayDestinations.index)
            ch.eveningremaining = list(ch.EveningDestinations.index)
            # Specifically store the LATS/LONS
            day = ch.DayDestinations[['LAT', 'LON']]
            evening = ch.EveningDestinations[['LAT', 'LON']]

            # Store information as a Series for later use
            s = pd.Series([ch.location[0], ch.location[1]],
                          index=['LAT', 'LON'], name=ch.name)
            day = day.append(s)
            evening = evening.append(s)

            # Do the same for all the sorting centres
            for sc in self.SortingCentres.values():
                s = pd.Series([sc.location[0], sc.location[1]],
                              index=['LAT', 'LON'], name=sc.name)
                day = day.append(s)
                evening = evening.append(s)
                sc.dayremaining = list(sc.DayDestinations.index)
                sc.eveningremaining = list(sc.EveningDestinations.index)


            # Defining the empty odm
            # The day variable contains all the lat/lon coordinates
            day_odm = np.zeros((len(day), len(day)))
            eve_odm = np.zeros((len(evening), len(evening)))

            for i, row in enumerate(day.index):
                # Store the lat/lon of the origin
                orig_lat = day.loc[row]['LAT']
                orig_lon = day.loc[row]['LON']
                day['OR_LAT'] = orig_lat
                day['OR_LON'] = orig_lon

                # Transform the lat/lons to radians for the calculation
                arr = day.as_matrix()
                arr = np.radians(arr)

                # Calculate the differences between the lats/lons
                dlat = arr[:,0] - arr[:,2]
                dlon = arr[:,1] - arr[:,3]

                # Add the differences above as columns to the matrix
                arr = np.concatenate((arr, np.array([dlat]).T), axis=1)
                arr = np.concatenate((arr, np.array([dlon]).T), axis=1)

                # Calculate the distances
                dist = self.distances(arr)
                day_odm[:, i] = dist

            # Store the odm in the Sorting Centre instance
            ch.day_odm = pd.DataFrame(day_odm, index=day.index, columns=day.index)

            for i, row in enumerate(evening.index):
                # Store the lat/lon of the origin
```

```
        orig_lat = day.loc[row]['LAT']
        orig_lon = day.loc[row]['LON']
        evening['OR_LAT'] = orig_lat
        evening['OR_LON'] = orig_lon

        # Transform the lat/lons to radians for the calculation
        arr = evening.as_matrix()
        arr = np.radians(arr)

        # Calculate the differences between the lats/lons
        dlat = arr[:,0] - arr[:,2]
        dlon = arr[:,1] - arr[:,3]

        # Add the differences above as columns to the matrix
        arr = np.concatenate((arr, np.array([dlat]).T), axis=1)
        arr = np.concatenate((arr, np.array([dlon]).T), axis=1)

        # Calculate the distances
        dist = self.distances(arr)
        eve_odm[:, i] = dist

    # Store the odm in the Sorting Centre instance
    ch.eve_odm = pd.DataFrame(eve_odm, index=day.index, columns=day.index)

    # If the Milieuzone is applied, SCs have their own demand that
    # has to be included too.
    if self.switch_MZ:
        for sc in self.SortingCentres.values():
            day = sc.DayDestinations[['LAT', 'LON']]
            evening = sc.EveningDestinations[['LAT', 'LON']]

            s = pd.Series([sc.location[0], sc.location[1]],
                          index=['LAT', 'LON'], name=sc.name)
            day = day.append(s)
            evening = evening.append(s)

            day_odm = np.zeros((len(day), len(day)))
            eve_odm = np.zeros((len(evening), len(evening)))

            for i, row in enumerate(day.index):
                # Store the lat/lon of the origin
                orig_lat = day.loc[row]['LAT']
                orig_lon = day.loc[row]['LON']
                day['OR_LAT'] = orig_lat
                day['OR_LON'] = orig_lon

                # Transform the lat/lons to radians for the calculation
                arr = day.as_matrix()
                arr = np.radians(arr)

                # Calculate the differences between the lats/lons
                dlat = arr[:,0] - arr[:,2]
                dlon = arr[:,1] - arr[:,3]

                # Add the differences above as columns to the matrix
                arr = np.concatenate((arr, np.array([dlat]).T), axis=1)
                arr = np.concatenate((arr, np.array([dlon]).T), axis=1)

                # Calculate the distances
                dist = self.distances(arr)
                day_odm[:, i] = dist
```

```
                # Store the odm in the Sorting Centre instance
                sc.day_odm = pd.DataFrame(day_odm, index=day.index, columns=day.

                for i, row in enumerate(evening.index):
                    # Store the lat/lon of the origin
                    orig_lat = day.loc[row]['LAT']
                    orig_lon = day.loc[row]['LON']
                    evening['OR_LAT'] = orig_lat
                    evening['OR_LON'] = orig_lon

                    # Transform the lat/lons to radians for the calculation
                    arr = evening.as_matrix()
                    arr = np.radians(arr)

                    # Calculate the differences between the lats/lons
                    dlat = arr[:,0] - arr[:,2]
                    dlon = arr[:,1] - arr[:,3]

                    # Add the differences above as columns to the matrix
                    arr = np.concatenate((arr, np.array([dlat]).T), axis=1)
                    arr = np.concatenate((arr, np.array([dlon]).T), axis=1)

                    # Calculate the distances
                    dist = self.distances(arr)
                    eve_odm[:, i] = dist

                # Store the odm in the Sorting Centre instance
                sc.eve_odm = pd.DataFrame(eve_odm, index=day.index, columns=day.

    # Otherwise, the ODM is only being calculated for the Sorting Centres
    else:
        for sc in self.SortingCentres.values():
            sc.dayremaining = list(sc.DayDestinations.index)
            sc.eveningremaining = list(sc.EveningDestinations.index)

            day = sc.DayDestinations[['LAT', 'LON']]
            evening = sc.EveningDestinations[['LAT', 'LON']]

            s = pd.Series([sc.location[0], sc.location[1]],
                          index=['LAT', 'LON'], name=sc.name)
            day = day.append(s)
            evening = evening.append(s)

            day_odm = np.zeros((len(day), len(day)))
            eve_odm = np.zeros((len(evening), len(evening)))

            for i, row in enumerate(day.index):
                # Store the lat/lon of the origin
                orig_lat = day.loc[row]['LAT']
                orig_lon = day.loc[row]['LON']
                day['OR_LAT'] = orig_lat
                day['OR_LON'] = orig_lon

                # Transform the lat/lons to radians for the calculation
                arr = day.as_matrix()
                arr = np.radians(arr)

                # Calculate the differences between the lats/lons
                dlat = arr[:,0] - arr[:,2]
                dlon = arr[:,1] - arr[:,3]
```

```
            # Add the differences above as columns to the matrix
            arr = np.concatenate((arr, np.array([dlat]).T), axis=1)
            arr = np.concatenate((arr, np.array([dlon]).T), axis=1)

            # Calculate the distances
            dist = self.distances(arr)
            day_odm[:, i] = dist

        # Store the odm in the Sorting Centre instance
        sc.day_odm = pd.DataFrame(day_odm, index=day.index, columns=day.inde

        for i, row in enumerate(evening.index):
            # Store the lat/lon of the origin
            orig_lat = day.loc[row]['LAT']
            orig_lon = day.loc[row]['LON']
            evening['OR_LAT'] = orig_lat
            evening['OR_LON'] = orig_lon

            # Transform the lat/lons to radians for the calculation
            arr = evening.as_matrix()
            arr = np.radians(arr)

            # Calculate the differences between the lats/lons
            dlat = arr[:,0] - arr[:,2]
            dlon = arr[:,1] - arr[:,3]

            # Add the differences above as columns to the matrix
            arr = np.concatenate((arr, np.array([dlat]).T), axis=1)
            arr = np.concatenate((arr, np.array([dlon]).T), axis=1)

            # Calculate the distances
            dist = self.distances(arr)
            eve_odm[:, i] = dist

        # Store the odm in the Sorting Centre instance
        sc.eve_odm = pd.DataFrame(eve_odm, index=day.index, columns=day.inde
```

The SIMULATION functions

Lastly, the model has two main functions for running the simulation. The first is determining what the next event will be. This is called *next_event()* in the *Model* class. Based on all the timing variables that all the instances have, the next event is determined and executed. Eventually, the model clock is updated as a consequence of executing the event.

Secondly, the *run()* function calls the *next_event()* function as long as the simulation time limit has not been exceeded. Both of these functions are defined below:

In [ ]:

```
    def next_event(self):
        # First, all timing variables are assessed
        times = [self.t_demand]
        for sc in self.SortingCentres.values():
            times += [sc.t_tsp]
            for v in sc.vehicles.values():
                times += [v.t_load, v.t_drop, v.t_collect, v.t_unload]

        for ch in self.CityHubs.values():
            times += [ch.t_tsp]
            for v in ch.vehicles.values():
                times += [v.t_load, v.t_drop, v.t_collect, v.t_unload]

        self.t_event = min(times)
        idx = times.index(self.t_event)

        # Update the clock
        self.clock = self.t_event

        # Execute the next event
        self.events[idx]()
```

In [ ]:

```
    def run(self):
        end = self.BasicInfo['Value'][self.BasicInfo['Variable'] == 'Simulation time
        while self.clock < end:
            self.next_event()

        # Optional functions to call:
        self.get_results()
        self.print_results()
```

References

Hedges, A. (2002). Finding distances based on Latitude and Longitude. Retrieved 07 30, 2018, from Andrew Hedges: https://andrew.hedges.name/experiments/haversine/ (https://andrew.hedges.name/experiments/haversine/)

APPENDIX VI: Distribution Centre class

Defining the DistributionCentre class and its operations

This Python Notebook is for reference only and will be added to the report as appendix. Running the code will only yield errors, as it is not set-up properly for running. It is for descriptive purposes only and aims to explain the *DistributionCentre* class in the simulation model in more detail.

Normally, a class is defined in only one block of code. This Notebook divides the class and its functions however among multiple code blocks to emphasize more on the functions in the class.

The *DistributionCentre* class is created to provide a little structure to the model with regard to the base locations of the vehicles. This class provides a little more flexibility in the route planning possibilities of the vehicles and the distinction between a Sorting Centre and the City Hub.

Import the necessary libraries

In [ ]:

```
import numpy as np
import pandas as pd
import geopandas as gpd

from shapely.geometry import Polygon, MultiPolygon, Point
```

Initialising the *DistributionCentre* instance

The *DistributionCentre* instance takes a few variables as input before it can be instantiated:

- Name
- Location [LAT, LON]
- Market share (Sorting Centre)
- Reference to the *Model* instance

These will first be stored inside the instance before proceeding with other functions within the class. The **init** function is shown in the code block below in more detail, followed by the *setup_vehicles()* function and *sort_vehicles* function, which are also used in the initialisation.

In [ ]:

```
class DistributionCentre:
    def __init__(self, name, loc, marketshare, model):
        self.name = name
        self.location = loc
        self.marketshare = marketshare
        self.model = model

        # Setup the time variables for DC-related events
        self.t_tsp = 0

        # Setup the DC-related events
        self.events = [self.TSP]

        # Define the empty demand DataFrames to be filled later on
        self.DayDestinations = pd.DataFrame(columns=['LAT', 'LON',
                                                     'Postal code',
                                                     'Amount'])
        self.EveningDestinations = self.DayDestinations.copy()

        # Setup the vehicles as defined in the Vehicle Fleet tab
        # of the model dashboard
        self.setup_vehicles()
```

The *setup_vehicles()* function creates instances of the *Vehicle* class, based on the VehicleFleet tab in the model dashboard. The code is shown below:

In [ ]:

```
    def setup_vehicles(self):
        self.vehicles = {}
        # Store the vehicle fleets
        vfs = self.model.db['VehicleFleet']
        # Extract the fleet for the respective SC
        self.VehicleFleet = vfs[vfs['Location'].str.contains(self.name)]

        vehicles = self.VehicleFleet[['Truck diesel', 'Van diesel', 'Van electri
c',
                                      'Bike electric', 'Stint electric']]
        self.lineshifts = self.VehicleFleet[['Line shift truck diesel',
                                             'Line shift van diesel',
                                             'Line shift van electric',
                                             'Line shift bike electric',
                                             'Line shift stint electric']]
        prefs = self.VehicleFleet[['Planning preference truck diesel',
                                   'Planning preference van diesel',
                                   'Planning preference van electric',
                                   'Planning preference bike electric',
                                   'Planning preference stint electric']]
        # Assign a plannning preference to the vehicle
        for col in vehicles.columns:
            for i in range(vehicles[col].values[0]):
                for j in prefs.columns:
                    if col.lower() in j:
                        pref = prefs[j].values[0]
                        break
                    else:
                        pref = len(prefs.columns)
                # Extract vehicle info of specific model
                v = self.model.VehicleInfo.loc[self.model.VehicleInfo['Vehicles'
].str.contains(col)].values[0]
                c = list(self.model.VehicleInfo.columns)
                vd = pd.Series(v, index=c)

                # Create vehicle instance
                v = Vehicle(col+' {}'.format(i), vd, pref, self, self.model)
                self.vehicles[v.name] = v

        # Sort vehicles according to their planning preference
        self.sort_vehicles()
```

The *sort_vehicles* function offers the functionality to sort vehicles according to their assigned planning preference. Thereby, the route planning takes vehicles according to their preference before the planning procedure is started. So, if all bikes have to be utilised before the electric vans, this function causes the planning to do so.

In [ ]:

```
    def sort_vehicles(self):
        vehs = list(self.vehicles.keys())
        prefs = []
        for v in self.vehicles.values():
            prefs.append(v.planning_preference)

        df = pd.Series(prefs, index=vehs)
        df = df.sort_values()
        self.tsp_order = df.index.values
```

Defining the DC related events

The *DistributionCentre* class only has one specific event that occurs during the simulation, which is creating a route planning for all its vehicles based on the demand. The route planning provides a shifttype and shift for the vehicle before calling the *Vehicle*-related *TSP()* function.

Furthermore, in the *DistributionCentre* class' *TSP()* function, both the day and evening shifts are planned until the full capacity of the vehicles is reached, and all of the vehicles are being used in both the day and evening shift. The route planning is done in the order as obtained by the *sort_vehicles()* function defined above. The code is shown below.

In [ ]:

```
    def TSP(self):
        # First the day planning if the City Hub has to be used
        if self.model.switch_CH and not self.model.switch_MZ:
            lineswitch = False
            for veh in self.tsp_order:
                v = self.vehicles[veh]
                for col in self.lineshifts.columns:
                    # As there is a line haul remaining -> plan line shift
                    if v.name[:-2].lower() in col and self.lineshifts[col].value
s[0] == 'yes' and self.linehaulremaining > 0:
                        st = 'Line'
                        s = self.model.lineshift
                        lineswitch = True
                        break
                # Otherwise, assign a regular shift for planning
                if not lineswitch:
                    if len(self.dayremaining) > 0:
                        st = 'Day'
                        s = self.model.dayshift
                    elif len(self.eveningremaining) > 0:
                        s = self.model.eveningshift
                        st = 'Evening'
                    else:
                        s = self.model.dayshift + 24
                        st = 'Useless'
                v.TSP(s, st)


            # Then, if necessary, another planning for the evening
            if len(self.dayremaining) > 0 or len(self.eveningremaining) > 0:
                print('----------------> FILLING UP THE EVENING SHIFTS <--------
--------')
                for veh in self.tsp_order:
                    v = self.vehicles[veh]
                    if len(v.shifts) >= 0 and len(v.shifts) < 2 and np.array_equ
al(v.shifts[0], self.model.dayshift) and len(self.eveningremaining) > 0:
                        v.TSP(self.model.eveningshift, 'Evening')
                    elif len(v.shifts) >= 0 and len(v.shifts) < 2 and np.array_e
qual(v.shifts[0], self.model.dayshift) and len(self.dayremaining) > 0:
                        v.TSP(self.model.eveningshift, 'DayDelayed')
                    else: pass

        elif self.model.switch_CH and self.model.switch_MZ:
            lineswitch = False
            for veh in self.tsp_order:
                v = self.vehicles[veh]
                for col in self.lineshifts.columns:
                    if v.name[:-2].lower() in col and self.lineshifts[col].value
s[0] == 'yes' and self.linehaulremaining > 0:
                        #print(self.linehaulremaining)
                        st = 'Line'
                        s = self.model.lineshift
                        lineswitch = True
                        break
                if not lineswitch:
                    if len(self.dayremaining) > 0:
                        st = 'Day'
                        s = self.model.dayshift
                    elif len(self.eveningremaining) > 0:
```

```
                        st = 'Evening'
                        s = self.model.eveningshift
                    else:
                        st = 'Useless'
                        s = self.model.dayshift + 24
                v.TSP(s, st)

            if len(self.dayremaining) > 0 or len(self.eveningremaining) > 0:
                print('----------------> FILLING UP THE EVENING SHIFTS <--------
--------')
                for veh in self.tsp_order:
                    v = self.vehicles[veh]
                    if len(v.shifts) >= 0 and len(v.shifts) < 2 and np.array_equ
al(v.shifts[0], self.model.dayshift) and len(self.eveningremaining) > 0:
                        v.TSP(self.model.eveningshift, 'Evening')
                    elif len(v.shifts) >= 0 and len(v.shifts) < 2 and np.array_e
qual(v.shifts[0], self.model.dayshift) and len(self.dayremaining) > 0:
                        v.TSP(self.model.eveningshift, 'DayDelayed')
                    else: pass

        else:
            for veh in self.tsp_order:
                v = self.vehicles[veh]
                if len(self.dayremaining) > 0:
                    v.TSP(self.model.dayshift, 'Day')
                elif len(self.eveningremaining) > 0:
                    v.TSP(self.model.eveningshift, 'Evening')
                else:
                    v.TSP(self.model.dayshift + 24, 'Useless')


            if len(self.dayremaining) > 0 or len(self.eveningremaining) > 0:
                print('----------------> FILLING UP THE EVENING SHIFTS <--------
--------')
                for veh in self.tsp_order:
                    v = self.vehicles[veh]
                    if len(v.shifts) >= 0 and len(v.shifts) < 2 and np.array_equ
al(v.shifts[0], self.model.dayshift) and len(self.eveningremaining) > 0:
                        v.TSP(self.model.eveningshift, 'Evening')
                    elif len(v.shifts) >= 0 and len(v.shifts) < 2 and np.array_e
qual(v.shifts[0], self.model.dayshift) and len(self.dayremaining) > 0:
                        v.TSP(self.model.eveningshift, 'DayDelayed')
                    else:    pass

        self.t_tsp += 24
```

References

APPENDIX VII: Vehicle class

Defining the Vehicle class and its operations

This Python Notebook is for reference only and will be added to the report as appendix. Running the code will only yield errors, as it is not set-up properly for running. It is for descriptive purposes only and aims to explain the *Vehicle* class in the simulation model in more detail.

Normally, a class is defined in only one block of code. This Notebook divides the class and its functions however among multiple code blocks to emphasize more on the functions in the class.

Importing the necessary libraries

First of all, the simulation model in general, but also the *Vehicle* class, use specific Python libraries and their functions. These are imported first.

In [1]:

```
import numpy as np
import pandas as pd
```

Furthermore, it is assumed that the *Model* class has already been defined, together with its 'structural' variables and DataFrames. Moreover, it is assumed that the following variables and DataFrames are defined in the *Model* class:

- The Dashboard
- The Shapefile
- The Population data
- Postal codes and their locations
- Locker locations
- The eventual 'Milieuzone'

The *Model* instance also sets the switches that define a configuration of the simulation and define the *Distribution Centre* instances to take into account.

Then, the *Model* instance has to be linked to the Vehicle class, together with the instance of the *Distribution Centre* class. Finally, some *Vehicle* class specific variables are entered and the initiation of the *Vehicle* instance can start.

In [ ]:

```
class Vehicle:
    def __init__(self, name, data, planning_preference, dc, model):
        # First, the basic variables are stored and linked in/to the instance
        self.name = name
        self.vehicle_data = data
        self.planning_preference = planning_preference
        self.dc = dc
        self.model = model

        # Then, some empty lists that will be filled during the route planning
        # have to be defined
        self.shifts = [] # The vehicle can have multiple shifts assigned
        self.shifttypes = []
        self.dayroutes = []
        self.everoutes = []
        self.allroutes = []
        self.loads_to_drop = []
        self.loads_to_collect = []
        self.charges = [] # To store when the vehicle has to be charged

        # Defining some individual vehicle data
        self.ParcelsOnBoard = 0
        self.ParcelsDelivered = 0

        self.EmissionsCO2 = 0
        self.EmissionsNOx = 0
        self.TotalKms = 0
        self.TotalTime = 0

        # Setting up the timing variables
        self.t_load = 0 # Time indication for the next loading event
        self.t_drop = 0 # Time indication for  the next drop event
        self.t_collect = 0 # Time indication for the next pick-up event
        self.t_unload = 0 # Time indication for whn the next unloading event

        # Setting up the events
        self.events = [self.Load, self.Drop, self.Collect, self.Unload]

        # Based on the implementation of the City Hub (or not), the initial
        # route of the vehicle can be determined. This is done by a separate
        # function, covered in the following code block.
        self.determine_initial_routes()
```

The function to determine the initial routes checks the option whether or not the vehicle can be used for a line haul. The line haul is defined as a service that transports parcels between a Sorting Centre and the City Hub. If the vehicle is not to be used as line haul, it will get a regular route assigned.

In [ ]:

```
    def determine_initial_routes(self):
        # First, the switches for the line haul shift have to be extracted
        # from the dashboard
        shifts = self.dc.VehicleFleet[['Line shift truck diesel',
                                       'Line shift van diesel',
                                       'Line shift van electric',
                                       'Line shift bike electric',
                                       'Line shift stint electric']]
        # Check if the City Hub option is switched ON
        if self.model.switch_CH:
            if 'City Hub' in self.dc.name:
                """If the dc.name contains the string 'City Hub', this vehicle
                belongs to the City Hub. From the definition, these vehicles
                can never be used as line haul.
                Therefore, they get the init_route as defined below"""

                self.init_route = [self.dc.name, self.dc.name]
            else:
                """If it is not the city hub, we have to check what type of vehi
cle
                this instance is, and if it is assigned to a line haul shift."""

                for col in shifts.columns:
                    if self.name[:-2].lower() in col and shifts[col].values[0] =
= 'yes':
                        self.init_route = [self.dc.name, 'City Hub', self.dc.nam
e]
                        break
                    else:
                        self.init_route = [self.dc.name, self.dc.name]
        # If City Hub option is switched OFF, then it is just the normal route
        # No line haul is ever being used
        else:
            self.init_route = [self.dc.name, self.dc.name]
```

Defining the Vehicle's Events

All of the setup functions for the *Vehicle* class have now been covered. As mentioned in the **init** function, the *Vehicle* class has four different events:

1. Load
2. Drop
3. Collect
4. Unload.

These events play a role in the actual simulation. In the following code blocks, each of these four events will be highlighted and explained in further detail.

LOADING THE VEHICLE

Loading the vehicle is usually the initiation of the simulated day for the vehicle. The route time and distance are reset to 0 and the vehicle is made ready to load the parcels for its route.

First, the shifttype has to be determined. Based on this shifttype, an Origin-Destination Matrix from the *Distribution Centre* can be assigned. These are split into a *day_odm* and *eve(ning)_odm*, based on the demand.

Then, the event only continues if the vehicle has more than 0 routes assigned. Hence, the parcels to load will be higher than 0 too. The number of parcels defined in the *loads_to_drop* list will be loaded onto the vehicle. Furthermore, these parcels will be extracted from *dc.ParcelsInStock* variable. If the vehicle has to be charged before it can start with the route, the *t_drop* will be calculated based on the extra charging time. Otherwise, the *t_drop* is calculated based on the *average loading time*, which is stored in the *vehicle_data* DataFrame.

Eventually, the emissions and route kilometres are updated based on the next destination in the route. Furthermore, the current destination (the DC) is deleted from the route. Been there, done that. Lastly, the *t_load* and *t_collect* are set to the next day, until interrupted otherwise.

Load - Code

In [ ]:

```
    def Load(self):
        # First, print some informatino in the terminal
        print('%.2f LOADING: '%self.model.clock, self.name, self.dc.name)
        # (Re)set the starting time and distance of the current route
        self.routestart = self.model.clock
        self.routekm = 0
        self.collected = 0

        # Assigning ODM based on shifttype
        if self.shifttype == 'Day':
            self.odm = self.dc.day_odm
        elif self.shifttype == 'Evening':
            self.odm = self.dc.eve_odm
        elif self.shifttype == 'Line' and self.model.switch_CH:
            self.odm = self.model.CityHubs['City Hub'].day_odm
        elif self.shifttype == 'DayDelayed':
            self.odm = self.dc.day_odm

        # Actually loading the vehicle, if it has a route assigned
        if len(self.routes) > 0:
            # Locally store the current (first) route
            r = self.routes[0]
            self.ParcelsOnBoard += self.loads_to_drop[0]
            self.dc.ParcelsInStock -= self.loads_to_drop[0]

            # Check if the vehicle needs a charge
            if self.charging[0]:
                print('-----> CHARGING <-----')
                # Set the t_drop variable based on charging time
                self.t_drop = self.model.clock \
                              + self.vehicle_data['Charging time'] \
                              + self.odm[r[0]].loc[r[1]] \
                              / self.vehicle_data['Speed average - highway']

            else:
                # Set the t_drop variable based on loading time
                self.t_drop = self.model.clock \
                              + self.vehicle_data['Fixed loading time'] \
                              + self.odm[r[0]].loc[r[1]] \
                              / self.vehicle_data['Speed average - highway']

            # Update the emissions
            self.EmissionsCO2 += self.odm[r[0]].loc[r[1]] \
                                 * self.vehicle_data['Emissions highway - CO2']
            self.EmissionsNox += self.odm[r[0]].loc[r[1]] \
                                 * self.vehicle_data['Emisssions highway - Nox/PM1
0']
            # Update kilometres
            self.routekm += self.odm[r[0]].loc[r[1]]
            # Update route, been there, done that
            self.routes[0].pop(0)

        else:
            self.t_drop += 24

        # Set the new loading time (and collection time)
        self.t_load += 24
        self.t_collect = self.t_load
```

DROPPING THE PARCELS

So, the parcels are loaded onto the *Vehicle* and it has arrived at its next destination. In other words, *t_drop* is reached and the *Drop* event is called. This event is subject to the hitrate as set in the Dashboard. There is always a certain change that the client is not answering the doorbell and the parcel has to be returned to the Sorting Centre / City Hub. Lets just assume that the parcel has to be dropped this time, what happens next?

The destinations can be distinguished in either *Day-*, *Evening-* or *Locker Destinations*. Eitherway, the parcel is dropped by removing the parcel from the *ParcelsOnBoard* and *model.ParcelsInSystem* variables. Furthermore, the *ParcelsDelivered* and *model.ParcelsDelivered* are updated.

Besides, the destinations are defined within the route the vehicle is travelling, so the current and the next destination are always known. Based on this information, the next *Drop* event is scheduled by calculating the *t_drop* variable.

However, if there are no parcels left to drop, in explicit, none of the aforementioned destinations is remaining, the activity looks at the next possible event: either *Collect* parcels as defined in the route, or *Unload* the parcels when returned to the Sorting Centre or City Hub.

Eventually, the outcome variables are updated as well. In explicit, the emissions and route distance are updated, together with the route itself. The current location is removed from the route, and the vehicle continues its journey to the next destination. The code is shown in the code block below

Drop - Code

In [ ]:

```
    def Drop(self):
        # Printing some information in the terminal
        print('%.2f DROP: ' %self.model.clock, self.name, self.dc.name,
                              self.shifttype)

        # Extract the hit-rate from the model dashboard
        hitrate=self.model.BasicInfo['Value'][self.model.BasicInfo['Variable']\
                  == 'Hit rate percentage'].values[0] / 100

        # Only proceed if there is a route being scheduled
        if len(self.routes) > 0:
            # Locally store the route
            r = self.routes[0]
            # Determine if the parcel can be dropped or not
            toDrop = np.random.choice([True, False], p=[hitrate, 1-hitrate])
            # Dropping day deliveries or lockers, if they fit
            if toDrop and 'day' in r[0] or 'Locker' in r[0]:
                colli = self.dc.DayDestinations['Amount'].loc[r[0]]
                self.ParcelsOnBoard -= colli
                self.model.ParcelsInSystem -= colli
                self.ParcelsDelivered += colli
                self.model.ParcelsDelivered += colli

                # Check what the next destination is
                if len(r) > 1 and 'Sorteercentrum' not in r[1] \
                              and 'City Hub' not in r[1]:
                    # Update the t_drop event
                    self.t_drop = self.model.clock \
                                  + self.vehicle_data['Fixed stop time'] \
                                  + self.odm[r[0]].loc[r[1]] \
                                  / self.vehicle_data['Speed average - City']
                    # Update the emissions
                    self.EmissionsCO2 += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - CO2'
]
                    self.EmissionsNOx += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - Nox/
PM10']

                elif len(r) > 1 and 'collection' in r[1]:
                    # If the next destination is a collection part, set t_collec
t
                    self.t_collect = self.model.clock \
                                     + self.vehicle_data['Fixed stop time'] \
                                     + self.odm[r[0]].loc[r[1]] \
                                     / self.vehicle_data['Speed average - City']
                    # Update the emissions
                    self.EmissionsCO2 += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - CO2'
]
                    self.EmissionsNOx += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - Nox/
PM10']
                    # Change the t_drop to t_load to prevent interferance
                    self.t_drop = self.t_load

                elif len(r) > 1 and ('Sorteercentrum' in r[1] \
                                     or 'City Hub' in r[1]):
                    # Print something in the terminal about returning to DC
```

```
                    print('-----> No afroming in this route, returning to base')
                    # Set the t_unload for the next event
                    self.t_unload = self.model.clock \
                                    + self.vehicle_data['Fixed stop time'] \
                                    + self.odm[r[0]].loc[r[1]] \
                                    / self.vehicle_data['Speed average - highwa
y']
                    # Update both the t_drop and t_collect to prevent interferan
ce
                    self.t_collect = self.t_load
                    self.t_drop = self.t_load

                # Update the route distance, route and demand DataFrame
                self.routekm += self.odm[r[0]].loc[r[1]]
                self.dc.DayDestinations.drop(r[0], inplace=True)
                self.routes[0].pop(0)

            # Define the same functionality for an evening delivery
            elif toDrop and 'evening' in r[0]:
                colli = self.dc.EveningDestinations['Amount'].loc[r[0]]
                self.ParcelsOnBoard -= colli
                self.model.ParcelsInSystem -= colli
                self.ParcelsDelivered += colli
                self.model.ParcelsDelivered += colli

                # Check what the next destination is
                if len(r) > 1 and 'Sorteercentrum' not in r[1] \
                              and 'City Hub' not in r[1]:
                    # Update the next drop activity
                    self.t_drop = self.model.clock \
                                  + self.vehicle_data['Fixed stop time'] \
                                  + self.odm[r[0]].loc[r[1]] \
                                  / self.vehicle_data['Speed average - City']
                    # Update the emissions of the route
                    self.EmissionsCO2 += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - CO2'
]
                    self.EmissionsNOx += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - Nox/
PM10']

                elif len(r) > 1 and ('Sorteercentrum' in r[1] \
                                     or 'City Hub' in r[1]):
                    # Update the model that the next activity is to unload at DC
                    self.t_unload = self.model.clock \
                                    + self.vehicle_data['Fixed stop time'] \
                                    + self.odm[r[0]].loc[r[1]] \
                                    / self.vehicle_data['Speed average - highwa
y']
                    # Update t_drop and t_collect to prevent interferance
                    self.t_collect = self.t_load
                    self.t_drop = self.t_load

                else: print('----------> MISTAKE <---------')
                # Update the routekm, demand DataFrame and the route
                self.routekm += self.odm[r[0]].loc[r[1]]
                self.dc.EveningDestinations.drop(r[0], inplace=True)
                self.routes[0].pop(0)

            # The City Hub as first destination can only occur at the Line Haul
            elif 'City Hub' in r[0]:
```

```
                self.model.CityHubs[r[0]].ParcelsInStock += self.ParcelsOnBoard
                self.ParcelsDelivered += self.ParcelsOnBoard
                self.ParcelsOnBoard -= self.ParcelsOnBoard

                # If the line haul has multiple routes, it returns
                # Otherwise it stays until the end of the day for collection
                if len(self.routes) > 1:
                    self.t_load = self.model.clock \
                                  + self.vehicle_data['Fixed stop time'] \
                                  + self.odm[r[0]].loc[r[1]] \
                                  / self.vehicle_data['Speed average - highway']
                    # Update t_drop and t_collect to prevent interferance
                    self.t_drop = self.t_load
                    self.routes.pop(0)
                else:
                    self.t_collect = self.shift[1]
                    self.t_drop = self.t_load

            # If the current destination is a collection point:
            # Change event to collection
            elif 'collection' in r[0]:
                self.t_collect = self.model.clock # Change immediately
                self.t_drop = self.t_load

            else: # Not toDrop -> not at home
                print('--------> {} not home'.format(r[0]))
                # Anyhow, update for the next event

                if len(r) > 1 and 'Sorteercentrum' not in r[1] \
                              and 'City Hub' not in r[1]:
                    self.t_drop = self.model.clock \
                                  + self.vehicle_data['Fixed stop time'] \
                                  + self.odm[r[0]].loc[r[1]] \
                                  / self.vehicle_data['Speed average - City']
                    self.EmissionsCO2 += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - CO2'
]
                    self.EmissionsNOx += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - Nox/
PM10']

                elif len(r) > 1 and 'collection' in r[1]:
                    self.t_collect = self.model.clock \
                                     + self.vehicle_data['Fixed stop time'] \
                                     + self.odm[r[0]].loc[r[1]] \
                                     / self.vehicle_data['Speed average - City']
                    self.EmissionsCO2 += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - CO2'
]
                    self.EmissionsNOx += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - Nox/
PM10']
                    self.t_drop = self.t_load

                elif len(r) > 1 and ('Sorteercentrum' in r[1] \
                                     or 'City Hub' in r[1]):
                    print('--------> No collection this route, returning to bas
e')
                    self.t_unload = self.model.clock \
                                    + self.vehicle_data['Fixed stop time'] \
                                    + self.odm[r[0]].loc[r[1]] \
```

```
                                    / self.vehicle_data['Speed average - highwa
y']
                    self.t_drop = self.t_load
                    self.t_collect = self.t_load

                else:
                    print('---------> MISTAKE <---------')

                # Update the routekm again, and the route
                self.routekm += self.odm[r[0]].loc[r[1]]
                self.routes[0].pop(0)
```

COLLECTION OF PARCELS

The collection of parcels is basically the same function as dropping a parcel. The only difference being that the *ParcelsOnBoard* increase and that there exists a risk that parcels do not fit in the vehicle. The collection procedure always takes place at the end of the route, if (almost) all parcels have been dropped. The only exception being if a destination was not at home or the parcel did not fit in the parcel locker.

Hence, the only important thing to note regarding the function of collecting parcels is that if a parcel does not fit, the vehicle returns to its DC. The parcel will be collected the next day instead. This differs from PostNL's approach, as they are obliged to collect parcels the same day. However, without this assumption this exception would overcomplicate the model for its current purpose. The code is shown in the code block below.

Collection - Code

In [ ]:

```
    def Collection(self):
        # Print something useful in the terminal when running
        print('%.2f AFROMING '%self.model.clock, self.name, self.dc.name)

        # Only proceed when there is a route scheduled.
        if len(self.routes) > 0:
            # Locally store the current route
            r = self.routes[0]
            if 'collection' in r[0]:
                # If indeed the parcel has to be collected, it is added onBoard
                colli = self.dc.DayDestinations['Amount'].loc[r[0]]
                self.ParcelsOnBoard += colli

                # Check if the parcels fit in the vehicle
                if self.ParcelsOnBoard > self.vehicle_data['Capacity average']:
                    self.ParcelsOnBoard -= colli
                    print('-> No space left in vehicle, coming back tomorrow')
                    if self.model.switch_CH:
                        # Search for the City Hub in the route
                        base = [i for i in r if 'City Hub' in i]
                        # Schedule the unload event
                        self.t_unload = self.model.clock \
                                        + self.vehicle_data['Fixed stop time'] \
                                        + self.odm[r[0]].loc[r[1]] \
                                        / self.vehicle_data['Speed average - hig
hway']
                        self.t_collect = self.t_load
                    else:
                        base = [i for i in r if 'Sorteercentrum' in i]
                        self.t_unload = self.model.clock \
                                        + self.vehicle_data['Fixed stop time'] \
                                        + self.odm[r[0]].loc[r[1]] \
                                        / self.vehicle_data['Speed average - hig
hway']
                        self.t_collect = self.t_load
                    # Update routekm and emissions
                    self.EmissionsCO2 += self.odm[r[0]].loc[base[0]] \
                                         * self.vehicle_data['Emissions highway - C
O2']
                    self.EmissionsNOx += self.odm[r[0]].loc[base[0]] \
                                         * self.vehicle_data['Emissions highway - N
ox/PM10']
                    self.routekm += self.odm[r[0]].loc[base[0]]

                elif len(r) > 1 and 'collection' in r[1]:
                    # Update the control variable and t_collect
                    self.collected += self.dc.DayDestinations['Amount'].loc[r[0
]]
                    self.t_collect = self.model.clock \
                                     + self.vehicle_data['Fixed stop time'] \
                                     + self.odm[r[0]].loc[r[1]] \
                                     / self.vehicle_data['Speed average - City']
                    # Update emissions and routekm
                    self.EmissionsCO2 += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - CO2'
]
                    self.EmissionsNOx += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['City emissions - Nox/
PM10']
```

```
                    self.routekm += self.odm[r[0]].loc[r[1]]
                    self.dc.DayDestinations.drop(r[0], inplace=True)
                    self.routes[0].pop(0)

                elif len(r) > 1 and ('Sorteercentrum' in r[1] \
                                     or 'City Hub' in r[1]):
                    self.collected += self.dc.DayDestinations['Amount'].loc[r[0
]]
                    self.t_unload = self.model.clock \
                                    + self.vehicle_data['Fixed stop time'] \
                                    + self.odm[r[0]].loc[r[1]] \
                                    / self.vehicle_data['Speed average - highwa
y']
                    # Update emissions and routekm
                    self.EmissionsCO2 += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['Emissions highway - C
O2']
                    self.EmissionsNOx += self.odm[r[0]].loc[r[1]] \
                                         * self.vehicle_data['Emissions highway - N
ox/PM10']
                    self.routekm += self.odm[r[0]].loc[r[1]]
                    self.dc.DayDestinations.drop(r[0], inplace=True)
                    self.routes[0].pop(0)

                else: print('-------> MISTAKE <-------')

            # If, by mistake, the current destination is already the DC:
            elif 'Sorteercentrum' in r[0] or 'City Hub' in r[0]:
                self.t_unload = self.model.clock
                self.t_collect = self.t_load
                self.routes[0].pop(0)

            else: print('-------> AFROMING GONE WRONG <--------')

        else: pass
```

UNLOADING COLLECTED PARCELS AT DC

The *Unload* activity returns the collected parcels to the Sorting Centre or City Hub and handles them as being delivered. Besides, the route statistics are calculated and stored, before continuing the process.

The remainder of the *Unload* activity is aimed at either setting up the vehicle for the next day, or setting up the vehicle for conducting the next route scheduled. If there are no routes left in the shift, it checks if the vehicle is scheduled for an optional following shift. If not, then it is prepared for the following day.

Unload - Code

In [ ]:

```
    def Unload(self):
        # Print something useful in the terminal
        print('%.2f RETURNED AND UNLOADING: ', %self.model.clock,
              self.name, self.dc.name)
        self.TotalKms += self.routekm
        self.TotalTime += (self.model.clock - self.routestart)

        # Updating the parcels delivered
        self.model.ParcelsDelivered += self.collected
        self.ParcelsDelivered += self.collected
        self.model.ParcelsInSystem -= self.collected
        self.ParcelsOnBoard -= self.ParcelsOnBoard

        # Check if the current route still exists and delete it
        if len(self.routes) > 0:
            self.routes.pop(0)
            self.loads_to_drop.pop(0)
            if len(self.loads_to_room) > 0:
                self.loads_to_room.pop(0)
            self.charging.pop(0)

            # If there is another route left, set it up
            if len(self.routes) > 0:
                self.t_load = self.model.clock
                self.t_drop = self.t_load
                self.t_collect = self.shift[1]
                self.t_unload = self.t_collect

            # If there is no other route left, but evening routes instead
            # Set these up and update the dayshift for the next day
            elif len(self.allroutes) > 1:
                if self.shifttype == 'Day':
                    self.shift += 24
                    self.model.dayshift = self.shift.copy()

                # Delete the day shift
                self.allroutes.pop(0)
                self.shifts.pop(0)
                self.shifttypes.pop(0)
                self.charges.pop(0)

                self.shift = self.shifts[0]
                self.shifttype = self.shifttypes[0]
                self.routes = self.allroutes[0]
                self.charging = self.charges[0]

                self.t_load = self.shift[0]
                self.t_drop = self.t_load
                self.t_collect = self.shift[1]
                self.t_unload = self.t_collect

            else:
                self.shift += 24
                # Check the type of shift in the model to update
                if self.shifttype == 'Line':
                    self.model.lineshift = self.shift.copy()
                elif self.shifttype == 'Day':
                    self.model.dayshift = self.shift.copy()
                elif self.shifttype == 'Evening':
```

```
                self.model.eveningshift = self.shift.copy()
            elif self.shifttype == 'DayDelayed':
                self.model.eveningshift = self.shift.copy()
            else:
                self.shifts = []
                self.shifttypes = []

            self.t_load = self.shift[0]
            self.t_drop = self.t_load
            self.t_collect = self.shift[1]
            self.t_unload = self.t_collect

            self.shifts.pop(0)
            self.shifttypes.pop(0)
            self.allroutes.pop(0)
            self.charges.pop(0)

        else:
            self.shift += 24
            self.t_load = self.shift[0]
            self.t_drop = self.t_load
            self.t_collect = self.shift[1]
            self.t_unload = self.t_collect

            self.shifts = []
            self.shifttypes = []
            self.allroutes = []
            self.charges = []
```

The Route Planning Function

The route planning function is, in contrast to the foregoing functions, not an 'event' of the *Vehicle* class. Instead, it is a regular function of the class that gets called by the route planning event of the parent *Distribution Centre*. The route planning algorithm is based on the Travelling Salesman Problem (TSP), which is why it is conveniently called *TSP* (Stein, 1978). Furthermore, the TSP in the simulation model is build upon a simple heuristic, namely that the next destination always is the one nearest to the current location. This way, routes remain within a 25% range of the optimal solution, but it saves a considerable amount of computing time.

The TSP algorithm is implemented in both the *Distribution Centre* class and the *Vehicles* class. Depending on the destinations remaining (day or evening), a working shift is assigned to the vehicles. Furthermore, if a city hub is to be used and the option to use a vehicle as line haul is set to ‘yes’, the vehicle gets the line shift assigned. After assigning a shift to the vehicle and assess the destinations remaining for the shift, the TSP function of the vehicle is called. This setup however is explained in more detail in the Notebook about the *Distribution Centre* events.

The algorithm is very simple. It follows the following steps:
As long as there is no limit reached AND there are destinations remaining:

1. Check what the last destination (or the current location) is.
2. Determine the nearest destination.
3. Insert the destination in the route (at the end, but before the vehicle has to return to the DC.
4. Calculate the current distance, travel time and capacity
5. Check if one of the limits is exceeded:
   If yes: Delete the last destination and add the route as it is now to the planned routes.
   If no: Delete the destination from the list of remaining destinations to visit and start at step 1.
   Continue for as long as there is time remaining in the shift.

Besides, if the route exceeds the range limit and the City Hub is to be used, the vehicle has to be charged in-between routes. A charging comment will be inserted in the right list, so the simulation knows when to charge.

As mentioned above, the planning first considers the parcels to drop. Then, the parcels to collect are planned. The collection is only conducted in the postal code areas the vehicle has already visited, to prevent vehicles from randomly driving through the city to collect parcels. This way, it is assumed that the collection is more connected to the original delivery route. The code is shown in the code block below

In [ ]:

```
    def TSP(self, shift, shifttype):
        # Print something useful in the terminal
        print('%.2f TSP: ', %self.model.clock, self.name, self.dc.name,
                            shifttype)
        loads = [] # Control list

        # Setup the lists with shifts and shifttypes
        # Useless will not be included in the planning
        if shifttype == 'Useless':
            self.shifts = [shift.copy()]
            self.shifttype = [shifttype]
        elif len(self.shifts) <= 1:
            self.shifts.append(shift.copy())
        else:
            self.shifts = [shift.copy()]

        # Setup the line shift vehicles as long as there is work
        if shifttype == 'Line':
            odm = self.model.CityHubs['City Hub'].day_odm
            self.lineroutes = []
            pcremaining = []
            charges = []
            lineroute_times = []
            demand = pd.DataFrame()
            parcelsleft = self.dc.linehaulremaining
            # Plan the line haul routes within the limits of
            # time and capacity
            while parcelsleft > 0:
                charges.append(False)
                self.lineroutes.append(list(self.init_route))
                # Check if all parcels fit in one vehicle
                if parcelsleft <= self.vehicle_data['Capacity average']:
                    self.loads_to_drop.append(parcelsleft)
                    parcelsleft -= parcelsleft
                    lineroute_times.append(self.vehicle_data['Fixed loading tim
e'] \
                                           * 2 \
                                           + odm[self.init_route[0]].loc[self.in
it_route[1]] \
                                           / self.vehicle_data['Speed average -
 highway'] * 2)
                    emptied = True
                else:
                    # Otherwise fill up to max_capacity
                    self.loads_to_drop.append(self.vehicle_data['Capacity averag
e'])
                    parcelsleft -= self.vehicle_data['Capacity average']
                    lineroute_times.append(self.vehicle_data['Fixed loading tim
e'] \
                                           * 2 \
                                           + odm[self.init_route[0]].loc[self.in
it_route[1]] \
                                           / self.vehicle_data['Speed average -
 highway'] * 2)
                    charges.append(False)
                    emptied = False
                # Calculate if time limit has been exceeded
                if len(self.lineroutes) > 1:
                    tempt = sum(lineroute_times)
```

```
                if tempt > (self.model.dayshift[0] \
                           - self.model.lineshift[0]):
                    if emptied:
                        parcelsleft += parcelsleft
                        self.lineroutes.pop(-1)
                    else:
                        parcelsleft += self.vehicle_data['Capacity average']
                        self.lineroutes.pop(-1)
                    break

        self.dc.linehaulremaining = parcelsleft
        routes = self.lineroutes
        self.shifttypes.append(shifttype)

    # Setup day shift, if that is the case
    elif shifttype == 'Day':
        # Set time limit for the shift, which is entered in the dashboard
        timelimit = self.model.BasicInfo['Value']\
                    [self.model.BasicInfo['Variable'] \
                     == 'Maximum route time'].values[0]
        # Filter the destinations that fit the capacity of the vehicle
        pcremaining = [x for x in self.dc.dayremaining if \
                       self.dc.DayDestinations['Amount'].loc[x] \
                       <= self.vehicle_data['Capacity average']]

        # Store ODM, demand and routes locally
        odm = self.dc.day_odm
        demand = self.dc.DayDestinations
        self.dayroutes = []
        self.daycharges = [False]
        self.shifttypes.append(shifttype)

        routes = self.dayroutes
        charges = self.daycharges

    # Do the same for the evening shift
    elif shifttype == 'Evening':
        timelimit = self.model.BasicInfo['Value']\
                    [self.model.BasicInfo['Variable'] \
                     == 'Maximum route time'].values[0] / 2
        pcremaining = [x for x in self.dc.eveningremaining if \
                       self.dc.EveningDestinations['Amount'].loc[x] \
                       <= self.vehicle_data['Capacity average']]

        odm = self.dc.eve_odm
        demand = self.dc.EveningDestinations
        self.everoutes = []
        self.evecharges = []
        self.shifttypes.append(shifttype)

        routes = self.everoutes
        charges = self.evecharges

    # And do the same for day demand that have to be delivered in the evenin
g
    elif shifttype == 'DayDelayed':
        timelimit = self.model.BasicInfo['Value']\
                    [self.model.BasicInfo['Variable'] \
                     == 'Maximum route time'].values[0] / 2
        pcremaining = [x for x in self.dc.dayremaining if \
                       self.dc.DayDestinations['Amount'].loc[x] \
```

```
                        <= self.vehicle_data['Capacity average']]

        odm = self.dc.day_odm
        demand = self.dc.DayDestinations
        self.delayedroutes = []
        self.delayedcharges = []
        self.shifttypes.append(shifttype)

        routes = self.delayedroutes
        charges = self.delayedcharges

    else:
        pcremaining = []
        routes = []
        charges = []
        demand = pd.DataFrame()

    # If there are no destinations remaining, planning procedure is skipped
    if len(pcremaining) > 0:
        pcr = True
    else: pcr = False

    # Assumed that vehicle is always fully charged for first route
    charging = False
    # First range limit, which is raised every time the vehicle is reloaded
    dist_limit = self.vehicle_data['Range average']

    times = []
    distances = []

    ################ Start planning procedure ################
    while pcr:
        limit = False
        route = list(self.init_route)
        load_limit = self.vehicle_data['Capacity average']

        # If there are more routes, check if charging is needed
        if len(routes) > 0:
            ds = sum([sum(x) for x in distances])
            if charging:
                print('-----> CHARGING <-----')
                times.append(self.vehicle_data['Charging time'])
                charges.append(True)
                dist_limit = ds + self.vehicle_data['Range average']

            elif self.model.switch_CH:
                times.append(self.vehicle_data['Fixed loading time'])
                charges.append(False)
                dist_limit = ds + self.vehicle_data['Fixed loading time'] \
                             / self.vehicle_data['Charging time'] \
                             * self.vehicle_data['Range average']

            else:
                times.append(self.vehicle_data['Fixed loading time'])
                charges.append(False)
        else:
            times.append(self.vehicle_data['Fixed loading time'])

        # First plan the delivery route, then the collection route
        # Handled by setting the switches below.
        # Furthermore, only collect parcels in postal code areas
```

```
            # we've already been
            switch_collection = False
            pcs_to_collect = []
            switch_delivery = True

            while not limit:
                # Check current/last location visited
                lc = route[-2]
                # Retrieve distances to remaining destinations
                dists = odm[lc].loc[pcremaining]

                if switch_delivery:
                    # Filter the applicable destinations:
                    dtemp = dists.loc[(dists.index.str.contains('day') \
                                       | dists.index.str.contains('Locker')\
                                       | dists.index.str.contains('evening'))]
                    if len(dtemp) > 0:
                        nearest = dtemp.min()
                        nearest_idx = dtemp.idxmin()
                        # Check postal code of destination
                        pcv = nearest_idx.split(' ')[2]
                        if pcv not in pcs_to_collect:
                            pcs_to_collect.append(pcv)

                        route.insert(-1, nearest_idx)
                    else:
                        # If there is no destination remaining in dtemp,
                        # Start collecting parcels instead
                        switch_delivery = False
                        switch_collection = True

                        pcs_to_collect = [x.split(' ')[2] for x in pcremaining \
                                          if 'Sorteercentrum' not in x \
                                          and 'City Hub' not in x]

                if switch_collection:
                    # Filter the demand for collection
                    collection_demand_idx = [x for x in dists.index if 'collecti
on' \
                                                in x and pcs_to_collect[-1] in x]
                    if len(collection_demand_idx) > 0:
                        # Get the distances, determine nearest and insert in rou
te
                        collection_demand = dists.loc[collection_demand_idx]
                        nearest = collection_demand.min()
                        nearest_idx = collection_demand.idxmin()
                        route.insert(-1, nearest_idx)

                    # If there is no demand left, but postal code area still has
 to
                    # be visited, the following code is entered.
                    elif len(collection_demand_idx) == 0 and len(pcs_to_collect)
 > 0:
                        pcs_to_collect.pop(-1)
                        if len(pcs_to_collect) == 0:
                            limit = True
                            print('---> No afroming left for this route')

                    else:
                        print('---> No afroming left for this route')
                        limit = True
```

```
                pcr = False

            # Start calculating the current route's properties
            # This is used as input to check whether or not limits are reach
ed
            tempd = 0 # Distance
            tempt = 0 # Time
            templ = 0 # Load

            for idx, pc in enumerate(route[:-1]):
                # Check distance and add to tempd
                d = odm[pc].loc[route[idx+1]]
                tempd += d
                if 'City Hub' in pc or 'Sorteercentrum' in pc:
                    # First stop, so add loading time too, speed is highway
                    tempt += d/self.vehicle_data['Speed average - highway']
                    tempt += self.vehicle_data['Fixed loading time']
                elif 'City Hub' in route[idx+1] or \
                     'Sorteercentrum' in route[idx+1]:
                        # Last stop back to base, so speed highway
                    tempt += d/self.vehicle_data['Speed average - highway']
                else:
                    # Interdrop simulation, city speed
                    tempt += d/self.vehicle_data['Speed average - City']

                if 'City Hub' not in pc and \
                   'Sorteercentrum' not in pc:
                    # Can only take demand if not DC type of destination
                    templ += demand['Amount'].loc[pc]

            # Start checking if limits have been reached
            if tempd > self.vehicle_data['Range average']:
                print('-> Range limit reached')
                route.pop(-2)
                limit = True
                charging = True

            elif tempt > timelimit:
                print('-> Time limit reached')
                route.pop(-2)
                limit = True
                pcr = False # Exit planning, no more time left

            elif templ > load_limit:
                print('-> Capacity limit reached')
                route.pop(-2)
                if len(pcs_to_collect) > 0:
                    switch_delivery = False
                    switch_collection = True
                    # Start collecting, extra load necessary
                    load_limit += self.vehicle_data['Capacity average']
                else:
                    limit = True

            else:
                # Check the total time and distance of all the routes
                td = 0
                tt = 0
                for d in distances:
                    if isinstance(d, float): td += d
                    else: td += sum(d)
```

```
                    for t in times:
                        if isinstance(t, float): tt += t
                        else: tt += sum(t)

                    # Check the total limits
                    if (td + tempd) > dist_limit:
                        print('-----> Overall range limit reached')
                        route.pop(-2)
                        limit = True
                        charging = True

                    elif (tt + tempt) > timelimit:
                        print('-----> Overall time limit reached')
                        route.pop(-2)
                        limit = True
                        pcr = False

                    else:
                        # If no limit has been reached: delete destination from
                        # remaining destinations and continue planning
                        if switch_collection \
                           and nearest_idx in collection_demand_idx:
                            collection_demand_idx.pop(collection_demand_idx.inde
x(nearest_idx))
                            pcremaining.pop(pcremaining.index(nearest_idx))
                            if len(collection_demand_idx) != 0:
                                pass
                            elif len(collection_demand_idx) == 0 and len(pcs_to_
collect > 0):
                                pcs_to_collect.pop(-1)
                                if len(pcs_to_collect) == 0:
                                    print('-----> No afroming left for this rout
e')
                                    limit = True
                            else: limit = True

                        elif switch_delivery:
                            pcremaining.pop(pcremaining.index(nearest_idx))

                        if len(pcremaining) == 0:
                            pcr = False
                            limit = True
                            # Exitting planning loop

            # Make sure that only meaningful routes are added to the planning
            if route != self.init_route:
                dists = []
                lds_d = []
                lds_c = []
                tms = []
                # Calculating route statistics for the final time
                for idx, pc in enumerate(route[:-1]):
                    d = odm[pc].loc[route[idx+1]]
                    dists.append(d)

                    if 'City Hub' in pc or 'Sorteercentrum' in pc:
                        tms.append(d/self.vehicle_data['Speed average - highway'
])
                    elif 'City Hub' in route[idx+1] \
                         or 'Sorteercentrum' in route[idx+1]:
                        tms.append(d/self.vehicle_data['Speed average - highway'
```

```
])
                    else:
                        tms.append(d/self.vehicle_data['Speed average - City'])

                    if 'City Hub' not in pc and 'Sorteercentrum' not in pc:
                        l = demand['Amount'].loc[pc]
                        if 'collection' in pc:
                            lds_c.append(l)
                        else:
                            lds_d.append(l)

                        tms.append(self.vehicle_data['Fixed stop time'])
                # Append the route to the planning
                # And all other statistics needed for the simulation
                routes.append(route)
                times.append(tms)
                distances.append(dists)
                loads.append(sum([sum(lds_c), sum(lds_d)]))
                self.loads_to_drop.append(lds_d)
                self.loads_to_collect.append(lds_c)

            if len(pcremaining) == 0:
                print('No destinations remaining')
                pcr = False
                # Exit the planning loop for this vehicle

            # Update the demand DataFrames in the DC instance
            if shifttype == 'Day':
                for r in routes:
                    for pc in r:
                        if pc in self.dc.dayremaining:
                            self.dc.dayremaining.pop(self.dc.dayremaining.index(
pc))
            elif shifttype == 'Evening':
                for r in routes:
                    for pc in r:
                        if pc in self.dc.eveningremaining:
                            self.dc.eveningremaining.pop(self.dc.eveningremainin
g.index(pc))
            elif shifttype == 'DayDelayed':
                for r in routes:
                    for pc in r:
                        if pc in self.dc.dayremaining:
                            self.dc.dayremaining.pop(self.dc.dayremaining.index(
pc))

        self.allroutes.append(routes)
        self.charges.append(charges)

        # Setup the simulation variables based on the planning
        if len(self.shifts) > 0:
            self.shift = self.shifts[0]
            self.routes = self.allroutes[0]
            self.shifttype = self.shifttypes[0]
            self.charging = self.charges[0]

            self.t_load = self.shift[0]
            self.t_drop = self.t_load
            self.t_collect = self.shift[1]
            self.t_unload = self.t_collect
```

```
        else:
            self.t_load += 24
            self.t_drop += 24
            self.t_collect += 24
            self.t_unload += 24
```

Summarising the *Vehicle* Class

In the code block below, the vehicle class with all its functions and events is summarised. The code of each function is covered in this Notebook, so this summary only shows an overview of the class

In [ ]:

```
class Vehicle:
    def __init__(self, name, data, planning_preference, dc, model):
        # Statements, calling functions and define variables
        # Code ...

    def determine_initial_routes(self):
        # Based on settings regarding City Hub. Input for TSP()
        # Code ...

    def Load(self):
        # Simulation event
        # Code ...

    def Drop(self):
        # Simulation event
        # Code ...

    def Collect(self):
        # Simulation event
        # Code ...

    def Unload(self):
        # Simulation event
        # Code ...

    def TSP(self, shift, shifttype):
        # Function called by DC instance
        # Code ...
```

References

Stein, D. M. (1978). An asymptotic, probablistic analysis of a routing problem. *Mathematics of Operations Research 3*(2). 89-101

APPENDIX VIII: Setup the Parcel Lockers

Showing the setup of the lockers - script

This Notebook describes the script that conducts the parcel locker setup. By no means this Notebook can be used to setup the lockers, as the script is broken into smaller pieces to explain. The script can be found in the attached *setup_lockers.py* file.

Import libraries

In [ ]:

```
import numpy as np
import pandas as pd
import geopandas as gpd

from shapely.geometry import Polygon, MultiPolygon
import tripy
```

Define the filenames to import

In [ ]:

```
dashboard = '../data/dashboard.xlsx'
sheet = 'Settings'
```

In [ ]:

```
# Read the file into memory
settings = pd.read_excel(dashboard, sheet_name=sheet)
```

Read the settings before setup

The dashboard is similar to the one used in the simulation model. Therefore, also the same settings can be extracted from the file, which are defined in the *Settings* sheet. Now, it is checked whether or not the *Parcel Locker* setting is switched on and what the settings with regard to the interdrop are.

In [ ]:

```
settings = settings.loc[['Parcel Lockers',
                         'Average walking time [min]',
                         'Average walking speed [km/h]']]
```

In [ ]:

```
# Check the locker setting
if settings['Setting'].loc['Parcel Lockers'] == 'True':
    switch_PL = True
else: switch_PL = False
```

In [ ]:

```
speed = settings['Setting'].loc['Average walking speed [km/h]']
time = settings['Setting'].loc['Average walking time [min]']
```

Now, the so-called interdrop distance can be calculated. In other words, what is the average distance consumers are willing to walk to the nearest parcel locker? The *i*, or the interdrop distance, is calculated by multiplying the speed by the time willing to walk. This value is used later on in another function.

In [ ]:

```
i = speed * time
```

Read the shapefile

First, the filename of the shapefile is defined. Then, the shapefile is read into memory and processed similarly as done in the normal setup file.

In [ ]:

```
fn = '../data/openpc4nl2015landonly/PC4_Nederland_2015.shp'
```

In [ ]:

```
def get_shapes(fn):
    geoframe = gpd.read_file(fn)
    # Select Amsterdam records
    adam = geoframe[:86]
    # Change orientation of the coordinates
    adam = adam.to_crs({'init': 'epsg:4326'})

    # Transform data to km instead of m
    adam['Shape_Area'] /= 1000000
    adam['Shape_Leng'] /= 1000

    return adam
```

Process the shapes

Then, the shapes are analysed to setup the distribution of parcel lockers properly. Before the lockers can be distributed, several variables about the postal code area have to be known/calculated:

1. The number of stops (n)
2. The area of the postal code area
3. The normalised values of the areas
4. The triangles obtained by triangulation
5. The centroids of these triangles
6. The corresponding postal code for later reference

These aspects have to be known as a part of the algorithm that will be used to randomly distribute the parcel lockers among the postal code area. The algorithm was found on a forum (https://gis.stackexchange.com/questions/6412/generate-points-that-lie-inside-polygon/6419) and goes as follows:

1. Decompose the polygon into triangles
2. Calculate the areas of these triangles
3. Normalise the areas
4. Sample a fixed number of points with a probability based on the normalised values

Calculating the number of lockers

Lastly, the number of stops in a certain area can be calculated based on the formula shown below. In this formula, *n* stands for the number of stops, *area* stands for the area of the postal code area and *i* is the average interdrop distance. This function was provided by a PostNL model.

$$n_{stops} = \frac{area}{i_{distance}^2}$$

Triangulation of polygon

For decomposing polygon into triangles, the python library *tripy* can be used. Moreover, this specific library is used to 'triangulate' the polygons of the postal code areas. This will be based on the so-called earclip algorithm, which is (of course) integrated in the *tripy* library. More information can be found here (https://en.wikipedia.org/wiki/Polygon_triangulation).

Calculating Centroids of a Triangle

Calculating centroids of a triangle is based on the formula below, where *Ox* is the x-coordinate and *Oy* is the y-coordinate of the *centroid*, or central point of the triangle. Furthermore, these coordinates are calculated by summing all of the corresponding coordinates of the points of the triangles and divide them by 3. So, for *Ox* all x-coordinates of the points A, B and C are summed up, while for *Oy* all y-coordinates of A, B and C are summed up.

$$O_x = \frac{A_x + B_x + C_x}{3} \qquad O_y = \frac{A_y + B_y + C_y}{3}$$

In [ ]:

```
def process_shapes(shapes, i):
    # Calculate the number of lockers in an area
    n = int(round(shapes['Shape_Area'] / i**2))
    n.index = shapes['pc4txt']
    n.name = 'Lockers'

    # Setup the empty lists for the variables to calculate
    areas = [] # Areas of triangles
    norms = [] # For normalised values
    triangles = []
    centroids = []
    postals = []

    # Iterate through shapefile to calculate variables
    for i in range(len(shapes)):
        # Get polygon
        poly = shapes['geometry'].iloc[i]
        # Get Postal Code
        pc = shapes['pc4txt'].iloc[i]
        if isinstance(poly, Polygon):
            # Extract outer coordinates of polygon
            x,y = poly.exterior.coords.xy
            # Generate coordinates
            p = [(x[i], y[i]) for i in range(len(x))]
            # Use earclip algorithm to calculate triangles
            tri = tripy.earclip(p)
            triangles.append(tri)
            # Calculate the areas
            a = [tripy._triangle_area(*t[0], *t[1], *t[2]) for t in tri]
            areas.append(a)
            c = []
            for t in tri:
                x = (t[0][0] + t[1][0] + t[2][0]) / 3
                y = (t[0][1] + t[1][1] + t[2][1]) / 3
                c.append((x,y))
            centroids.append(c)
            norm = [area / sum(a) for area in a]
            norms.append(norm)
            postals.append(pc)
        else:
            for part in poly:
                x, y = part.exterior.coords.xy
                p = [(x[i], y[i]) for i in range(len(x))]
                tri = tripy.earclip(p)
                triangles.append(tri)
                c = []
                for t in tri:
                    x = (t[0][0] + t[1][0] + t[2][0]) / 3
                    y = (t[0][1] + t[1][1] + t[2][1]) / 3
                    c.append((x,y))
                centroids.append(c)
                a = [tripy._triangle_area(*t[0], *t[1], *t[2]) for t in tri]
                areas.append(a)
                norm = [area / sum(a) for area in a]
                norms.append(norm)
                postals.append(pc)
        print('PC {} done'.format(i))

    return areas, norms, triangles, centroids, postals, n
```

Reshape shape data

As some of the polygons were actually MultiPolygons, some outcomes have a different shape than we would expect. Therefore, the data obtained by the previous function is being processed again to 'flatten' the outcomes. Hence, the distribution of parcel lockers can be executed more easily.

In [ ]:

```
def reshape(data):
    # Data is a tuple that contains all the variables
    areas, norms, tris, centroids, pcs, stops = data

    # Define new lists to fill up with processed data
    a2 = []
    n2 = []
    t2 = []
    c2 = []
    # Only store unique postal codes.
    pcs2, counts = np.unique(pcs, return_counts=True)

    # Iterate through postal codes
    for i, pc in enumerate(pcs2):
        idx = pcs.index(pc)
        # If postal code only occurs once, data can be stored directly
        if counts[i] == 1:
            a2.append(areas[idx])
            n2.append(norms[idx])
            t2.append(tris[idx])
            c2.append(centroids[idx])
        # Otherwise, it is from a MultiPolygon and it has to be split up
        else:
            ta = []
            tc = []
            tt = []
            for i2 in range(counts[i]):
                ta += areas[idx + i2]
                tc += centroids[idx + i2]
                tt += tris[idx + i2]

            a2.append(ta)
            t2.append(tt)
            c2.append(tc)
            # Normalise again for MultiPolygon
            n2.append([a / sum(ta) for a in ta])

    return a2, n2, t2, c2, pcs2, stops
```

Sample locker locations

Now, the data is ready to be used for sampling the locker locations. The locker locations will be sampled from the *Centroid* coordinates, based on the normalised values of the areas of the corresponding triangles, as defined in the algorithm before. The locations can only be sampled once. The size is defined as the calculated number of stops. These locations will be stored in a DataFrame, so it can easily be stored in Excel for later reference. The function is shown below:

In [ ]:

```
def sample_locker_locations(data):
    # Unpack data tuple
    areas, norms, tris, centroids, pcs, stops = data

    # Define locker locations list and iterate through postal codes
    lockerlocs = []
    for i in range(len(pcs)):
        # Sample for Postal code
        locs = np.random.choice(np.arange(len(centroids[i])), \
                                size=int(stops.iloc[i]), \
                                p=norms[i], replace=False)
        # Select centroid
        locs = [centroids[i][v] for v in locs]
        # Store location
        lockerlocs.append(locs)

    # Predefine dataframe by list
    lockerlocsdf = []

    # Fill up the list by iterating through locations
    for i, ll in enumerate(lockerlocs):
        pc = pcs[i]
        for idx, loc in enumerate(ll):
            l = list(loc)
            n = 'Locker {} {}'.format(pc, idx)
            l += [str(pc), n]
            lockerlocsdf.append(l)

    lockerlocsdf = pd.DataFrame(lockerlocsdf, columns=['LON', 'LAT', 'Postcode',
 'Name'])
    lockerlocsdf = lockerlocsdf[['Name', 'LAT', 'LON', 'Postcode']]

    return lockerlocsdf
```

APPENDIX IX: Setup the simulation model

Describing the model setup - script

In this Notebook, the setup script as used in the simulation model has been further explained. Important to note is that the code in this Notebook cannot be used to run the setup for the simulation and is only for educational purposes.

This Notebook is written in a slighlty different order than the other Notebooks that can be found in the report, as first the *main* function is explained. In this main function, other functions will be called that will be defined later in this Notebook too. However, the main function is the one executed in the initiation of the simulation model and is therefore the most important to know. Besides, it is the simplest one as well.

Import necessary libraries

In [ ]:

```
import numpy as np
import pandas as pd
import geopandas as gpd
```

Main ¶

So, in the main function all the references to the files used are defined. Thus, this Notebook does not differ from that and the files are defined first.

Then, the shapes are read into memory, followed by the population data from the CBS. These are executed by two different functions that will be covered later in this Notebook.

Then, the similar postal codes are stored and both the shapes and population data are filtered for the similar postal codes. This, since it makes no sense to include postal code areas without demographic data and vice-versa.

Finally, the locker locations as defined in the *setup_lockers.py* script are read into memory and passed on to the simulation model. This is covered in the *main* function below.

In [ ]:

```
def main():
    dashboard = '../data/dashboard.xlsx'
    shape_data = '../openpc4nl2015landonly/PC4_Nederland_2015.shp'
    adam_data = '../../../data_and_indicators/bevolkingscijfers_amsterdam.xls'
    milieuzone_file = '../../../data_and_indicators/milieuzone_amsterdam.xls'
    pc_location_file = '../../../data_and_indicators/4pp-master/4pp.csv'
    locker_location_file = 'LockerLocations.xlsx'

    dashboard = read_dashboard(dashboard_file)
    shapes = read_shapes(shapefile)
    amsterdam = read_population_data(adam_file)
    shapes, amsterdam, similarpcs = check_similarities(shapes,
                                                       amsterdam)
    pc_locs, locker_locs = read_location_data(pc_location_file,
                                              locker_location_file,
                                              similarpcs)

    milieuzone = read_milieuzone(milieuzone_file)

    return dashboard, shapes, amsterdam, similarpcs, pc_locs, \
            locker_locs, milieuzone
```

Read Dashboard

Now, we will explain all the functions executed in the *main* function in the same order. So, first the *read_dashboard* function. This is the easiest, as it only opens the simulation model's dashboard file and passes it on. The code is shown below:

In [ ]:

```
def read_dashboard(fn):
    """Returns the dashboard as a dictionary with the different sheets"""
    db = pd.read_excel(fn, sheet_name=None)
    return db
```

Read shapes

The *read_shapes* function reads the shapefile of the Dutch postal code areas into memory. These postal codes will be filtered for only the ones in Amsterdam before being passed on to the simulation model.

Besides, the function transforms the projection of the shapefile from Dutch coordinates to the universal Latitude/Longitude projection and the area/lengths are re-calculated to kilometres (squared).

In [ ]:

```
def read_shapes(fn):
    """Returns a geoframe with lat/lon coordinates and postal codes
    of Amsterdam"""
    # Store Geoframe
    geoframe = gpd.read_file(fn)
    # After inspection it is determined that Amsterdam is present
    # until line 86
    geoframe = geoframe.iloc[:86]
    # Change the orientation of coordinates to lat/lon
    geoframe = geoframe.to_crs({'init': 'epsg:4326'})
    # Transform area and length of shapes to kilometers (squared)
    geoframe['Shape_Area'] /= 1000000
    geoframe['Shape_Leng'] /= 1000

    return geoframe
```

Read population data

The *read_population_data* function reads the Excel file from the Dutch CBS into memory. Only the columns of interest will be stored and renamed for easier further reference. Also, several density variables are being calculated, as can be noticed from the code below.

In [ ]:

```
def read_population_data(fn):
    """Analyse the data from CBS and store the relevant data in
    a separate dataframe. First, all the postal codes have to be
    filtered from the file. Then, the combined records for each
    postal code have to be created. """
    df = pd.read_excel(fn, header=[0,1])
    df.columns = df.columns.droplevel(-1)

    # Filter for 'Buurt', as they only contain the postal codes.
    df = df[df['Soort regio'] == 'Buurt']
    # Then, all postal codes have to be assessed.
    pcs = df['||Meest voorkomende postcode'].unique()
    # If there is still a missing record, we have to drop it.
    #These are indicated by a '.'
    pcs = pcs[np.where(pcs != '.')]
    df = df.replace('.', np.nan)

    # Now, we have to calculate the needed data for each postal code.
    #Data needed is:
    # Population and Population density
    # Companies and Company density
    # Distribution of Population and Companies among postal codes.
    variables = ['Population', 'Population density', 'Horeca', 'Horeca density',
                 'Companies', 'Company density']
    # First, a new DataFrame has to be defined, to append the new records to.
    newdf = pd.DataFrame(columns=variables)

    for pc in pcs:
        recs = df[df['||Meest voorkomende postcode'] == pc]
        inw = recs['||Aantal inwoners'].sum()
        hor = recs['|||G+I Handel en horeca'].sum()
        ovb = (recs['||Bedrijfsvestigingen totaal'] \
               - recs['|||G+I Handel en horeca']).sum()

        area = recs['Oppervlakte land'].sum()

        inw_dens = inw/area
        hor_dens = hor/area
        ovb_dens = ovb/area


        s = pd.Series([inw, inw_dens, hor, hor_dens, ovb, ovb_dens],
                      index=variables, name=pc)
        newdf = newdf.append(s)

    newdf.sort_index(inplace=True)

    # Now the distribution is calculated
    for var in ['Population', 'Horeca', 'Companies']:
        newdf[var+' distribution'] = newdf[var]/newdf[var].sum()

    # Change the order of the columns for appropriateness
    cols = ['Population', 'Population density', 'Population distribution',
            'Horeca', 'Horeca density', 'Horeca distribution',
            'Companies', 'Company density', 'Companies distribution']
    newdf = newdf[cols]

    return newdf
```

Check similar pcs

As mentioned before, the *check_similar_pcs* function only checks whether or not the same postal codes are present in the two different DataFrames of population data and shapes. The code is shown below.

In [ ]:

```
def check_similarities(geoframe, populationdf):
    """Check the extent to which the geoframe and dataframes are similar.
    If similar, combine the records.
    """
    geoPC = geoframe['PC4'].values
    popPC = populationdf.index.values

    toReturn = []
    if np.array_equal(geoPC, popPC):
        return populationdf.index.values
    elif len(geoPC) < len(popPC):
        for pc in geoPC:
            if pc in popPC:
                toReturn.append(pc)
            else:
                print(pc, 'not in arr')
    else:
        for pc in popPC:
            if pc in geoPC:
                toReturn.append(pc)
            else:
                print(pc, 'not in arr')
    geoframe['PC4'] = geoframe['PC4'].astype(int)
    geoframe = geoframe[geoframe['PC4'].isin(toReturn)]
    populationdf = populationdf.loc[toReturn]

    return geoframe, populationdf, toReturn
```

Read location data

The *read_location_data* function reads the locker locations and the postal code area locations into memory. The code is shown below.

In [ ]:

```
def read_location_data(*fns):
    """Open and process the files with all locations of
    Postal codes and Lockers"""
    pc_locations, locker_locations, pcs = fns

    # Postal code locations
    pcdf = pd.read_csv(pc_locations, index_col=0)
    pcdf = pcdf[pcdf['postcode'].isin(pcs)]

    pcdf = pcdf[['postcode', 'latitude', 'longitude']]
    pcdf.columns = ['Postcode', 'LAT', 'LON']
    pcdf.reset_index(drop=True, inplace=True)


    # Locker locations
    lockerlocsdf = pd.read_excel(locker_locations)

    return pcdf, lockerlocsdf
```

Mileuzone

Lastly, the *read_milieuzone* file is read into memory. This can be used to select the postal codes to be delivered from a city hub.

In [ ]:

```
def read_milieuzone(fn):
    df = pd.read_excel(fn)
    #print(df.head())
    milieuzone = df['Milieuzone Postcodes'].values
    arr = np.array(milieuzone)
    arr = arr.astype(str)
    return arr
```

APPENDIX X: Sensitivity Analysis

Conducting the Sensitivity analysis

In this Notebook, the sensitivity analysis for this study has been conducted. The sensitivity analysis consists two main parts: the scripts that define the different cases to run, and this Notebook to analyse the results of these runs.

The cases are defined as either increasing or decreasing a certain factor or variable in the model dashboard and run 5 different experiments with these settings. That 5 runs are needed is determined by van Soest's formula (van Soest, 1992) in chapter 5.6 in the report.

The variables under investigation in this Notebook are:

1. The route time limit
2. Vehicle capacities
3. Loading and stopping times
4. The number of transporters in the simulation
5. Inter-distance between lockers

The cases have already been generated and only their results will be covered in this analysis. The outcomes are stored in separate Excel files that are provided in the *Data* directory of this study.

Importing libraries

In [1]:

```
import numpy as np
import pandas as pd
```

1. Varying the route time limit

The route time limit is used to make sure that the routes together do not exceed the time of a normal working day. This normal time is assumed to be 8 hours.

For this analysis however, the effects of shortening and extending this route time is analysed. Furthermore, the route time limit is shortened to 6 hours and extended to 10 hours. The results are shown below.

In [2]:

```
fn = '../Data/Outcomes_Routetime.xlsx'

outcomes = pd.read_excel(fn, sheet_name=None)
```

The outcomes consist of the following cases:

1. Reference case (8 hours)
2. Regular config (10 hours)
3. Regular config (6 hours)
4. City Hub config (8 hours)
5. City Hub config (10 hours)
6. City Hub config (6 hours)

Each of these is analysed below. First, the statistics are calculated. These statistics are put together in DataFrames. Then there will be reflected upon the differences and if it strokes with the expectations.

In [3]:

```
cases = list(outcomes.keys())
cases
```

Out[3]:

```
['Case 0', 'Case 1', 'Case 2', 'Case 3', 'Case 4', 'Case 5']
```

In [4]:

```
# Prepare df
df_mean = pd.DataFrame(columns=outcomes[cases[0]].columns)
df_std = df_mean.copy()
```

In [5]:

```
for c in cases:
    case = outcomes[c]

    mean = case.mean(axis=0)
    std = case.std(axis=0)

    mean.name = c
    std.name = c

    df_mean = df_mean.append(mean)
    df_std = df_std.append(std)
```

In [6]:

```
df_mean
```

Out[6]:

	Total simulated time [h]	Parcels in system [pcs]	Parcels delivered [pcs]	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	
Case 0	24.0	2166.0	2014.0	1725.108366	0.856536	179.079072	0.088917	4
Case 1	24.0	2183.8	1996.2	1550.448885	0.776759	172.931484	0.086635	3
Case 2	24.0	2170.6	2009.4	2036.455483	1.013443	191.272441	0.095188	4
Case 3	24.0	2221.6	1958.4	1889.204378	0.964890	494.180944	0.252369	2
Case 4	24.0	2243.4	1936.6	1758.694288	0.908146	490.183450	0.253118	2
Case 5	24.0	2213.8	1966.2	1997.759704	1.016034	496.286243	0.252411	2

In [7]:

```
df_std
```

Out[7]:

	Total simulated time [h]	Parcels in system [pcs]	Parcels delivered [pcs]	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]
Case 0	0.0	9.137833	9.137833	27.509218	0.011139	1.927481	0.000873
Case 1	0.0	11.987493	11.987493	32.119641	0.018749	2.176464	0.001411
Case 2	0.0	7.700649	7.700649	55.498615	0.026288	3.522693	0.001625
Case 3	0.0	20.719556	20.719556	42.539060	0.030099	2.521495	0.003642
Case 4	0.0	6.066300	6.066300	53.825535	0.028105	3.046145	0.001855
Case 5	0.0	6.833740	6.833740	29.585170	0.013093	1.435955	0.001145

Calculate the advantages in terms of percentages

In [8]:

```
df_perc = df_mean / df_mean.iloc[0] * 100
```

In [9]:

```
df_perc = df_perc.apply(pd.Series.round)
```

In [10]:

```
oois = ['Total distance driven [km]', 'Distance per parcel [km]',
        'Total time operative [h]', 'Average time/parcel [h]',
        'Total emissions CO2 [kg]', 'CO2 emissions/parcel [kg]']
df_perc[oois]
```

Out[10]:

	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	Total emissions CO2 [kg]	CO2 emissions/parcel [kg]
Case 0	100.0	100.0	100.0	100.0	100.0	100.0
Case 1	90.0	91.0	97.0	97.0	98.0	99.0
Case 2	118.0	118.0	107.0	107.0	110.0	110.0
Case 3	110.0	113.0	276.0	284.0	59.0	61.0
Case 4	102.0	106.0	274.0	285.0	56.0	58.0
Case 5	116.0	119.0	277.0	284.0	59.0	61.0

What can be noticed from the table above, is that varying the route time has less of an impact on the City Hub configuration, compared to the regular configuration.

Increasing the route time limit considerably improves the statistics in both cases, while decreasing the route time limit drastically worsens the performance.

2. Varying vehicle capacity

The vehicle capacity is defined as the average number of parcels that fit into a specific type of vehicle. In general it is assumed that bicycles have the lowest capacity, followed by stints and then the (electric) vans.

What happens with the system performance if the capacity of each of this vehicles is increased? And what happens if capacity is decreased?

In [11]:

```
fn = '../Data/Outcomes_Capacity.xlsx'

outcomes = pd.read_excel(fn, sheet_name=None)
```

The outcomes consist of the following cases:

1. Reference case
2. Regular config High Capacity (+50%)
3. Regular config Low Capacity (-50%)
4. City Hub config
5. City Hub config High Capacity (+50%)
6. City Hub config Low Capacity (-50%)

Each of these is analysed below. First, the statistics are calculated. These statistics are put together in DataFrames. Then there will be reflected upon the differences and if it strokes with the expectations.

In [12]:

```
cases = list(outcomes.keys())
```

In [13]:

```
# Prepare df
df_mean = pd.DataFrame(columns=outcomes[cases[0]].columns)
df_std = df_mean.copy()
```

In [14]:

```
for c in cases:
    case = outcomes[c]

    mean = case.mean(axis=0)
    std = case.std(axis=0)

    mean.name = c
    std.name = c

    df_mean = df_mean.append(mean)
    df_std = df_std.append(std)
```

In [15]:

```
df_perc = df_mean / df_mean.iloc[0] * 100
df_perc = df_perc.apply(pd.Series.round)
df_perc[oois]
```

Out[15]:

	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	Total emissions CO2 [kg]	CO2 emissions/parcel [kg]
Case 0	100.0	100.0	100.0	100.0	100.0	100.0
Case 1	99.0	100.0	100.0	101.0	100.0	101.0
Case 2	100.0	101.0	100.0	101.0	101.0	102.0
Case 3	108.0	111.0	277.0	284.0	57.0	58.0
Case 4	99.0	101.0	270.0	278.0	58.0	60.0
Case 5	120.0	123.0	284.0	291.0	60.0	62.0

In [16]:

```
df_mean
```

Out[16]:

	Total simulated time [h]	Parcels in system [pcs]	Parcels delivered [pcs]	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	
Case 0	24.0	2171.0	2009.0	1720.949570	0.856644	177.762892	0.088485	4
Case 1	24.0	2182.6	1997.4	1704.101666	0.853206	178.527654	0.089384	4
Case 2	24.0	2179.4	2000.6	1722.788987	0.861158	178.513055	0.089231	4
Case 3	24.0	2225.6	1954.4	1859.316588	0.951427	491.849788	0.251671	2
Case 4	24.0	2223.2	1956.8	1699.727722	0.868627	480.759054	0.245692	2
Case 5	24.0	2219.2	1960.8	2063.657816	1.052481	505.730478	0.257932	2

What can be noticed from the figures above is that changing the vehicle capacity has no influence on the outcomes of the model. Both a 50% increase and decrease result in similar figures of the simulation.

However, changing the vehicle capacity for vehicles individually, thus not all at once, might have a bigger effect on the outcomes. This can be considered in a future research, as it is outside the scope of this study.

3. Loading and stopping times

The average loading time is defined as the average time a vehicle spends at the City Hub or Sorting Centre to completely fill up with parcels.
The average stop time is defined as the average time the vehicle has to stop at its destination to drop a parcel.

By varying these variables, the route planning can be made either more efficient, or less efficient, depending on increasing or decreasing the time for these variables.

The outcomes consist of 15 different cases.
The first case is the regular case, followed by increasing the loading time by 25% and 50%.
Then, loading time is decreased by the same numbers.

This is followed by the same prodedure regarding the stop time. Finally, these same experiments are conducted with a City Hub configuration.

In [17]:

```
fn = '../data/outcomes_loadstop_times.xlsx'

outcomes = pd.read_excel(fn, sheet_name=None)
```

In [18]:

```
cases = list(outcomes.keys())
```

In [19]:

```
# Prepare df
df_mean = pd.DataFrame(columns=outcomes[cases[0]].columns)
df_std = df_mean.copy()
```

In [20]:

```
for c in cases:
    case = outcomes[c]

    mean = case.mean(axis=0)
    std = case.std(axis=0)

    mean.name = c
    std.name = c

    df_mean = df_mean.append(mean)
    df_std = df_std.append(std)
```

In [21]:

```
df_perc = df_mean / df_mean.iloc[0] * 100
df_perc = df_perc.apply(pd.Series.round)
df_perc[oois]
```

Out[21]:

	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	Total emissions CO2 [kg]	CO2 emissions/parcel [kg]
Case 0	100.0	100.0	100.0	100.0	100.0	100.0
Case 1	103.0	103.0	104.0	104.0	103.0	103.0
Case 2	107.0	107.0	107.0	107.0	105.0	105.0
Case 3	98.0	98.0	98.0	98.0	100.0	100.0
Case 4	97.0	97.0	96.0	95.0	99.0	99.0
Case 5	106.0	107.0	115.0	116.0	104.0	105.0
Case 6	108.0	109.0	128.0	129.0	106.0	107.0
Case 7	95.0	95.0	86.0	86.0	98.0	98.0
Case 8	90.0	90.0	72.0	72.0	99.0	99.0
Case 9	101.0	102.0	101.0	101.0	102.0	102.0
Case 10	103.0	103.0	103.0	104.0	102.0	103.0
Case 11	107.0	107.0	107.0	107.0	104.0	104.0
Case 12	96.0	97.0	96.0	96.0	98.0	98.0
Case 13	96.0	96.0	95.0	95.0	98.0	98.0
Case 14	105.0	105.0	115.0	115.0	104.0	104.0
Case 15	114.0	114.0	131.0	131.0	110.0	110.0
Case 16	97.0	96.0	87.0	87.0	99.0	99.0
Case 17	91.0	91.0	72.0	72.0	101.0	101.0

What can be noticed from the Table above, is that that neither loading or stopping time heavily affects the performance of the regular system.

4. Changing the number of transporting companies

The simulation model offers the opportunity to change the number of transporting companies to include in the simulation. For all the former cases, the biggest companies have been included. But what will be the effects if only one company is included, or what if 5 companies are included?

That are exactly the two cases analysed in this Notebook. Furthermore, these three (including a reference case) are combined with a City Hub configuration to see if the City Hub has more effect if more transporters are included.

In [27]:

```
fn = '../data/outcomes_transporters-2.xlsx'
outcomes = pd.read_excel(fn, sheet_name=None)
```

In [28]:

```
cases = list(outcomes.keys())
```

In [29]:

```
# Prepare df
df_mean = pd.DataFrame(columns=outcomes[cases[0]].columns)
df_std = df_mean.copy()
```

In [30]:

```
for c in cases:
    case = outcomes[c]

    mean = case.mean(axis=0)
    std = case.std(axis=0)

    mean.name = c
    std.name = c

    df_mean = df_mean.append(mean)
    df_std = df_std.append(std)
```

In [31]:

```
df_perc = df_mean / df_mean.iloc[0] * 100
df_perc = df_perc.apply(pd.Series.round)
df_perc[oois]
```

Out[31]:

	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	Total emissions CO2 [kg]	CO2 emissions/parcel [kg]
Case 0	100.0	100.0	100.0	100.0	100.0	100.0
Case 1	117.0	111.0	113.0	107.0	118.0	112.0
Case 2	59.0	86.0	61.0	90.0	56.0	81.0
Case 3	109.0	112.0	276.0	283.0	57.0	59.0
Case 4	108.0	110.0	276.0	281.0	56.0	57.0
Case 5	74.0	108.0	120.0	176.0	35.0	51.0

In [32]:

```
df_mean
```

Out[32]:

	Total simulated time [h]	Parcels in system [pcs]	Parcels delivered [pcs]	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	
Case 0	24.0	2178.0	2002.0	1722.970126	0.860632	178.684171	0.089253	4
Case 1	24.0	2294.6	2105.4	2009.287219	0.954352	201.351337	0.095637	4
Case 2	24.0	1491.8	1368.2	1010.332152	0.738481	109.641178	0.080140	2
Case 3	24.0	2224.8	1955.2	1886.111073	0.964702	493.404444	0.252360	2
Case 4	24.0	2214.2	1965.8	1852.571687	0.942462	492.539489	0.250563	2
Case 5	24.0	1488.8	1371.2	1278.461931	0.932506	214.984034	0.156795	1

Text blabla

5. Varying inter-distance of lockers

Every analysis including the parcel locker infrastructure uses the same reference file. This file has placed lockers approximately within 500 meters apart from each other.

However, would the locker infrastructure still yield promising alternatives if the distance is changed?

In [33]:

```
fn = '../data/outcomes_lockers.xlsx'
outcomes = pd.read_excel(fn, sheet_name=None)
```

In [34]:

```
cases = list(outcomes.keys())
```

In [35]:

```
# Prepare df
df_mean = pd.DataFrame(columns=outcomes[cases[0]].columns)
df_std = df_mean.copy()
```

In [36]:

```
for c in cases:
    case = outcomes[c]

    mean = case.mean(axis=0)
    std = case.std(axis=0)

    mean.name = c
    std.name = c

    df_mean = df_mean.append(mean)
    df_std = df_std.append(std)
```

In [37]:

```
df_perc = df_mean / df_mean.iloc[0] * 100
df_perc = df_perc.apply(pd.Series.round)
df_perc[oois]
```

Out[37]:

	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	Total emissions CO2 [kg]	CO2 emissions/parcel [kg]
Case 0	100.0	100.0	100.0	100.0	100.0	100.0
Case 1	100.0	100.0	100.0	100.0	100.0	100.0
Case 2	99.0	99.0	100.0	100.0	99.0	99.0

In [ ]:

```
```

APPENDIX XI: Determining the number of replications (Van Soest)

Analysing outcomes of regular system

In this Notebook, the outcomes of the standard configuration are analysed. Basic statistics like mean and standard deviations are calculated, so van Soest's (1992) formula can be applied.

First, the necessary imports of libraries and the output file are conducted. This is followed by calculating the statistical data. Finally, van Soest's formula is applied.

In [1]:

```
# Import necessary libraries
import numpy as np
import pandas as pd
```

In [2]:

```
# Set file name
fn = 'Outcomes_multiple.xlsx'
# Open file in pandas DataFrame
cases = pd.read_excel(fn, sheet_name=None)
```

In [3]:

```
names = list(cases.keys())
```

Only extract regular cases

The cases with the regular system (without city hub) are the cases 0-7.

In [4]:

```
regular = names[:8]
print(regular)
```

```
['Case 0', 'Case 1', 'Case 2', 'Case 3', 'Case 4', 'Case 5', 'Case
6', 'Case 7']
```

Determine statistical data

In [5]:

```
# Create empty DF for statistical data
df_avg = pd.DataFrame(columns=cases['Case 0'].columns)
df_std = df_avg.copy()
```

In [7]:

```
for name in regular:
    df = cases[name]
    avg = df.mean(axis=0)
    avg.name = name
    std = df.std(axis=0)
    std.name = name


    df_avg = df_avg.append(avg)
    df_std = df_std.append(std)
```

In [8]:

```
df_avg
```

Out[8]:

	Total simulated time [h]	Parcels in system [pcs]	Parcels delivered [pcs]	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Avera time/parc
Case 0	24.0	29484.2	26945.8	9857.419763	0.365824	1655.713506	0.061446
Case 1	24.0	29371.5	27058.5	9396.025831	0.347248	1570.246440	0.058032
Case 2	24.0	29110.9	27319.1	8537.910829	0.312526	1352.750407	0.049517
Case 3	24.0	28863.2	27566.8	8445.813159	0.306376	1128.138921	0.040924
Case 4	24.0	28598.6	27831.4	8003.766329	0.287581	904.995950	0.032517
Case 5	24.0	29385.3	27044.7	10926.449539	0.404016	1630.512876	0.060290
Case 6	24.0	31822.9	24607.1	8976.062846	0.364776	1149.924035	0.046732
Case 7	24.0	32227.5	24202.5	9880.527178	0.408244	1469.636924	0.060723

In [9]:

```
df_std
```

Out[9]:

	Total simulated time [h]	Parcels in system [pcs]	Parcels delivered [pcs]	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	
Case 0	0.0	43.865451	43.865451	56.414212	0.002106	5.000673	0.000183	25
Case 1	0.0	47.103314	47.103314	60.817216	0.002133	5.223967	0.000216	20
Case 2	0.0	38.865723	38.865723	58.074571	0.002201	4.931453	0.000190	19
Case 3	0.0	22.660784	22.660784	89.091872	0.003246	4.986828	0.000182	18
Case 4	0.0	19.811332	19.811332	67.044700	0.002425	3.513421	0.000126	19
Case 5	0.0	46.984749	46.984749	53.213018	0.002138	5.882406	0.000272	20
Case 6	0.0	57.893485	57.893485	55.054596	0.002158	3.754589	0.000177	14
Case 7	0.0	53.219149	53.219149	44.130436	0.001612	3.593352	0.000174	16

Determine optimal number of replications

So, van Soest's formula is defined below, where *n* is the required number of replications, *n_testrun* is the number of experiments run for the test run under investigation, *sigma* is the standard deviation and *max(test run)* is the maximum value obtained by the test run.

$$n = \frac{n_{test\ run} * \sigma}{0.5 * \max(test\ run) * 0.05}$$

The formula is applied for all the outcomes of interest, so the required number of experiments is determined based on the outcomes that requires the highest number of experiments.

In [ ]:

```
avg = df_avg['Total distance driven [km]'].loc['Case 0']
std = df_std['Total distance driven [km]'].loc['Case 0']

avg, std
```

In [11]:

```
# Desired number of reps
temp = cases['Case 0']
maxim = temp['Total distance driven [km]'].max()
dnr = 10 * std / (maxim * 0.05 / 2)
dnr
```

Out[11]:

```
2.2637222183713113
```

In [13]:

```
std = df_std['Parcels delivered [pcs]'].loc['Case 0']
maxim = temp['Parcels delivered [pcs]'].max()
dnr = 10 * std / (maxim * 0.05 / 2)
dnr
```

Out[13]:

```
0.6498585310273552
```

In [16]:

```
std = df_std['Total time operative [h]'].loc['Case 0']
maxim = temp['Total time operative [h]'].max()
dnr = 10 * std / (maxim * 0.05 / 2)
dnr
```

Out[16]:

```
1.2034370871514184
```

In [12]:

```
temp = cases['Case 0']
std = df_std['Total emissions CO2 [kg]'].loc['Case 0']
maxim = temp['Total emissions CO2 [kg]'].max()
dnr = 10 * std / (maxim * 0.05 / 2)
dnr
```

Out[12]:

```
4.732776039594699
```

So, from all the applications of van Soest's formula above, it can be concluded that the required number of experiments for this simulation study is **5** experiments. This number will be used for any further analysis during this study.

References

Van Soest, J. (1992). *Elementaire Statistiek*. Delft: VSSD

APPENDIX XII: How to set-up the simulation model to conduct experiments?

How to setup experiments for simulation

This Notebook presents the way one could setup different experiments for a simulation study with the simulation model. This Notebook is for educational purposes only and is not able to run by itself.

Contents
First, the required libraries will be imported, including the *.py* file that contains the simulation model.
Then, a little more insight in the *Model* class is provided to understand why the following experiments are defined in a certain way.
Finally, the experiments will be conducted and stored in a useful way for later reference.

Importing libraries

In [ ]:

```
import numpy as np
import pandas as pd

from Model import * # Import all classes in script
```

The *Model* class

As mentioned before, the Model class is a little more special then shown in throughout the actual report, since the simulation model can be used in two ways:

1. As normal, based on the model dashboard
2. Based on (a) changed sheet(s) of the model dashboard.

The first is just the normal way, the simulation model is defined by instantiating the *Model* class from the script. See the code block below. In this code block, all the settings and variables as set in the dashboard's Excel file are applied.

In [ ]:

```
model = Model()
```

The latter method follows-up on the first, by first copying one or more of the DataFrames of the following sheets of the model dashboard:

1. *model.BasicInfo*
2. *model.Settings*
3. *model.VehicleInfo*

These are the only three options included for experiments at the moment, as no other variations have been investigated in this simulation study. If other sheets have to be changed for experiments, the *init* function of the *Model* class has to be adapted.

Anyway, the three sheets above can be changed according to the desired experiment. The experiments then have to be parsed into the simulation model before conducting the experiment. This can be done by instantiating the *Model* class again, but parse the adapted dashboard sheets between the brackets. This workflow, from the beginning to the instantiation, is shown in the code blocks below:

In [ ]:

```
# Instantiate the initial model
model = Model()
```

In [ ]:

```
# Copy the BasicInfo and Settings sheet
bi_init = model.BasicInfo.copy()
st_init = model.Settings.copy()
```

In [ ]:

```
# Define two cases:
# An initial
case0 = {'BasicInfo': bi_init,
         'Settings': st_init}
# One with the changed City Hub option
# First copy the settings
st_1 = st_init.copy()
st_1.at[0, 'Setting'] = 'True' # Set option to True
# Define the case
case1 = {'BasicInfo': bi_init,
         'Settings': st_1}
```

In [ ]:

```
# Instantiate the model based on case1
model = Model(case1['BasicInfo'], case1['Settings'])
```

Now, the model is instantiated with an alternative model dashboard, without having changed the source file of the model dashboard. Now, more experiments could be defined to include in the simulation study. Say for instance, that also *case2, case3* and *case4* have been defined in a similar matter as above and have to be included. These cases can be inserted in a list of all cases, as is shown below:

In [ ]:

```
# Define a list of cases to go through
cases = [case0, case1, case2, case3, case4]
```

Setup the simulation

So, all experiments have been defined and are ready to be conducted. However, before the system is ready, some more preparations have to be done.

1. First, the number of runs per experiments has to be defined (see van Soest for required number of experiments)
2. Prepare a DataFrame to store the results
3. Prepare a file to store the results of all different cases

In [ ]:

```
# Define the number of runs per experiments
n_exp = 5
```

In [ ]:

```
# Prepare empty DataFrame
# Indices of the Model's outcomes
idx = ['Total simulated time [h]', 'Parcels in system [pcs]',
       'Parcels delivered [pcs]', 'Total distance driven [km]',
       'Distance per parcel [km]', 'Total time operative [h]',
       'Average time/parcel [h]', 'Total emissions CO2 [kg]',
       'CO2 emissions/parcel [kg]', 'Total emissions NOx [kg]',
       'NOx emissions/parcel [kg]']
# Define empty DataFrame
df = pd.DataFrame(columns=idx)
```

In [ ]:

```
# Prepare the file with a 'Writer' object from Pandas
writer = pd.ExcelWriter('OutcomesFile.xlsx', engine='xlsxwriter')
```

Conduct the experiments

The time has come to actually conduct the experiments with the defined number of runs per experiment. This is done in the following order:

1. Iterate through the list of cases
2. Instantiate the simulation model with the new dashboard records
3. Run the simulation
4. Store data in DataFrame
5. Store DataFrame in Excel File
6. Close the Excel File

In [ ]:

```
for i, c in enumerate(cases):
    # Copy empty DataFrame
    oc_df = df.copy()
    # Conduct the number of experiments
    for j in range(n_exp):
        # Instantiate simulation model
        m = Model(c['BasicInfo'], c['Settings'])
        # Run simulation model
        m.run()
        s = pd.Series(list(m.Results.values()),
                      index=list(m.Results.keys()),
                      name='Run {}'.format(j))
        # Append outcomes to DataFrame
        oc_df = oc_df.append(s)

    # Store outcomes in
    oc_df.to_excel(writer, sheet_name='Case {}'.format(i))

writer.save()
```

APPENDIX XIII: Analysing the experiments

Analysing City Hub and non-City Hub configurations

In this notebook, the two different Excel files will be analysed that contain the outcomes of the simulation experiments. One file contains all cases, including faulty city hub configurations. These will be replaced by the outcomes stored in the other file.

Furthermore, the first file contains 10 runs per experiments, while the latter only has the 5 runs required for the desired accuracy. Comparing the two configurations with these differing number of runs should not make a difference, as they both comply with the rule of van Soest (1992).

Start-up the analysis

In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
# Define the file names
fn1 = 'Outcomes_multiple.xlsx' # Regular cases
fn2 = 'Outcomes_multiple_moreVeh.xlsx' # Only City Hub configurations
```

In [3]:

```
# Open the files and store in a dictionary with DataFrames
regulars = pd.read_excel(fn1, sheet_name=None)
city_hubs = pd.read_excel(fn2, sheet_name=None)
```

Processing the files

As mentioned before, the regular file also contains faulty city hub cases. These have to be filtered out first. From experience it can be stated that only the first 8 cases are needed. These will be filtered and stored separately.

In [4]:

```
# Retrieve the keys of the dictionary
regulars_keys = list(regulars.keys())
# Define the keys to extract from the dictionary
to_filter = regulars_keys[:8]
# Define filtered regulars dictionary and store the desired DataFrames
regulars_filtered = {}
for key in to_filter:
    regulars_filtered[key] = regulars[key]
```

In [5]:

```
# Check if the correct cases are filtered and stored
regulars_filtered.keys()
```

Out[5]:

```
dict_keys(['Case 0', 'Case 1', 'Case 2', 'Case 3', 'Case 4', 'Case
5', 'Case 6', 'Case 7'])
```

Then, the averages of the outcomes for each case can be calculated together with the standard deviations. These will be stored in separate DataFrames for further analyses.

Before this can be executed, a special Python function is defined. Hence, both dictionaries can be processed by the same function and time and space get spared. The Python function can be found in the code-block below:

In [6]:

```
def analyse_dictionaries(dic):
    # First, store the cases in the dictionary
    keys = list(dic.keys())
    # Define an empty DataFrame with the same column names as the DataFrames of
 the cases
    df_mean = pd.DataFrame(columns=dic[keys[0]].columns)
    df_std = df_mean.copy()

    # Iterate through the cases -> determine mean and std -> append to df
    for case in keys:
        # Extract the DataFrame
        df = dic[case]
        # Calculate the mean over the rows
        mean = df.mean(axis=0)
        # Calculate the std over the rows
        std = df.std(axis=0)

        # Set the names of the resulting Series objects
        mean.name = case
        std.name = case

        # Append records of the case to the DataFrame
        df_mean = df_mean.append(mean)
        df_std = df_std.append(std)

    # Return the results
    return df_mean, df_std
```

In [7]:

```
# Apply the function above for both files
df_mean_reg, df_std_reg = analyse_dictionaries(regulars_filtered)
df_mean_ch, df_std_ch = analyse_dictionaries(city_hubs)
```

Calculate the differences to the base case

Before the differences are calculated, all the DataFrames are combined and stored together. The means are put together and the standard deviations DataFrames too. Hence, two new DataFrames with all cases will be generated, replacing the four as obtained above.

In [8]:

```
df_mean = df_mean_reg.append(df_mean_ch, ignore_index=True)
df_std = df_std_reg.append(df_std_ch, ignore_index=True)

df_mean
```

Out[8]:

	Total simulated time [h]	Parcels in system [pcs]	Parcels delivered [pcs]	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]
0	24.0	29484.2	26945.8	9857.419763	0.365824	1655.713506	0.061446
1	24.0	29371.5	27058.5	9396.025831	0.347248	1570.246440	0.058032
2	24.0	29110.9	27319.1	8537.910829	0.312526	1352.750407	0.049517
3	24.0	28863.2	27566.8	8445.813159	0.306376	1128.138921	0.040924
4	24.0	28598.6	27831.4	8003.766329	0.287581	904.995950	0.032517
5	24.0	29385.3	27044.7	10926.449539	0.404016	1630.512876	0.060290
6	24.0	31822.9	24607.1	8976.062846	0.364776	1149.924035	0.046732
7	24.0	32227.5	24202.5	9880.527178	0.408244	1469.636924	0.060723
8	24.0	30750.6	25679.4	13343.102115	0.519605	1945.401748	0.075758
9	24.0	30642.0	25788.0	13360.740523	0.518098	1912.496389	0.074162
10	24.0	30396.0	26034.0	12783.384877	0.491031	2390.715398	0.091822
11	24.0	29866.2	26563.8	10731.186047	0.403978	2545.038142	0.095809
12	24.0	29854.0	26576.0	10613.077171	0.399349	2547.081459	0.095842
13	24.0	30440.2	25989.8	12384.033046	0.476495	3134.647242	0.120612
14	24.0	30418.0	26012.0	12487.812330	0.480078	3138.284864	0.120649
15	24.0	30492.4	25937.6	12440.165392	0.479622	3136.588253	0.120928

In [9]:

```
# Define the outcomes of interest to calculate the percentages from
oois = ['Total distance driven [km]', 'Distance per parcel [km]', 'Total time op
erative [h]',
        'Average time/parcel [h]', 'Total emissions CO2 [kg]', 'CO2 emissions/pa
rcel [kg]',
        'Total emissions NOx [kg]', 'NOx emissions/parcel [kg]']
```

In [11]:

```
# Store the base case
base_case = df_mean.iloc[0]
other_cases = df_mean.iloc[1:]

# Define the DataFrame to store the percentages
df_perc = pd.DataFrame(columns=oois)
#df_perc = df_perc.append(base_case[oois]/base_case[oois] * 100)

for i in range(len(df_mean)):
    s = df_mean.iloc[i]
    soi = s[oois]
    perc = soi / df_mean[oois].iloc[0] * 100
    df_perc = df_perc.append(perc, ignore_index=True)
```

In [12]:

```
df_perc
```

Out[12]:

	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	Total emissions CO2 [kg]	CO2 emissions/parcel [kg]
0	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
1	95.319323	94.922137	94.838052	94.443190	91.993785	91.610431
2	86.614054	85.430609	81.701961	80.585587	97.088538	95.761997
3	85.679756	83.749590	68.136119	66.601167	87.644867	85.670937
4	81.195348	78.611643	54.658970	52.919666	82.315838	79.696504
5	110.844925	110.439789	98.477960	98.118187	100.171612	99.806326
6	91.058949	99.713340	69.451873	76.052928	87.127798	95.410517
7	100.234416	111.595636	88.761547	98.822681	91.793171	102.197072
8	135.361002	142.036675	117.496278	123.291125	48.615553	51.014467
9	135.539937	141.624842	115.508896	120.694487	48.314996	50.484874
10	129.682870	134.225920	144.391852	149.435049	48.478771	50.176658
11	108.864047	110.429473	153.712471	155.923000	45.342908	45.994801
12	107.665874	109.164185	153.835881	155.976471	44.963156	45.589556
13	125.631589	130.252491	189.323046	196.288995	54.413926	56.414899
14	126.684392	131.231781	189.542747	196.349370	54.966065	56.939989
15	126.201031	131.107031	189.440277	196.803660	54.898058	57.031947

In [13]:

```
# Round the variables in the DataFrame for clarification
df_perc = df_perc.apply(pd.Series.round)
df_perc
```

Out[13]:

	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	Total emissions CO2 [kg]	CO2 emissions/parcel [kg]	Tota emission NOx [kg
0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1	95.0	95.0	95.0	94.0	92.0	92.0	86.0
2	87.0	85.0	82.0	81.0	97.0	96.0	94.0
3	86.0	84.0	68.0	67.0	88.0	86.0	76.0
4	81.0	79.0	55.0	53.0	82.0	80.0	73.0
5	111.0	110.0	98.0	98.0	100.0	100.0	89.0
6	91.0	100.0	69.0	76.0	87.0	95.0	79.0
7	100.0	112.0	89.0	99.0	92.0	102.0	82.0
8	135.0	142.0	117.0	123.0	49.0	51.0	52.0
9	136.0	142.0	116.0	121.0	48.0	50.0	52.0
10	130.0	134.0	144.0	149.0	48.0	50.0	52.0
11	109.0	110.0	154.0	156.0	45.0	46.0	49.0
12	108.0	109.0	154.0	156.0	45.0	46.0	49.0
13	126.0	130.0	189.0	196.0	54.0	56.0	57.0
14	127.0	131.0	190.0	196.0	55.0	57.0	57.0
15	126.0	131.0	189.0	197.0	55.0	57.0	57.0

Store the DataFrame in Excel for further use in the report

In [14]:

```
df_perc.to_excel('Percentages_experiments.xlsx')
```

References

Van Soest, J. (1992). *Elementaire Statistiek*. Delft: VSSD

For further reference, the standard deviations are shown below. These are used in the report to sketch the accuracy of the outcomes

In [16]:

```
df_std / df_mean * 100
```

Out[16]:

	Total simulated time [h]	Parcels in system [pcs]	Parcels delivered [pcs]	Total distance driven [km]	Distance per parcel [km]	Total time operative [h]	Average time/parcel [h]	T emissi CO2 [
0	0.0	0.148776	0.162791	0.572302	0.575610	0.302025	0.298294	1.2042
1	0.0	0.160371	0.174080	0.647265	0.614149	0.332685	0.371608	1.0478
2	0.0	0.133509	0.142266	0.680196	0.704138	0.364550	0.383667	0.9595
3	0.0	0.078511	0.082203	1.054864	1.059558	0.442040	0.445700	0.9966
4	0.0	0.069274	0.071183	0.837664	0.843306	0.388225	0.388121	1.1003
5	0.0	0.159892	0.173730	0.487011	0.529285	0.360770	0.451706	0.9405
6	0.0	0.181924	0.235271	0.613349	0.591681	0.326508	0.378668	0.7725
7	0.0	0.165136	0.219891	0.446641	0.394797	0.244506	0.287359	0.8294
8	0.0	0.301300	0.360801	0.387544	0.309631	0.251720	0.216351	1.4509
9	0.0	0.139245	0.165454	0.327026	0.202591	0.422410	0.311100	1.1736
10	0.0	0.105071	0.122676	1.215352	1.304553	10.212521	10.095362	1.3786
11	0.0	0.081412	0.091533	0.210226	0.222256	0.203518	0.268001	2.0963
12	0.0	0.111624	0.125392	1.020228	1.046623	0.223380	0.245545	1.4465
13	0.0	0.378684	0.443529	0.667242	0.478844	0.266758	0.388542	1.8990
14	0.0	0.398320	0.465789	1.199976	1.084916	0.220947	0.383634	2.0985
15	0.0	0.159934	0.188020	0.682802	0.752936	0.252442	0.173635	0.9397