Optimal allocation of customers to sorting centers - PostNL

Fetsje Bijma[1], Yoeri E. Boink[2], Ajinkya Kadu[3], Jan C. de Munck[1], Rémi de Verclos[4], Tom C. van der Zanden[5], Edwin Zondervan[6]

Abstract

We study the logistics problem associated with transporting parcels from web shops (customers) to sorting centers (depots). Each customer must be assigned to one or more depots where their parcels are sorted. The core of the problem is the trade-off between the transportation costs and the running costs of depots. The organization of depots is bound by several constraints such as minimal working time, maximum work load and maximum storage capacity. We present two approaches to solve this problem, determining the required sorting capacity at each depot and the routing of trucks from customers to depots in an optimal way.

[1]PostNL, The Netherlands fetsje.bijma@postnl.nl, jan.de.munck@postnl.nl

[2]Department of Applied Mathematics, University of Twente, The Netherlands y.e.boink@utwente.nl

[3]Mathematical Institute, Utrecht University, The Netherlands a.a.kadu@uu.nl

[4]Radboud University Nijmegen, r.deverclos@math.ru.nl

[5]Department of Data Analytics and Digitalisation, Maastricht University, The Netherlands T.vanderZanden@maastrichtuniversity.nl

[6]Department of Production Engineering, Bremen University, Germany , edwin.zondervan@uni-bremen.de

2.1 Introduction

PostNL is a mail, parcel and e-commerce company based in the Netherlands. The most important service of PostNL after mail delivery is to transport parcels ordered at web shops (PostNL's customers) to consumers. This service needs to be provided in a timely manner (most parcels have to be delivered within 24 hours) and deals with considerable volumes, which requires a complex logistics process. The PostNL process for transporting parcels from web shops to consumers is divided into separate processes that can be studied and optimized more or less independently: collection, first sorting (to level of rough destination area), transport (cross docking), second sorting (to consumer addresses) and distribution from second sorting center to consumer. In this paper, we focus on the collection process, which comprises the connection between customer and sorting center, and the sorting process.

The problem arises from the regional unbalance of web shops, i.e. most of PostNL's customers are located in the south of The Netherlands, whereas the destination locations of the consumers are more concentrated around the big cities in the west. Therefore, the naive solution that connects each customer to its nearest sorting center would overload the southern sorting centers whereas the northern ones would run under their capacity.

2.1.1 Background

The transportation between customers and sorting centers gives rise to two costs: a *transportation cost*, which is roughly proportional to the total travel time of trucks between customers and sorting center, and a *sorting cost*, that depends on the amount of workforce that is needed to handle the load in a given time frame. In modeling collection systems, decisions are aimed at balancing these two costs. The two cost types are interconnected by the effects of transportation delays on the sorting process, causing potential overloads at some sorting centers. This connection makes it impossible to treat both problems separately.

The problem is complicated further by time constraints. Customers often only have their packages ready for collection late in the evening. This means that sorting centers experience a peak of incoming parcels in the evening, requiring a large capacity/workforce to finish the sorting in time before the deadline. This may cause the sorting center to be overcapacitated for the process earlier in the day, resulting in workers being idle. Furthermore, if parcels have to be driven a large distance (from a customer in the south to a processing center in the north) this will cause the parcels to arrive later, exacerbating the problem. Thus, to ensure efficient utilization of resources, it is important to balance the arrival of parcels over the day as evenly as possible, while taking into account driving times.

2.1.2 Mathematical problem statement

The main question that needs to be answered is "to which depots should each customer be connected?" (see Figure 2.1). The trivial solution of connecting each customer to the nearest depot is not viable, since it causes an overload at some depots, while underutilizing others. Apart from knowing the customer-depot assignment, we also need to know a feasible schedule for collecting parcels from the customers and processing them (in a timely manner) in the depot.

A general mathematical description is as follows. Given are K customers and D depots. The driving times between them are given by a distance matrix $\mathbf{T}$ of size $D \times K$, with each element τ_{dk} representing a distance from depot $d \in \{1, \ldots, D\}$ to customer $k \in \{1, \ldots, K\}$.

- The main goal is to find an optimal assignment between customers and depots, where each customer is only assigned to one depot. this connection is described by $f_{dk} \in \{0, 1\}$. $f_{dk} = 1$ means that customer k is connected to depot d, and 0 means no connection. The task is to find a matrix $\mathbf{F}$ (an assignment matrix of size $D \times K$ with elements f_{dk}) such that the transportation and sorting cost are as small as possible.

- For computing the collection schedule, we also need information about the number of parcels $p(t)_K$ available at time t at customer K. For this goal we make the assumption that each truck is fully loaded, unless it is the last truckload of the day for this customer (there are not enough parcels to fill a truck). This creates the

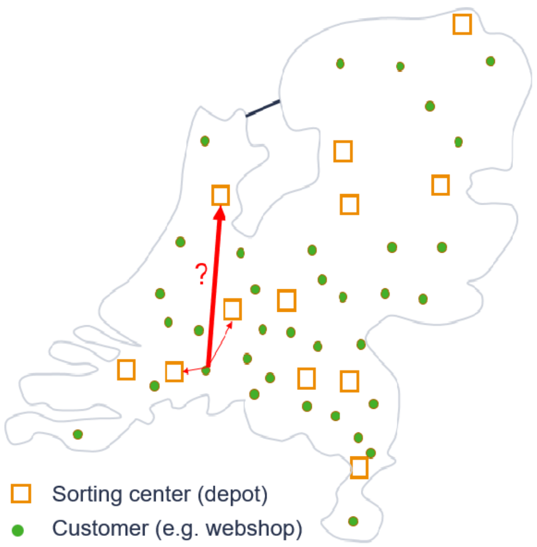

Figure 2.1: Graphical representation of the problem.

variable A: a list of 'customer-truck-loads', whose elements $a_{K,t}$ are all assigned to a customer K with time t.

The sorting speed of depots depends linearly on the number of conveyor belts $\mathbf{n} \in \{0, \ldots, 12\}^D$ used. Each depot is operational between times T_d^{begin} and T_d^{end}. In a simplified case, the sorting costs depend linearly on the number of belts and the total operational time $T_d^{\text{end}} - T_d^{\text{begin}}$, see (2.4). In practice, there is also some constant overhead in the sorting costs. Moreover, using 11 or 12 belts does not actually give a linear increase over using 10 belts. This is because the main carousel belt will become overloaded, resulting in diminishing returns.

We present two approaches to solve the problem, a Convex Programming-based one (section 2.3) and a Bilevel optimization approach (section 2.4). In the respective sections, we provide more specified problem descriptions. The Convex Programming approach is guaranteed to yield an optimal solution but is computationally expensive. The Bilevel optimization approach is not guaranteed to yield an optimal solution, but is faster and does not require the use of an external solver.

Because both methods make use of slightly different assumptions and due to the different nature of both approaches, it is interesting to compare the results of both methods on a example data set. This will be done in section 2.5. Conclusions will be drawn in section 2.6.

2.2 Data and model

The data provided for this case study consists of an anonymized and randomized version of customer orders. Randomization has been performed in such a way that the main characteristics, particularly the imbalance of customers and consumers, is reflected in the pseudo data. Data are given for five weekdays from Monday to Friday. Numerical values of sorting costs and transportation costs used in the model, only reflect the main component of the variable costs. Also, the model description presented here implies a simplification of reality. In practice, parcels of different customers have different sizes and transport and sorting costs depend on parcel weight and size. These parameters are largely dependent on the customer. In addition, parcels with odd sizes cannot be sorted by the sorting machines and need to be sorted by hand. Furthermore, transport capacity is not available in unlimited amounts at all times and in practice transports from small customers to depots are combined in order to achieve much more efficiency in use of transport capacity than suggested by the data set provided as input this case study.

With these simplifications the essence of the depot-customer allocation problem is maintained and methods developed in this paper form a solid basis for future simulation tools for strategic decision making of PostNL.

2.3 Convex Programming

2.3.1 Model development

For the convex programming method, we model the cumulative supply of customer k by a non-decreasing function $0 \leq x_k(t) \leq X_k$, where X_k is the total number of parcels to have been collected by the end of the day from customer k, and t denotes the time. For this method, we make the simplification that each customer is only connected to one depot. this means that it is the task to find $\mathbf{F}$ such that the transportation and sorting cost are as small as possible.

Transportation cost

The *transportation cost* depends linearly on the assignment matrix F. The total transportation cost is

$$\mathcal{J}_T(\mathbf{F}) \triangleq \lambda_T \sum_{d,k} \left\lceil \frac{X_k}{Y} \right\rceil f_{dk}\, \tau_{dk}, \tag{2.1}$$

where λ_T is a known constant converting time per truck in Euros, Y represents the maximum number of parcels that fit in one truck, and $\lceil\ \rceil$ denotes the ceiling operator. To represent the above cost function in terms of matrices and vectors, we introduce a diagonal matrix $\mathbf{L}$ with elements $l_{kk} = \lceil \frac{X_k}{Y} \rceil$ that denotes the number of trucks for each customer k. Hence, an algebraic representation of the *transportation cost* (2.1) is

$$\mathcal{J}_T(\mathbf{F}) \triangleq \lambda_T\, \mathbf{Tr}\left(\mathbf{F}\mathbf{L}\mathbf{T}^T\right), \tag{2.2}$$

where $\mathbf{Tr}\,(\mathbf{X}) = \sum_i x_{ii}$ denotes the trace of the matrix $\mathbf{X}$, and $\mathbf{X}^T$ denotes the transpose of the matrix $\mathbf{X}$. It is important to note few assumptions here: (i) Each customer is connected to a single depot everyday. This may not be the case in general. (ii) We assume that the minimum number of transports are used. Alternatively, more than $\lceil \frac{X_k}{Y} \rceil$ transports could be used, with partial loads to obtain more efficient supply lines at the depot. (iii) We do not account for the possibility to combine transports of different customers in one single trip. We discuss the changes in *transportation cost* later if we do not assume the above assumptions.

Sorting cost

The *sorting cost* requires the information about the supply lines at each depot. We represent the cumulative number of parcels to be sorted at depot d as

$$a_d(t) = \sum_k f_{dk} \hat{S}\left(x_k\left(t - \tau_{dk}\right)\right),$$

where the operator $\hat{S}$ rounds off the customer supply to whole truck loads. Sorting tasks are executed by choosing the number of conveyor belts $n_d \in \{0, 1, \ldots, 12\}$, and hiring a number of labor forces proportional to n_d. The maximum speed by which parcels are sorted is proportional to n_d. Labor forces start sorting parcels at T_d^{begin} and completes the sorting at T_d^{end}. Let us represent the number of conveyor belts as vector $\mathbf{n} \in \{0, \ldots, 12\}^D$. Hence, the *sorting cost* is given by

$$\mathcal{J}_S(\mathbf{F}, \mathbf{n}) = \lambda_S \sum_d n_d \left(T_d^{\text{end}} - T_d^{\text{begin}}\right), \tag{2.3}$$

where λ_S is a known constant. To represent the sorting cost in an algebraic format, we need to introduce a (cumulative) parcel-sort matrix $\mathbf{P} \in \mathbb{R}^{T \times D}$, where T represents the total number of time stamps. An each element of $\mathbf{P}$, denoted by p_{td}, represents the number of parcels sorted till time t at the depot d. Denoting the last row of a matrix $\mathbf{P}$ by $\mathbf{P}_T$ (it is a row vector of length D), we get an algebraic expression for the *sorting cost* in (2.3):

$$\mathcal{J}_S(\mathbf{P}) = \lambda_S \sum_d P_{Td} = \lambda_S \mathbf{P}_T \mathbf{1}, \tag{2.4}$$

where $\mathbf{1}$ is a column vector of length D with every element equal to 1. It is important to note here that $\mathbf{P}$ implicitly depends on the number of conveyor belts $\mathbf{n}$, and the working times of the labor force. Also, the dependency of the parcel-sort matrix $\mathbf{P}$ on an assignment matrix $\mathbf{F}$ will be incorporated through constraints (discussed below).

Constraints

Let us first discuss the constraints on the assignment matrix $\mathbf{F}$. Each element of the matrix $\mathbf{F}$ takes value 0 or 1, *i.e.*, $\mathbf{F} \in \{0, 1\}^{D \times K}$. In other words, matrix $\mathbf{F}$ is a binary matrix. Furthermore, each customer is assigned only one depot:

$$\sum_d f_{dk} = 1 \quad \forall k = 1, \dots, K \qquad \Longrightarrow \qquad \mathbf{F}^T \mathbf{1} = \mathbf{1},$$

where $\mathbf{1}$ is the vector of all ones of an appropriate size. The constraints on the number of conveyor belts are also discrete, *i.e.*, $\mathbf{n} \in \{0, \dots, 12\}^D$.

Now, let us look at the constraints on the parcel-sort matrix. First of all, the parcel-sort matrix represents the cumulative number of parcels sorted at each depot. Hence, all the entries of matrix $\mathbf{P}$ must be non-negative. Furthermore, each column must be non-decreasing. To set this constraint, we look at the gradient of each column of matrix $\mathbf{P}$, $\mathbf{g}_d = \nabla \mathbf{P}_d$, for $d = 1, \dots, D$, where ∇ represents the gradient operator whose discrete form (using forward difference scheme) is given by

$$\nabla = \begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & & \\ & & & -1 & 1 \end{pmatrix},$$

and $\mathbf{P}_d$ denotes the d^{th}-column of matrix $\mathbf{P}$. The vector $\mathbf{g}_d$ gives the time-derivative of parcel-sort function for depot d. The (time-derivative) constraints are:

$$0 \leq \mathbf{g}_d \leq \alpha_d n_d, \qquad \Longrightarrow \qquad 0 \leq \nabla \mathbf{P}_d \leq \alpha_d n_d, \quad \forall\, d = 1, \dots, D,$$

where α_d is a given constant per depot. It is also important to note that $\mathbf{P}_d$ is a piecewise linear. To enforce this constraint, we look at the second-derivative of $\mathbf{P}_d$, and count the size of non-zero elements. For a piecewise linear function, the count is equal to two, *i.e.*,

$$\|\nabla^2 \mathbf{P}_d\|_0 = 2 \qquad \forall d = 1, \dots, D,$$

where $\|\cdot\|_0$ denotes an ℓ_0 norm that counts the number of non-zero elements ($\|\mathbf{x}\|_0 = \sum_i 1_{x_i \neq 0}$), and ∇^2 is a second-order derivative, whose discrete form is represented by

$$\nabla^2 = \begin{pmatrix} 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & & \ddots & & \\ & & & 1 & -2 & 1 \end{pmatrix}.$$

The ℓ_0 norm is a non-convex function, and hence, we reformulate the constraint using its convex approximation. In this case, the second-derivative of the parcel-sort vector for each depots should have jumps that amounts to twice the number of belts. This constraint is imposed as

$$\|\nabla^2 \mathbf{P}_d\|_1 \leq 2n_d \qquad \forall d = 1, \ldots, D,$$

where $\|\cdot\|_1$ is an ℓ_1-norm or Manhattan norm ($\|\mathbf{x}\|_0 = \sum_i |x_i|$). Figure 2.2 shows the behavior of the first-order and second-order derivative for a piecewise linear function. Note that the second-order derivative has only two non-zero values, and they exactly occur during the start and the end time. The sum of absolute values of these two peaks is twice the slope of the function p, and hence $\|\nabla^2 p\|_1 = 2n$, where n is the slope. We argue that the above constraints on the parcel-sort matrix $\mathbf{P}$ makes it a cumulative matrix with respect dimension t and piecewise linear in the same dimension. We enlist these constraints again below:

$$\begin{aligned} &\mathbf{P} \geq 0, \\ &0 \leq \nabla \mathbf{P}_d \leq \alpha_d n_d, && \forall\, d = 1, \ldots, D, \\ &\|\nabla^2 \mathbf{P}_d\|_1 \leq 2n_d && \forall\, d = 1, \ldots, D, \end{aligned}$$

Next, we look at the buffer constraints on the parcel-sort matrix P. Before we jump to the constraints, we define a parcel-receive quantity. We call the matrix $\mathbf{Q}$ as a parcel-receive matrix. It has the same size as matrix $\mathbf{P}$ *i.e.*, $T \times D$. Each element of matrix $\mathbf{Q}$, denoted by q_{td}, represents the number of parcels received until time t at depot d. We can represent each column of matrix $\mathbf{Q}$ using the supply lines of customer and the assignment matrix $\mathbf{F}$. To do this, let us denote the supply lines of customers by matrix $\mathbf{X}$ of size $T \times K$. To account for the

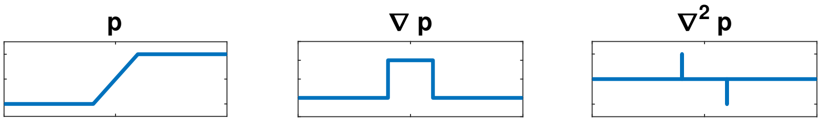

Figure 2.2: Demonstration of piecewise linear function and the behavior of their first-order and second-order derivative.

delay from customer k to depot d (for each pair), we define matrices $\mathbf{X}_d$, for $d = 1, \ldots, D$. They are of size $T \times K$, with each element of matrix $\mathbf{X}_d$ is a delayed version of $\mathbf{X}$ with respect to the time required to reach depot d. Matrices $\mathbf{X}_d$ denotes the supply lines from all customers that is available to depot d, if all of the customers are routed to d. Since the assignment matrix $\mathbf{F}$ denotes the customer-depot relation, we can now rewrite each column of matrix $\mathbf{Q}$ as

$$\mathbf{Q}_d = \mathbf{X}_d \mathbf{F}_d^T \qquad \forall\, d = 1, \ldots, D,$$

where $\mathbf{F}_d$ represents the d^{th}-row of matrix $\mathbf{F}$. We note here that the matrices X_d are known from the supply lines at customers $\mathbf{X}$, and the distance matrix $\mathbf{T}$. The buffer constraints are:

$$\begin{aligned} \mathbf{Q}_d - \mathbf{P}_d &\geq \mathbf{0}, & \forall\, d &= 1, \ldots, D, \\ \mathbf{Q}_d - \mathbf{P}_d &\leq A_d^{\max} \mathbf{1}, & \forall\, d &= 1, \ldots, D, \end{aligned}$$

where $A_d^{\max}$ is the maximum buffer size at depot d. Hence, the buffer constraint on the parcel-sort matrix $\mathbf{P}$ is

$$\mathbf{0} \leq \mathbf{X}_d \mathbf{F}_d^T - \mathbf{P}_d \leq A_d^{\max} \mathbf{1}, \forall\, d = 1, \ldots, D,$$

which gives the relationship between the parcel-sort matrix and the assignment matrix. Finally, we have to take the time-constraints of the labor force into account. Let T_d^{start} and T_d^{end} represents the start and end time of the shift of labor at depot d. Correspondingly, we define vectors $\mathbf{\Gamma}_{d0}$ and $\mathbf{\Gamma}_{d1}$ of length T, such that they denote non-zero elements with respect to T_d^{start} and T_d^{end}. That is,

$$\mathbf{\Gamma}_{d0} = [\underbrace{1\,1\,\ldots\,1}_{0:T_d^{\text{start}}}\,0\,\ldots\,0]^T, \qquad \mathbf{\Gamma}_{d1} = [0\,\ldots\,0\,\underbrace{1\,1\,\ldots\,1}_{T_d^{\text{end}}:T}]^T.$$

Using these vectors, we can write the constraints on the parcel-sort matrix $\mathbf{P}$: (i) No sorting of the parcels till T_d^{start}, and (ii) All parcels must be sorted by T_d^{end}. Hence, the (work) constraints are:

$$\begin{aligned}
&\forall\, d = 1, \dots D \\
&\qquad \boldsymbol{\Gamma}_{d0}^T \mathbf{P}_d = 0 && \text{(no work before } T_d^{\text{start}}) \\
&\qquad \boldsymbol{\Gamma}_{d1}^T \left(\mathbf{X}_d \mathbf{F}_d^T - \alpha_d \mathbf{P}_d\right) = 0 && \text{(parcels sorted by } T_d^{\text{end}})
\end{aligned}$$

Optimization Problem

The goal is to find the optimal assignment matrix $\mathbf{F}^\star$, the number of active belts at sorting depots $\mathbf{n}^\star$ and the work schedules at each depots by minimizing the sum of *transportation cost* and the *sorting cost* subject to the certain constraints described in the subsection above. Our main contribution lies in the introduction of a parcel-sort matrix $\mathbf{P}$ that help in making the cost function *convex*, and most of the constraints *affine*. The optimization problem (2.5) describes the goal:

$$\begin{aligned}
\underset{\mathbf{F},\mathbf{n},\mathbf{P}}{\text{minimize}} \quad & \underbrace{\lambda_T\, \mathbf{Tr}\left(\mathbf{FLT}^T\right)}_{\mathcal{J}_T} \quad + \quad \underbrace{\lambda_S \mathbf{P}_T \mathbf{1}}_{\mathcal{J}_S} \\
\text{subject to} \quad & \mathbf{F} \in \{0,1\}^{D\times K} \quad \mathbf{F}^T\mathbf{1} = \mathbf{1} \\
& \mathbf{n} \in \{0, \dots, 12\}^D \\
& \mathbf{P} \geq \mathbf{0}, \quad \mathbf{0} \leq \nabla \mathbf{P}_d \leq n_d \mathbf{1}, \quad \|\nabla^2 \mathbf{P}_d\|_1 \leq 2n_d \quad \forall\, d = 1, \dots D \\
& \mathbf{0} \leq \mathbf{X}_d \mathbf{F}_d^T - \alpha_d \mathbf{P}_d \leq A_d^{\max} \mathbf{1} \qquad \forall\, d = 1, \dots D \\
& \boldsymbol{\Gamma}_{d0}^T \mathbf{P}_d = 0, \quad \boldsymbol{\Gamma}_{d1}^T \left(\mathbf{X}_d \mathbf{F}_d^T - \alpha_d \mathbf{P}_d\right) = 0 \qquad \forall\, d = 1, \dots D
\end{aligned} \tag{2.5}$$

We note here that the cost parameters λ_T and λ_S, the (diagonal) load matrix $\mathbf{L}$, time matrix $\mathbf{T}$, depot parameters: α_d, $A_d^{\max}$, $\boldsymbol{\Gamma}_{d0}$, $\boldsymbol{\Gamma}_{d1}$ for all $d = 1, \dots, D$, and supply matrices $\mathbf{X}_d$ are given. Except the two constraints, binary structure of $\mathbf{F}$ and integer constraints on $\mathbf{n}$, all other constraints are either affine or semi-definite cones. By relaxing these two constraints (by setting $\mathbf{F} \in [0,1]^{D\times K}$ and $0 \leq \mathbf{n} \leq 12$), we get a relaxed problem that is convex and can be solved very efficiently. But formulation in (2.5) has a minor issue: the requirement on having less number of $\mathbf{n}$ is not taken into account explicitly. One can choose $n_d = 12$ for all $d = \{1, \ldots, D\}$, and still get a solution. In this case, we may lose out the piecewise linearity of $\mathbf{P}$. To mitigate this issue, we add a penalty for the number of belts. The new formulation is

$$\begin{array}{lll}
\underset{\mathbf{F},\mathbf{n},\mathbf{P}}{\text{minimize}} & \underbrace{\lambda_T\, \mathbf{Tr}\left(\mathbf{F}\mathbf{L}\mathbf{T}^T\right)}_{\mathcal{J}_T} \quad + \quad \underbrace{\lambda_S \mathbf{P}_T \mathbf{1}}_{\mathcal{J}_S} \quad + \quad \lambda_n \mathbf{n}^T\mathbf{1} & \\
\text{subject to} & \mathbf{F} \in \{0,1\}^{D\times K} \quad \mathbf{F}^T\mathbf{1} = \mathbf{1} & \\
 & \mathbf{n} \in \{0, \ldots, 12\}^D & \\
 & \mathbf{P} \geq \mathbf{0}, \quad \mathbf{0} \leq \nabla \mathbf{P}_d \leq n_d \mathbf{1}, \quad \|\nabla^2 \mathbf{P}_d\|_1 \leq 2n_d & \forall\, d = 1, \ldots D \\
 & \mathbf{0} \leq \mathbf{X}_d \mathbf{F}_d^T - \alpha_d \mathbf{P}_d \leq A_d^{\max}\mathbf{1} & \forall\, d = 1, \ldots D \\
 & \mathbf{\Gamma}_{d0}^T \mathbf{P}_d = 0, \quad \mathbf{\Gamma}_{d1}^T\left(\mathbf{X}_d\mathbf{F}_d^T - \alpha_d \mathbf{P}_d\right) = 0 & \forall\, d = 1, \ldots D
\end{array} \tag{2.6}$$

where λ_n is the penalty weight for the number of active belts. It can be chosen to have a high value with respect to the transportation cost and sorting cost. For a sufficiently large value of λ_n, we argue that the optimal $\mathbf{P}^\star$ will be piecewise linear with respect to time. This convex optimization problem is a mixed integer program wherein the variables $\mathbf{F}$ and $\mathbf{n}$ are integer types, and $\mathbf{P}$ is continuous. Since the cost function is convex, and the constraints are semi-definite cones (note that affine cones are a subset of semi-definite cones), the optimization problem will give you an optimal solution if it is feasible [Boyd and Vandenberghe (2004)].

2.3.2 Implementation

We use a CVX toolbox [Grant and Boyd (2014)] in MATLAB to solve the optimization problem (2.6). A Gurobi solver [Gurobi Optimization (2020)] helps to speed up the iterative process. Gurobi solver uses simplex method to solve the mixed-integer program. The solver also informs about in-feasibility of the solution when constraints are not satisfied exactly. The computation takes about 2 hours for the non-relaxed problem. The relaxed problem in $\mathbf{n}$ can be solved within a minute.

2.4 Bi-Level optimization

2.4.1 Decomposing the problem

The second approach we propose is a bilevel one. We decompose the problem into two levels:

1. For each depot, choose the paramters: number of conveyor belts $\mathbf{n}$, opening time T_d^{begin}, closing time T_d^{end}.
2. Given these paremeters, find a feasible assignment of customers to depots that minimizes the driving cost.

More mathematically described, we minimize

$$\min_{\mathbf{n}, T_d^{\text{begin}}, T_d^{\text{end}}} \mathcal{J}_T(\mathbf{n}, T_d^{\text{begin}}, T_d^{\text{end}}), \tag{2.7}$$

where $\mathcal{J}_T$ is the solution of the transport problem.

If we make some simplifying assumptions, the lower-level problem can be solved using minimum weight matching. This problem can be efficiently solved using e.g., the Hungarian algorithm (see Pentico (2007) for an overview of algorithms for solving matching problems).

The upper-level problem is in general not easy to solve. However, in part thanks to the relatively low dimension of the problem (24 depots, 12 conveyor belts), we can find a solution by using a heuristic approach (such as local search), by discrete optimization methods or by consulting domain experts and manually changing depot parameters.

To solve the lower-level problem, we need to assume all trucks carry the same number of parcels (i.e., there are no partially loaded trucks) and we need to assume that either the buffer capacities are unlimited or we are allowed to delay collection of parcels from customers. Moreover, we allow one customer to be assigned to multiple distinct depots (if that customer produces more than one full truck load of parcels).

2.4.2 Solving the lower-level problem using minimum weight matching

To solve the lower-level problem, we first discretize the time for each depot into time slots, during which one full truck load of parcels (1400 parcels) can be processed. We build a bipartite graph $G = (A, B, E)$ where the vertices in B correspond to these time slots.

Next, for each customer, we determine how many trucks are needed to collect the total number of parcels. For each customer-truck-load, we create a vertex in A. For each such vertex, we store the time at which the parcels are ready for collection: i.e., we determine what is the first point in time the customer has 1400 parcels available for collection, and create a vertex corresponding to that point in time, then we determine at what point 2800 parcels are available and create a second vertex, etc..

We now create edges between vertices in A and B as follows: For a vertex $a \in A$ and a vertex $b \in B$, we create an edge (a, b) if the time at which the parcels in a are ready for collection, plus the driving time from the customer associated with a to the depot associated with b, is less than the start time of the processing slot associated with b. The weight of this edge is equal to the cost of driving a truck from the customer associated with a to the depot associated with b.

Clearly, minimum weight maximal matching of size $|A|$ in this graph provides an assignment of customer-trucks to depots in which all parcels are processed in time (by construction) and with minimal driving costs (by minimality of the solution). If no maximal matching of size $|A|$ exists, the problem is infeasible.

2.4.3 Dealing with partially filled trucks

The approach described in the previous section can not deal with partially filled trucks. In the provided dataset, there are around 500 trucks which are fully loaded and around 1000 trucks that are partially loaded. Rounding up to full trucks is not an option, as this massively overestimates the total number of parcels and leads to infeasibility. Rounding down is not an option either, since this would ignore a substantial number of customers completely.

Many of the partially-loaded trucks contain only a small number of parcels. Intuitively, it does not make sense to route these trucks far away from the closest depot: if the closest depot is overloaded, it makes much more sense to re-route a full truck to an underutilized depot that is farther away, than to reroute a truck with only 60 parcels to that same alternative depot. Rerouting a full truck gains a much larger redistribution of load for the same driving cost.

Therefore, as a heuristic, we assign partially loaded trucks to the closest depot. This is done as a preprocessing step when building the graph G described in the previous section: for each depot, we start creating time slots (of the length required to process one full truck load) from the opening time of the depot. As soon as a partially filled truck from a customer is available, we create a gap in the schedule to accommodate it. We then continue creating full truck slots, until the next partial truck is available.

For customers that have a number of parcels greater than 1400 but that is not evenly divisible by 1400, we assume the partially filled truck is the one at the end of the day (so that we have some flexibility in case the number of parcels turns out greater than expected).

2.4.4 Buffers: cost and modelling as flow

The previously described algorithm is not able to explicitly take the buffer at the depot into account. The algorithm simply returns a matching between customer truck loads (which are ready for collection at some given time) and a depot time slot (during which time slot the parcels are processed). If the start of the assigned time slot is (much) later than the collection time, we either need to store the parcels in the buffer or delay collection of the parcels from the customer. The latter option is not desirable for PostNL, as the customers generally do not have a large buffer capacity themselves and thus want the parcels picked up as soon as possible.

One possible way to model this in the previous model is to enforce that parcels ready for collection at some time must be matched to a time slot not much later in time (by simply not creating the corresponding edges); the maximum delay can be related to the processing speed of the depot and size of the buffer to obtain a guarantee the buffer never overflows. However, this approach means the buffer may be underutilized. A longer delay can be used, but this runs the risk of returning an infeasible solution with buffer overflows. Alternatively, a cost (possibly superlinear) could be associated with delaying the pick up of parcels.

It is however possible to take the buffers into account explicitly by switching from a matching formulation to a minimum-cost flow formulation. We have nodes corresponding to customer-truck-loads in A as before, nodes corresponding to processing time slots in B as before, but also have one source node s and one sink node t. We create an edge with capacity 1 and cost 0 from s to every node in A, for each node $a \in A$ and every depot, we create an edge from a to the node $b \in B$ corresponding to the first possible processing time slot at that depot; the capacity of this edge is 1 and the cost is equal to the corresponding driving cost. Further, possible extra buffers associated with a cost (such as the cost of keeping a retaining a truck parked next to a depot) can be also be integrating to the model.

Next, for each depot, the buffers are modelled as follows: from each time slot node $b_1 \in B$ we create an edge with capacity equal to the buffer size (in whole trucks) and cost 0 to the immediately succeeding time slot node $b_2 \in B$. Sending flow over an edge $b_1 b_2$ models keeping the corresponding truck loads in the buffer. Finally, we create an edge from every node $b \in B$ to t with capacity 1 and weight 0; sending flow over this edge corresponds to processing one truck load at that point in time. An example of this construction is illustrated on Figure 2.3.

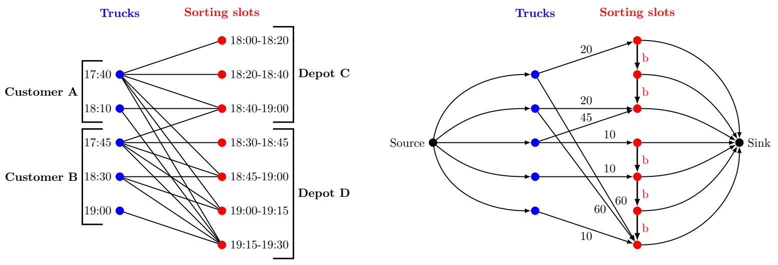

Figure 2.3: Construction of the minimum-cost flow problem. Here the driving times are $\tau_{A,C} = 20$, $\tau_{A,D} = 60$, $\tau_{B,C} = 45$ and $\tau_{B,D} = 10$ in minutes. Left: the graph for the maximum matching problem. Right: The graph for the minimum-cost problem. The label on the edges correspond to their cost. Every edge has capacity 1 except the edges between red vertices whose capacity is the buffer capacity of the depot.

An integral flow form s to t with value n in this graph corresponds to the assignment of n trucks to the depot that respect both buffer and sorting constraints. The cost of such a flow correspond to the corresponding traveling cost. Therefore, looking for a flow of minimal cost and value equal to the total number of trucks solves our problem.

The minimum-cost flow problem can be solved in polynomial time by the Network Simplex Algorithm [Orlin (1997) and Tarjan (1997)]. Moreover, the solution provided by this algorithm is guaranteed to have integral values provided that all capacities are integers, which is the case in our application.

2.4.5 Metaheuristics

The upper-level problem could be solved using any of a number of metaheuristics, such as local search, simulated annealing or gradient descent. Alternatively, the upper-level problem could be solved manually, by using expert domain knowledge and making informed, iterative guesses at the solution. Since there are currently only 24 depots, it could also be possible to explore a large part of the search space by brute force (e.g., if we wanted to completely close a number of depots, it is easily possible to explore all possible subsets of depots that can be closed given that we may only close a small number due to capacity constraints).

As an example, we implemented a simple heuristic to optimize the upper-level problem. With each depot i we associate a real number $d_i \in [0, 1]$, representing which fraction of the depot's maximum capacity is used (e.g., $d_i = 1$ corresponds to using all 12 belts from the earliest possible starting time to the latest possible closing time).

Given a vector of utilization fractions for all depots, for each depot we compute the minimum number of belts required to achieve this fraction (rounding up). We fix the closing time at the latest possible closing time, and then compute the required starting time to achieve exactly the requested fraction of processing capacity.

This approach again simplifies the problem somewhat, since we do not consider the possibility of closing the depot earlier or using more belts and starting later to process the same number of parcels. However, PostNL prefers solutions using fewer belts (as this makes the solution more robust) and closing a depot earlier does not seem to be a beneficial choice since most processing capacity is required late in the evening.

We now repeatedly apply the following procedure: we pick a uniform random depot i, and with probability 1/2 we increase its load d_i by a small amount ϵ, and with probability 1/2 we decrease it by the same amount. This amount ϵ decreases as the algorithm progresses. We then recompute the solution to the lower-level problem, and accept the change if the score decreases, and reject the change if the score increases.

2.4.6 Implementation and computation time

We prepared an implementation of the approach using minimum weight matching in C#, using Google's ORTools as solver for the matching problem. The underlying matching problem can be solved in a fraction of a second, and within two minutes our simple local search heuristic finds a solution.

2.5 Results and discussion

In this section we discuss and compare results for both methods. Since Monday is the busiest day of the week, this is the most challenging to optimize for. Therefore we only consider this day for our results.

2.5.1 Comparison between both methods

Without disclosing the given numbers for the sorting costs, transport costs and the data sets, we will provide some rough numbers and visualizations as results from both methods. As a benchmark, we computed the minimum sorting costs and the minimum transport costs, provided in Table 2.1.

The minimum sorting costs comprise the costs to sort all packages by any depot, the minimum transport costs comprise the costs of transporting all parcels to the nearest depot, independent of buffer capacity and other constraints. In the same table, the solution for our methods is provided. It can be seen that both methods perform very similar

	benchmark	Convex programming	Bilevel optimization
Transport costs	22.3k€	24.1k€	24.3k€
Sorting costs	164.4k€	164.4k€	164.4k€
Total costs	186.7k€	188.4k€	188.7k€

Table 2.1: Costs for our solutions compared to benchmark.

with only a small difference (less than 2.0k€) from the benchmark solution. Both methods have found a viable solution where the sorting costs are the same for both methods and the benchmark: this means that there is no idle time at the depots. It can be seen that both methods find a solution within 10% of the benchmark transport costs. This is not very surprising, because the bulk of the parcels are expected to be transported to the nearest depot. Since the transport costs are only a small fraction of the total costs, the total costs only increase by 1.1% compared to the benchmark.

Figure 2.4 shows a visualization of the matching between customers with depots for both methods. It can be seen that both methods provide a solution in which most customers are connected to a depot nearby: there is no transport over large distances. Note that in Figure 2.4b some customers are connected with multiple depots, which is not possible in Figure 2.4a, due the the assumptions. This figure only shows the matching of customers with depots, without any indication of the time of the transport.

A different visualization from Figure 2.4b, using the same bilevel solution, is given in Figure 2.5. Here it can be nicely seen that many depots mainly take customers south of its location, especially in the south of the Netherlands (Noord-Brabant) and in the north (Friesland and Groningen). This is an expected result, since it was already stated in the problem description that there are relatively few depots for the number of parcels in the south. Because of this, the 'mean' transport has to be from south to north.

In both methods, there is no constraint on the number of trucks available. This is done intentionally, because the number of available

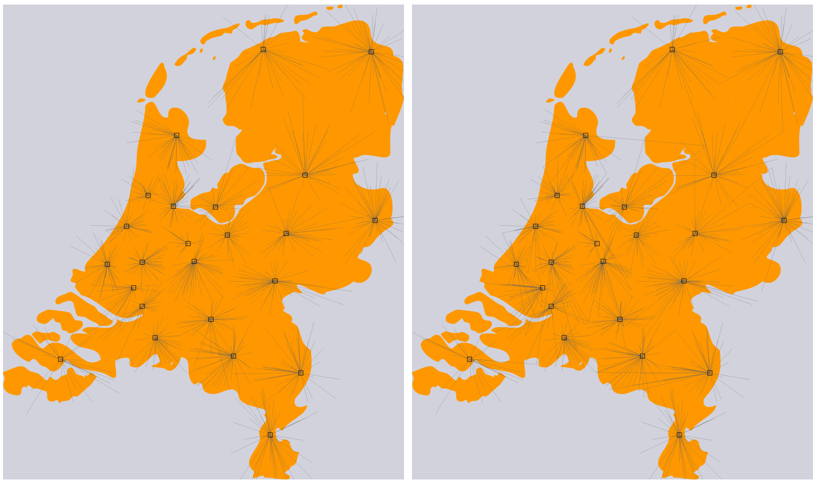
(a) Convex programming (b) Bilevel optimization

Figure 2.4: Visual matching of customers with depots for both methods. In this visualization, two depots were mistakenly closed in (a).

trucks depends on other parts of the full process, described in the introduction (section 2.1). In Figure 2.6, the number of necessary trucks is plotted against time of the day. The feasibility of such a demand is beyond the scope of this paper.

2.5.2 Closing early or closing a depot

PostNL notices that due to various reasons, depots do not always manage to finish the sorting before they officially close. If the closing time would be set 30 minutes earlier, this would create a 'buffer' of 30 minutes to handle this unfinished sorting. Moreover, it would be interesting to see the effects of closing one depot entirely. This could be done deliberately, to save certain costs for instance, or it can have an unintentional external reason, such as a malfunction.

In Table 2.2, the normal scenario is compared with the scenario where all depots are closed 30 minutes early and the scenario where

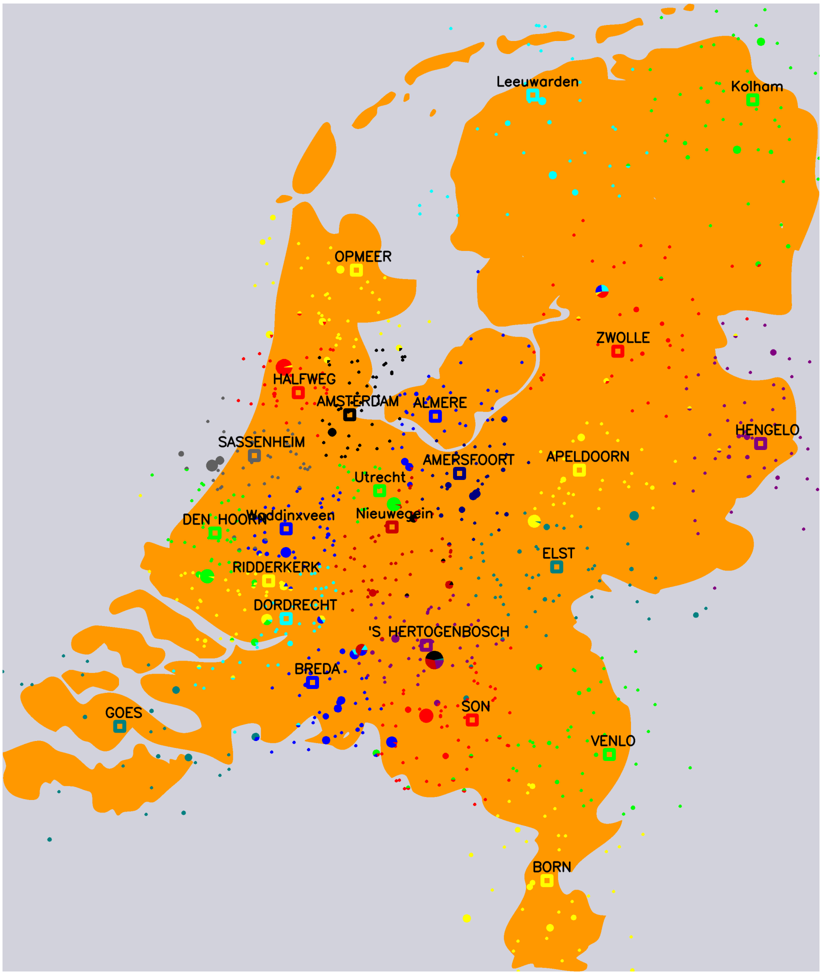

Figure 2.5: Colorful representation of the bilevel solution matching customers (disks) with depots (squares).

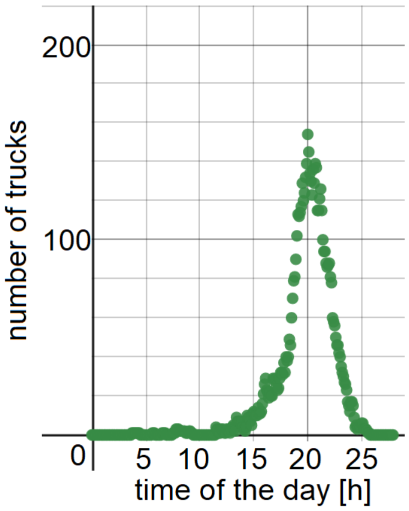

Figure 2.6: Number of trucks necessary as function of time (hours) for the bilevel solution. A time after 24h means the truck is driving after midnight.

the depot in Opmeer is closed respectively. We state the number of conveyor belts used and the usage of the depot as percentage of the maximum capacity, when all belts are used for the maximum possible time. Note that the maximum capacity decreases when all depots close 30 minutes early. For this comparison, we made use of the bilevel optimization method.

First of all, in the bottom of Table 2.2 it can be seen that the total costs do not increase a lot when all depots are closed early or if one depot is closed entirely. Secondly, in case of a 30 minute early closure, depots are generally used more, as expected. However, some of them (e.g. Opmeer, Amersfoort) are used significantly more, while others (e.g. Breda) are even used less. Something similar can be seen in the case where Opmeer is closed entirely: Amersfoort is used a lot more, while Breda is used less. Finally, in both alterations, Kolham is used to almost full capacity. This shows that the alterations truly have a 'global' (or national) effect in the solution, which shows the necessity for finding the solution automatically.

2.5.3 Advantages and disadvantages of both methods

Here we summarize some advantages and disadvantages of both methods by comparing them in several aspects.

- Convex programming provides an exact solution in finite time, while it is not guaranteed that Bilevel optimization solves the given optimization problem, due to the heuristic approach.
- Convex programming solves a problem where each customer needs to be assigned to one depot, which might not be optimal. Bilevel optimization does not have this restriction.
- It is relatively easy to add additional constraints to the convex programming method, while it is not clear if this is possible for the bilevel method.
- Convex programming is relatively slow (unrelaxed) or does not provide a usable solution (relaxed), while bilevel optimization provides a usable solution in less than 2 minutes.

	normal		30 minutes early		Opmeer closed	
Depot	#belts	usage	#belts	usage	#belts	usage
Apeldoorn	10	80%	10	88%	10	84 %
Waddinxveen	10	80%	10	78%	10	81 %
Halfweg	10	88%	10	92%	10	90 %
Opmeer	**8**	**70**%	**10**	**82**%	**0**	**0** %
Den Hoorn	10	84%	10	86%	10	88 %
Amersfoort	**8**	**67**%	**10**	**76**%	**10**	**78** %
Den Bosch	8	62%	8	69%	8	66 %
Nieuwegein	8	72%	8	68%	10	78 %
Kolham	**10**	**92**%	**12**	**94**%	**12**	**98** %
Zwolle	6	38%	6	46%	6	33 %
Goes	8	67%	8	69%	8	68 %
Amsterdam	10	86%	10	90%	10	88 %
Sassenheim	10	80%	10	86%	10	86 %
Ridderkerk	10	76%	10	78%	10	78 %
Utrecht	6	44%	6	52%	6	48 %
Hengelo	8	58%	8	60%	8	68 %
Elst	10	84%	10	84%	10	80 %
Dordrecht	8	62%	8	70%	8	65 %
Breda	**10**	**77**%	**8**	**70**%	**8**	**70** %
Venlo	8	62%	8	62%	8	64 %
Born	10	74%	10	76%	10	78 %
Almere	8	74%	10	78%	8	72 %
Son	10	77%	10	83%	8	70 %
Leeuwarden	10	80%	10	86%	10	84 %
Total costs	188.7k€		189.6k€		190.0k€	

Table 2.2: number of conveyor belts and usage as percentage of the maximum capacity.

- Bilevel optimization can compute a very fast solution (less than a second) once the operational constraints (number of belts, operating time) has been set. This can be very useful for 'real-time' distribution of parcels once the depots have opened. Moreover it can be used in case a depot suddenly has to close due to unexpected circumstances. This is not possible for the convex programming method.
- The bilevel optimization method first assigns all non-full trucks to the nearest depot. If this already creates an overflow in the depots, the rest of the method does not work. This could be circumvented by assigning a lower 'average' truckload to all trucks, which results in a less exact solution. However, it is not likely that the non-full trucks already create an overflow in the depots.

2.6 Conclusion

In this paper, we have investigated two methods that solve an optimal allocation problem for a parcel delivery company in the Netherlands. Both methods, using convex programming and a bilevel optimization approach respectively, find an optimal solution to the problem. The solutions are slightly different due to different assumptions. It was found that the optimal solution for a Monday, the busiest day, has a transport cost that is only 10% higher than allocating each customer to the nearest depot. This means that near-optimal allocation is possible. Moreover, idle time at depots is avoided completely, resulting in a sorting cost that is optimal.

Our two solution methods make it possible to quickly compute the effect of many changes, such as opening times of depots, location of (new) depots, closure of a depot and sorting speed of depots. This enables the company to quickly investigate the effect from changes in their operation on the parcel collection process and its finances. As an example, we have shown that closing 30 minutes early or closing one depot entirely is possible and not very expensive.

The focus of this paper was in the allocation of parcels to depots coming from customers. This is only a part of the full process of handling all parcels, therefore our optimal solutions might not be optimal for the full process. Further research could be put in other parts of the process and connecting all of them in one large optimization problem. Since this might not be feasible, another option is to include stochasticity in the data and the assumptions to find a suboptimal, but robust solution.

References

Boyd, Stephen and Lieven Vandenberghe (2004). *Convex Optimization*. Cambridge University Press.

Grant, Michael and Stephen Boyd (Mar. 2014). *CVX: Matlab Software for Disciplined Convex Programming, version 2.1*. `http://cvxr.com/cvx`.

Gurobi Optimization, LLC (2020). *Gurobi Optimizer Reference Manual*. URL: `http://www.gurobi.com`.

Orlin, James B. (1997). "A polynomial time primal network simplex algorithm for minimum cost flows". In: *Math. Program.* 77, pp. 109–129. DOI: `10.1007/BF02614365`.

Pentico, David W. (2007). "Assignment problems: A golden anniversary survey". In: *European Journal of Operational Research* 176.2, pp. 774–793. ISSN: 0377-2217. DOI: `https://doi.org/10.1016/j.ejor.2005.09.014`.

Tarjan, Robert Endre (1997). "Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm". In: *Math. Program.* 77, pp. 169–177. DOI: `10.1007/BF02614369`.