



PLAATSING AMBULANCE LOCATIES

ANNIKA VAN ADRICHEM
NEDERLAND

MAAIKE BUKMAN
NEDERLAND

STEPHAN DEMMENDAL
NEDERLAND

JORIS VAN DER WAGT
NEDERLAND

3 juni 2021

Samenvatting

Inhoudsopgave

Samenvatting	2
Lijst met variabelen	5
1 Introductie	6
2 Intuïtieve methode	7
3 Lineair programmeringsmodel	9
3.1 Maximalisering bereikte inwoners	9
3.2 Minimalisering reistijd	11
3.3 Uitbreiding	12
4 Resultaten	15
4.1 Intuïtieve methode	15
4.2 lineaire programmeringsmethode	15
5 Conclusies	17
Bibliography	18
A Intuïtieve methode	19

B	Lineair programmeringsmodel	22
C	Lineair programmeringsmodel uitbreiding1	26

Lijst met variabelen


I	Verzameling van de te bereiken postcodes.
J	Verzameling van postcodes waarin de ambulanceposten staan.
i	De te bereiken postcode waarbij $i \in I$.
j	Postcode waar een ambulancepost staat waarbij $j \in J$
t_{ji}	Afstand tussen de plaats met een ambulancepost, j , en postcode, i , in seconden.
d_i	Aantal inwoners van de betreffende postcode.
p_i	
y_{ji}	Binaire variabele die afhangt van t_{ji} .
z_j	Postcodes waar een ambulancepost staat.
n	Aantal postcode in de betreffende regio.
k	Aantal ambulanceposten dat geplaatst mag worden in de betreffende regio.
q_{ji}	
k	
l	

Hoofdstuk 1

Introductie

Wanneer er een ongeluk plaatsvindt, is het belangrijk dat een ambulance zo snel mogelijk ter plekke is om iemand met zwaar letsel alhier te behandelen en haastig naar het ziekenhuis te vervoeren. Als de ambulance niet binnen een bepaalde tijd aanwezig is bij de patiënt, kan het voorkomen dat het letsel fataal is en de patiënt het niet haalt. Om deze kans zo klein mogelijk te maken, moeten de ambulances op de meest efficiënte locaties geplaatst worden, zodat ze een zo groot mogelijke populatie binnen 15 minuten bereiken. Wanneer een ambulance niet gebruikt wordt, staat deze dus op deze aangewezen plaats gestationeerd.

In dit onderzoek wordt gekeken naar de beste locaties voor de ambulanceposten in twee verschillende gebieden: De regio Flevoland en regio Noordoost-Gelderland, die respectievelijk uit 91 en 201 postcodes bestaan. Is het mogelijk om met drie of vijf ambulances zoveel mogelijk inwoners te bereiken binnen 15 minuten vanaf de ambulanceposten en de reistijd tot alle inwoners te minimaliseren?

Deze onderzoeksvraag wordt beantwoord door middel van een paar stappen. Eerst wordt er in hoofdstuk 2  naar een intuïtieve methode. Dit houdt in dat er geprobeerd is logischerwijs is nagedacht over het probleem en er een eerste stap richting de optimale oplossing wordt gezet. Vervolgens wordt er in hoofdstuk 3 een algemeen geheeltallig lineair programmeringsmodel opgesteld. Dit houdt in dat de opgestelde functies in *Python* worden gezet en met behulp van *PuLP* wordt een oplossing gevonden en dit model wordt daarna geoptimaliseerd voor de tijd. Dit omdat het wenselijk is om alle inwoners in zo min mogelijk tijd te bereiken, ook buiten de gewenste 15 minuten.

Hoofdstuk 2

Intuïtieve methode

Het doel van dit onderzoek is om binnen 15 minuten zo veel mogelijk mensen te kunnen bereiken met ambulances, hiervoor wordt gekeken naar de plaatsing van een bepaald aantal ambulanceposten, afhankelijk van de grootte van het gebied. Hiervoor is een intuïtieve methode bedacht. Deze code is te zien in Appendix A.

Om te beginnen wordt er bij deze code de verkregen informatie uit het Excelbestand in lijsten geconverteerd, betreffende de tabbladen *Traveltimes (seconds)* en *Demand* die respectievelijk in de lijsten *traveltimes_lst* en *demand_lst* worden omgezet. Dit bevordert de snelheid van het programma om reden dat het programma de gegevens nu rechtstreeks uit de lijst kan halen en hiervoor niet steeds het Excelbestand hoeft te lezen.

Vervolgens wordt er gekeken naar het aantal te bereiken postcodes dat elke ambulancepost in een postcode kan bereiken. Dit wordt gedaan door het implementeren van een functie die het aantal bereikte postcodes per ambulancepost in postcodes telt en opslaat welke specifieke postcodes de individuele postcodes bereiken. Deze gegevens worden gebruikt om het daadwerkelijke aantal mensen die bereikt worden te berekenen.

Nu is het doel om uiteindelijk zoveel mogelijk mensen te bereiken met het aantal ambulanceposten, dus intuïtief wordt er voor de plek van de eerste ambulancepost gekozen voor de postcode die de meeste mensen kan bereiken.

Om de volgende postcode om een ambulancepost te plaatsen te bepalen, wordt er gekeken naar de grootste postcode die het meeste inwoners kan bereiken na de eerst gekozen postcode. Hierbij worden de al bereikte postcodes weggehaald, omdat deze inwoners al bereikt zijn. Dit wordt gedaan door voor alle postcodes nog een keer af te gaan welke postcodes bereikt kunnen worden door de specifieke postcode, hierbij worden de al bereikte postcodes wegge-

haald. Vervolgens wordt er gekeken naar de postcode die dan de grootste hoeveelheid mensen kan bereiken. En deze postcode wordt dan gekozen voor het plaatsen van een ambulancepost.

Dit proces wordt herhaald tot er evenveel postcodes in de lijst *uitcode* staan als dat er ambulance centrales geplaatst mogen worden. Uiteindelijk staan er in deze lijst dus de rijnummers van de gewenste postcode die eenvoudig terug omgezet worden naar de echte postcodes. Hiermee is het beoogde resultaat behaald.

Hoofdstuk 3

Lineair programmeringsmodel

Na het opstellen van een intuïtieve methode wordt er met behulp van o.a. *PuLP* en *CPLEX* een Algemeen geheeltallig lineair programmeringsmodel opgesteld en uiteindelijk opgelost.

Dit wordt gedaan in twee stappen: Eerst wordt er gekeken naar het maximaal aantal inwoners dat bereikt kan worden binnen 15 minuten. Daarna wordt er gekeken wat de minimale tijd is om alle inwoners te kunnen bereiken zonder dat het aantal mensen binnen het bereik van 15 minuten afneemt.

3.1 Maximalisering bereikte inwoners

Allereerst het maximale inwoners die te bereiken zijn binnen 15 minuten. Er moeten een aantal variabelen worden gedefinieerd, hierbij is er gekozen om te werken met veelal binaire variabelen.

J is de set met postcodes die een ambulancepost hebben en t_{ji} geeft de reistijd van postcode j naar postcode i aan.

De binaire variabele y_{ji} geeft aan of t_{ij} minder is dan 900 seconden.

$$y_{ji} = \begin{cases} 1 & \text{Als } t_{ji} \leq 900 \\ 0 & \text{Als } t_{ji} > 900 \end{cases}$$



De variabele z_j geeft aan of postcode j in de set J zit, dus of postcode j een ambulancepost heeft.

$$z_j = \begin{cases} 1 & \text{Als } j \in J \\ 0 & \text{Als } j \notin J \end{cases}$$

De binaire variabele p_i geeft aan of de postcode i door een ambulancepost, postcode in set J , bereikt kan worden.

$$p_i = \begin{cases} 1 & \text{Als } y_{ji} = 1 \wedge z_j = 1 \\ 0 & \text{Als } y_{ji} = 0 \vee z_j = 0 \end{cases}$$

De variabele d_i geeft het aantal inwoners aan in postcode i . Er is ook een parameter k nodig, die aangeeft hoeveel ambulanceposten geplaatst kunnen worden.

$$\text{Maximaliseer } \sum_{j=1}^n d_j p_i \quad (3.1)$$

$$\text{Zódat } \sum_{j=1}^n z_j = k \quad (3.2)$$

$$\sum_{j \in J} z_j y_{ji} \geq p_i \quad \text{voor } i \in I \quad (3.3)$$

In de objectieve maximalisatie functie 3.1 word het inwonersaantal van iedere postcode vermenigvuldigd met de binaire variabele p_i .

Het uiteindelijke doel is in beginsel het maximaliseren van het bereiken van de mensen in het betreffende gebied. Dit wordt behaald door het inwoner-aantal van iedere postcode te vermenigvuldigen met een binaire variabele die aangeeft of de postcode zal worden bereikt door de plek van de ambulancepost.

Hiervoor is er eerst per postcode gekeken welke postcodes deze wel en niet binnen de voorgeschreven 15 minuten kan bereiken, dit wordt opgeslagen in de parameter y_{ji} . De beperking 3.2 geeft aan dat er een aantal k ambulanceposten geplaatst worden. In dit geval zijn dat er voor Flevoland en Noordoost-Gelderland respectievelijk 3 en 5 posten.

De variabele p_i is afhankelijk van zowel y_i als z_j . De beperking 3.3 zorgt ervoor dat p_i 1 is als deze bereikt kan worden door één of meer ambulanceposten en 0 als dat niet het geval is. Met andere woorden: Wanneer er meerdere postcodes door verschillende ambulanceposten bereikt worden deze postcodes maar één keer bij de totaal bereikte postcodes worden geteld, zo wordt overlapping voorkomen.

Uit de maximalisering functie komen de drie beste postcodes waar de ambulanceposten het best geplaatst kunnen worden zodat de er zoveel mogelijk inwoners binnen 15 minuten bereikt worden.

In Appendix B is te zien hoe dit in een model is gezet met behulp van *PuLP*. Door het definiëren van de variabelen, de objectieve functie en de beperkingen berekent *PuLP* het optimale resultaat. Zoals in Appendix B te zien is, is in het stuk "1" maximaliseren van het aantal te bereiken mensen eerst het maximalisatie probleem gedefinieerd en zijn daarna de variabelen benoemd. Vervolgens is de objectieve functie aangemaakt en zijn de beperkingen aangegeven, waarna het probleem wordt opgelost. Nu *Python* berekent heeft hoeveel inwoners binnen 15 minuten bereikt worden en wat de ideale postcodes zijn om een ambulancepost neer te zetten moeten deze nog wel geprint worden zoals onderaan Appendix B te zien is.

3.2 Minimalisering reistijd

De volgende stap is het kijken naar de hoeveelheid tijd die het kost om alle inwoners in een gebied te bereiken. Hierbij wordt onderscheid gemaakt tussen de postcodes die binnen en buiten 15 minuten bereikt worden door de postcodes met een ambulancepost. Dit wordt verkregen met een minimalisering en twee beperkingen. Hierbij is de binaire variabele q_{ji} nodig die aangeeft of postcode j de dichtstbijzijnde ambulancepost is voor i .

$$q_{ji} = \begin{cases} 1 & \text{Als } t_{j1i} \leq (t_{j2i} \wedge \dots \wedge t_{jk i}), \quad \text{met } j_1 \dots j_k \in J \\ 0 & \text{Als } t_{j1i} \not\leq (t_{j2i} \wedge \dots \wedge t_{jk i}), \quad \text{met } j_1 \dots j_k \in J \end{cases}$$

Met deze nieuwe variabelen q_{ji} kan een nieuwe functie opgesteld worden. Dit is de minimaliseringsfunctie van de reistijd waarbij ook een aantal beperkingen horen.

$$\text{Minimaliseer} \quad \sum_{j=1}^n \sum_{i=1}^n q_{ji} t_{ji} \quad (3.4)$$

$$\text{Zódat} \quad \sum_{j=1}^n z_j = k \quad (3.5)$$

$$\sum_{j \in J} z_j y_{ji} \geq p_i \quad \text{voor } i \in I \quad (3.6)$$

$$\sum_{j=1}^n d_i p_i \geq u \quad (3.7)$$

$$q_{ji} \leq z_j \quad \text{voor } i \in I \text{ en } j \in J \quad (3.8)$$

$$\sum_{j=1}^n q_{ji} = 1 \quad \text{voor } i \in I \quad (3.9)$$

Wat uiteindelijk geminimaliseerd moet worden is de totale reistijd waarin alle inwoners bereikt worden. De objectieve minimalisering functie 3.4 is daarom gedefinieerd als $\sum_{j=1}^n \sum_{i=1}^n q_{ji} t_{ji}$, waarin q een matrix is met één 1 in elke kolom en t een matrix met de reistijden als waarde. Doordat er in elke kolom maar één 1 staat, wordt er maar een reistijd bij elkaar opgeteld. Dit is de snelste reistijd naar de te bereiken postcode. Alle snelste reistijden worden bij elkaar opgeteld en deze moet zo minimaal mogelijk zijn.

De beperkingen die hierbij horen zijn de beperkingen van objectieve functie 3.1 en 3 extra beperkingen.

De eerste extra beperking is functie 3.7. Hierin wordt er een eis gesteld aan hoeveel inwoners bereikt worden in 15 minuten. De maximalisatie van het aantal inwoners is dus bepaald met functie 3.1. De minimalisering van de reistijd mag het aantal bereikte inwoners binnen 15 minuten niet verminderen en dus is een beperking dat het aantal mensen die bereikt worden in 15 minuten in 3.1 gelijk aan of kleiner dan het aantal dat nu bereikt gaat worden.

De beperking 3.8 is een erg simpele beperking. Deze beperking geeft aan dat wanneer de postcode $j \notin J$, oftewel postcode j is geen ambulancepost en $z_j = 0$, dan is q_{ji} kleiner of gelijk aan z_j en dus is $q_{ji} = 0$. Als $j \in J$ dan is $z_j = 1$ en dus geldt $q_{ji} = 1$ of $q_{ji} = 0$. Om te bepalen of deze waarde 0 of 1 is, is beperking 3.9 opgesteld. Deze zegt dat voor het aantal postcodes met een ambulancepost, alleen diegene met de kortste reistijd naar postcode i een $q_{ji} = 1$ heeft en de andere zijn postcodes met een ambulancepost hebben een $q_{ji} = 0$.

In appendix B is uitgewerkt onder het kopje "Het minimaliseren van de reistijd". Eerst is een minimalisatie probleem opgesteld en vervolgens de variabelen die hiervoor nodig zijn. Daarna zijn de objectieve functie en de beperkingen benoemd met daaropvolgend de oplossing. Tenslotte worden de postcodes waar de ambulanceposten moeten komen te staan voor een optimale reistijd geprint.

3.3 Uitbreiding

Als uitbreiding op het model is er gekeken naar de marginale verschillen tussen het variëren van het aantal ambulanceposten en hoeveel ambulanceposten er nodig zouden zijn om alle mensen te bereiken binnen 15 minuten.

Aan het bestaande model B zijn een aantal stukken code toegevoegd. Zo wordt er bepaald hoeveel mensen er in zich in totaal op de verschillende post-

codes bevinden. Vervolgens worden het "aantalposts" niet meer als vaste input data gelezen maar wordt er met behulp van een while loop gekeken naar het verschil tussen een verschillend aantal posten is. Deze stopt wanneer het totaal aantal mensen wordt bereikt. De output van zowel de gekozen postcodes met hierbij het aantal bereikte mensen en de geminimaliseerde totale reistijd wordt voor elke extra ambulancepost geprint. Dit levert de volgende tabellen op:

Aantal ambulanceposten	Bepaalde postcodes	Aantal bereikte mensen	Totale reistijd <i>Seconden</i>
1	1339	210749	86111
2	1331, 8255	328944	69001
3	1349, 8219, 8301	374949	51563
4	1351, 3891, 8219, 8301	386184	36565

Aantal ambulanceposten	Bepaalde postcodes	Aantal bereikte mensen	Totale reistijd <i>Seconden</i>
1	7341	234435	322884
2	7054, 7341	434065	180621
3	7054, 7382, 8072	575725	158977
4	3843, 7051, 7251, 7341	676750	137778
5	3843, 7005, 7141, 7231, 7341	754150	113133
6	3843, 7005, 7141, 7231, 7341, 8094	806530	106632
7	3843, 7007, 7109, 7204, 7273, 7341, 8094	809530	96332
8	3843, 7007, 7109, 7204, 7273, 7331, 7397, 8094	809865	91975

De eerste tabel is representatief voor de kleine data set en tweede tabel voor de grote data set. In de eerste tabel is te zien dat er met een extra ambulancepost alle mensen bereikt kunnen worden. Wat opvalt is dat er bij de toevoeging van een extra ambulancepost de voorgaande ambulanceposten zich niet op dezelfde plek bevinden. Ondanks dat het aantal bereikte mensen met 12000 toeneemt neemt de reistijd met een vergelijkbaar aantal af ten op zichte van de afname van 2 naar 3 ambulanceposten.

Bij het extra plaatsen van ambulanceposten voor de grote data set in de tweede tabel geldt dat alle postcodes en hierbij dus ook alle mensen worden bereikt

met 8 ambulanceposten. In tegenstelling tot het toevoegen van slechts 1 ambulancepost voor de eerste data-set zijn er in de tweede data-set 3 extra posten nodig. Het toevoegen van de zesde ambulance post heeft marginaal het grootste effect op het totaal aantal bereikte mensen vergeleken met de zevende en achtste ambulancepost. Dit is te verklaren doordat de data-set groter is en dus meer postcodes bevat die niet in een kluster bevat zijn met daarin meerdere postcodes. Er zijn dus een groter aantal postcodes die uit de buurt liggen van de rest. Wat opvalt is dat de marginale afname van de reistijd na het plaatsen van de 5e ambulancepost het grootste is voor het plaatsen van de 7e ambulancepost.

Hoofdstuk 4

Resultaten

In dit onderzoek wordt er voor de beste locaties voor de ambulanceposten naar twee specifieke regio's gekeken: Flevoland en Noordoost-Gelderland. Voor de kleine data set van Flevoland mogen er drie posten geplaatst worden. Voor de grote data set van Noordoost-Gelderland zijn dit er vijf.

4.1 Intuïtieve methode

Het intuïtieve model geef als resultaat dat de drie ambulanceposten in Flevoland het best geplaatst kunnen worden in de postcodes: 1339, 8255, 8315. Hiermee worden in totaal 346414 van de 386184 inwoners in deze regio bereikt binnen de 15 minuten.

In de regio Noordoost-Gelderland worden er vijf ambulanceposten geplaatst. Hierbij komt er uit het model dat de ambulanceposten het best geplaatst kunnen worden in de postcodes: 7341, 7054, 3847, 7251, 7141. En bij deze gekozen postcodes worden 738465 van de 809865 inwoners bereikt binnen de 15 minuten.

4.2 lineaire programmeringsmethode

Het lineaire programmeringsmodel geeft voor de regio Flevoland aan dat de drie ambulanceposten het best geplaatst kunnen worden in de postcodes: 1349, 8219, 8301. Hierbij worden 374949 van de 386184 inwoners bereikt binnen 15 minuten. De totale tijd van het bereiken van alle inwoners is precies 51563 seconden.

Voor de regio Noordoost-Gelderland kunnen de vijf ambulanceposten het best geplaatst worden in de postcodes: 3843, 7005, 7141, 7231, 7341. Hierbij worden 754150 van de 809865 inwoners bereikt binnen 15 minuten. Alle inwoners worden binnen 113133 seconden bereikt.

Hoofdstuk 5

Conclusies

Bibliografie

Bijlage A

Intuïtieve methode

```
import pandas as pd
import math

# Het Excelbestand omzetten in lijsten per tabblad
traveltimes = pd.read_excel('Data assignment ambulance 1 Small.xlsx',
    skiprows = 1, sheet_name = "Traveltimes (seconds)")
traveltimes_lst = traveltimes.values.tolist()
count = -1
for i in traveltimes_lst:          # de lege hokjes omzetten naar een 0
    count = count + 1
    count2 = -1
    for k in i:
        count2 = count2 + 1
        if math.isnan(k) == True:
            traveltimes_lst[count][count2] = 0

demand = pd.read_excel('Data assignment ambulance 1 Small.xlsx',
    skiprows = 0, sheet_name = "Demand")
demand_lst = demand.values.tolist()

general = pd.read_excel('Data assignment ambulance 1 Small.xlsx',
    skiprows = 0, sheet_name = "General Information")
general_lst = general.values.tolist()

# Het aantal postcodes aflezen uit de gemaakte lijsten
aantal = general_lst[0][1]
aantalposts = general_lst[2][1]
```

```

# De lege lijsten maken die later nodig zijn
postclst = []
mensenlst = []
codelst = []

# Postcode is het kolomnummer van de postcode in het Excelbestand

# Functie die het aantal postcodes berekent dat kan worden bereikt
def aantalbereikt(aantal,postcode):
    count = 0
    lst = []
    for i in range(0,aantal):
        data = traveltimes_lst[postcode][i+1]
        if data < 900:
            count = count + 1
            lst.append(i)
    return count,lst

# Functie die het aantal mensen teruggeeft die een postcode bereikt
def aantalmensen(lst):
    mensen = 0
    for i in lst:
        aantal = demand_lst[i][1]
        mensen = mensen + aantal
    return mensen

# Functie die het aantal bereikte postcodes berekent minus die al bereikt zijn

def hoogsteweg(aantal,postcode,uitcodeplus):
    lst = []
    for n in range(0,aantal):
        if n not in uitcodeplus:
            data = traveltimes_lst[postcode][n+1]
            if data < 900:
                lst.append(n)
    return lst

def tweede_elem(elem):
    return elem[1]

for k in range(0,aantal):
    # Welke postcodes specifiek
    postclst.append([k,aantalbereikt(aantal,k)[1]])

```

```

for m in range(0,aantal):          # [postcode,mensen bereiken]
    mensenlst.append([m,aantalmensen(postclst[m][1])])

mensenlst.sort(key=tweede_elem)
meestemensen = mensenlst[-1]
uitcode = [meestemensen[0]]        # Eerst gekozen postcode
                                     # Lijst met bereikte postcodes
                                     # voor de 'grootste' postcode
postcodelst = aantalbereikt(aantal,meestemensen[0])[1]
uitcodeplus = postcodelst
mensencount = meestemensen[1]

for z in range(0,aantalposts-1):
    postclst = []
    mensenlst = []
    for j in range(0,aantal):        # [postcode,[postcodes]]
        postclst.append([j,hoogsteweg(aantal,j,uitcodeplus)])
    for p in range(0,aantal):        # [postcode,mensen bereiken]
        mensenlst.append([p,aantalmensen(postclst[p][1])])

    mensenlst.sort(key=tweede_elem)
    meestemensen = mensenlst[-1]
    uitcode.append(meestemensen[0]) # Tweede gekozen postcode
    postcodelst = aantalbereikt(aantal,meestemensen[0])[1]
        # Lijst met de bereikte postcodes voor de tweede grootste postcode
    uitcodeplus = uitcodeplus + postcodelst
    mensencount = mensencount + meestemensen[1]

# Postcode rijnummer naar postcode zelf
for t in uitcode:
    codelst.append(demand_lst[t][0])

print(codelst)
print("Aantal bereikte mensen:",mensencount)

```

Bijlage B

Lineair programmeringsmodel

```
from pulp import *
import pandas as pd
import math
# Verzamelen data door Excelbestanden te lezen en deze informatie toe
# te voegen aan drie verschillende lijsten

df = pd.read_excel('Data assignment ambulance 1 Large.xlsx', skiprows = 1,
sheet_name = "Traveltimes (seconds)")
df_lst = df.values.tolist()

dp = pd.read_excel('Data assignment ambulance 1 Large.xlsx', skiprows = 0,
sheet_name = "Demand")
dp_lst = dp.values.tolist()

dq = pd.read_excel('Data assignment ambulance 1 Large.xlsx', skiprows = 0,
sheet_name = "General Information")
dq_lst = dq.values.tolist()

# Nul maken van lege vakjes
count = -1
for i in df_lst:
    count = count + 1
    count2 = -1
    for k in i:
        count2 = count2 + 1
        if math.isnan(k) == True:
            df_lst[count][count2] = 0
```

```

aantal = dq_lst[0][1]      # Aantal te bereiken postcodes toegevoegd aan aantal
aantalposts = dq_lst[2][1] # Aantal posten toegevoegd aan aantalposts

# Vier lege lijsten
postclst = []
mensenlst = []
codelst = []
y = []

# Functie voor het omzetten van de tijd die het kost voor een postcode
om een andere postcode te bereiken, binair

def aantalbereikt(aantal,postcode):
    count = 0
    yj = []
    for i in range(0,aantal):
        data = df_lst[postcode][i+1] # Lezen van de tijd die het kost om
                                     # een andere postcode te bereiken
        if data < 900:                # Als de gelezen tijd minder is dan 900
                                     # wordt er een 1 toegevoegd aan yj
            yj.append(1)
        else:                         # Als de gelezen tijd meer is dan 900
                                     # wordt er een 0 toegevoegd aan yj
            yj.append(0)
    return yj

# Functie aantalbereikt wordt voor alle postcodes aangeroepen
# Deze lijsten worden toegevoegd aan de lijst y zodat er een matrix ontstaat
for k in range(0,aantal):
    y.append(aantalbereikt(aantal,k))

# MAXIMALISEREN VAN HET AANTAL TE BEREIKEN MENSEN

# Creëren LP Probleem voor het maximaliseren van het aantal te bereiken mensen
Ambulances = LpProblem("Ambulances", LpMaximize)

# Introduceren van de pj variabele
pj = LpVariable.dicts("Bewoners", (range(aantal)),cat = "Binary")

# Introduceren van de zj variabele
zj = LpVariable.dicts("Base_Locations",(range(aantal)),cat = "Binary")

# Objective Function
Ambulances += lpSum([dp_lst[i][1]*pj[i] for i in range(aantal)])

```

```

# Constraint Aantal Ambulanceposten
Ambulances += lpSum([zj[i] for i in range(aantal)]) == aantalposts

# Constraint p koppelen aan z met behulp van yji
for i in range(aantal):
    Ambulances += lpSum([zj[j] * y[j][i] for j in range(aantal)]) >= pj[i]

# Oplossen van het probleem
Ambulances.solve(CPLEX_CMD(msg=0))

# Print status
print("Status:", LpStatus[Ambulances.status])

# Printen waarde van de objective function
print("Totale inwoners:", value(Ambulances.objective))

# Printen oplossing
count = -1
ideale_postcodes = [] #Lijst met de ideale postcodes
for i in range(0,len(zj)):
    count = count + 1
    if value(zj[i]) == 1:
        print(dp_lst[count][0])
        ideale_postcodes.append(dp_lst[count][0])

# HET MINIMALISEREN VAN DE REISTIJD

# Creëren LP Probleem voor het minimaliseren van de tijd
Tijd = LpProblem("Tijd", LpMinimize)

# Introduceer qij variabele
qji = LpVariable.dicts("MatrixQ",(range(aantal),range(aantal)), cat = "Binary")

# Introduceer di variabele
pj = LpVariable.dicts("Bewoners", (range(aantal)),cat = "Binary")

# Introduceer pj variabele
zj = LpVariable.dicts("Base_Locations",(range(aantal)),cat = "Binary")

# Objective Function
Tijd += lpSum([qji[j][i]*df_lst[j][i+1] for i in range(aantal-1)
               for j in range(aantal)])

```



```

# Constraint Aantal Ambulanceposten
Tijd += lpSum([zj[i] for i in range(aantal)]) == aantalposts

# Constraint p aan z koppelen met y
for i in range(aantal):
    Tijd += lpSum([zj[j] * y[j][i] for j in range(aantal)]) >= pj[i]

# Constraint aantal bereikte mensen
Tijd += lpSum([dp_lst[i][1]*pj[i] for i in range(aantal)]) >=
    value(Ambulances.objective)

# Constraint q aan z koppelen
for j in range(aantal):
    for i in range(aantal):
        Tijd += lpSum([qji[j][i]]) <= zj[j]

# Constraint q alleen de laagste nemen
for i in range(aantal):
    Tijd += lpSum([qji[j][i] for j in range(aantal)]) == 1

# Oplossen van het probleem
Tijd.solve(CPLEX_CMD(msg=0))

# Print status
print("Status:", LpStatus[Tijd.status])

# Print objective function value
print("Totale reistijd:", value(Tijd.objective))

# Printen oplossing
count = -1
for i in range(0, len(zj)):
    count = count + 1
    if value(zj[i]) == 1:
        print(dp_lst[count][0])

```

Bijlage C

Lineair programmeringsmodel uitbreiding1

```
from pulp import *
import pandas as pd
import math
# Verzamelen data door Excelbestanden te lezen en deze informatie toe
te voegen aan drie verschillende lijsten

df = pd.read_excel('Data assignment ambulance 1 Large.xlsx',skiprows = 1,
sheet_name = "Traveltimes (seconds)")
df_lst = df.values.tolist()

dp = pd.read_excel('Data assignment ambulance 1 Large.xlsx',skiprows = 0,
sheet_name = "Demand")
dp_lst = dp.values.tolist()

dq = pd.read_excel('Data assignment ambulance 1 Large.xlsx',skiprows = 0,
sheet_name = "General Information")
dq_lst = dq.values.tolist()

# Nul maken van lege vakjes
count = -1
for i in df_lst:
    count = count + 1
    count2 = -1
    for k in i:
        count2 = count2 + 1
        if math.isnan(k) == True:
            df_lst[count][count2] = 0
```

```

x = 0
for i in dp_lst:          # Totaal aantal inwoners tellen
    x = x+ i[-1]
print(x)

aantal = dq_lst[0][1]      # Aantal te bereiken postcodes toegevoegd aan aantal
aantalposts = dq_lst[2][1] # Aantal posten toegevoegd aan aantalposts

# Vier lege lijsten
postclst = []
mensenlst = []
codelst = []
y = []

# Functie voor het omzetten van de tijd die het kost voor een postcode
om een andere postcode te bereiken, binair

def aantalbereikt(aantal,postcode):
    count = 0
    yj = []
    for i in range(0,aantal):
        data = df_lst[postcode][i+1] # Lezen van de tijd die het kost om
                                     # een andere postcode te bereiken
        if data < 900:                # Als de gelezen tijd minder is dan 900
                                     # wordt er een 1 toegevoegd aan yj
            yj.append(1)
        else:                         # Als de gelezen tijd meer is dan 900
                                     # wordt er een 0 toegevoegd aan yj
            yj.append(0)
    return yj

# Functie aantalbereikt wordt voor alle postcodes aangeroepen
# Deze lijsten worden toegevoegd aan de lijst y zodat er een matrix ontstaat

for k in range(0,aantal):
    y.append(aantalbereikt(aantal,k))
posts = 1

k = 0
while k < x:          # Waarbij x het totaal aantal inwoners
    print("Aantal postst is: ",posts)

```

```

# HET MAXIMALISEREN VAN HET AANTAL TE BEREIKEN MENSEN

# Creëren LP Probleem voor maximaliseren van het aantal te bereiken mensen
Ambulances = LpProblem("Ambulances", LpMaximize)

# INTRODUCEREN VAN DE VARIABELEN

# Introduceren van de pj variabele
pj = LpVariable.dicts("Bewoners", (range(aantal)), cat = "Binary")

# Introduceren van de zj variabele
zj = LpVariable.dicts("Base_Locations", (range(aantal)), cat = "Binary")

# Objective Function
Ambulances += lpSum([dp_lst[i][1]*pj[i] for i in range(aantal)])

# Constraint Aantal Ambulance-posten
Ambulances += lpSum([zj[i] for i in range(aantal)]) == posts

# Constraint p koppelen aan z met behulp van yji
for i in range(aantal):
    Ambulances += lpSum([zj[j] * y[j][i] for j in range(aantal)]) >= pj[i]

# Oplossen van het probleem
Ambulances.solve(CPLEX_CMD(msg=0))

# Print status
print("Status:", LpStatus[Ambulances.status])

# Printen waarde van de objective function
print("Totale inwoners:", value(Ambulances.objective))
k = value(Ambulances.objective)

# Printen oplossing
count = -1
ideale_postcodes = [] #Lijst met de ideale postcodes
for i in range(0,len(zj)):
    count = count + 1
    if value(zj[i]) == 1:
        print(dp_lst[count][0])
        ideale_postcodes.append(dp_lst[count][0])

```

```

# HET MINIMALISEREN VAN DE REISTIJD

# Creëren LP Probleem voor het minimaliseren van de tijd
Tijd = LpProblem("Tijd", LpMinimize)

# Introduceer qij variabele
qji = LpVariable.dicts("MatrixQ", (range(aantal), range(aantal)),
    cat = "Binary")

# Introduceer di variabele
pj = LpVariable.dicts("Bewoners", (range(aantal)), cat = "Binary")

# Introduceer pj variabele
zj = LpVariable.dicts("Base_Locations", (range(aantal)), cat = "Binary")

# Objective Function
Tijd += lpSum([qji[j][i]*df_lst[j][i+1] for i in range(aantal-1)
    for j in range(aantal)])

# Constraint Aantal Ambulanceposten
Tijd += lpSum([zj[i] for i in range(aantal)]) == posts

# Constraint p aan z koppelen met y
for i in range(aantal):
    Tijd += lpSum([zj[j] * y[j][i] for j in range(aantal)]) >= pj[i]

# Constraint aantal bereikte mensen
Tijd += lpSum([dp_lst[i][1]*pj[i] for i in range(aantal)]) >=
    value(Ambulances.objective)

# Constraint q aan z koppelen
for j in range(aantal):
    for i in range(aantal):
        Tijd += lpSum([qji[j][i]]) <= zj[j]

# Constraint q alleen de laagste nemen
for i in range(aantal):
    Tijd += lpSum([qji[j][i] for j in range(aantal)]) == 1

# Oplossen van het probleem
Tijd.solve(CPLEX_CMD(msg=0))

# Print status
print("Status:", LpStatus[Tijd.status])

```

```
# Print objective function value
print("Totale reistijd:", value(Tijd.objective))

# Printen oplossing
count = -1
for i in range(0,len(zj)):
    count = count + 1
    if value(zj[i]) == 1:
        print(dp_lst[count][0])
posts += 1
```