KU LEUVEN

FACULTY OF
ENGINEERING SCIENCE

Voorkeuren analyseren van dagelijkse routeplanners met conjoint analysis

Achilles Demey

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen, hoofdoptie
Artificiële intelligentie

Promotor:
Prof. dr. T. Guns

Evaluatoren:
Dr. ir. A. van den Berghe
S. Kolb

Begeleider:
Dr. V. Bucarey

Academiejaar 2021 – 2022

Voorwoord

Deze thesis concludeert mijn Master in Ingenieurswetenschappen: Computerwetenschappen aan KU Leuven. Ik bedank mijn promotor Prof. dr. Tias Guns om mij dit interessant onderwerp toe te vertrouwen. Daarnaast bedank ik mijn dagelijkse begeleider dr. Victor Bucarey van harte om tijd vrij te maken voor dit onderzoek en mij sporadisch te bij te staan bij het uitpuzzelen van de details met zijn brede kennis over het domein. Graag zou ik Jayanta Mandi willen bedanken om mij technisch te assisteren bij het onderzoeken van de dataset.

Daarnaast wil ik graag mijn ouders, familie en vrienden bedanken voor hun onverbiddelijke steun door de jaren heen. Tot slot wil ik mijn dankwoord richten aan *Dire Straits* om met hun harmonieuze muziek mijn creatieve inspiratie iedere dag aan te wakkeren.

Ik wens u veel plezier bij het lezen van deze tekst in de hoop dat u nieuwe inzichten zal verwerven.

Achilles Demey

Inhoudsopgave

Voorwoord i

Samenvatting iv

Lijst van figuren en tabellen v

Lijst van afkortingen en symbolen vii

1 Inleiding 1

2 Achtergrond 5
- 2.1 Travelling Salesman Problem . . . 5
- 2.2 Capacitated Vehicle Routing Problem . . . 6
- 2.3 Conjoint Analysis . . . 9
- 2.4 Genetische Algoritmen . . . 11

3 Gerelateerd Werk 13
- 3.1 Transitieprobabiliteiten . . . 13

4 Probleemstelling 17

5 Aanpak 19
- 5.1 Abstractieniveaus . . . 20
- 5.2 Twee Fasen . . . 24

6 Voorkeuren 27
- 6.1 Datageneratie . . . 27
- 6.2 Conjoint Analysis . . . 28
- 6.3 Preferentiemodel . . . 33

7 Clustering 35
- 7.1 Preferenties . . . 35
- 7.2 Algoritme . . . 41

8 Routing 43
- 8.1 Preferenties . . . 43
- 8.2 Algoritme . . . 47

9 Dataset 49
- 9.1 Beschrijving . . . 49
- 9.2 Extractie van informatie . . . 50
- 9.3 Analyse . . . 51

10 Experimenten **53**
10.1 Methodologie . . . 53
10.2 Clustering . . . 54
10.3 Routing . . . 56
10.4 Discussie . . . 58

11 Besluit **61**

A Algoritmen **67**
A.1 Datageneratie . . . 67
A.2 Training van het preferentiemodel . . . 68
A.3 Evaluatie van het preferentiemodel . . . 70
A.4 Perturbatiemechanisme voor clustering . . . 72
A.5 Algoritme voor clustering . . . 73
A.6 Algoritme voor routing . . . 74
A.7 Padreconstructie . . . 76

B Historisch voorbeeld **79**

C Experimentele resultaten **81**
C.1 Clustering . . . 81
C.2 Routing . . . 84

D Poster **89**

E Fiche Masterproef **91**

Bibliografie **93**

Samenvatting

Dit werk brengt een beslissingsondersteunend systeem voor routeplanning tot stand voor kleine en middelgrote bedrijven. Het systeem is in staat om impliciete voorkeuren te leren uit historische trajecten die door een menselijke planner gemaakt werden. Deze persoon brengt vaak een aantal persoonlijke voorkeuren en doelstellingen van het bedrijf in het plan zoals een gelijke verdeling van de klanten onder de bestuurders. We gaan ervan uit dat dezelfde klanten vaak terugkomen in oplossingen. Het systeem is in staat om aan de hand van de geleerde voorkeuren nieuwe plannen op te stellen zoals de menselijke planner het zou doen. In de praktijk stelt men echter voor elk bedrijf een eigen *vehicle routing problem* op met een eigen set aan objectieven en *constraints*. De methode die dit werk voorstelt, is een alternatieve manier om oplossingen te genereren die zowel de planner als de bestuurders kunnen bekoren. Het onderliggend model die de voorkeuren leert en nieuwe oplossingen evalueert, is gebaseerd op *conjoint analysis*. Deze techniek analyseert een aantal attributen van de historische trajecten zoals geometrische aspecten of de lengte van de routes. Voor elk van deze attributen leert het model wat de geprefereerde waarden zijn. Op basis van het *conjoint model* zoekt een metaheuristisch algoritme naar de oplossing met de meest geprefereerde attribuutwaarden. Het werk neemt een twee-fasen aanpak waarbij de klanten eerst onder de bestuurders worden verdeeld en vervolgens voor iedere bestuurder het beste pad wordt gezocht.

Lijst van figuren en tabellen

Lijst van figuren

2.1 Travelling Salesman Problem 5
2.2 Capacitated Vehicle Routing Problem 7
2.3 Oplossingsmethodes voor het CVRP 9
2.4 Nutsvergelijkingen 11
2.5 Framework voor genetische algoritmen 12

3.1 Voorbeeld van transitieprobabiliteiten 15

5.1 Grafische voorstelling van de abstractieniveaus 20
5.2 Framework 26

6.1 Attribuutprobabiliteit 32
6.2 Bilateraal dalende probabiliteiten 32

7.1 Attributen van een cluster (Knoop 4) 38

8.1 Attributen van een pad (Knoop 4) 46

9.1 Analyse van de stops 51

10.1 Accuraatheid van het clusteralgoritme 54
10.2 Accuraatheid van het routingalgoritme 56
10.3 Afvlakken van attribuutprobabiliteit 58

11.1 Algemeen framework 64

A.1 Probleemgeval voor padreconstructie 77

Lijst van tabellen

2.1 Voorbeeld van choice-based conjoint analysis 9

5.1 Attributen van de abstractieniveaus 21

6.1 Voorbeeld van een discrete regressie-analyse 30

7.1 Tabel met attributen van clusters 37
7.2 Contextvariabelen van het clusteringprobleem 39
7.3 Voorbeeld van contextvariabelen 39
7.4 Metrieken voor contextsimilariteit 40

8.1 Tabel met attributen van paden 45

9.1 Inconsistenties in de dataset 52

10.1 Vergelijking tussen de preferentiescore van historische clusters en voorspelde clusters 55
10.2 Vergelijking tussen de preferentiescore van historische oplossingen en voorspelde oplossingen 57

B.1 Historisch voorbeeld 80

C.1 Resultaten van clustering voor *data drift* ($\tau = 0, \rho = 90, \omega = 125$) 82
C.2 Resultaten van clustering na *data drift* ($\tau = 125, \rho = 160, \omega = 200$) . . . 83
C.3 Resultaten van routing ($lineair, \tau = 130, \rho = 165, \omega = 200$) 85
C.4 Resultaten van routing ($ellipsvormig, \tau = 130, \rho = 165, \omega = 200$) 86
C.5 Resultaten van routing ($exponentieel, p = 0.1, \tau = 130, \rho = 165, \omega = 200$) 87
C.6 Resultaten van routing ($exponentieel, p = 0.01, \tau = 130, \rho = 165, \omega = 200$) 88

Lijst van afkortingen en symbolen

Afkortingen

KMO	Kleine of middelgrote onderneming
VRP	Vehicle routing problem
CVRP	Capacitated vehicle routing problem
CA	Conjoint analysis
MIP	Mixed Integer Programming
HMM	Hidden Markov Model
CBC	Choice-based Conjoint Analysis
TSP	Traveling Salesman Problem
ILP	Integer Linear Programming
GA	Genetisch Algoritme
LR	Logistic Regression
MNL	Multinomial Logit Model
OLS	Ordinary Least Squares
PCA	Principal Component Analysis
MDS	Multidimensional Scaling
RD	Route Difference
AD	Arc Difference

Symbolen

X	Historische oplossing
C	Cluster
P	Pad
k	Knoop
Pr	Probabiliteit
M	*Conjoint* model
U	Nut of *utility*
A	Attribuut
Pr_A	Attribuutprobabiliteit
β	Nut van een discreet attribuutsniveau
w	Gewicht
$f(x)$	Bilateraal dalende probabiliteit
τ	Index begin training set
ρ	Index eind training set en begin test set
ω	Index eind test set

Hoofdstuk 1

Inleiding

In de logistieke afdeling van kleine en middelgrote ondernemingen (KMO's) is het *vehicle routing problem* (VRP) een gekend onderwerp. Deze bedrijven hebben een beperkt aantal voertuigen, bestuurders en klanten die op regelmatige basis bezocht moeten worden. Het is vaak de taak van een werknemer om elke dag een nieuw routeplan op te stellen. De planner probeert doorgaans een objectieve kost zoals de totale afstand, het brandstofverbruik of de CO_2-emissie te minimaliseren. Daarnaast moet de planner rekening houden met beperkingen op het aantal wagens, de capaciteit van de voertuigen en de *timeframes* waarin de bestuurders de klanten kunnen bezoeken. Er bestaan softwarepakketten die specifiek gemaakt zijn voor het produceren van optimale routes door enkele knopen wanneer de objectieve kost tussen deze knopen gegeven is. Deze oplossingen voldoen echter vaak niet aan de verwachtingen van de planner en de bestuurders [5]. Een onderzoek naar de werkwijze van menselijke planners leidde tot de observatie dat planners eerst een gelijkaardig probleem zoeken uit het verleden of een optimaal plan genereren met optimalisatiesoftware. De planners gebruiken deze oplossing als *template* waaraan ze iteratief aanpassingen maken. Dit resulteert in het finale plan dat de bestuurders moeten uitvoeren [4]. Op deze manier brengen planners hun eigen voorkeuren in de oplossing. De ene planner kan bijvoorbeeld rekening houden met een correcte verdeling het werk tussen de bestuurders. Een andere planner kan denken aan de eisen van de bestuurders. Sommige bestuurders willen bepaalde snelwegen vermijden tijdens de ochtendspits en weigeren te rijden op wegen die in slechte staat zijn. Daarnaast heeft elke bestuurder een aantal preferenties over de plaats waar ze middagpauze nemen of waar ze brandstof bijvullen. Al deze informatie is impliciet vervat in de historische plannen en daar valt dus veel uit te leren.

De opgenoemde voorkeuren zijn complex en bijna onmogelijk formeel neer te schrijven. Dit werk ontwikkelt een nieuw *framework* om voorkeuren impliciet uit historische data te leren. Het voorgestelde model zal dus iets over de voorkeuren leren zonder exact te specifiëren wat de voorkeuren zijn. Daarnaast ontwikkelt dit werk een metaheuristisch algoritme om oplossingen te vinden voor nieuwe VRP's aan de hand van de geleerde preferenties. Het algoritme is in staat om een oplossing te vinden die zowel de menselijke planner als de bestuurders kan bekoren.

Het *vehicle routing problem* is een verzamelnaam voor varianten van distributieproblemen en te algemeen om te onderzoeken in deze thesis [24]. Daarom ligt de focus in dit werk op het *capacitated vehicle routing problem* waarin er één centraal depot is en de wagens een limiet hebben op de capaciteit. Het model dat het mogelijk maakt om de voorkeuren te leren, vindt zijn oorsprong in *conjoint analysis*. Dit is een statistische marktonderzoekmethode waarmee men te weten kan komen hoe klanten verschillende eigenschappen van een product waarderen en keuzes maken tussen meerdere alternatieve producten.

Dit werk is bestaat uit meerdere hoofdstukken. Hoofdstuk 2 laat de lezer kennis maken met de belangrijkste onderwerpen in dit werk zoals het *travelling salesman problem*, het *capacitated vehicle routing problem*, *conjoint analysis* en *genetische algoritmen*. Hoofdstuk 3 geeft de essentie van de onderzoeken die dit werk voorafgaan. Deze onderzoeken leggen de fundamenten voor modellen die impliciet voorkeuren leren uit historische oplossingen van *vehicle routing problems*.

Hoofdstuk 4 geeft de probleemstelling van deze thesis in het kader van de geïntroduceerde concepten. Vervolgens bespreekt hoofdstuk 5 de methodologie die dit werk hanteert. Het hoofdstuk leidt de lezer door verschillende niveaus waarop men een historisch routeplan kan analyseren en stelt vervolgens de algemene objectieve vergelijking op van het voorkeurgebaseerde optimalisatieprobleem.

Hoofdstuk 6 bespreekt zeer nauwkeurig hoe *conjoint analysis* geïncorporeerd is in het *preferentiemodel*. De lezer komt hier in contact met zowel de populairste vorm van *conjoint analysis* onder marketeers als een volledig nieuw model dat gericht is op het leren maken van soortgelijke beslissingen. Alle aspecten zoals datageneratie, training en evaluatie van het model zijn ondersteund door algoritmen in de appendices. Het opstellen van het preferentiemodel is slechts de eerste helft van het werk. Het zal voor de lezer duidelijk zijn dat de objectieve vergelijking incompatibel is met alle *state of the art solvers*. Daarom stelt dit hoofdstuk een twee-fasen aanpak voor om het meest geprefereerde routeplan te vinden in de ruimte van alle mogelijke oplossingen. Beide fasen gebruiken een eigen preferentiemodel met verschillende attributen.

Hoofdstuk 7 behandelt de technieken die we in de eerste fase aanwenden. Deze fase verdeelt de klanten onder de bestuurders en noemen we het *clusterprobleem*. Hoofdstuk 8 bespreekt de tweede fase waarin het *routingalgoritme* het meest geprefereerde pad door de knopen van de clusters vindt. Beide hoofdstukken zijn op dezelfde manier opgebouwd. De eerste helft van elk hoofdstuk bespreekt het preferentiemodel voor elk deelprobleem in detail. Hier komt de lezer voor elk probleem de attributenkeuze te weten en hoe elk model de context van nieuwe problemen in rekening brengt. De tweede helft van elk hoofdstuk introduceert een metaheuristisch algoritme voor het vinden van de meest geprefereerde oplossing. Voor transparantie naar de lezer toe bevat dit werk alle algoritmen in de appendices.

Hoofdstukken 9 en 10 ondersteunen de theoretische bevindingen van dit onderzoek met enkele experimenten en resultaten. Hoofdstuk 9 geeft een analyse van de dataset met enkele visualisaties. Daarnaast bespreekt dit hoofdstuk enkele *preprocessingstappen* en een aantal inherente problemen aan de dataset. Hoofdstuk 10 is volledig toegewijd aan het experimentele design, de uitvoering van de experimenten en een kritische bespreking van de resultaten.

Aangezien dit onderzoek een onbetreden pad afwandelt bij het leren maken van gelijkaardige beslissingen, vormt er zich een volledige nieuwe horizon aan mogelijkheden. Hoofdstuk 11 concludeert dit werk met alle bijdragen en stelt een aantal potentiële uitbreidende onderzoeken voor.

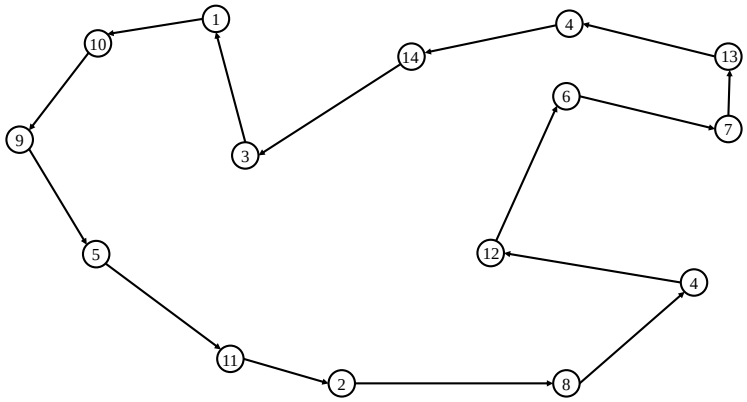
Hoofdstuk 2

Achtergrond

Dit hoofdstuk geeft op een intuïtieve manier een beschrijving van de wetenschappelijke domeinen waar deze thesis zich in afspeelt. Het is noodzakelijk de geïntroduceerde concepten volledig te begrijpen vooraleer verder te gaan naar latere delen. Secties 2.1 en 2.2 geven een introductie tot de wereld van de *vehicle routing problems*. Het eerste deel behandelt het *travelling salesman problem* en het tweede deel generaliseert dit probleem tot het *capacitated vehicle routing problem*. In sectie 2.3 maakt de lezer op een luchtige manier kennis met *conjoint analysis* aan de hand van een schoolvoorbeeld. Het laatste deel 2.4 bespreekt *genetische algoritmen* op hoog niveau.

2.1 Travelling Salesman Problem

Het *travelling salesman problem* (TSP) is een klassiek combinatorisch optimalisatieprobleem. In het probleem probeert men het kortste Hamiltoniaanse pad door een aantal knopen te vinden. Het pad moet elke knoop exact éénmaal bezoeken en van minimale totale lengte zijn. Het TSP is de meest elementaire vorm van alle *vehicle routing problems* en daarom relevant voor deze studie. Het probleem is NP-hard en dus niet berekenbaar in polynomiale tijd.



FIGUUR 2.1: Travelling Salesman Problem

In de loop der jaren hebben veel wetenschappers zowel exacte als heuristische methodes ontwikkeld voor het TSP. Omdat exacte methodes heel veel mogelijke combinaties moeten uitproberen, zijn deze computationeel duur. Exacte algoritmen zijn vaak gebaseerd op *integer linear programming* (ILP) waarbij het *branch-and-bound*-principe het zoekproces in de niet-veelbelovende takken vervroegd stopzet [25]. De andere exacte algoritmen voor het TSP vinden hun oorsprong in *dynamic programming* [13]. Exacte methodes leggen beperkingen op de keuze van de objectieve functie en de representatie van het probleem. De ILP modellen voor TSP's hebben voor elke boog tussen de knopen één beslissingsvariabele. De objectieve vergelijking is in dat paradigma lineair in functie van de afstand van de bogen en de beslissingsvariabelen. De restricties op de objectieve vergelijking maken deze exacte algoritmen onbruikbaar in het kader van dit onderzoek. Daarom lijkt een heuristische methode aangewezen.

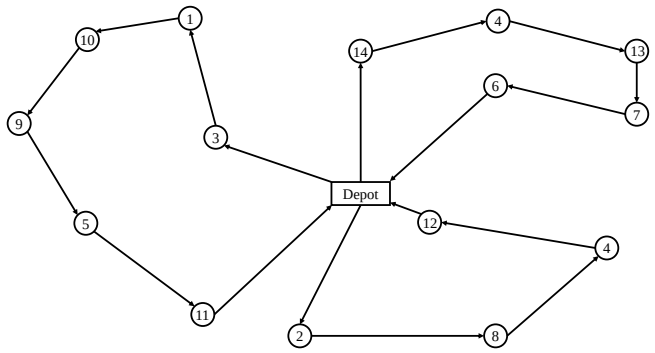
Heuristische algoritmen zijn opgedeeld in drie klassen: constructieve, verbeterende en samengestelde methodes. Constructieve methodes bouwen een pad door telkens één boog aan de oplossing toe te voegen. Het algoritme loopt ten einde wanneer het pad volmaakt is en alle knopen bevat. Meestal kiest men de nieuwe boog op basis van de *nearest neighbour* of *greedy* heuristiek. Verbeterende methodes of *improvement heuristics* starten met een initiële oplossing. Het algoritme verbetert dit pad door in elke iteratie het pad te perturberen (2-opt, 3-opt en Lin-Kernighan) tot een betere oplossing. Voor het TSP zijn de populairste *improvement heuristics tabu search*, *simulated annealing* en *genetische algoritmen*. Samengestelde methodes zijn combinaties van constructieve en verbeterende methodes [19]. Tot slot bestaan er ook nog memetische algoritmen voor het TSP. Dit zijn evolutionaire algoritmes met een lokale zoekprocedure.

Genetische algoritmen zijn zeer flexibel in de keuze van de objectieve functie. Daarnaast zijn genetische algoritmen modulair opgebouwd waarbij de keuze van de operatoren de balans bepaalt tussen *exploreerbaarheid* (de grootte van de zoekruimte) en *exploitativiteit* (het opzoeken van optima in de lokale structuur).

2.2 Capacitated Vehicle Routing Problem

Het *capacitated vehicle routing problem* (CVRP) is een distributieprobleem met één centraal depot en een verzameling klanten die rond het depot verspreid liggen. Dit is een generalisatie van het *travelling salesman problem*. In het probleem hebben de klanten een vraag naar een product waarvan de hoeveelheid tussen de klanten onderling kan verschillen. Het aantal wagens staat voor elk probleem vast en is gegeven. De bestuurders van de wagens moeten een lus maken die vertrekt en eindigt in het depot. De bestuurders mogen het depot niet bezoeken om de producten in hun wagen aan te vullen. In een CVRP heeft elk voertuig dezelfde capaciteit en moet elke klant door exact één bestuurder bezocht worden. De oplossing is een routeplan die elke klant van zijn vraag voorziet en waarbij de capaciteit van de wagens niet overschreden wordt. Het klassieke CVRP minimaliseert een objectieve kost zoals de

totale afstand.



FIGUUR 2.2: Capacitated Vehicle Routing Problem

2.2.1 Mixed Integer Programming Model

Men formuleert het CVRP meestal als een *mixed integer programming* (MIP) probleem. Neem aan dat $G = (V, A)$ een grafe is met $V = (0, 1, 2, ..., N)$ een verzameling knopen en $A = \{(i, j) : i, j \in V, i \neq j\}$ een verzameling bogen tussen deze knopen. Knoop 0 stelt het depot voor en knopen $1...N$ zijn de klanten. De kostenmatrix $\mathbf{C}$ bevat voor elke boog (i, j) een objectieve kost c_{ij}. De kost kan bijvoorbeeld de afstand of de tijd tussen twee knopen beduiden. De kost kan ook een combinatie zijn van meerdere eigenschappen van de boog. Er zijn m voertuigen beschikbaar die allemaal met dezelfde capaciteit Q uitgerust zijn. Elke knoop $i \in V, i > 0$ heeft zijn eigen vraag $Q_i > 0$ naar producten.

Het MIP model geeft aan elke boog (i, j) in G een booleaanse beslissingsvariabele x_{ij}. Wanneer $x_{ij} = 0$, behoort de gerelateerde boog niet tot de oplossing. Wanneer $x_{ij} = 1$, neemt een bestuurder de boog in de oplossing. Aan de hand van deze notatie formuleert men het MIP model voor het CVRP als volgt.

$$Minimaliseer \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.1}$$

$$\sum_{j\in V, j\neq i} x_{ij} = 1 \qquad i = 1, ..., N \tag{2.2}$$

$$\sum_{i\in V, i\neq j} x_{ij} = 1 \qquad j = 1, ..., N \tag{2.3}$$

$$\sum_{j=1}^{n} x_{0j} \leq m \qquad j = 1, ..., N \tag{2.4}$$

$$x_{ij} = 1 \Rightarrow u_i + q_j = u_j \qquad (i, j) \in A : j \neq 0, i \neq 0 \tag{2.5}$$

$$q_i \leq u_i \leq Q \qquad i = 1...N \tag{2.6}$$

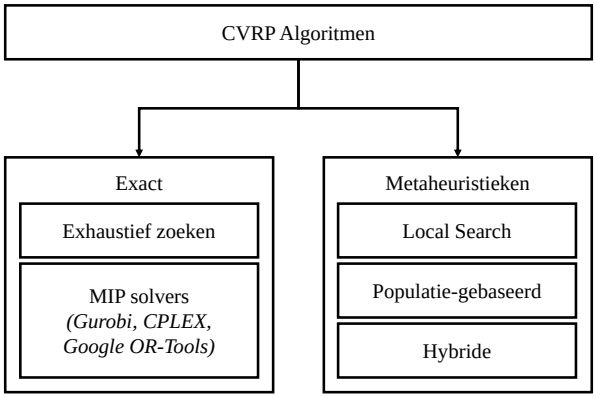
$$x_{ij} \in \{0,1\} \qquad (i,j) \in A \tag{2.7}$$

Constraints 2.2 en 2.3 geven aan dat er exact één voertuig aankomt en vertrekt in elke knoop (behalve het depot). *Constraint* 2.4 geeft aan dat er maximaal m routes uit het depot kunnen vertrekken. In *constraint* 2.5 geeft u_i de cumulatieve lading van de wagen aan in knoop i. De variabelen u_i nemen *integers* als waarde. Aan de ene kant zorgt constraint 2.5 ervoor dat knoop i geen lus introduceert. Aan de andere kant limiteren *constraints* 2.5 en 2.6 samen de cumulatieve lading van de wagens tot de maximale capaciteit. Door *constraint* 2.7 kunnen de beslissingsvariabelen x_{ij} enkel de waarden $True$ en $False$ aannemen. Wanneer de oplossing exact m voertuigen moet gebruiken, verandert het ongelijkheidsteken in 2.4 in een gelijkheidsteken.

2.2.2 Solvers

Sinds de introductie van het *vehicle routing problem* in 1959 in het *"Truck Dispatching Problem"* artikel van *Dantzig en Ramser* [7], werd het VRP één van de meest bestudeerde combinatorische optimalisatieproblemen omwille van het logistieke nut ervan. In de afgelopen decennia werd het klassieke CVRP tot in de fijnste details onderzocht met het idee dat men het CVRP kan generaliseren naar complexere VRP varianten [14]. Voor kleine instanties van het CVRP gebruikt men exacte MIP solvers. Deze solvers proberen veel mogelijke combinaties en gebruiken complexe algoritmes met heuristieken en pruning om het zoekproces te versnellen [18]. De *state-of-the-art solvers* zijn in zowel *open source* als commerciële softwarepakketten (*Gurobi, CPLEX of Google OR-Tools*) beschikbaar.

Door de ontwikkeling van deze snelle methodes onderzocht men sinds 2014 steeds meer realistische VRP varianten onder de verzamelnaam *Rich-VRP* [1]. Deze varianten hebben complexe eigenschappen zoals *time windows*, meerdere depots en proberen rekening te houden met milieuvriendelijkheid en de ervaring van de bestuurders [2]. De objectieven van *Rich-VRP* varianten zijn vaak moeilijk te formuleren in het MIP paradigma. Daarnaast is de computationele kost voor problemen met meer dan honderd klanten vaak te hoog om een optimale oplossing te bekomen. Hierdoor ontwikkelden VRP experts algoritmen die goede, niet-optimale oplossingen kunnen vinden binnen voorspelbare tijd. Deze algoritmen zijn gebaseerd op constructieve en verbeterende heuristieken (*constructive and improvement heuristics*). De accumulatie van al dit onafhankelijk onderzoek leidde in de laatste jaren tot de ontwikkeling van krachtige metaheuristieken die in een paar seconden oplossingen kunnen vinden die minder dan één procent van de optimale waarde afwijken [15]. De meeste metaheuristieken zijn gebaseerd op principes zoals *local search* (*simulated annealing, tabu search, iterated local search*), populatie-gebaseerd (*ant colony optimization, genetische algoritmes*) of hybride methodes (memetische algoritmes).



FIGUUR 2.3: Oplossingsmethodes voor het CVRP

2.3 Conjoint Analysis

Conjoint analysis (CA) vindt zijn oorsprong in het domein van de marketingmodellen. De techniek modelleert het gedrag van individuen bij het maken van keuzes tussen verschillende producten of diensten. CA neemt de assumptie dat elk product een decompositie heeft in onafhankelijke attributen waarvan elk attribuut verschillende waarden kan aannemen. Tabel 2.1 geeft een voorbeeld van de decompositie van een wagen in de attributen merk, type, motor en brandstof. De attribuutwaarden voor het merk zijn *Volkswagen*, *Mercedes* en *BMW*. De meest voorkomende experimenten met *conjoint analysis* bevatten een vragenlijst of *survey* die meerdere testpersonen moeten invullen. In deze vragen moeten de testpersonen een keuze maken tussen meerdere profielen. Een profiel is een combinatie van attributen waar een waarde is aan toegekend. In andere varianten van *conjoint analysis* moeten de testpersonen een verzameling profielen ordenen naar persoonlijke voorkeur of moeten ze profielen een score geven op honderd. Daarnaast kunnen onderzoekers ook direct aan testpersonen vragen welke attribuutwaarden ze het belangrijkst vinden. [9]

Na het uitvoeren van het experiment kunnen statistisch onderbouwde modellen het relatieve belang van de attributen schatten en de meest geprefereerde attribuutwaarden bepalen. Met *conjoint analysis* kunnen marketeers aan marktsegmentatie doen of kunnen productdesigners te weten komen welke eigenschappen van een product in ontwikkeling individuen belangrijk vinden.

Selecteer een voertuig			
Merk	Volkswagen	Mercedes	BMW
Type	MPV	Sedan	Coupé
Motor	V6, 3.6L	Elektrisch	V8, 3.6L
Brandstof	Diesel	Geen	Benzine
Geselecteerd	Nee	Nee	Ja

TABEL 2.1: Voorbeeld van choice-based conjoint analysis

2.3.1 Choice-based Conjoint

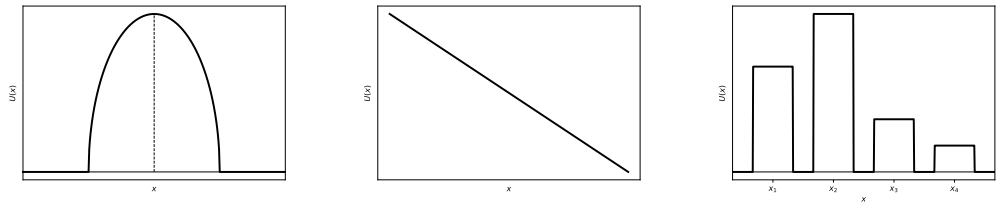
Conjoint anaysis is een verzamelnaam voor technieken die het nut (*utility*) van attribuutwaarden meten. De populairste variant onder marketeers is *choice-based conjoint analyis* (CBC). In dit paradigma krijgen alle testpersonen meerdere keren een drietal producten te zien waaruit ze het beste profiel moeten kiezen. Meer dan 80 % van de van de experimenten met *conjoint analysis* zijn CBC. Daardoor is de focus van de meeste researchers op deze variant gericht. Wegens de toegankelijkheid van CBC in de literatuur en de populariteit van de techniek, ligt de focus in dit werk op deze variant.

Een conjoint experiment heeft altijd meerdere attributen en niveaus waardoor het aantal combinaties snel kan exploderen. Voor vijf attributen met elk drie niveaus zijn er bijvoorbeeld $3 \times 3 \times 3 \times 3 \times 3 = 243$ verschillende profielen. Voor een testpersoon is het onbegonnen werk om al deze opties te overwegen en vervolgens het beste profiel te kiezen. Om dit probleem te voorkomen, bestaan er experimentele designs die het aantal verschillende profielen drastisch verminderen [9]. In dit werk zal een computer de keuze maken waardoor een reductie van het aantal profielen niet nodig is.

2.3.2 Attributen en nut

In *conjoint analysis* splitst men de attributen op in twee klassen: categorisch en kwantitatief. Een voorbeeld van een categorisch attribuut is een automerk. Een voorbeeld van een kwantitatief attribuut is de maximale snelheid van de wagen. Kwantitatieve attribuutwaarden kunnen categorisch gemaakt worden door het domein te discretiseren in intervallen. [22]

Er bestaan verschillende modellen (*utility models*) die een maat van nut relateren aan alle attribuutwaarden. Sommige modellen nemen zowel continue als discrete waarden aan, terwijl andere modellen enkel categorische waarden aannemen. Het *ideaal punt* model in figuur 2.4a veronderstelt dat er een optimale waarde bestaat voor elk attribuut en dat het nut op dat punt bijgevolg het hoogst is. In het *ideaal punt* model is er een negatieve correlatie tussen de afstand van de optimale waarde en het nut. Het vector model uit figuur 2.4b modelleert een lineair verband tussen de attribuutwaarde en het nut. Het *part-worth* model in figuur 2.4c gebruikt enkel discrete of categorische attributen en maakt a priori geen assumpties over de interacties tussen verschillende niveaus. Deze manier is aangewezen wanneer er geen ordening is tussen de waarden van het attribuut. Voor attributen met kwantitatieve waarden kunnen andere niet-lineaire nutsvergelijkingen (*utility functions*) opgesteld worden. De exacte vorm van deze functie hangt sterk af van het onderzoeksdomein en het doel van het experiment. Het onderzoek in dit werk analyseert de keuzes van de planner eerst aan de hand van het *part-worth* model. Daarna stelt deze thesis een nieuwe nutsvergelijking voor die speciaal ontworpen is voor het analyseren van CVRP oplossingen. Elk model heeft een eigen wiskundige formulering met parameters. De keuze van het model bepaalt de techniek voor het schatten van de parameters. [22]



(A) Ideaal punt (B) Vector (C) Part-Worth

FIGUUR 2.4: Nutsvergelijkingen

2.3.3 Het Additieve Model

Conjoint analysis neemt de kernassumptie dat alle attributen onafhankelijk zijn van elkaar. *Rao* spreekt in [22] van een additief conjoint model $Utility(X)$ waarin men het nut van een profiel berekent als de de som van de individuele nutsscores U_i van de attribuutwaarden x_i in het profiel.

$$Utility(X) = U_1(x_1) + U_2(x_2) + ... + U_r(x_r) + \epsilon \tag{2.8}$$

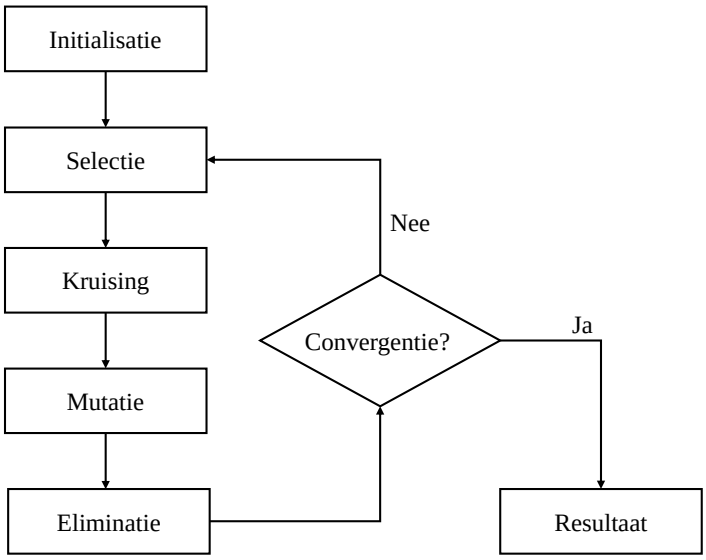
In deze vergelijking zijn U_i de *utility* modellen en x_i de attribuutwaarden van het profiel. Merk op dat de keuze van de *utility* modellen van de attributen onderling kan verschillen. Welke waarden het additieve nut exact kan aannemen, is ook vrij te kiezen. De ontwerper van het model kan enerzijds de uitvoerwaarden beperken tot een vast interval en anderzijds alle reële waarden accepteren als score. De uitvoer van het model hangt bijgevolg af van de toepassing.

2.4 Genetische Algoritmen

Een genetisch algoritme (GA) categoriseert men als een populatie-gebaseerd, metaheuristisch algoritme. Populatie-gebaseerde metaheuristieken gebruiken meerdere kandidaat-oplossingen in het zoekproces. De methode zorgt voor diversiteit binnen de populatie en voorkomt dat de oplossingen in een lokaal optimum blijven vastzitten. Genetische algoritmen zijn geïnspireerd op het Darwiniaanse evolutionair proces en het *survival-of-the-fittest* principe. [20]

De bouwstenen van een GA zijn de voorstelling van de individuen, initialisatie, selectie, kruising, mutatie en eliminatie. Daarnaast is er ook een altijd een convergentiecriterium en een objectieve vergelijking die aangeeft hoe optimaal een individu is. Een GA start met de initialisatie de eerste generatie. Het initialisatieproces genereert de individuen op willekeurige wijze of met behulp van constructieve heuristieken. Daarna maakt een GA meerdere iteraties waarin een nieuwe generatie de oude populatie telkens vervangt. De nieuwe generatie is het product van het sequentieel toepassen van selectie-, kruising-, mutatie- en eliminatie-operatoren. Figuur 2.5 geeft het proces grafisch weer. De selectie-operator selecteert een aantal paren (ouders) uit de populatie. De kruising-operator transformeert elk geselecteerd

paar tot een nieuw individu (nageslacht) met de goede eigenschappen van beide ouders. De mutatie-operator introduceert perturbaties in het nageslacht en zorgt voor diversiteit binnen de populatie. Eliminatie zorgt ervoor dat het aantal individuen in de populatie na elke iteratie gelijk blijft. De eliminatie-operator beslist welke individuen uit de verzameling van de vorige generatie en de nakomelingen samen het eliminatieproces overleven. Dit gebeurt op basis van de objectieve waarde van de individuen. Genetische algoritmen bieden veel keuze in de representatie van individuen, de operatoren en de kruisings- en mutatiekansen. [6]



Figuur 2.5: Framework voor genetische algoritmen

Hoofdstuk 3

Gerelateerd Werk

Dit hoofdstuk haalt een belangrijk onderzoek aan die de basis legt voor het impliciet leren van voorkeuren uit historische plannen. De auteurs stelden het artikel voor in *CP 2019, the 25th International Conference on Principles and Practice of Constraint Programming* en draagt de titel *"Vehicle routing by learning from historical solutions"* [4].

3.1 Transitieprobabiliteiten

De auteurs stellen voor om de voorkeuren van de planner te vatten in een probabilistische transitiematrix. Ze gebruiken vervolgens deze matrix in plaats van de objectieve kostenmatrix om nieuwe oplossingen te vinden. Zo slagen de auteurs erin de voorkeuren te leren en deze te incorporeren in eender welke CVRP solver. [4]

De methode is gebaseerd op een *hidden Markov model* (HMM). Op basis van historische trajecten berekent het algoritme een transitie probabiliteitsmatrix $\mathbf{T} = [t_{ij}]$ waarin $t_{ij} = Pr(j \mid i)$ de conditionele kans voorstelt dat de volgende knoop j is wanneer de bestuurder zich in knoop i bevindt. Het doel van het voorgestelde model is om een oplossing X te vinden die de gezamenlijke probabiliteit van de gekozen bogen maximaliseert zoals in vergelijking 3.1. In de tweede stap is het product maximaliseren equivalent aan het maximaliseren van de som van de logaritmen. Analoog aan het MIP model van het CVRP in 2.2 introduceren we in de laatste uitdrukking de beslissingsvariabelen x_{ij}. We transformeren dit maximalisatieprobleem tot een minimalisatieprobleem door het teken van het logaritme om te draaien.

$$\begin{aligned} X^* &= \arg\max_X \prod_{(i,j)\in X} Pr(j \mid i) \\ &= \arg\max_X \sum_{(i,j)\in X} log(t_{ij}) \\ &= \arg\min_X \sum_{(i,j)\in A} -log(t_{ij}) x_{ij} \end{aligned} \tag{3.1}$$

Deze objectieve vergelijking is volledig analoog aan de standaard vergelijking van het CVRP in 2.1. Hierin vervangt het negatief logaritme van de conditionele transitieprobabiliteiten de afstand in de objectieve kostenmatrix. Door de matrix $\mathbf{C} = [c_{ij}]$ op te stellen waarin $c_{ij} = -log(t_{ij})$ en deze te gebruiken in eender welke CVRP *solver*, bekomt men de meest waarschijnlijke oplossing. [4]

De productregel voor de conditionele probabiliteit leidt tot uitdrukking 3.2. Deze vergelijking geeft de kans weer dat stop j de directe opvolger is van stop i in de oplossing. Men bekomt de tweede stap door de noemer te marginaliseren over alle stops die bereikbaar zijn vanuit i. Bij de uitwerking gebruikt men de Markov eigenschap die zegt dat de keuze van de volgende stop alleen afhangt van de huidige stop. [4]

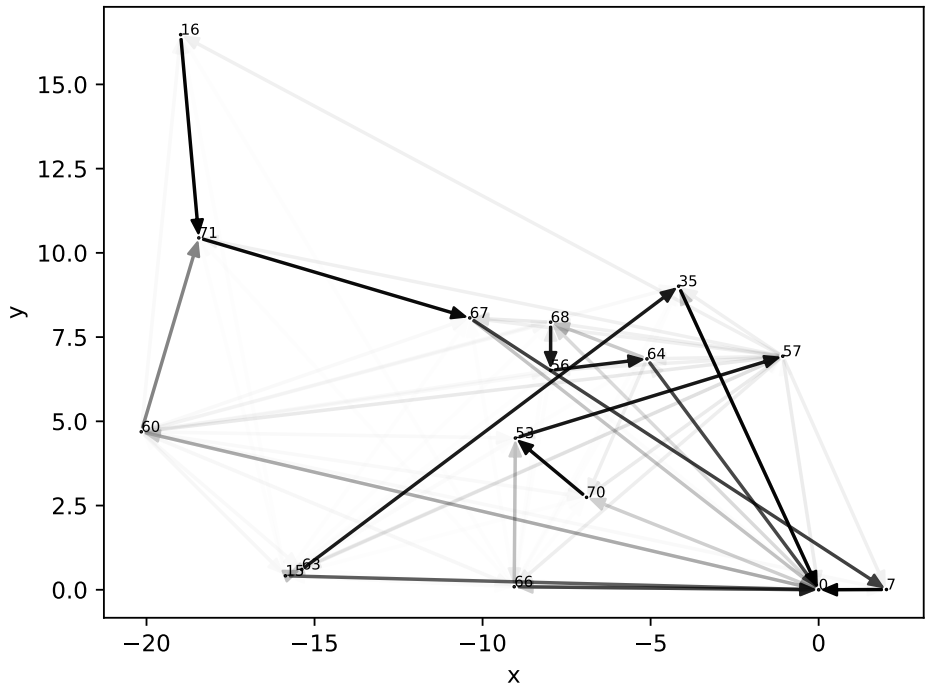
$$\begin{aligned} Pr(j \mid i) &= \frac{Pr(j,i)}{Pr(i)} \\ &= \frac{Pr(j,i)}{\sum_k Pr(k,i)} \\ &= \frac{f_{ij}}{\sum_k f_{ik}} \end{aligned} \tag{3.2}$$

Intuïtief stelt de teller f_{ij} de frequentie van de boog (i,j) in alle historische oplossingen voor en de noemer $\sum_k f_{ik}$ het aantal keer dat knoop i voorkomt in alle oplossingen. Bijgevolg moet het algoritme voor het opstellen van de transitie probabiliteitsmatrix vooral bogen en knopen in de oplossingen tellen. De auteurs voorkomen het probleem dat probabiliteiten nul kunnen zijn met *Laplace smoothing*. Daarnaast stellen de auteurs enkele mogelijke gewichten voor om oude oplossingen de uitkomst minder te beïnvloeden dan recente trajecten. Tot slot stellen ze nog een matrix voor die zowel de afstand als de transitieprobabiliteit in achting neemt. Figuur 3.1 geeft de transitieprobabiliteiten tussen enkele knopen in een historische dataset. De transparantie van de bogen geeft de sterkte van de probabiliteit aan. [4]

Het succes van het onderzoek leidde in 2021 tot een soortgelijke studie met een Markov model van de tweede orde [2]. De objectieve vergelijking die het artikel opbouwt is daardoor kwadratisch waardoor het veel langer duurt om een oplossing te vinden. De auteurs van het tweede paper concluderen dat de kleine verbetering in accuraatheid niet significant genoeg is om de verveelvoudiging van de computationele kost te accepteren.

In artikel [17] nemen dezelfde auteurs een andere wending met een neuraal net. Hier beredeneren ze dat elk probleem een context heeft die bestaat uit de weekdag, het aantal bestuurders, de capaciteit, de klanten en de vraag. Omdat de voorgenoemde methodes geen rekening houden met de context van het probleem beredeneren de auteurs dat het een goed idee is om de probabiliteiten afhankelijk te maken van de context. Omdat context moeilijk te formaliseren is, verkiezen de auteurs een *black-box* aanpak met neurale netten. Dit model geeft zeer goede resultaten en het lijkt dat er op dit moment geen directe verbeteringen in aanmerking komen in de

richting van dit soort modellen. Daarom neemt dit werk een stap terug van dit idee om na te denken of er geen andere technieken bestaan die het toe laten de voorkeuren impliciet te leren en toe te passen. Het recentste werk van dezelfde auteurs leert de voorkeuren van de planner ook impliciet met transitieprobabiliteiten in een TSP setting. [3]



FIGUUR 3.1: Voorbeeld van transitieprobabiliteiten

Hoofdstuk 4

Probleemstelling

Het *capacitated vehicle routing problem* is gelimiteerd door het aantal voertuigen en de capaciteit van deze voertuigen. Dit probleem is relatief eenvoudig op te lossen met een *mixed integer programming* model wanneer we een objectieve kost, zoals afstand, willen minimaliseren. Dit is de manier waarop de meeste planners te werk gaan wanneer er geen oude routeplannen ter beschikking zijn.

Dit werk is gebaseerd op een dataset van een KMO met historische routeplannen van 201 verschillende dagen. Uit analyse van de data blijkt dat de beslissing van de planner niet alleen gebaseerd is op het minimaliseren van de totale afstand, maar dat ook enkele persoonlijke voorkeuren ook een rol spelen. Bij gelijkaardige problemen werden oude plannen hergebruikt en aangepast om nieuwe oplossingen te vinden. We geven enkele observaties:

- Sommige knopen komen frequent samen voor in hetzelfde pad. Dit wijst erop dat de bestuurder een voorkeur heeft om deze knopen samen te bezoeken op een dag. De onderliggende reden is bijvoorbeeld familiariteit van de bestuurder met de regio waarin deze knopen zich bevinden.
- Sommige paden bestaan slechts uit één of twee knopen. Bij deze knopen kan de tijd van het bezoek langer zijn en is daarom het aantal stops in het pad klein. Een ander idee dat dit fenomeen kan verklaren is dat deze knopen ver verwijderd zijn van het depot en dat de bestuurder grote afstanden moet afleggen om beide knopen te bezoeken.
- Knopen kunnen een geprefereerde positie hebben in het pad. De positie van een knoop in een pad correleert met het tijdstip waarop men de klant moet bezoeken. Wanneer een knoop bijvoorbeeld vaak vroeg in een pad voorkomt, verkiest de klant een bezoek in de voormiddag.

Deze observaties tonen aan dat de planner vaak meerdere doelstellingen heeft. Bij het opstellen van het finale routeplan, kunnen deze objectieven niet allemaal tegelijk voldaan zijn. Het is dan aan de planner zelf om een beslissing te maken. Het is moeilijk en tijdrovend om deze voorkeuren te formeel neer te pennen in een multi-objectief

optimalisatiemodel. En als er een zo model zou bestaan, is het bijna onmogelijk om een beslissingsondersteunend systeem te ontwikkelen die net zoals de planner een oplossing kan kiezen uit het Pareto-front.

Een andere manier om een oplossing te bekomen met historische data, is het automatisch leren van de voorkeuren zonder deze voorkeuren te specifiëren. Op deze manier bekomen we een systeem dat door elke planner geaccepteerd kan worden, ongeacht de aard en omvang van de voorkeuren.

Het doel van dit onderzoek is om met *conjoint analysis* de voorkeuren van planners en bestuurders impliciet te leren uit oplossingen van *vehicle routing problems*. Dit werk ontwikkelt ten eerste een model die de voorkeuren kan leren. Ten tweede bouwt dit werk een systeem op die plannen kan genereren op basis van de subjectieve voorkeuren. We maken de assumptie dat er vergelijkbare historische routeplannen ter beschikking zijn en leren keuzes te maken zoals de planner.

Hoofdstuk 5

Aanpak

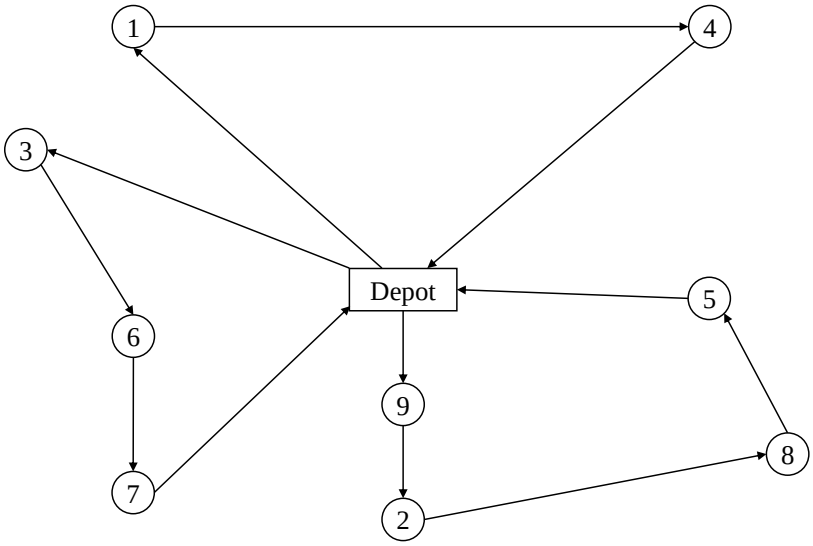
Het idee om de voorkeuren van planners en bestuurders te leren zonder deze te specificeren aan de hand van *conjoint analysis* fungeert als de rode draad doorheen dit werk. Het gerelateerde werk in deel 3.1 gebruikt de transitieprobabiliteit als de beslissende factor bij het opstellen van nieuwe plannen. De auteurs hebben deze eigenschap van historische oplossingen onderzocht tot op het punt dat er geen nieuwe methodes lijken te bestaan die de accuraatheid van het model kunnen verbeteren. Deze technische barrière drijft ons ertoe enkele stappen terug te nemen en opnieuw na te denken over andere interessante eigenschappen van historische oplossingen. Omdat *conjoint analysis* de *trade-offs* onderzoekt tussen meerdere attributen, moet het voorgestelde model de historische oplossing ontbinden in enkele verschillende eigenschappen.

Dit werk beschrijft een systeem dat uit twee componenten bestaat: een preferentiemodel en een *solver*. Het preferentiemodel leert aan de hand van *conjoint analysis* (2.3) de voorkeuren van de planners. De solver heeft de taak om de geleerde voorkeuren te incorporeren in een aangepast CVRP optimalisatieprobleem en vervolgens de beste oplossing te vinden. Om dit mogelijk te maken definieert het eerste deel 5.1 van dit hoofdstuk systematisch drie abstractieniveaus. Het hoogste niveau analyseert eigenschappen van oplossingen in hun geheel. Het laagste niveau bekijkt eigenschappen van individuele knopen. Het abstractieniveau bepaalt de mate van detail waarin het systeem werkzaam is. Elk niveau is begrensd in de attributenkeuze van het preferentiemodel. Daarnaast geeft deel 5.1 voor elk niveau een objectieve vergelijking voor het aangepaste CVRP optimalisatieprobleem.

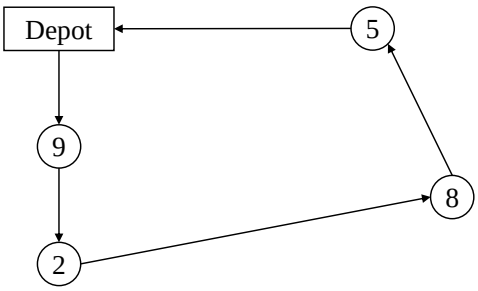
Deel 5.2 introduceert een twee-fasen aanpak voor het oplossen van het *capacitated vehicle routing problem*. Door de complexiteit van de voorgestelde objectieve vergelijking en de incompatibiliteit met bestaande algoritmen, splitsen we het probleem op in *clustering* en *routing*.

5.1 Abstractieniveaus

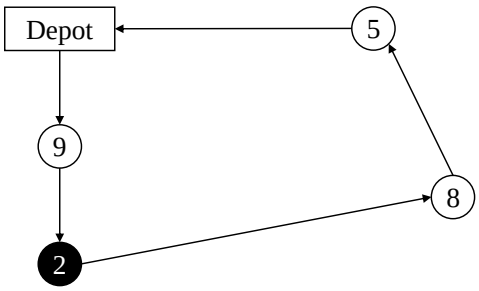
In de analyse van historische oplossingen kan men drie abstractieniveaus onderscheiden. Het hoogste niveau legt de focus op het volledige routeplan. Het middelmatige niveau analyseert van paden. Het laagste en het gedetailleerdste niveau onderzoekt de eigenschappen van knopen in een pad. Figuur 5.1 geeft grafisch weer wat deze abstractieniveaus exact voorstellen en welk deel van de oplossing de verschillende niveaus analyseren. Tabel 5.1 geeft voor elk niveau enkele mogelijke attributen. Het vervolg van dit deel bespreekt de abstractieniveaus in verder detail en stelt de objectieve vergelijking voor de solver van elk niveau op.



(a) Oplossing - *high level*



(b) Pad - *medium level*



(c) Knoop - *low level*

Figuur 5.1: Grafische voorstelling van de abstractieniveaus

Hoog niveau	
0 → 9 → 2 → 8 → 5 → 0 **0 → 1 → 4 → 0** **0 → 3 → 6 → 7 → 0**	
Totale afstand	24 km
Totale tijd	8 uur
Gemiddeld aantal knopen per pad	5

Middelmatig niveau	
0 → 9 → 2 → 8 → 5 → 0	
Afstand	8 km
Tijd	3 uur
Aantal knopen	6

Laag niveau	
Knoop **2** in **0 → 9 → 2 → 8 → 5 → 0**	
Afstand voor bezoek (0 → 9 → 2)	2 km
Afstand na bezoek (2 → 8 → 5 → 0)	6 km
Tijd voor bezoek (0 → 9 → 2)	1 uur
Tijd na bezoek (2 → 8 → 5 → 0)	2 uur
Positie in pad	2

TABEL 5.1: Attributen van de abstractieniveaus

5.1.1 Eerste niveau: de oplossing

Het model op hoog abstractieniveau focust op eigenschappen van de oplossing in zijn geheel zoals de totale lengte en het gemiddeld aantal knopen per pad. Daarom kan dit model alleen informatie verzamelen die geaggregeerd is over alle paden. Dit zijn doorgaans statistieken zoals een som, het gemiddelde of de variantie van eigenschappen van paden. Vergelijking 5.1 ontbindt een oplossing X in *high level* attributen A en berekent een *utility score* aan de hand van het additieve model van *conjoint analysis* (2.3.3). Het preferentiemodel voor het bepalen van het nut van eender welke oplossing bestaat dus uit één *conjoint model* M_{high}. In de volgende vergelijkingen heeft U_A twee effecten. Eerst berekent de functie de attribuutwaarde van attribuut A gegeven het argument. Vervolgens geeft de functie de nutsscore van de berekende waarde terug. Dit vereenvoudigt de notatie.

$$\begin{aligned} Utility(X) &= M_{high}(X) \\ &= \sum_{A \in M} U_A(X) \end{aligned} \tag{5.1}$$

Vergelijking 5.2 geeft de objectieve vergelijking voor het aangepaste *capacitated vehicle routing problem* weer. Deze vergelijking maximaliseert van het nut van de *high level* attribuutwaarden van de oplossing. Helaas is dit framework niet krachtig genoeg om de voorkeuren te leren. Hierdoor komen we terecht in het tweede abstractieniveau.

$$X^* = \arg\max_X \Big(M_{high}(X) \Big) \tag{5.2}$$

5.1.2 Tweede niveau: het pad

Het tweede abstractieniveau bekijkt de individuele paden van een oplossing. Het model ontbindt een pad aan de hand van *conjoint analysis* in *medium level* attributen zoals afstand, tijdsduur, aantal knopen en het geografisch centrum van het pad. Dit model verwerft dus verfijndere informatie over de oplossingen dan het model van hoge abstractie. Een oplossing bestaat uit meerdere paden waardoor het *medium level* model de individuele *utility scores* van de paden moet aggregeren. Om dit te kunnen doen, is het noodzakelijk om assumpties te maken over de uitvoer van het *conjoint* model die de paden evalueert. Stel dat de uitvoer een probabiliteit beduidt die waarden tussen nul en één aanneemt, dan kan men een algemene nutsscore voor probleem X bekomen door het product te nemen van de individuele *utility scores* van de paden P. De onderliggende nutsvergelijkingen van het additieve model en de gewichten van de attributen moeten deze eigenschap garanderen. Vergelijking 5.3 werkt de toewijzing van een score aan een oplossing aan de hand van het *medium* abstractieniveau. Het is belangrijk aan te kaarten dat er hier nog steeds één *conjoint model* is zoals in het eerste abstractieniveau. Met dit ene model krijgen alle paden een nutsscore.

$$\begin{aligned} Utility(X) &= \prod_{P \in X} M_{mid}(X, P) \qquad\qquad M_{mid} \in [0, 1] \\ &= \prod_{P \in X} \sum_{A \in M_{mid}} U_A(X, P) \end{aligned} \tag{5.3}$$

Vergelijking 5.4 geeft de objectieve vergelijking voor het CVRP weer aangepast aan het tweede abstractieniveau. Omdat de eerste vorm een product neemt van probabiliteiten kan de score zeer klein worden. Om de methode numeriek stabiel te maken nemen we het negatieve logaritme van de uitvoer van het conjoint model waardoor we het probleem omvormen tot een minimalisatieprobleem. Dit abstractieniveau voldoet

ook niet aan de verwachtingen van het onderzoek. Observeer dat een historisch pad in de omgekeerde richting dezelfde attribuutwaarden krijgt als het origineel pad en daarom dezelfde score krijgt. Dit zijn ongewenste effecten en om deze reden verdiept het onderzoek zich in het abstractieniveau van de knoop.

$$\begin{aligned} X^* &= \arg\max_X \left(\prod_{P \in X} M_{mid}(X, P) \right) \\ &= \arg\min_X \left(\sum_{P \in X} -log\left(M_{mid}(X, P)\right) \right) \end{aligned} \tag{5.4}$$

5.1.3 Derde niveau: de knoop

Aangezien de attributen van de hoge en middelmatige abstractieniveaus beide een te lage resolutie hebben voor het leren van voorkeuren van planners en bestuurders, is *conjoint analysis* op het laagste abstractieniveau aangewezen. In het *low level* model spelen er meerdere *conjoint modellen* een rol bij het bepalen van het nut van een oplossing. Vergelijking 5.5 geeft aan hoe dit model de scores aggregeert over de knopen k van alle paden$P$ in de oplossing X. In dit paradigma heeft elke knoop k zijn eigen *conjoint model* $M_{k,low}$. Omdat deze vergelijking het product neemt over de uitkomst van meerdere modellen, limiteren we het bereik van het de uitvoer van het model opnieuw tussen nul en één. Hierdoor kunnen alle knopen individueel informatie verzamelen over de preferenties van de planner. We kunnen het feit dan een knoop een pad in twee delen opsplitst, exploiteren bij het opstellen van nieuwe attributen. Daarnaast is het ook mogelijk om eigenschappen te definiëren in de directe omgeving van een knoop. Voorbeelden van individuele attributen zijn de afstand voor en na het bezoek van de knoop, de richting waaruit de bestuurders komen en de positie in het pad. Het *low level* model biedt daarnaast het voordeel dat de informatie van de hogere abstractieniveaus impliciet vervat is in de combinatie van de individuele *conjoint modellen* van de knopen. Omgekeerd is dit zeker niet het geval. Dit abstractieniveau biedt de nodige resolutie voor een preferentiemodel. We noemen het model dat gelinkt is met dit abstractieniveau vanaf voortaan het *knopenmodel*.

$$\begin{aligned} Utility(X) &= \prod_{P \in X} \prod_{k \in P} M_{low,k}(X, P, k) \qquad\qquad M_{k,low} \in [0, 1] \qquad \forall k \\ &= \prod_{P \in X} \prod_{k \in P} \sum_{A \in M_k} U_{A,k}(X, P, k) \end{aligned} \tag{5.5}$$

Vergelijking 5.2 geeft de objectieve vergelijking voor het CVRP weer voor het knopenmodel. Het negatieve logaritme komt opnieuw in de vergelijking voor omwille van numerieke stabiliteit bij de evaluatie van nieuwe paden. We sluiten dit deel af

met de volledige uitwerking in functie van het additieve model van *conjoint analysis*. Noteer dat men het knopenmodel op elke VRP variant kan toepassen.

$$
\begin{aligned}
X^* &= \arg\max_X \left(\prod_{P \in X} \prod_{k \in P} M_{low,k}(X, P, k) \right) \\
&= \arg\min_X \left(\sum_{P \in X} \sum_{k \in P} -log\left(M_{low,k}(X, P, k)\right) \right) \\
&= \arg\min_X \left(\sum_{P \in X} \sum_{k \in P} -log\left(\sum_{A \in M_k} U_{A,k}(X, P, k) \right) \right)
\end{aligned}
\tag{5.6}
$$

5.2 Twee Fasen

De *state of the art mixed integer programming solvers* leggen een beperking op de objectieve vergelijking van het model. Softwarepakketten zoals *Gurobi* bieden enkel de mogelijkheid om modellen op te lossen met een lineair (MILP) of kwadratisch (MIQP) objectief. Vergelijking 5.6 geeft de objectieve vergelijking voor het evalueren van voorkeuren in functie van van het knopenmodel. Een computer kan deze vergelijking gemakkelijk evalueren. Maar om complexe interacties tussen de knopen te onderzoeken en vrijheid in de keuze van de attributen te garanderen, moeten we accepteren dat er geen *off-the-shelf solver* voorhanden is en bijgevolg een nieuwe *solver* ontwerpen.

Sectie 2.2.2 bespreekt enkele alternatieve oplossingsstrategieën voor het CVRP uit de literatuur. Omdat het optimalisatieprobleem van het knopenmodel volledig verschillend is aan het minimaliseren van een objectieve kost zoals de afstand, zijn de meeste recent ontworpen heuristische *solvers* incompatibel met het preferentiemodel. Deze algoritmes gebruiken constructieve en verbeteringsheuristieken die speciaal ontwikkeld zijn voor eenvoudige objectieve functies. Naïeve enumeratie van alle mogelijke oplossingen is absoluut geen optie wanneer het probleem meer dan tien knopen bevat. Constructieve algoritmes zijn onmogelijk te implementeren wanneer de objectieve functie afhankelijk is van complexe interacties tussen knopen. Lokale zoekprocedures voor het CVRP zijn al zeer complex op zich en het introduceren van het knopenmodel binnen dit soort procedures lijkt bijgevolg onbegonnen werk.

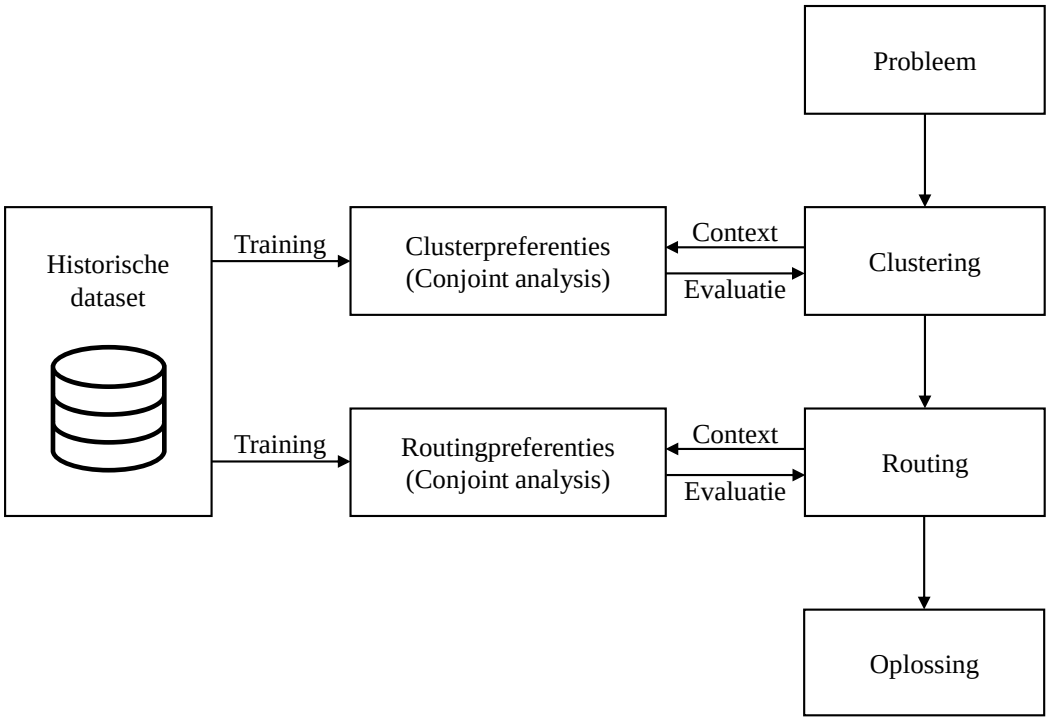
Om deze redenen nemen we de beslissing om het CVRP op te splitsen in twee deelproblemen: clustering en routing

Het eerste deelprobleem is *clustering*. In deze stap worden de knopen van het probleem over de bestuurders verdeeld. Het clusteralgoritme moet ervoor zorgen dat

knopen die frequent samen voorkomen in dezelfde cluster belanden en er tegelijk voor zorgen dat de knopen uit dezelfde cluster niet te ver uit elkaar liggen. Daarnaast moet het algoritme garanderen dat de capaciteit van de wagens niet overschreden wordt. Hoofdstuk 7 geeft een gedetailleerde aanpak van dit probleem.

Het tweede deelprobleem is *routing*. Het routingalgoritme vindt het meest waarschijnlijke Hamiltoniaanse pad door de knopen van een gegeven cluster. Deze stap is identiek aan het *travelling salesman problem* (zie 2.1) maar gebruikt de objectieve vergelijking van het knopenmodel 5.6 als optimalisatiecriterium. Met behulp van de geleerde voorkeuren moet het algoritme ervoor zorgen dat alle knopen de meest geprefereerde positie in het pad krijgen en dat het pad verwant is aan de historische trajecten. Hoofdstuk 8 bespreekt alle elementen van het clusterprobleem.

Figuur 5.2 geeft een visualisatie van het voorgestelde framework. In het framework heeft elk deelprobleem zijn eigen preferentiemodel. Beide modellen leren de voorkeuren van de planner uit historische data op basis van *conjoint analysis* en evalueren de oplossing aan de hand van het knopenmodel. Clustering legt een focus op attributen zoals de ingesloten oppervlakte, het geografisch centrum en de breedte van clusters. Het preferentiemodel voor clusters moet aan elke knoop dus een goed beeld geven over de geometrische clustereigenschappen waarin de knoop zich meestal bevindt. Belangrijke attributen voor het routingprobleem zijn de positie van de knoop in het pad en de richting van het inkomende en uitgaande pad. Met dit soort eigenschappen kunnen knopen leren hoe de historische paden opgebouwd zijn. Hoofdstuk 6 behandelt het leren van voorkeuren in het kader van *conjoint analysis* en de opbouw van het achterliggende preferentiemodel voor beide deelproblemen.



Figuur 5.2: Framework

Hoofdstuk 6

Voorkeuren

Dit hoofdstuk behandelt de werkwijze waarop een knoop de voorkeuren van de planner en de bestuurders leert aan de hand van conjoint analysis. Deel 6.1 bespreekt de methode waarop negatieve voorbeelden gegenereerd worden uit historische trajecten. Deel 6.2 geeft de interne details van twee verschillende *conjoint models* om de geprefereerde attribuutwaarden te modelleren. Deel 6.3 incorporeert het meestbelovende *conjoint model* in de objectieve vergelijking van het knopenmodel. Dit resulteert in het *preferentiemodel* met een online trainingsalgoritme en een eenvoudig evaluatie-algoritme.

6.1 Datageneratie

Om een beslissingsondersteunend systeem te bouwen zijn er meestal positieve en negatieve voorbeelden nodig. De dataset bevat echter enkel historische trajecten. Het systeem moet deze goede oplossingen gebruiken een template voor nieuwe oplossingen. Het struikelblok die zich op dit punt vormt, is de afwezigheid negatieve voorbeelden. We definiëren drie oplossingen die dit probleem verhelpen: *thresholding*, *similarity scoring* en negatieve datageneratie.

- *Thresholding* is een eenvoudige methode die een interval van vaste grootte plaatst rond de attribuutwaarden van de historische oplossing. Wanneer de attribuutwaarden van een nieuw traject binnen de intervallen liggen, krijgt het traject een goede score. Deze aanpak wordt niet verder onderzocht.

- Bij *similarity scoring* is de score van een traject omgekeerd evenredig met de afstand tussen de attribuutwaarden van de historische oplossingen. Deze techniek is dus gelijkaardig aan *thresholding* en wordt niet verder onderzocht.

- Negatieve datageneratie perturbeert de trajecten uit de dataset tot negatieve voorbeelden. Elk negatief voorbeeld leidt vervolgens tot een datarecord dat bestaat uit een aantal attribuutwaarden en het *label False*. De attribuutwaarden van de originele trajecten worden voorzien van het *label True*. Deze aanpak heeft het doel om een binair classificatiemodel het onderscheid aan te leren

tussen goede en slechte attribuutwaarden en daar een probabiliteit aan te koppelen. Daarnaast brengt deze methode domeinspecifieke informatie over het CVRP in het onderliggend classificatiemodel.

Het onderzoek in deze thesis is enkel gebaseerd op negatieve datageneratie. Algoritme A.1 beschrijft gedetailleerd hoe men een gelabelde dataset voor een bepaalde knoop kan genereren aan de hand van een aantal historische trajecten.

In principe kan eender welke traditionele machine learning techniek het binaire classificatieprobleem uitvoeren. Deze methodes zijn bijvoorbeeld beslissingsbomen, logistische regressie, het dichtste-buren algoritme en neurale netten. De techniek in dit werk is volledig gebaseerd op conjoint analysis waarin het additieve model (2.3.3) veronderstelt dat de attributen (vrije variabelen in het classificatieprobleem) onafhankelijk zijn van elkaar. De voorgenoemde machine learning technieken maken deze assumptie niet en zijn daarom niet geschikt in het kader van dit onderzoek.

6.2 Conjoint Analysis

Het knopenmodel (5.1) stelt voor elke knoop één *conjoint model* op. Herinner dat vergelijking 5.5 de uitvoer van de modellen begrenst tussen nul en één waardoor elk model intuïtief de probabiliteit van een knoop bepaalt in de oplossing. Objectieve vergelijking (5.6) maximaliseert dan de gezamenlijke knopenprobabiliteit van een oplossing. Vergelijking 6.1 geeft nogmaals het additieve model van *conjoint analysis*. Dit model geeft aan dat de *utility* van een profiel de som is van de individuele *utilities* van de attribuutwaarden in het profiel. In functie van de gegenereerde dataset kan men de attribuutwaarden x_i direct invullen. De beslissingsvariabele $Utility(X)$ resulteert in waarde 1 voor positief gelabelde records en 0 voor negatief gelabelde records. De keuze van de nutsvegelijkingen bepaalt de manier waarop de parameters van de additieve vergelijking geschat kunnen geschat worden.

$$Utility(X) = U_1(x_1) + U_2(x_2) + ... + U_r(x_r) + \epsilon \tag{6.1}$$

6.2.1 Logistische regressiemodel

De meeste wetenschappers focussen zich op de variant van conjoint analysis die enkel categorische attributen bevat. Deze variant is bijgevolg het toegankelijkst voor nieuwe onderzoeken en het populairst bij marketeers. Er is een algemene consensus dat conjoint analisten zich bij experimenten moeten beperken tot zeven attributen met elk maximaal vijf verschillende niveaus [10].

Om in dit paradigma de gegenereerde dataset uit 6.1 te gebruiken, moeten de attribuutwaarden categorisch zijn. Daarom discretiseert het model de continue variabelen eerst in vijf niveaus. Hoewel deze stap eenvoudig is, kan men kiezen om het domein op te splitsen in uniforme intervallen of om de grenzen te kiezen zodat elk niveau even veel voorbeelden bevat. Daarna zet het model de niveaus om aan de

hand van dummy-encoding. Met deze codering wordt een attribuut $i, i \in 1..r$ met k_i verschillende niveaus omgezet in $k_i - 1$ dummy-variabelen $X_{ij}, j \in 1..k-1$ waarbij het laatste niveau ter referentie niet gecodeerd is. De dummy-variabelen nemen de waarde 1 aan wanneer het respectievelijke niveau in het datarecord geselecteerd is. De waarde van een dummy-variabele van een niet-geselecteerd niveau is altijd 0. Vergelijking 6.2 geeft de wiskundige uitwerking van de additieve vergelijking met dummy-encoding. Het totale nut is een gewogen som van de dummy-variabelen waarin β_{ij} het nut aangeeft van het niveau j van attribuut i. β_0 is het snijpunt van de vergelijking en bevat de som van de nutsscores van de niveaus die ter referentie werden weggelaten bij het dummy-encoderen. [21]

$$\begin{gathered}\textstyle\sum_{i\in 1..r} U_i(X) = \beta_0 \\ + \\ \beta_{11}X_{11} + \beta_{12}X_{12} + ... + \beta_{1(k_1-1)}X_{1(k_1-1)} \\ + \\ \beta_{21}X_{21} + \beta_{22}X_{22} + ... + \beta_{2(k_2-1)}X_{2(k_2-1)} \\ + \\ ... \\ + \\ \beta_{r1}X_{r1} + \beta_{r2}X_{r2} + ... + \beta_{r(k_r-1)}X_{r(k_r-1)} \\ = \\ \beta_0 + \vec{\beta} \cdot \vec{X}\end{gathered} \tag{6.2}$$

In klassieke experimenten bieden *multivariabele logistische regressie* (LR) en *multinomiale logistische regressie* (MNL) de mogelijkheid om de parameters β te schatten. Vergelijking 6.3 geeft de probabiliteit weer van een combinatie attribuutwaarden wanneer de parameters β geschat zijn met LR. De formule geeft aan dat de uitkomsten enkel tussen 0 en 1 liggen en stelt dus werkelijk een probabiliteit voor. Herinner dat dit een nodige voorwaarde is van het knopenmodel. Dit *conjoint model* kan men dus voor elke knoop trainen en invullen in de objectieve vergelijking van het knopenmodel om nieuwe oplossingen te evalueren. Met MNL bekomt op een gelijkaardige manier een probabiliteit bij het maken van keuzes tussen verschillende profielen. Omdat MNL ingewikkelder is en minder relevant is bij het leren van voorkeuren, bespreekt dit onderzoek het MNL model niet verder. Het is niet toegelaten om *ordinary least squares* (OLS) direct toe te passen op de additieve vergelijking bij het schatten van de parameters β omdat dit zowel positieve als negatieve scores kan geven. [21]

$$M(\vec{X}) = \frac{exp(\beta_0 + \vec{\beta} \cdot \vec{X})}{1 + exp(\beta_0 + \vec{\beta} \cdot \vec{X})} \tag{6.3}$$

Formule 6.4 toont aan hoe men de nutsscores van de weggelaten niveaus berekent. Met formule 6.5 berekent men het relatieve belang α van de attributen. Deze parameter bepaalt hoe doorslaggevend een attribuut is bij het evalueren van oplossingen. Tabel 6.1 geeft ter illustratie een overzichtelijk voorbeeld van de geschatte parameters met

het LR model.

$$\beta_{ik_i} = -\sum_{j \in 1..(k_i-1)} \beta_{ij} \tag{6.4}$$

$$\alpha_i = \frac{max(\beta_{ij}, j = 1..k) - min(\beta_{ij}, j = 1..k)}{\sum_{l=1..r} (max(\beta_{lj}, j = 1..k) - min(\beta_{lj}, l = 1..k))} \tag{6.5}$$

LR Model (Knoop 51)		
Attribuutnaam	**Interval**	**Nut β**
Afstand voor bezoek ($\alpha = 50\%$)	$0-2$ km	-4.60
	$2-8$ km	5.40
	8+ km	-0.8
Afstand na bezoek ($\alpha = 30\%$)	$0-4$ km	-2.2
	$4-6$ km	-1.6
	6+ km	3.8
Positie in pad ($\alpha = 20\%$)	$0-1$	-1.8
	$2-3$	2.2
	3+	-0.4

Tabel 6.1: Voorbeeld van een discrete regressie-analyse

Het logistische regressiemodel heeft drie grote nadelen. Ten eerste verliest het model enorm veel informatie bij de discretisatie van continue attribuutwaarden. De stap zorgt er zelfs voor dat attribuutwaarden van goede en slechte oplossingen in hetzelfde niveau kunnen belanden. Daarnaast kan het model onmogelijk een variabel gewicht geven aan de historische records. Dit is belangrijk om de context van een probleem in rekening te brengen bij het evalueren van nieuwe paden. Het logistische regressiemodel biedt ten derde geen mogelijkheid tot online training en moet dus bij elk nieuw traject volledig opnieuw getraind worden.

6.2.2 Additief bilateraal dalende model

Om de drie nadelen te voorkomen, introduceert dit deel een nieuw *conjoint model*. Het kernidee van dit model is dat de scores van de continue attribuutwaarden van historische trajecten lokale maxima zijn in de nutsvergelijkingen en dat deze scores dalen wanneer de attribuutwaarden afwijken van de optima. Het conjoint model $M_{abd}(X)$ dat we voorstellen, noemen we het *additieve bilateraal dalende model* en moet voldoen aan de volgende eisen:

- $M_{abd}(X)$ modelleert een probabiliteit voor continue attribuutwaarden waardoor de uitkomsten enkel tot het interval [0,1] behoren.
- $M_{abd}(X)$ is een gewogen som van attribuutprobabiliteiten $Pr_A(X)$. De vergelijking is dus gelijkaardig aan het additieve model 6.1, maar met het belangrijke

verschil dat de nutsscores $U_i(X)$ gemodelleerd zijn als probabiliteiten en begrensd zijn tot het interval [0,1]. Het gewicht van de attribuutprobabiliteiten zorgt er voor dat de uitvoer van het model tussen nul en één ligt wat een nodige voorwaarde is van het knopenmodel.

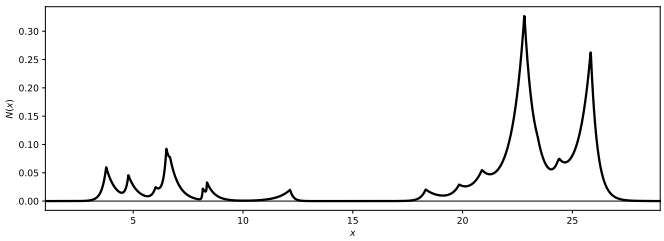
- De attribuutprobabiliteit $Pr_A(X)$ van attribuutwaarden van historische oplossingen x_{opt} moeten zich in een (lokaal) maximum bevinden. De attribuutprobabiliteit van de geassocieerde negatieve records moet laag zijn.
- Om gelijkaardige oplossingen te accepteren met dit model, moet de attribuutprobabiliteit geleidelijk dalen rond de (lokale) maxima.
- Het moet mogelijk zijn om historische oplossingen een variabel gewicht te geven om in het evaluatie-algoritme rekening te houden met de context van problemen.
- Het model moet gemakkelijk up te daten zijn met nieuwe trajecten zonder het hele model opnieuw te trainen.

De eerste twee eigenschappen zorgen ervoor dat het *conjoint model* opgebouwd is als een gewogen som van attribuutprobabiliteiten $Pr_A(X)$. Vergelijking 6.6 schrijft dit formeel neer. De gewichten w_A realiseren het onderling belang tussen de attributen en moet men manueel toekennen. De keuze voor uniforme gewichten is het meest voor de hand liggend. De voorwaarde die aan de gewichten verbonden zijn, zorgen ervoor dat de uitvoerwaarden van het model binnen het interval $[0, 1]$ blijven. In deze notatie betekent X de oplossing van een probleem.

$$M_{abd}(X) = \sum_{A \in M} w_A Pr_A(X) \qquad \sum_A w_A = 1 \text{ en } w_A \geq 0 \qquad \forall i \tag{6.6}$$

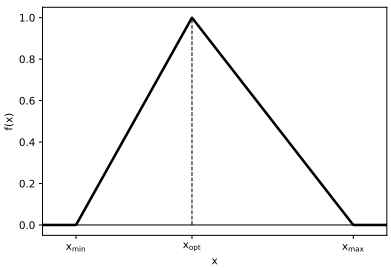
Uit de laatste vier eigenschappen kan men de conclusie trekken dat de curve van de attribuutprobabiliteit Pr_A over het domein van de attribuutwaarden eruit ziet als een accumulatie van pieken rond de historische attribuutwaarden waarbij de relatieve hoogte van elke piek afhangt van de context van het probleem. Figuur 6.1 geeft een voorbeeld van een attribuutprobabiliteit grafisch weer. Deze vorm kan men bekomen door Pr_A te ontbinden in meerdere componenten f_i waarbij elke component een piek voorstelt. Voor elke historische instantie bestaat er exact één component in de attribuutprobabiliteit. Men kan de context van problemen in rekening brengen door aan elke component een gewicht w_i toe te kennen. Dit gewicht berekent men aan de hand van de context zodat de hoogte van de pieken kan variëren. Vergelijking 6.7 geeft een formele beschrijving van de attribuutprobabiliteit. In deze vergelijking berekent en evalueert f_i de attribuutwaarde van X en berekent w_i de similariteit van de context van X met het historisch probleem. Deze context verschillende waarden aannemen zoals de dag, het aantal bestuurders of de stops.

$$Pr_A(X) = \sum_{i \in H} w_i(X) f_i(X) \qquad \sum_{i \in H} w_i = 1 \text{ en } w_i \geq 0 \qquad \forall j \forall i \tag{6.7}$$

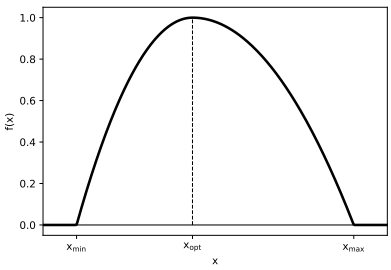


Figuur 6.1: Attribuutprobabiliteit

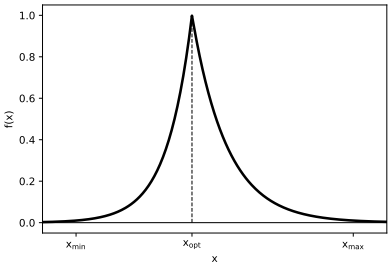
Tot slot moeten we de vorm van de pieken $f(x)$ kiezen. Figuur 6.2 geeft grafisch enkele mogelijkheden. Het hoogste punt van elke component bevindt zich in de attribuutwaarde van de historische oplossing x_{opt} en krijgt de score één. Vervolgens zoeken we tussen de attribuutwaarden van de negatieve datarecords die gegenereerd werden uit dezelfde historische instantie naar de waarden x_{min} en x_{max}. Dit zijn respectievelijk de attribuutwaarden in de records die het dichtst rondom x_{opt} liggen waarbij $x_{min} < x_{opt}$ en $x_{max} > x_{opt}$. In deze waarden moet de score dicht bij nul liggen. De drie parameters x_{opt}, x_{min} en x_{max} bepalen dus hoe ver de piek zich langs beide zijden van de optimale waarde uitstrekt. Omdat beide flanken een verschillende hellingsgraad hebben en de waarde van de curve een probabiliteit voorstelt, noemen we dit soort curve een *bilateraal dalende probabiliteit*. De vorm van de flanken kan men vrij kiezen en geeft een indicatie van hoe streng het model moet optreden tegen niet-optimale attribuutwaarden.



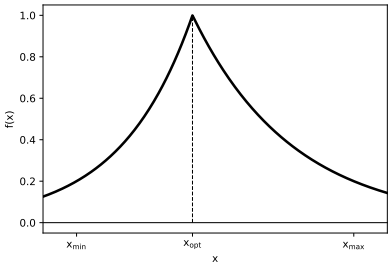
(a) Lineair



(b) Ellipsvormig



(c) Exponentieel ($p = 0.001$)



(d) Exponentieel ($p = 0.2$)

Figuur 6.2: Bilateraal dalende probabiliteiten

Een eerste vorm voor de flanken is een dalende rechte vanuit $(x_{opt}, 1)$ naar de punten $(x_{min}, 0)$ en $(x_{max}, 0)$ en daarbuiten de waarde nul. De piek die zich vormt is de asymmetrische driehoek uit figuur 6.2a.

$$f(x) = \begin{cases} \frac{x - x_{min}}{x_{opt} - x_{min}} & x \in [x_{min}, x_{opt}] \\ \frac{x_{max} - x}{x_{max} - x_{opt}} & x \in (x_{opt}, x_{max}] \\ 0 & x \in (-\infty, x_{min}) \text{ or } x \in (x_{max}, +\infty) \end{cases} \tag{6.8}$$

Een tweede mogelijkheid vindt zijn inspiratie het ideale-punt model van *conjoint analysis*. Vanuit de top $(x_{opt}, 1)$ volgt de curve twee ellipsvormige banen naar de punten $(x_{min}, 0)$ en $(x_{max}, 0)$. Daarbuiten neemt de functie de waarde nul. De piek ziet eruit als twee verschillende ellipsvormige kwarten met dezelfde hoogte en verschillende breedte. De raaklijn aan de top van deze piek is horizontaal waardoor het model aan attribuutwaarden die net niet optimaal zijn ook hoge scores geeft. Figuur 6.2b geeft een grafische voorstelling van deze piek.

$$f(x) = \begin{cases} 1 - \frac{(x_{opt} - x)^2}{(x_{opt} - x_{min})^2} & x \in [x_{min}, x_{opt}] \\ 1 - \frac{(x - x_{opt})^2}{(x_{max} - x_{opt})^2} & x \in (x_{opt}, x_{max}] \\ 0 & x \in (-\infty, x_{min}) \text{ or } x \in (x_{max}, +\infty) \end{cases} \tag{6.9}$$

De bilateraal exponentieel dalende probabiliteit is een voorbeeld van een zeer uitgesproken piek. Wanneer de attribuutwaarde niet optimaal is, daalt de probabiliteit snel. Beide zijden van het ideale punt strekken zich uit tot in het oneindige. Deze piekvorm vindt zijn maximum in het punt $(x_{opt}, 1)$ en neemt de extra parameter $p \geq 0$ die de probabiliteit bepaalt in de punten x_{min} en x_{max}. Figuren 6.2c en 6.2d geven een grafische voorstelling van deze piek voor twee verschillende waarden van p.

$$f(x) = \begin{cases} e^{-\lambda(x_{opt} - x)} \text{ met } \lambda = \frac{-ln(p)}{x_{opt} - x_{min}} & x \in (-\infty, x_{opt}) \\ e^{-\lambda(x - x_{opt})} \text{ met } \lambda = \frac{-ln(p)}{x_{max} - x_{opt}} & x \in [x_{opt}, +\infty) \end{cases} \tag{6.10}$$

6.3 Preferentiemodel

Het knopenmodel uit 5.1 geeft de objectieve vergelijking die het mogelijk maakt om nieuwe oplossingen aan de hand van *conjoint analysis* te evalueren op het laagste abstractieniveau. Deel 6.2.2 werkt het additief bilateraal dalende model tot in de details uit: een *conjoint model* die speciaal ontworpen is om gelijkaardige beslissingen te maken. De combinatie van deze twee modellen leidt tot een het preferentiemodel. Dit systeem geeft een algemeen framework die in staat is om de voorkeuren van eender welke VRP variant impliciet te leren en deze voorkeuren te gebruiken bij het evalueren van nieuwe oplossingen. Dit fundamentele model voor het leren maken van gelijkaardige beslissingen is de centrale contributie van deze thesis.

Deel 6.3.1 werkt het preferentiemodel analytisch uit. Deel 6.3.2 geeft een introductie tot het trainings- en evaluatie-algoritme van het preferentiemodel.

6.3.1 Objectieve vergelijking

Door het additief bilateraal dalende model 6.2.2 in te vullen in de objectieve vergelijking van het knopenmodel 5.5 bekomt men uitdrukking 6.11. Merk op dat alle termen in deze vergelijking meerdere indices hebben om duidelijk te maken over welke knoop, welk attribuut en welk historisch probleem het gaat. De buitenste twee lussen sommeren over de knopen k in alle paden P van oplossing X. De twee lussen in het logaritmen sommeren eerst over alle attributen i in het *conjoint model* van knoop k. De binnenste lus maakt de som over alle historische trajecten waar knoop k in voorkomt. Ter herhaling liggen alle w_{ki} vast op voorhand. De gewichten w_{kij} bepaalt het model aan de hand van de contextsimilariteit tussen probleem X en het historisch probleem j. De piekvorm f_{kij} evalueert de attribuutwaarde van attribuut i van knoop k in pad P van oplossing X in de j'de component.

$$
\begin{aligned}
X^* &= \arg\max_X \left(\prod_{P \in X} \prod_{k \in P} M_k(X, P, k) \right) \\
&= \arg\min_X \left(\sum_{P \in X} \sum_{k \in P} -log\left(M_k(X, P, k)\right) \right) \\
&= \arg\min_X \left(\sum_{P \in X} \sum_{k \in P} -log\left(\sum_{i \in M_k} w_{ki} Pr_{ki}(X, P, k) \right) \right) \\
&= \arg\min_X \left(\sum_{P \in X} \sum_{k \in P} -log\left(\sum_{i \in M_k} w_{ki} \sum_{j \in H_k} w_{kij}(X) f_{kij}(X, P, k) \right) \right)
\end{aligned}
\tag{6.11}
$$

6.3.2 Training en evaluatie

Appendices A.2 en A.3 geven het trainings- en evaluatie-algoritme voor het preferentiemodel. Het model laat online training toe omdat het gemakkelijk nieuwe componenten kan toevoegen aan de attribuutprobabiliteiten. De appendices geven ook nog enkele algoritmische details die belangrijk zijn bij de implementatie.

Hoofdstuk 7

Clustering

Het eerste deelprobleem bij het bepalen van nieuwe trajecten door een gegeven verzameling knopen is clustering. Informeel betekent clustering het verdelen van knopen onder de bestuurders waarbij de totale vraag van de toegewezen knopen de capaciteit van de wagens niet mag overschreden. Het clusteralgoritme moet ervoor zorgen dat de knopen die vaak samen voorkomen in historische paden in dezelfde cluster belanden. Daarnaast moeten de clusters ook gelijkaardige eigenschappen hebben als de historische clusters.

Het algoritme gebruikt het preferentiemodel om de impliciete voorkeuren uit historische clusters te leren en deze voorkeuren te gebruiken bij de evaluatie van nieuwe clusters. Deel 7.1 bespreekt de componenten van het preferentiemodel in functie van het leren van clusterpreferenties uit historische data. Dit deel bouwt het model systematisch op door eerst de attributen te definiëren, vervolgens het perturbatiemechanisme voor te stellen en ten slotte de contextsimilariteit te bespreken. Het tweede deel van dit hoofdstuk 7.2 geeft een beknopte uitleg over het clusteralgoritme die de meest geprefereerde verdeling van knopen zoekt aan de hand van het geleerde model.

In dit hoofdstuk is V de verzameling knopen die geclusterd moet worden. Het clusteralgoritme zoekt een verdeling van de knopen B waarin elke knoop exact tot één cluster C_i behoort en waarbij er geen lege clusters zijn. De parameter v beduidt het aantal voertuigen en is altijd gegeven. Omdat elke bestuurder minstens één knoop moet hebben, is het aantal clusters altijd gelijk aan het aantal voertuigen.

$$V = \{k_1 ... k_n\}$$

$$B = \{C_1 ... C_v\} \quad \text{met} \quad C_i = \{k_i | k_i \in N\}$$

7.1 Preferenties

Het preferentiemodel moet in staat zijn goede clusters van slechte clusters te onderscheiden. Daarom moet de attributenkeuze de focus leggen op kwaliteitsvolle eigenschappen van clusters. Een goede attributenkeuze leidt uiteraard tot een hogere

accuraatheid van het model. Deel 7.1.1 identificeert de eigenschappen van clusters. Daarnaast bespreekt deel 7.1.2 het perturbatiemechanisme om clusters te genereren die gelijkaardig zijn aan de historische clusters. Deel 7.1.3 bespreekt een aantal manieren om context in rekening te brengen bij de evaluatie van clusters.

Het trainingsalgoritme voor het preferentiemodel A.2 neemt een historische oplossing X argument en maakt een lus over elk pad P in deze oplossing. Cluster C, bepaald door pad P, bevat alle knopen k_i van P bevat behalve het depot k_0. Het depot k_0 behoort nooit tot een cluster, omdat die knoop triviaal tot elk pad behoort. Het includeren van het depot kan *bias* in de attribuutwaarden introduceren en dus het de accuraatheid van het model contamineren. Alle attribuutprobabiliteiten van de knopen in de historische clusters krijgen één nieuwe component. De parameters van deze componenten worden berekend na het perturberen van de oorspronkelijke clusters.

$$P = k_0 \rightarrow k_1 \rightarrow k_2 \rightarrow ... \rightarrow k_{i-1} \rightarrow k_i \rightarrow k_{i+1} \rightarrow ... \rightarrow k_n \rightarrow k_0$$

$$C = \{k_1, k_2, ..., k_{i-1}, k_i, k_{i+1}, ..., k_n\}$$

7.1.1 Attributen

De attributenkeuze moet de clusters zo goed mogelijk ontleden in enkele eigenschappen. De eigenschappen die belangrijk kunnen zijn bij het leren van voorkeuren zijn statistieken over het aantal knopen in een cluster en de afstand tussen deze knopen. Deze statistieken zorgen ervoor dat het model kan leren hoe groot de clusters moeten zijn. Andere geometrische eigenschappen zoals de breedte, de hoogte en het geografische centrum van een cluster zorgen ervoor dat het preferentiemodel historische clusterregio's kan bepalen.

Van elke clustereigenschap bestaat er zowel een globale als een lokale variant. De globale variant beschouwt alle knopen van cluster C bij het berekenen van de attribuutwaarden. De lokale variant beschouwt alleen de n dichtste buren NN_n van knoop k_i in cluster C. De waarde van n is een vaste parameter van het preferentiemodel en de afstand tussen de knopen is altijd Euclidisch. Onderstaande formule geeft formeel de beschrijving van de lokale cluster $C_l(i)$ rond knoop k_i.

$$C_l(i) = \{k_n \mid k_n \in NN_n(C, k_i)\}$$

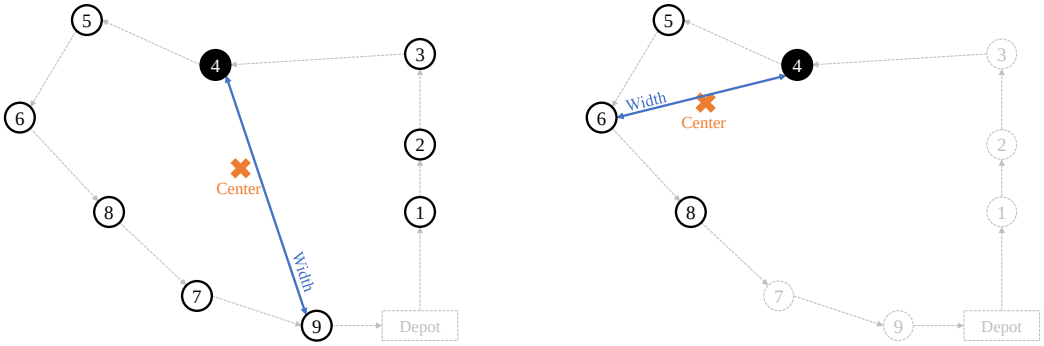
Wanneer n kleiner is dan het aantal knopen in C, dan is de lokale cluster identiek aan de globale cluster. Knoop k_i behoort altijd tot $NN_n(C, k_i)$. Op deze manier kan men altijd lokale clusters definiëren, zelf wanneer C slechts één knoop bevat. De eigenschappen van een lokale cluster zijn dus sterk afhankelijk van de positie van k_i.

In de historische dataset zijn de stops V in bijna elk probleem verschillend. Om deze reden vertonen de paden (en de gerelateerde clusters) in de historische oplossingen meestal kleine verschillen. Daardoor is het plausibel dat een preferentiemodel met alleen globale clusterattributen niet de nodige resolutie bieden om de voorkeuren te leren. We nemen de logische assumptie dat naburige knopen in een cluster vaker in hetzelfde pad voorkomen dan uiteenliggende knopen. Wanneer we vervolgens lokale clusterattributen in het preferentiemodel introduceren, kan het beslissingsondersteunend systeem veel nauwkeuriger bepalen welke knopen het best samen voorkomen in een pad. Tabel 7.1 geeft de attributenkeuze voor het clustermodel waarbij elk attribuut een naam, een wiskundige formulering en een intuïtieve uitleg krijgt. Figuur 7.1 geeft een illustratie van de globale en de lokale clustereigenschappen. De grijze elementen in de figuren hebben geen invloed op de berekening van de attribuutwaarden.

Naam **Formule**	**Intuïtie**
Aantal knopen $\#(C)$	Dit attribuut geeft een aanwijzing over het aantal knopen in de clusters. Met dit attribuut leert elke knoop iets over het geprefereerd aantal knopen in het pad. Het attribuut leert aan het preferentiemodel hoeveel stops de bestuurders bij voorkeur moeten bedienen.
Centrum van globale cluster $center(C)$	Dit zijn in principe twee attributen, namelijk een x- en een y-coördinaat. Dit attribuut geeft aan waar het geografisch centrum van een cluster zich zou moeten bevinden. Dit attribuut is nuttig bij het bepalen van de geprefereerde regio's van een knoop.
Breedte van globale cluster $width(C, k_i)$	De breedte van een globale cluster is gedefinieerd als de maximale afstand van knoop k_i tot de knopen in C. Dit attribuut leert het model iets over de geprefereerde afstand tussen knoop k_i het verste punt in de cluster. Dit attribuut geeft samen met het centrum van de globale cluster een ruw idee van het gebied waarin de knopen van een cluster liggen.
Centrum van de lokale cluster $center(C_l(i))$	Deze eigenschap introduceert een lokale clustereigenschap in het model. Het centrum bevat opnieuw een x- en y-coördinaat en daarom zijn dit twee attributen. Deze eigenschap brengt de mogelijkheid om het preferentiemodel iets bij te leren over de geprefereerde lokale clusterregio's.
Breedte van de lokale cluster $width(C_l(i), k_i)$	De breedte van een lokale cluster is gedefinieerd als de maximale afstand van knoop k_i tot de knopen in $C_l(i)$. Hiermee komt het preferentiemodel te weten hoever de lokale clusterregio van knoop k zich mag uitspreiden. Samen met het centrum van de lokale cluster bepaalt het model het gebied van de geprefereerde dichtste knopen van k_i.

Tabel 7.1: Tabel met attributen van clusters

De attributenkeuze in tabel 7.1 heeft twee doelen. Ten eerste stelt de combinatie van de attributen het model in staat om geografische regio's te identificeren waarin de geprefereerde clusters zich bevinden. Ten tweede moeten de attribuutwaarden gemakkelijk te berekenen zijn. Dit geeft een groot computationeel voordeel bij het trainen en evalueren van het preferentiemodel.



(A) Globaal (B) Lokaal ($n = 4$)

Figuur 7.1: Attributen van een cluster (Knoop 4)

Er bestaan andere attributen die de vorm van een cluster veel nauwkeuriger kunnen beschrijven. Een eerste voorbeeld is de omtrek en de oppervlakte van de complexe omhullende van de knopen. Daarnaast kan de variantie van de twee principiële componenten bepaald met *Prinipal Component Analysis* (PCA) op de positie van de knopen een goede metriek geven over de geografische spreiding van de knopen rond het centrum van een cluster. Hoewel deze twee voorbeelden veel potentieel bieden voor het verhogen van de accuraatheid, includeert het finale preferentiemodel deze attributen niet wegens de hoge computationele kost bij het berekenen van de attribuutwaarden. Dit soort attributen maken de solver traag.

7.1.2 Perturbatiemechanisme

Om het preferentiemodel te trainen met domeinspecifieke informatie voor clustering, moet het perturbatiemechanisme clusters genereren die gelijkaardig zijn aan de clusters in de historische trajecten. De perturbatie mag zowel knopen verwijderen uit clusters als nieuwe knopen toevoegen die tot een ander pad behoren in de oplossing. De gegenereerde clusters moeten aan twee voorwaarde voldoen. Ten eerste moet een geperturbeerde cluster altijd minstens één knoop hebben. Daarnaast mag het perturbatiemechanisme het depot nooit toevoegen aan een cluster. Algoritme A.4 geeft de beschrijving van het perturbatiemechanisme voor clustering in detail.

7.1.3 Contextsimilariteit

In het trainingsalgoritme van het preferentiemodel (A.2) is de functie *add_component* verantwoordelijk voor twee zaken. Ten eerste voegt de functie een nieuwe component aan de attribuutprobabiliteiten. Ten is de functie verantwoordelijk om de context van het historisch probleem te koppelen aan de nieuwe componenten. Het evaluatie-algoritme van het preferentiemodel (A.3) brengt vervolgens de probleemcontext in rekening door een gewicht toe te kennen aan iedere component. Het gewicht geeft aan in welke mate de historische en de nieuwe problemen op elkaar lijken. Tabel 7.2 geeft alle contextvariabelen die beschikbaar zijn in de dataset. Tabel 7.3 schetst een eenvoudige probleemcontext die alle elementen bevat. De verscheidenheid aan contextvariabelen leidt tot een explosie van mogelijkheden om contextsimilariteit te definiëren. Tabel 7.4 definieert één mogelijke similariteitsfunctie per contextvariabele.

Variabele	**Beschrijving**
Nummer	Historische trajecten zijn genummerd van oud naar recent.
Weekdag	Elk probleem speelt zich af op een bepaalde dag van de week (inclusief zaterdag en zondagen).
Stops	Alle knopen met een vraag die groter is dan nul. Dit zijn de klanten in het probleem.
Vraag	Deze variabele bepaalt de vraag van elke klant.
Bestuurders	Het aantal bestuurders in het probleem. Dit is inmiddels ook het aantal paden in de oplossing.
Capaciteit	Deze contextvariabele geeft de maximale capaciteit van de wagens in een probleem.

Tabel 7.2: Contextvariabelen van het clusteringprobleem

Probleem 53	
Aantal bestuurders : 3	
Capaciteit : 13	
Weekdag : 3 (Woensdag)	
Knoop	**Vraag**
0	0
1	1
2	1
3	2
4	1
5	4
6	8
7	2
8	1

Tabel 7.3: Voorbeeld van contextvariabelen

Variabele	**Metriek**
Nummer	Een tijdgebaseerde contextsimilariteit (TS) geeft oude trajecten een kleiner gewicht dan nieuwe trajecten. Het gewicht kan een lineair, kwadratisch of exponentieel verband hebben met de ouderdom van het historisch traject. Exponentieel dalende gewichten geven de beste resultaten [4, 2]. Vergelijking 7.1 geeft een voorbeeld waarin T en t respectievelijk het nummer van het nieuwe en het historische probleem zijn. $$TS(T,t) = \left(\frac{t}{T}\right)^i \qquad (i \geq 1) \tag{7.1}$$
Weekdag	Om de weekdag een rol te laten spelen in het evalueren van oplossingen, is het gewicht van componenten die gekoppeld zijn aan een context van dezelfde weekdag dubbel zo groot als het gewicht van de andere componenten.
Stops	De triviaalste metriek voor overeenkomst tussen de stops van twee probleemcontexten is de Jaccard-index tussen de knoopverzamelingen V_1 en V_2. $$J(V_1,V_2) = \frac{V_1 \cap V_2}{V_1 \cup V_2} \tag{7.2}$$
Vraag	Het gewicht is omgekeerd evenredig met het verschil tussen de vraag q_{k1} en q_{k2} van knoop k in twee contexten. Een groter verschil in vraag resulteert in een kleiner gewicht. $$Similarity(q_{k1}, q_{k2}) = \frac{1}{(\mid q_{i1} - q_{i2} \mid +1)^i} \qquad (i \geq 1) \tag{7.3}$$
Bestuurders	Een groter verschil in aantal bestuurders geeft een kleinere contextsimilariteit. Gebruik hiervoor vergelijking 7.3 met $i = 1$.
Capaciteit	Een groter verschil in de capaciteit van de wagens resulteert in een kleinere contextsimilariteit. Gebruik hiervoor vergelijking 7.3 met $i = 2$ waardoor het gewicht veel sneller klein wordt dan het gewicht het vorige voorbeeld.

Tabel 7.4: Metrieken voor contextsimilariteit

De dataset waar dit werk zich op baseert, is afkomstig van één bedrijf waarvan de vloot niet vaak verandert. Daarom kan men veronderstellen dat de capaciteit van de wagens ongeveer constant blijft doorheen de historische problemen. Dit is echter niet het geval en daarom is de capaciteit een controversiële contextvariabele voor dit

onderzoek. Analyse van de dataset resulteerde in de observatie dat elke klant een relatief vaste vraag heeft en dat de vraag niet veel invloed heeft op de toewijzing van de knopen aan de bestuurders. Om deze twee redenen gebruikt het preferentiemodel de vraag en de capaciteit niet bij het bepalen van de contextsimilariteit. Omdat er geen triviale similariteitsfunctie bestaat voor weekdag en het aantal bestuurders, laat het model deze contextvariabelen ook vallen. In de historische oplossingen is er een duidelijke correlatie tussen de stops en de weekdag. Bepaalde stops komen enkel voor op specifieke weekdagen. Omdat de similariteit tussen de stops van twee problemen direct meetbaar is aan de hand van de *Jaccard*-index neemt het preferentiemodel deze variabele op bij het bepalen van het gewicht. Daarnaast is het menselijk dat persoonlijke voorkeuren na verloop van tijd veranderen. Om deze reden is het een goed idee om de ouderdom van de historische oplossing ook in rekening te brengen bij het bepalen van het gewicht. Vergelijking 7.4 geeft een uitdrukking voor contextsimilariteit die dit werk later zal gebruikten in de experimenten. In deze vergelijking zijn t en T respectievelijk het nummer van het historisch en het nieuw probleem en V_1 en V_2 de stops van beide problemen.

$$w_{kij}(X) = CS(T, t, V_1, V_2) = \frac{t}{T} \cdot \frac{V_1 \cap V_2}{V_1 \cup V_2} \tag{7.4}$$

Door de goed beredeneerde verklaring die vergelijking 7.4 schuilt, onderzoeken we de keuze van de metriek voor contextsimilariteit op de accuraatheid van het model niet verder in deze thesis.

7.2 Algoritme

Het clusteralgoritme zoekt een oplossing aan de hand van *iterated local search* (ILS). Deze metaheuristiek gebruikt een constructieve heuristiek om een eerste kwaliteitsvolle oplossing te vinden. Vervolgens doet er zich een iteratief proces voor dat een keten van oplossingen genereert. Het iteratief proces van ILS perturbeert eerst aan de huidige oplossing. Dit leidt tot een tussenoplossing die verbeterd wordt aan de hand van de lokale zoekprocedure. Het perturbatiemechanisme moet sterk genoeg zijn om lokale optima te kunnen ontsnappen. De lokale zoekprocedure mag niet in staat zijn de perturbatie ongedaan te maken omdat het algoritme dan vaak op hetzelfde lokale optimum zou terugvallen. Een acceptatiecriterium beslist welke oplossing behouden moet blijven voor de volgende iteratie. [23]

Er zijn twee redenen waarom we voor dit probleem ILS boven genetische algoritmen verkiezen. Ten eerste bestaat er geen één-op-één relatie tussen de representatie van een oplossing en de oplossing zelf. De representaties (1112322) en (2223133) waarin het nummer aangeeft welke stops samen horen, geven bijvoorbeeld exact dezelfde verdeling onder de bestuurders. Ten tweede zijn er voor dit probleem niet veel kruisingsoperatoren beschikbaar in de literatuur.

Appendix A.5 beschrijft het clusteralgoritme. De initiële clusters worden gege-

nereerd door de paden van de oplossing met transitieprobabiliteiten (3.1) om te vormen tot clusters. Vervolgens past het clusteralgoritme het iteratief proces uit. Het perturbatiemechanisme selecteert twee clusters uit de oplossing en ruilt twee stops in de eerste cluster met één stop in de tweede cluster. De perturbaties zijn willekeurig. De lokale zoekprocedure zoekt de beste uitwisseling van één knoop tussen twee clusters. Omdat het perturbatiemechanisme *'twee-in-ruil-voor-één'* is, kan de lokale zoekprocedure nooit op de initiële oplossing terugvallen.

Hoofdstuk 8

Routing

Routing is het tweede deelprobleem bij het vinden van nieuwe oplossingen. Deze stap past het *travelling salesman problem* herhaaldelijk toe op de clusters. Maar in plaats van het kortste Hamiltoniaanse pad te vinden door de knopen, zoekt het routingalgoritme het pad die de voorkeuren van de planners en de bestuurder maximaliseert aan de hand van het preferentiemodel. Dit model is verantwoordelijk voor het evalueren van de voorgesteld paden. Deel 8.1 geeft eerst de attributen van het preferentiemodel voor routing. Daarna volgt het perturbatiemechanisme waarmee het trainingsalgoritme uitgerust wordt. Deel 8.2 stelt een eenvoudig genetisch algoritme die gebruik maat van de objectieve vergelijking van het preferentiemodel (6.11).

Dit hoofdstuk hanteert enkele notationele conventies. Een cluster wordt voorgesteld als een ongeordende verzameling knopen C. Uit de definitie van het CVRP volgt dat het depot (k_0) triviaal het begin- en eindpunt is van ieder pad. Om deze reden neemt een cluster deze knoop niet op in de verzameling. Een pad P is een ordening van deze knopen, waarbij het depot aan beide het vertrek- en eindpunt is.

$$C = \{k_1, k_2, ..., k_{i-1}, k_i, k_{i+1}, ..., k_n\}$$

$$P = k_0 \to k_1 \to k_2 \to ... \to k_{i-1} \to k_i \to k_{i+1} \to ... \to k_n \to k_0$$

8.1 Preferenties

De accuraatheid van het preferentiemodel hangt af van de keuze van de attributen. Een slechte attributenkeuze leidt meteen tot een zwak presterend preferentiemodel. Om deze reden is het nodig goed na te denken over de interessante aspecten van paden. Deel 8.1.1 onderzoekt hoe het preferentiemodel een pad ontbindt in enkele attributen. Deel 8.1.2 geeft het perturbatiemechanisme waarmee het preferentiemodel in de trainingsfase negatieve voorbeelden genereert om de parameters van de componenten van de attribuutprobabiliteiten te schatten. Deel 8.1.3 bespreekt contextsimilariteit in het kader van het routingprobleem.

Herinner dat het trainingsalgoritme voor het preferentiemodel (A.2) een historische oplossing X neemt als input en een lus maakt over elk pad P in deze oplossing. Vervolgens maakt het algoritme een lus over elke knoop k in het pad. Na perturbatie van het historisch pad, berekent het algoritme de attribuutwaarden in functie van de geselecteerde knoop en krijgt elke attribuutprobabiliteit één nieuwe component.

8.1.1 Attributen

De attributen van het preferentiemodel voor routing moeten een knoop k_i in staat stellen iets bij te laten leren over het pad P waarin het zich bevindt. De attributen kunnen enerzijds informatie halen uit het globale pad. Anderzijds kan een attribuut een kleiner deel van het pad rond de knoop analyseren. Globale eigenschappen zeggen iets over het pad P in zijn geheel.

$$P = k_0 \to k_1 \to k_2 \to ... \to k_{i-1} \to k_i \to k_{i+1} \to ... \to k_n \to k_0$$

Lokale attributen beschouwen enkel het pad in een vaste omgeving rond een knoop. Onderstaand voorbeeld geeft een lokaal pad rond knoop k_i met twee knopen voor en twee knopen na k_i. Het subscript l indiceert dat het om het lokaal pad gaat en index i bepaalt rond welke knoop we de omgeving nemen. De grootte van de omgeving is een parameter van het model.

$$P_l(i) = k_{i-2} \to k_{i-1} \to k_i \to k_{i+1} \to k_{i+2}$$

Door het feit dat het pad een lus maakt in het depot, kan er altijd een lokaal pad gemaakt worden van de gekozen lengte. In het knoop k_1 vinden we bijvoorbeeld het volgende lokaal pad.

$$P_l(1) = k_n \to k_0 \to k_1 \to k_2 \to k_3$$

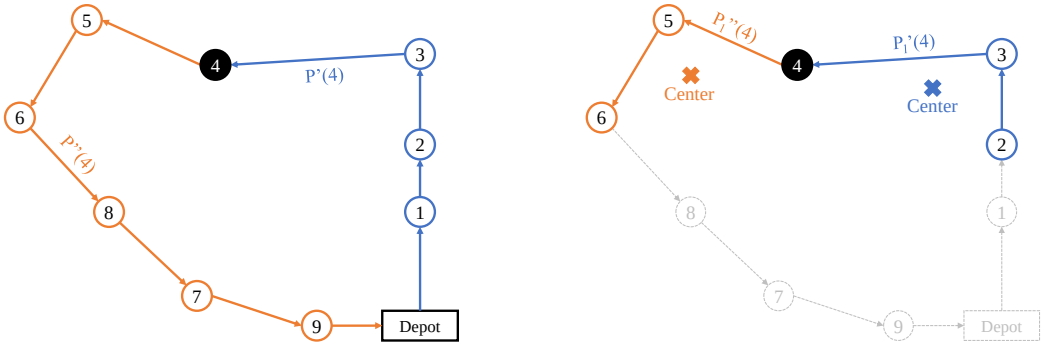
Ten slotte maken we ook de observatie dat elke knoop zowel een globaal pad als een lokaal pad opsplitst in een deel voor en een deel na de knoop. Deze opsplitsing wordt voor knoop k_i hieronder gegeven.

Globaal voor k_i	$P'(i) = k_0 \to k_1 \to ... \to k_{i-1} \to k_i$
Globaal na k_i	$P''(i) = k_i \to k_{i+1} \to ... \to k_n \to k_0$
Lokaal voor k_i	$P'_l(i) = k_{i-2} \to k_{i-1} \to k_i$
Lokaal na k_i	$P''_l(i) = k_i \to k_{i+1} \to k_{i+2}$

Nu de definities van globale paden, lokale paden, omgeving en opsplitsing van een pad in een knoop gekend zijn, is het mogelijk een aantal attributen te definiëren voor het preferentiemodel aangepast aan het routingprobleem. Tabel 8.1 geeft aan elk attribuut een naam, een formule en een intuïtieve uitleg waarom het attribuut een meerwaarde biedt voor in het preferentiemodel. Figuur 8.1 stelt visueel de eigenschappen van lokale en globale paden voor.

Naam **Formule**	**Intuïtie**
Positie in pad i	Dit attribuut geeft een indicatie over de positie waarop een knoop het liefst voorkomt in een pad. Op deze manier kan de planner beslissen of een knoop beter vroeger of later bezocht wordt.
Totale afstand voor bezoek $distance(P'(i))$	Door de afstand van het eerste deel van het globale pad te onderzoeken komt het model te weten hoeveel afstand er normaal gezien wordt afgelegd door de bestuurder voor het bezoek aan een knoop.
Totale afstand na bezoek $distance(P''(i))$	Dit attribuut wordt om exact dezelfde reden als het voorgaande attribuut aan het model toegevoegd. Merk ook op dat de combinatie van deze twee attributen leidt tot een idee van de totale lengte van het pad.
Richting van het inkomende pad $center(P'_l(i))$	Dit zijn in principe twee attributen, namelijk een x- en y-coördinaat. Dit betekent dat in het preferentiemodel voor zowel de x- als de y-coördinaat een attribuutprobabiliteit moet toevoegen. Het attribuut berekent het geografisch centrum van de knopen die zich in het eerste deel van het lokale pad bevinden. Op deze manier heeft elke knoop een voorkeursrichting waaruit het pad moet komen. Merk op dat het centrum van het eerste deel van het globale pad nemen ook een optie kan zijn als attribuut. We kiezen met opzet voor het lokale pad om ervoor te zorgen dat het preferentiemodel rekening houdt met lokale kenmerken van de oplossingen.
Richting van het uitgaande pad $center(P''_l(i))$	Dit attribuut is gelijkaardig aan het attribuut dat hierboven beschreven is, maar met het verschil dat hier het geografisch centrum van het lokale uitgaande pad wordt berekend. Hierdoor krijgt elke knoop in het model een voorkeursrichting van het uitgaande pad. Deze informatie in combinatie met het vorige attribuut geaggregeerd over alle knopen vormt een zeer sterke basis voor het leren van geometrische lokale vormaspecten van de paden.

Tabel 8.1: Tabel met attributen van paden



(A) Globaal (B) Lokaal ($n = 2$)

Figuur 8.1: Attributen van een pad (Knoop 4)

Daarnaast is er nog veel ruimte om andere attributen van paden te definiëren. We kunnen bijvoorbeeld kijken naar het aantal stops voor en na het bezoek van een knoop. Een andere mogelijkheid is om een attribuut op te stellen die iets leert over de belasting van de voertuigen op basis van capaciteit en de vraag van de klanten die eerder bezocht werden. Een laatste voorbeeld is om statistieken van het pad te verzamelen zoals de gemiddelde lengte van de bogen in het pad. Men kan veel onderzoek uitvoeren op de attributenkeuze en op de interacties tussen de attributen. Het doel van dit werk is echter om een framework te ontwikkelen die aan de hand van attributen van oplossingen keuzes leert maken in nieuwe oplossingen. Om deze reden onderzoeken we de exacte keuze van de attributen niet verder.

8.1.2 Perturbatiemechanisme

Het routingprobleem krijgt als input een cluster en moet het meest geprefereerde pad door de knopen van dit cluster vinden. In het perturbatiemechanisme moet het model specifiek informatie proberen te genereren over het pad zelf en niet over clusters. Daarom limiteren de perturbatiemogelijkheden zich tot het door elkaar schudden van de volgorde van de knopen in een historisch pad. Het is dus niet toegelaten knopen onderling tussen de bestuurders uit te wisselen. Het gepermuteerde pad L moet voldoen aan de volgende eigenschappen bij het bepalen van de attributen voor knoop k_i om opgenomen te worden in het model:

- Het geperturbeerde pad L is een permutatie van het originele pad P en bezoekt daarom alle knopen van P exact éénmaal, uitgezonderd het depot k_0.
- L begint en eindigt in het depot k_0.
- Het lokale pad $L_l(i)$ is verschillend van $P_l(i)$ bij het bepalen van de attributen voor knoop i.

Er bestaan drie manieren om de geshuffelde paden te genereren. De eerste manier ligt voor de hand en is enkel bruikbaar bij korte paden. Door exhaustieve enumeratie van de permutaties van het oorspronkelijk pad zonder het depot bekomen we alle

mogelijkheden van de zoekruimte. Daarna moet het depot nog aan het begin en het eind van de permutaties geplakt worden en moeten de voorwaarden gecheckt worden. De tweede manier is bijna identiek aan de eerste manier, maar verschilt omdat de methode een vast aantal permutaties willekeurig genereert in plaats van deze te enumereren. Deze manier werkt benaderend, maar is wel veel sneller voor langere paden. Een derde optie is het ontwikkelen van een efficiënt algoritme die de parameters x_{min} en x_{max} direct kan vinden zonder gepermuteerde paden te generen. Naar de derde methode werd geen verder onderzoek gedaan omdat deze algoritmen afhankelijk zijn van de attributenkeuze.

8.1.3 Contextsimilariteit

De enige context die in het routingprobleem belangrijk is, zijn de clusters zelf. Het preferentiemodel voor het clusterprobleem hield reeds rekening met probleemcontext zoals de weekdag, het aantal bestuurders en de capaciteit.

Wanneer we een component toevoegen aan de attribuutprobabiliteit van knoop i in historisch pad P, voegen we C_P, de cluster bepaald door P, toe als contextvariabele aan de component. Bij het evalueren van een nieuw pad door de punten van een cluster C_C, gebruiken we de Jaccard-index $J(C_P, C_C)$ (7.2) om de context van de componenten te vergelijken.

8.2 Algoritme

Deel 2.1 geeft een overzicht van de mogelijke methodes voor het oplossen van het *travelling salesman problem*. Deel 5.2 maakte duidelijk dat de objectieve vergelijking van het preferentiemodel niet goed samengaat met de *state of the art* exacte algoritmen voor het oplossen van het TSP probleem. Daarom stellen we in voor het routingalgoritme een *solver* voor die zijn oorsprong vindt in het domein van de genetische algoritmen. Het achtergrondmateriaal hiervoor staat in sectie 2.4. In de dataset bestaan de paden uit maximaal 10 knopen. Dit vormt een relatief kleine zoekruimte waardoor het genetisch algoritme eenvoudig kan blijven. Algoritme A.6 in de appendices geeft overzichtelijk weer hoe dit algoritme opgebouwd is en legt de keuze van de genetische operatoren in detail uit.

Hoofdstuk 9

Dataset

Dit hoofdstuk geeft een inleiding tot de dataset die gebruikt wordt in de experimenten van hoofdstuk 10. Deel 9.1 beschrijft de informatie die aanwezig is in de historische voorbeelden. Vervolgens haalt deel 9.2 twee technieken voor het extraheren van niet-expliciete informatie in de dataset. Deel 9.3 sluit dit hoofdstuk af met een aantal belangrijke observaties over de dataset en een grafisch voorbeeld van een historisch probleem met de oplossing.

Het onderzoek in dit werk is gebaseerd op dataset die *J. Mandi, R. Canoy, V. Bucarey, en T. Guns* gebruikten in publicaties [4, 2, 17]. De data is geanonimiseerd en publiek toegankelijk op *Github* [16].

9.1 Beschrijving

De historische dataset bestaat uit 201 dagelijkse plannen die verzameld werden in een tijdspanne van 39 weken. De plannen werden gegenereerd door een menselijke planner en gebruikt door het bedrijf in hun dagelijkse operaties. Elk voorbeeld is genummerd en geordend in tijd. Een historisch probleem wordt gekenmerkt door de stops, de vraag van elke stop, de weekdag, de capaciteit en het aantal voertuigen. [17]

De dataset bevat 73 unieke klanten waarbij elke klant verschillend is van het depot. De historische problemen bevatten gemiddeld 31 stops die van een bepaalde vraag moeten voorzien worden door gemiddeld 8 voertuigen. De voorbeelden zijn gegroepeerd per weekdag (inclusief zaterdag en zondag) met een gemiddelde van 29 voorbeelden per weekdag. Deze groepering bootst de operationele wijze van het bedrijf na. [4, 2].

Voor elk probleem in de historische dataset is de oplossing gegeven als een verbindingsmatrix (*adjacency matrix*). Deze matrix geeft aan welke bogen er genomen worden in de oplossing. Appendix B bespreekt één voorbeeld uit de dataset in detail door de de context te schetsen en de oplossing te visualiseren. Het laatste element in

de dataset is een matrix met de afstanden tussen alle stops.

9.2 Extractie van informatie

Om experimenten te kunnen uitvoeren op de dataset en de performantie van het preferentiemodel te valideren, moet bepaalde informatie die in de dataset vervat is, getransformeerd worden. Ten eerste is elke historische oplossing gegeven als een verbindingsmatrix of *adjacency matrix*. Het trainingsalgoritme neemt echter paden als invoer. Sectie 9.2.1 behandelt de extractie van de paden uit de oplossingen. Daarnaast definieerden hoofdstukken 7 en 8 geometrische attributen voor clusters en paden. Om de attribuutwaarden te berekenen moet het algoritme de coördinaten van de knopen kennen. Sectie 9.2.2 geeft een methode om deze coördinaten te bepalen.

9.2.1 Paden

Voor een perfecte dataset is het vinden van de paden uit de verbindingsmatrix eenvoudig. Aangezien het CVRP exact één bestuurder toekent aan elke knoop, is er in elke knoop één inkomende en één uitgaande boog (behalve in het depot). Dit betekent dat het pad gereconstrueerd kan worden door de bogen die vanuit het depot vertrekken eenvoudigweg te volgen.

De historische trajecten in de dataset voldoen echter niet altijd aan deze *constraint*. Het komt voor dat meerdere bestuurders dezelfde knoop bezoeken waardoor er occasioneel meerdere inkomende en uitgaande bogen zijn in de knopen van de verbindingsmatrix. Hierdoor kan men in een knoop waar paden kruisen niet deterministisch beslissen welke combinatie van de inkomende en uitgaande paden de historische paden van de bestuurders vormen. Dit probleem is in de gevallen waarin lussen voorkomen nog erger. Deel A.7 in de appendices geeft een algoritme op de paden te extraheren uit de verbindingsmatrix. Dit algoritme kan nooit honderd procent zeker zijn of het de juiste beslissing heeft gemaakt bij het construeren van paden door een knoop met meerdere bestuurders.

9.2.2 Coördinaten

De dataset geeft de afstand tussen de knopen in de vorm van een afstandsmatrix. Bij het impliciet leren van de voorkeuren van planners en bestuurders, kijkt het preferentiemodel naar geometrische eigenschappen zoals het geografische centrum van een cluster of de richting van het uitgaande pad. Om de waarden van deze eigenschappen te bepalen, moet de positie van de knopen gekend zijn. Daarom is het nodig om de coördinaten van alle knopen in tweedimensionale ruimte te schatten op basis van de afstandsmatrix. Formeel betekent dit dat elke knoop een 2D-coördinaat moet krijgen waarbij het verschil tussen de waarden in de afstandsmatrix en de respectievelijke Euclidische afstand tussen de coördinaten minimaal is. Vergelijking 9.1 geeft de wiskundige formulering van dit mimalisatieprobleem.

$$\arg\min_{X,Y} \sum_{i \in V} \sum_{j \in V} \left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - D(i,j) \right) \tag{9.1}$$

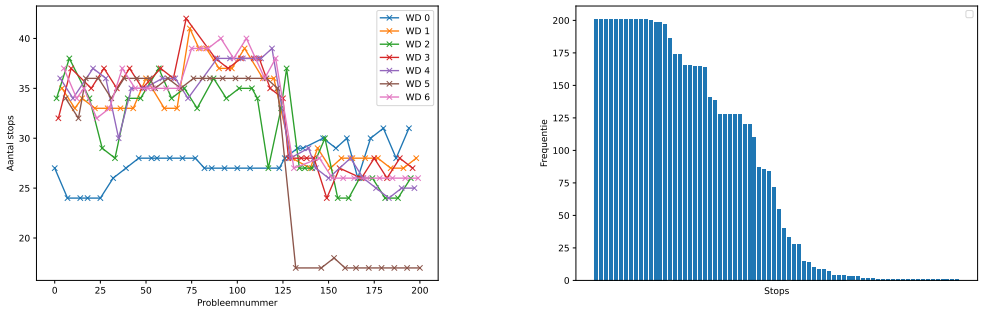
Multidimensional Scaling (MDS) of *Principal Coordinates Analysis* is een geometrische methode voor het construeren van een configuratie gegeven de afstand tussen punten. Deze techniek maakt het mogelijk de structuur bloot te leggen die in hoogdimensionale ruimten besloten ligt. MDS beeldt punten in hogere dimensies af op lagere dimensies waarbij de afstand of dissimilarieit tussen de punten zo goed mogelijk bewaard blijft [11]. Deze techniek lost het probleem van de ontbrekende coördinaten dus meteen op. Merk op dat translaties en rotaties van deze transformatie niets verandert aan de afstand tussen de punten en dus een even goede schatting geeft. De transformatie bepaalt dus de relatieve positie tussen de knopen.

9.3 Analyse

9.3.1 Stops

Figuur 9.1a geeft het aantal stops van de problemen gegroepeerd per weekdag. Tussen probleemnummer 0 en 125 zijn deze aantallen redelijk stabiel. Daarna volgt er rond nummer 125 een *data drift*. Na dit punt stabiliseren de waarden zich opnieuw. Hierdoor zal in de experimenten altijd vermeld worden of de gehele dataset of enkel het deel voor *data drift* gebruikt wordt. Deze observatie toont aan dat het belangrijk is om het gewicht van een component in het preferentiemodel afhankelijk te maken van de ouderdom van de historische oplossing.

Figuur 9.1b toont de frequentie van alle stops geordend van hoog naar laag. Ongeveer 30% van de stops komt in meer dan 75% van de historische instanties voor. Omdat deze stops vaak bediend worden, bezitten deze knopen veel informatie over de voorkeuren van de planner. De laatste 50% van de stops in de figuur komen eerder voor bij uitzondering.



(A) Aantal stops per weekdag (B) Frequentie van de stops

FIGUUR 9.1: Analyse van de stops

9.3.2 Inconsistenties

In de historische oplossingen komt het voor dat een knoop door meerdere bestuurders bezocht wordt. Hierdoor schenden enkele oplossingen de CVRP *constraint* dat elke knoop behalve het depot exact één inkomende en één uitgaande boog heeft. Daarnaast zijn er ook oplossingen die de *constraint* van de maximale capaciteit van de wagens niet respecteren. Tabel 9.1 splitst de het *Id* van historische problemen op in enkele intervallen (deze zijn genummerd op datum). *CVRP* geeft per interval het aantal oplossingen weer die minstens één knoop met meerdere bestuurders bevatten. *Capaciteit* geeft per interval het aantal instanties weer waarin de capaciteitsconstraint geschonden wordt.

Id	**CVRP**	**Capaciteit**
0-19	6	3
20-39	9	5
40-59	14	4
60-79	14	3
80-99	11	3
100-119	14	3
120-139	9	2
140-159	10	0
160-179	3	0
180-200	2	0

Tabel 9.1: Inconsistenties in de dataset

Hoofdstuk 10

Experimenten

10.1 Methodologie

Onderzoekers evalueren machine learning modellen doorgaans door de dataset op te splitsen in een *training set* en een *test set*. Eerst trainen ze het model met de data in de *training set*. Vervolgens evalueren ze het model met de data in de *test set*. Deze aanpak noemt men *batch* evaluatie omdat alle test *instances* in een keer geëvalueerd worden. De dataset die hoofdstuk 9 beschrijft, koppelt echter een tijdsaspect aan elk historisch voorbeeld. De achterliggende reden hiervoor is dat het bedrijf elke dag een nieuw plan opstelt. Bijgevolg stelt elke oplossing één dag voor in de dataset, waardoor we het model kunnen evalueren door de data incrementeel te laten groeien en steeds opnieuw iets bij te leren. Deze aanpak komt volledig overeen met de filosofie dat menselijke planners uit vorige ervaringen leren en dat ze hun kennis iedere dag uitbreiden. Omdat dit ook de meest evidente manier is om een voorkeurgebaseerde planner in praktijk te gebruiken, zullen we het preferentiemodel en de voorgestelde solvers testen aan de hand van *incrementele evaluatie*. [4, 2]

Algorithm 1 Incrementele evaluatie van de dataset

Input: Dataset $H = \{(X^t, Z^t)\}$ waarbij t de volgorde van de historische voorbeelden bepaalt. In deze notatie is X^t de oplossing en Z^t de probleembeschrijving. De probleembeschrijving omvat contextvariabelen zoals de stops, het aantal voertuigen, de capaciteit, de vraag en de weekdag. Variabele M representeert het preferentiemodel. τ beduidt het begin van de *training set*, ρ het begin van de *test set* en ω het einde van de *test set*.

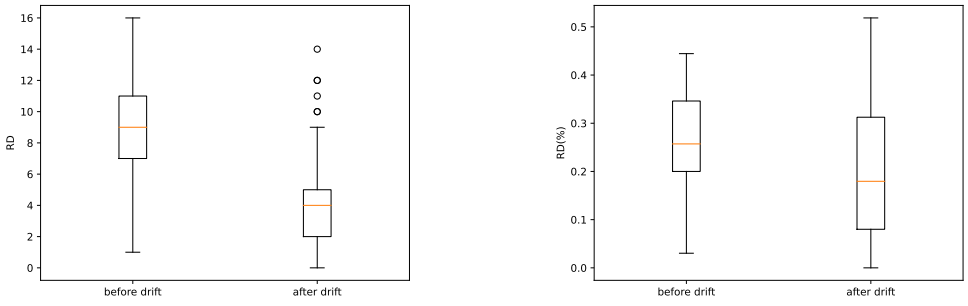
for $\sigma = \tau, ..., \rho$ **do**
    Train preference model M on instance (X^σ, Z^σ)
end for
for $\sigma = \rho + 1, ..., \omega$ **do**
    Solve problem Z^σ using preference model M
    Evaluate the solution against X^σ
    Train preference model M on instance (X^σ, Z^σ)
end for

10.2 Clustering

Deze sectie gebruikt de metriek *route difference* (RD) om de accuraatheid van het clusteralgoritme te meten de evaluatie van oplossingen. RD geeft het aantal stops weer die incorrect toegewezen zijn aan een andere cluster. Intuïtief kunnen we RD voorstellen als het aantal aanpassingen die we aan de voorspelde clusters moeten maken om de historische clusters te bekomen. Om deze metriek te berekenen, maken we een paarsgewijze vergelijking tussen de voorspelde en historische clusters. De paren clusters met het kleinste verschil in stops wordt *greedily* uitgekozen zonder vervanging. Incorrect toegewezen stops worden geteld en het totale aantal bepaalt de RD. We bekomen een percentage door RD te delen door het totaal aantal stops in het hele probleem. [2]

Deel 9.3.1 toont aan dat er een *data drift* aanwezig is rond probleemnummer 125. Daarom splitsen we bij de evaluatie van het clusteralgoritme de dataset op in een deel voor (0-125) en een deel na (125-200) *data drift*. Appendix C.1 bespreekt in detail hoe de evaluatie van het clusteralgoritme uitgevoerd werd en geeft alle resultaten getabelleerd weer.

Figuur 10.1 vat de accuraatheid van beide datasets samen in termen van absolute en procentuele RD. Deze figuur geeft aan hoe goed het clusteralgoritme voorspellingen kan maken. We merken direct op dat het model relatief goed presteert met een mediaan van het procentueel RD op 25% en 19% voor de datasets respectievelijk voor en na *data drift*. In termen van absoluut RD presteert de dataset na drift veel beter dan de dataset voor drift. Hier kunnen we twee oorzaken aan koppelen. Ten eerste zijn het aantal stops per probleem veel kleiner vanaf probleemnummer 125 (figuur 9.1a bevestigt dit). Men kan de grotere variantie in procentueel RD op de tweede helft van de dataset om dezelfde reden verklaren. Ten tweede bevatten de voorbeelden van de tweede helft van de dataset veel minder fouten tegen de CVRP *constraints*. In het deel voor *data drift* is het eerder regel dan uitzondering dat de voorbeelden de CVRP *constraints* schenden.



(A) RD (Absoluut) (B) RD (Procentueel)

Figuur 10.1: Accuraatheid van het clusteralgoritme

Tabel 10.1 geeft het aantal resultaten weer waarvan de preferentiescore van de voorspelde clusters (P) kleiner, gelijk aan of groter is dan de score van de historische clusters (H). Ter duidelijkheid beduidt de preferentiescore de objectieve waarde die het getrainde preferentiemodel aan de clusters geeft. Wanneer de score van de voorspelling groter is dan de score van de oplossing ($H < P$), bestaan er andere combinaties in de zoekruimte met een betere objectieve waarde. De historische oplossing is dan één van deze meer geprefereerde combinaties. In die gevallen is het clusteralgoritme er niet in geslaagd te convergeren naar het globale optimum. In het omgekeerde geval, wanneer de score van de predictie lager is dan de score van de oplossing ($H > P$), beslist het preferentiemodel dat de historische oplossing minder geprefereerd wordt dan de voorspelling. In die gevallen liggen de historische oplossingen zeker niet in het globaal optimum. Wanneer de score van de predictie gelijk is aan de score van de oplossing ($H = P$), heeft het algoritme een perfecte oplossing gevonden. Hoewel dit de gewenste uitkomst is, kunnen we bij deze resultaten niets zeggen over convergentie van het algoritme en het globaal optimum van het preferentiemodel.

Dataset	**H < P**	**H = P**	**H > P**
Voor drift	27	0	5
Na drift	24	2	14
Totaal	70.8%	2.8%	26.4%

TABEL 10.1: Vergelijking tussen de preferentiescore van historische clusters en voorspelde clusters

Wanneer we deze inzichten aan de resultaten toetsen, merken we direct op dat het clusteralgoritme in meer dan 70% van de gevallen convergeerde tot een suboptimale oplossing. In al deze gevallen was er een kans dat de historische oplossing op het globaal optimum van het preferentiemodel lag maar dat het het algoritme dit optimum niet vond. Dit wijst er op dat de mutatie-operator van het ILS algoritme te zwak is of dat de lokale zoekprocedure er niet in slaagt om de lokale optima te vinden omdat de waarde van de onderliggende objectieve vergelijking te veel verandert bij het maken van kleine aanpassingen aan de clusters. Wanneer het model de voorspelling boven de historische oplossing prefereert ($H > P$), is de voorspelling op zich niet slecht aangezien het RD binnen accepteerbare grenzen blijft.

Tot slot hebben dat de historische voorbeelden die de CVRP *constraints* schenden een grote invloed op de accuraatheid van het clusteralgoritme. Ten eerste wordt het model getraind op slechte data en ten tweede kunnen we niet zeker zijn of we de voorspelling vergelijken met de werkelijke toewijzing van de stops door onzekerheid bij het reconstrueren van paden (zie A.7). In tabel C.2 hebben de instanties waar de CVRP *constraints* geschonden worden een hoger procentueel RD. De significant lagere accuraatheid van het model op de eerste helft van de dataset (die veel conflicterende instanties bevat) geeft aan dat de dataset niet helemaal geschikt is voor deze taak. Gemiddeld duurt het 4.48 seconden om een het clusteralgoritme uit te voeren.

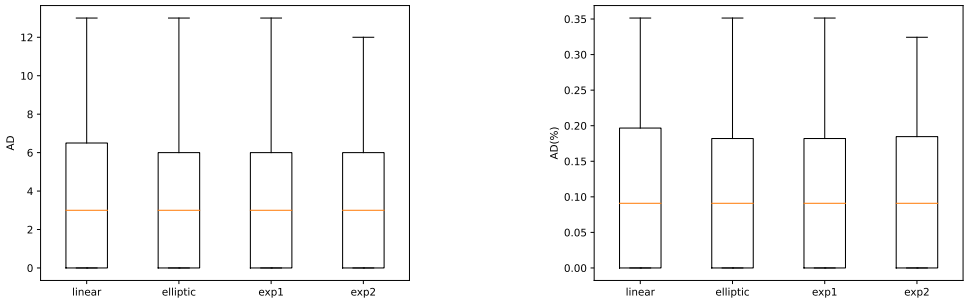
10.3 Routing

Dit deel introduceert de metriek *arc difference* (AD) om de accuraatheid van het clusteralgoritme te meten bij het evalueren van nieuwe oplossingen. AD meet het verschil tussen de bogen die wel in de historische paden en niet in de voorspelde paden genomen worden. We berekenen AD door het verschil van de verzameling van de historische bogen en de voorspelde bogen te delen door de unie van de bogen in beide oplossingen zoals in vergelijking 10.1. [2]

$$AD(A_H, A_P) = \frac{A_H - A_P}{A_H \cup A_P} \tag{10.1}$$

Secties 9.3.2 en 10.2 maken duidelijk dat de dataset in de periode voor de *data drift* veel conflicterende instanties bevat. Om een goed beeld te krijgen over de accuraatheid van het routingalgoritme, is het aangeraden om instanties te gebruiken die geen fouten bevatten. Daarom wordt het routingalgoritme enkel geëvalueerd op data uit de periode na *data drift*. Deze aanpak biedt ook de mogelijkheid om een betrouwbare comparatieve studie uit te voeren op de verschillende soorten pieken van de bilateraal dalende probabiliteit. Bij de volgende experimenten nemen we de assumptie dat de clusters volledig correct geïdentificeerd werden door het clusteralgoritme. Dit wil zeggen dat deze experimenten volledig los staan vaan de clusterexperimenten. Appendix C.2 bevat gedetailleerde informatie over de methodologie, de keuze van de parameters en de getabelleerde experimentele resultaten.

Eerst onderzoeken we de accuraatheid van het routingalgoritme door gebruik te maken van AD. We onderzoeken simultaan het effect van enkele pieken op de accuraatheid. De bestudeerde pieken zijn lineair, ellipsvormig, exponentieel ($p = 0.1$) en exponentieel$_2$ ($p = 0.01$) zoals in deel 6.2.2. Figuur 10.2 vat de resultaten samen.



(A) AD (Absoluut) (B) AD (Procentueel)

Figuur 10.2: Accuraatheid van het routingalgoritme

Uit de figuur kunnen we afleiden dat het routingalgoritme goed presteert. De mediaan van het procentuele *arc differences* ligt in alle gevallen op 10 % wat equivalent is met drie verkeerde bogen per oplossing. Daarnaast zijn er altijd meer dan één vierde

van de *test instances* volledig correct. Deze figuur toont aan dat de keuze van de piek niet veel invloed heeft op de accuraatheid van de oplossing. Het is wel duidelijk dat de exponentieel bilateraal dalende probabiliteit met de kleinste p-waarde beter presteert dan de lineaire tegenhanger. Dit verschil is echter te klein om te bestuderen. Het is zelfs mogelijk dat deze observatie te wijten is aan willekeurige randeffecten van het genetisch algoritme.

Voor het evalueren van de performantie van het genetisch algoritme en het preferentiemodel doen we opnieuw beroep op de preferentiescores van de historische en de voorspelde oplossingen. Net zoals in deel 10.3 betekent $(H < P)$ dat de het algoritme niet convergeerde naar het globale optimum, betekent $(H = P)$ dat de meest geprefereerde oplossing identiek is aan de werkelijke oplossing en betekent $(H > P)$ dat de werkelijke oplossing minder geprefereerd is dan de voorspelde oplossing. De getallen in de tabel beduiden het aantal resultaten met de met de bovenstaande eigenschap. Tabel 10.2

Vorm piek	**H < P**	**H = P**	**H > P**
Lineair	2	12	21
Ellipsvormig	1	11	23
Exponentieel ($p = 0.1$)	4	11	20
Exponentieel ($p = 0.01$)	1	14	20
Totaal	5.7%	34.3%	60%

TABEL 10.2: Vergelijking tussen de preferentiescore van historische oplossingen en voorspelde oplossingen

Deze tabel geeft veel interessante informatie. Ten eerste is het genetisch algoritme relatief goed in het vinden van het globaal optimum. We kunnen met zekerheid zeggen dat het algoritme in minstens 5.7% van de testen niet convergeerde naar het globale optimum. In de andere gevallen kunnen we niets met zekerheid vertellen over convergentie naar het globale optimum. Aangezien de zoekruimte van het routingprobleem relatief klein is, bestaat er een grote kans dat het algoritme convergeerde naar het globaal optimum. Hoewel 34.3% instanties volledig correct voorspeld werden, geeft het preferentiemodel in 60% van de gevallen een slechtere preferentiescore aan de historische oplossing dan aan de voorspelling. Dit betekent dat het preferentiemodel niet altijd in staat is is om de exacte oplossingen te vinden.

De gemiddelde tijd voor het vinden van alle paden van één probleem is 71.76 seconden met een standaarddeviatie van 48.46 seconden. Deze tijd is acceptabel voor praktisch gebruik in KMO's. Men kan het preferentie-gebaseerde routingalgoritme bijvoorbeeld gebruiken om in enkele seconden een plan op te stellen voor een bestuurder wanneer deze een aantal klanten selecteert.

We concluderen dit experiment met twee stellingen. Het het routingalgoritme vindt aan de hand van het preferentiemodel ofwel ideale oplossingen ofwel goede,

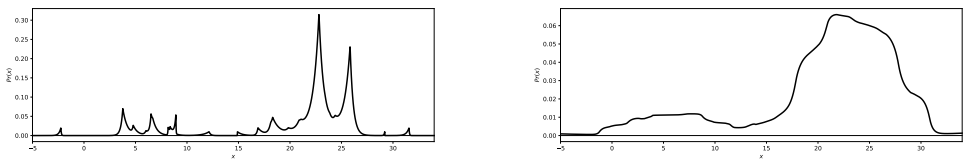
suboptimale oplossingen. Ten tweede convergeert het genetische algoritme bijna altijd tot het globaal optimum.

10.4 Discussie

Dit deel sluit het hoofdstuk over de experimenten af met een discussie van de resultaten en enkele diepgaandere inzichten in het volledige framework. Dit stuk bespreekt problemen met de dataset, de implementatie, de objectieve vergelijking, convergentie van de algoritmen en stelt voor de problemen enkele aanpassingen voor.

De fouten tegen de CVRP *constraints* in de dataset beïnvloeden de resultaten in een negatieve zin. Het preferentiemodel traint veelal op slechte voorbeelden en vergelijkt de uitkomsten soms met incorrecte oplossingen. Om verder onderzoek te doen op het voorgestelde framework, is het aangewezen de dataset op te kuisen of om een nieuwe dataset te genereren met voorbeelden die de *constraints* volledig respecteren. In een andere aanpak kan men afstand nemen van het CVRP en algoritmen ontwikkelen voor algemenere VRP varianten met het preferentiemodel als onderliggend evaluatiemechanisme.

Het clusteralgoritme slaagt er meestal niet in te convergeren naar het lokale optimum. De bron van dit probleem ligt aan twee verschillende oorzaken. Ten eerste is de mutatie-operator te zwak om uit lokale optima te ontsnappen. Ten tweede is de objectieve vergelijking te complex om de lokale structuur ervan te exploiteren in een *local search* algoritme. In ons voorbeeld heeft de objectieve vergelijking voor clustering zeven dimensies en liggen de beste objectieve waarden van elke dimensie verspreid over meerdere pieken. Dit leidt ertoe dat de objectieve vergelijking een zeer oneffen zeven-dimensioneel oppervlak heeft. Wanneer we deze observatie koppelen aan de grote zoekruimte van het clusterprobleem, kunnen we stellen dat het bijna onmogelijk is om het globale optimum te vinden in dit ruwe reliëf. Men kan dit probleem op twee verschillende manieren aanpakken. Ten eerste kan men nieuwe mutatie- en *local search*-operatoren introduceren die zich richten tot het verbeteren van specifieke attributen. Dan kan een zoekprocedure aan de hand van *round robin* de dimensies één voor één optimaliseren. De tweede aanpak is het afvlakken van de objectieve vergelijking zodat er minder lokale optima zijn. Dit kan men bereiken door elke attribuutprobabiliteit te transformeren met eender welke tweedimensionale *smoothing* techniek zoals in figuur 10.3.



(a) Origineel (b) Na smoothing

Figuur 10.3: Afvlakken van attribuutprobabiliteit

In het routingalgoritme geeft het preferentiemodel in veel gevallen een betere preferentiescore aan een suboptimale oplossing dan aan een historische oplossing. Dit kan te wijten zijn aan twee onderliggende verklaringen. Ten eerste kan de menselijke planner veel variatie in de oplossingen aanbrengen zodat de scores van de goede, suboptimale oplossingen zeer dicht bij de scores van de historische oplossingen liggen en de keuze bijgevolg een gok is tussen goede oplossingen. Een tweede verklaring vindt men bij de attributenkeuze. Het is plausibel dat er betere attributen bestaan om de voorkeuren te leren. Een interessant verder onderzoek kan de impact bestuderen van de attributenkeuze op het predictievermogen van het preferentiemodel voor verschillende VRP varianten.

Wat performantie betreft, vinden beide algoritmen een oplossing binnen aanvaardbare tijd waardoor deze algoritmen in praktische toepassingen zouden kunnen geïncorporeerd worden. Om de snelheid van de algoritmen te verhogen, is er in de code nog veel ruimte voor optimalisatie. Het parallelliseren van de berekeningen van de attribuutprobabiliteiten kan bijvoorbeeld een grote impact hebben. Specifiek voor het genetisch routingalgoritme kan elk nageslacht gevormd en geëvalueerd worden in een aparte *thread*. Een efficiënte datastructuur voor het opslaan van de gewichten bepaald door contextsimilariteit en het hergebruik van deze gewichten bij de evaluatie kan de berekening ook versnellen.

Hoofdstuk 11

Besluit

Het impliciet leren van voorkeuren uit historische trajecten is een onderwerp die niet veel bestudeerd wordt in de literatuur. Uit dit soort studies kwam men tot de conclusie dat planners meestal gelijkaardige keuzes maken bij het opstellen van nieuwe routeplannen met soortgelijke klanten. In plaats van de achterliggende voorkeuren van de planners en de bestuurders formeel neer te schrijven, probeert deze aanpak kenmerken van de trajecten te zoeken die vaak opnieuw voorkomen. Het eerste onderzoek begaf zich in de richting van transitieprobabiliteiten waarin elke boog een subjectieve probabiliteit krijgt in plaats van een objectieve kost. Op deze manier kan men de meest waarschijnlijke oplossing vinden door de totale probabiliteit van alle bogen te maximaliseren. Door het succes van de methode, volgden snel twee uitbreidende onderzoeken op dit onderwerp. Het lijkt erop dat het onderzoek naar transitieprobabiliteiten zich momenteel op een punt bevindt dat er geen methodes bestaan om het model accurater te maken. Daarom neemt dit werk een stap terug om te onderzoeken of de historische voorbeelden andere interessante eigenschappen vertonen.

Dit werk ontwikkelt een nieuw *preferentiemodel* voor het leren van voorkeuren van planners, bestuurders en klanten uit historische trajecten aan de hand van *conjoint analysis*. Naar de filosofie van *conjoint analysis* beschouwt het preferentiemodel de attributen als onafhankelijk. Het grote verschil met de traditionele *conjoint* technieken is dat het model een nieuwe nutsvergelijking gebruikt, namelijk de bilateraal dalende probabiliteit. Het model gebruikt vervolgens de geleerde voorkeuren om nieuwe oplossingen te evalueren en ze een preferentiescore toe te kennen. Hierdoor kan het model gebruikt worden in metaheuristische solvers om voor een gegeven VRP voorbeeld de meest geprefereerde oplossing te vinden. Omdat het model complexe interacties tussen knopen en bogen analyseert, is dit model incompatibel met de *state of the art* exacte MIP solvers. Het preferentiemodel is uitgerust met een eigen online trainingsalgoritme en een snel evaluatie-algoritme. Omdat het model steunt op *conjoint analysis*, moet de ontwerper altijd een aantal attributen definiëren die eigenschappen van de trajecten beduiden. Dit is de informatie waarop het model zich zal baseren bij het maken van keuzes. Het preferentiemodel is op een manier

ontworpen dat het op eender welke VRP variant toegepast worden. Dit komt ten eerste door de variabiliteit in de attributenkeuze en ten tweede doordat elke knoop een eigen conjoint model heeft. Dit betekent dat de ontwerper van het model kan bepalen wat een knoop moet leren over zijn positie in een oplossing. Het kan dan bijvoorbeeld gaan om allerlei attributen zoals *time windows*, het depot waarvan de bestuurder komt, richtingen vanwaar de bestuurders moeten komen en de totale lengte van een pad.

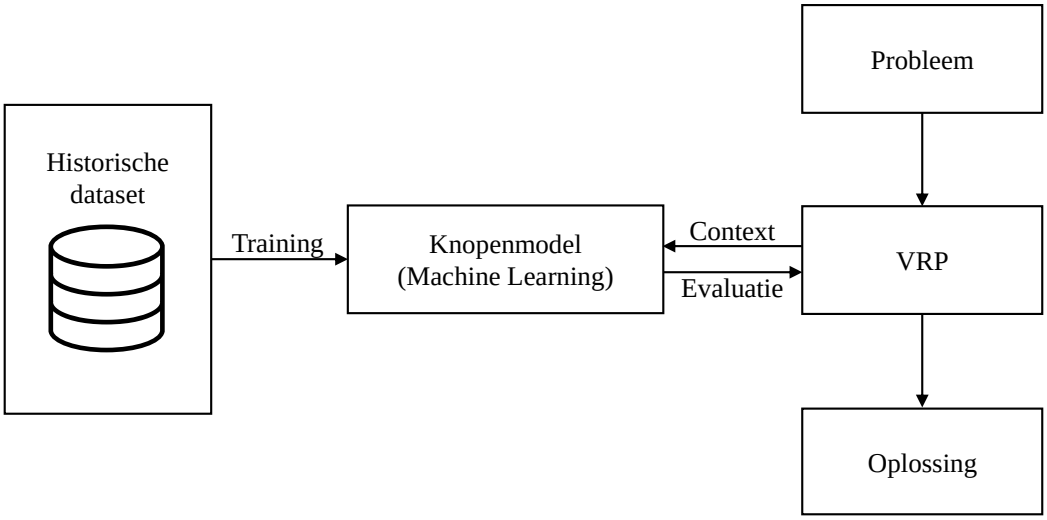
De VRP variant waar deze thesis de focus op legt, is het *capacitated vehicle routing problem*. Omdat het moeilijk is om voor dit probleem een metaheuristische solver op te stellen waarvan het objectief niet het minimaliseren is van een objectieve kost, splitsen we het probleem op in clustering en routing. Elk deelprobleem krijgt een eigen preferentiemodel met eigen zorgvuldig gekozen attributen. Clustering verdeelt de knopen onder de bestuurder aan de hand van een algoritme dat geïnspireerd is op *iterated local search*. Routing vindt het meest geprefereerde pad door al deze clusters met een genetisch algoritme. De experimenten maakten het mogelijk om de voorgestelde algoritmen te testen op accuraatheid, convergentie en performantie. Daarnaast gaven de experimentele resultaten ook inzicht in de onderliggende werking van de metaheuristische algoritmen en de complexiteit van de objectieve vergelijking van het preferentiemodel.

Voor clustering concludeerden we dat de objectieve vergelijking te veel lokale optima bevat en te sterk afhankelijk is van interacties tussen de knopen om de lokale structuur te kunnen exploiteren in een lokale zoekprocedure. Kleine aanpassingen in de oplossingen kunnen leiden tot grote veranderingen in de preferentiescore wat een ongewenst effect is voor dit soort algoritmen. Daarom is het huidige clusteralgoritme niet in staat om te convergeren tot het globale optimum. Het is wel zo dat de historische oplossing in vele gevallen nog een betere preferentiescore heeft dan uitkomst van het algoritme. Hoewel clustering veel problemen met zich meebrengt, zijn de resultaten in termen van accuraatheid en performantie goed. Om deze reden is verder onderzoek op het clusteralgoritme, de attributenkeuze en afvlakking van de objectieve vergelijking voor clustering zeker aangewezen.

Het routingprobleem faalt niet bij het convergeren naar het globale optimum omwille van twee redenen. Ten eerste is de zoekruimte veel kleiner en ten tweede heeft een genetisch algoritme veel meer kandidaat-oplossingen dan *iterated local search* en is de zoektocht in de ruimte van alle mogelijke combinaties veel verspreider. Hoewel het routingalgoritme erin slaagt te convergeren, komt er hier een nieuw probleem aan het licht. Het is vaak zo dat goede oplossingen die minder dan 10% verschillen met de historische oplossingen een betere preferentiescore krijgen. Dit kan te wijten zijn aan de keuze van de attributen, willekeur in de keuze van de planner of *overfitting* van het preferentiemodel. Daarnaast zagen we dat de vorm van de bilateraal dalende probabiliteit niet veel invloed had op de accuraatheid van het routingalgoritme.

Het grootste probleem waar deze techniek echter mee kampt, is dat *conjoint analysis*

de assumptie neemt dat attributen onafhankelijk zijn van elkaar. We illustreren dit met een eenvoudig voorbeeld. Beschouw het aantal afgeleverde producten voor en na bezoek aan een knoop als attributen. Wanneer het aantal afgeleverde producten voor het bezoek bijna gelijk is aan de capaciteit van de wagen, zullen er weinig knopen liggen in het tweede deel van het pad. Men kan een soortgelijke analyse maken voor de totale afstand voor en na bezoek. Het is in principe bijna onmogelijk om attributen van trajecten te definiëren die werkelijk onafhankelijk zijn van elkaar. Om deze redenen zou het een zeer interessante studie zijn om het *conjoint model* te vervangen door traditionele *machine learning* technieken die binaire classificatie kunnen uitvoeren op de attribuutwaarden. De techniek moet in staat zijn om goede van slechte oplossingen te onderscheiden en daar een probabiliteit aan koppelen. Deze probabiliteiten kan men vervolgens gebruiken in het knopenmodel om de preferentiescore voor een oplossing te berekenen. Efficiënte perturbatiemechanismen moeten negatief gelabelde training data genereren uit de historische trajecten. Enkele voorbeelden zijn logistische regressie, *decision trees* en *support vector machines*. Figuur 11.1 geeft het algemeen framework voor dit soort *vehicle routing* systemen. Dit framework kan het beginpunt zijn van enorm veel nieuw onderzoek door de keuzevrijheid aan attributen, perturbatiemechanismen, contextrepresentaties, *machine learning* technieken en VRP varianten.



Figuur 11.1: Algemeen framework

Bijlagen

Bijlage A

Algoritmen

A.1 Datageneratie

Algorithm 2 Genereren van een dataset voor een knoop uit historische trajecten

```
function GENEREER DATASET(H, k)                    ▷ historische trajecten H, knoop k
    dataset ← list()
    for X ∈ H do                                   ▷ historische oplossing X
        for P ∈ X do                               ▷ pad P
            if k ∈ P then
                dataset.add_labeled_record(attributes(X, P, k), True)
                P'_set ← perturb(X, P, k)          ▷ geperturbeerde paden P'_set
                for P' ∈ P'_set do
                    dataset.add_labeled_record(attributes(X, P', k), False)
                end for
            end if
        end for
    end for
    return dataset
end function
```

Dit algoritme zoekt in de gegeven verzameling historische trajecten naar alle paden waarin de gegeven knoop voorkomt. De attribuutwaarden van deze paden worden eerst positief gelabeld. Vervolgens maakt het algoritme een set met geperturbeerde oplossingen in de dichte omgeving van het originele pad. De attribuutwaarden van de geperturbeerde oplossingen worden in de dataset opgeslagen onder het negatieve label. De exacte perturbatiemethode en de attributen verschillen bij de clustering en routing deelproblemen en worden respectievelijk besproken in secties 7.1 en 8.1.

A.2 Training van het preferentiemodel

Algorithm 3 On-line training van het preferentiemodel

```
function ADD SOLUTION(X, M)   ▷ historische oplossing X, preferentiemodel M
    for P ∈ X do                                                    ▷ pad P
        for k ∈ P do                                                ▷ knoop k
            M_k ← M.get_node_model(k)
            if M_k = None then
                M_k ← M.create_empty_node_model(k)
            end if
            P'_set ← perturb(X, P, k)                  ▷ geperturbeerde paden P'_set
            for i ∈ M_k.nb_attributes do
                Pr_ki ← M_k.get_attribute_probability_model(i)
                Pr_ki ← Pr_ki.add_component(X, P, P'_set, k)
            end for
        end for
    end for
    return M
end function
```

Het trainingsalgoritme toont aan hoe men preferentiemodel moet trainen. Het model is opgebouwd zodat het online training toelaat. Hierdoor kunnen we nieuwe voorbeelden aan het model toevoegen zonder de parameters steeds opnieuw te moeten schatten. De volgende puntjes geven een overzicht van de datastructuren en de functies die terug te vinden zijn in het algoritme.

- De variabele X is een oplossing, deze oplossing bestaat uit meerdere onderdelen. Ten eerste bevat een oplossing een aantal paden. Een pad P wordt gekenmerkt door een sequentie knopen $k_0 \rightarrow k_1 \rightarrow k_2 \rightarrow ...k_n \rightarrow k_0$. Merk op dat het een pad steeds start en eindigt in het depot (knoop 0). Daarnaast heeft elke oplossing ook een context waar het probleem zich in afspeelde.

- Het preferentiemodel dat we trainen wordt aangeduid door de variabele M. Dit model is opgebouwd uit één *conjoint model* M_k per knoop k. Voor elke knoop die werd gezien in de eerdere trainingsdata, bestaat er reeds een model. Wanneer er in de oplossing die we aan het model willen toevoegen een knoop aanwezig die nooit eerder gezien in het preferentiemodel, moet moet er een nieuw knopenmodel gemaakt worden met de functie *create_empty_node_model*(k). Deze functie initialiseert voor elke attribuut een leeg attribuutprobabiliteitsmodel Pr_{ki} (i.e. zonder componenten). Hier nemen we de assumptie dat het lege knoopmodel een uniform gewicht w_{ki} geeft aan de attributen tenzij dit anders gespecificeerd is in de code.

- Elk knoopmodel M_k bestaat uit een aantal attribuutprobabiliteitsmodellen die we aanduiden met Pr_{ki}. Voor elk attribuut is er exact één attribuutprobabili-

teitsmodel. Deze modellen zijn opgebouwd uit een aantal bilateraal dalende probabilitscomponenten die samen opgeslagen zijn met hun context.

- De functie $perturb(X, P, k)$ perturbeert het gekozen pad P in oplossing X waarbij rekening gehouden wordt met knoop k. De variabele P'_{set} slaat de geperturbeerde oplossingen op. Het perturbatiemechanisme hangt af van het probleem waarin het preferentiemodel gebruikt wordt en van de keuze van de attributen.

- De functie $add\_component(X, P, P'_{set}, k)$ heeft de taak om een nieuwe component aan het attribuutprobabiliteitsmodel Pr_{ki} toe te voegen. Deze stap berekent de eerst de attribuutwaarde van zowel het oorspronkelijk pad als de geperturbeerde paden en bepaalt vervolgens de parameters x_{opt}, x_{min} en x_{max} in de formule van de bilateraal dalende probabiliteit f_{kij} zoals in vergelijkingen 6.8, 6.9 en 6.10. Een tweede effect van deze functie is de context van de historische oplossing aan de component gekoppeld wordt. Dit is noodzakelijk om het gewicht te bepalen bij de evaluatie van het model in algoritme A.3.

A.3 Evaluatie van het preferentiemodel

Algorithm 4 Evaluatie van het preferentiemodel

```
function EVALUATE(X, M)                          ▷ oplossing X, preferentiemodel M
    node_scores ← list()
    for P ∈ X do                                                          ▷ pad P
        for k ∈ P do                                                      ▷ knoop k
            M_k ← M.get_node_model(k)
            if M_k = None then
                continue                                  ▷ geen training data voor knoop
            else
                attributes ← get_attributes(X, P, k)
                node_score ← 0
                for i ∈ M_k.nb_attributes do
                    Pr_ki ← M_k.get_attribute_probability_model(i)
                    w_ki ← M_k.get_attribute_weight(i)
                    w_kij ← list()
                    f_kij ← list()
                    for j ∈ Pr_ki.nb_components do
                        w_kij.add(Pr_ki.get_component_weight(j, X.get_context))
                        f_kij.add(Pr_ki.get_component_probability(j, attributes[i]))
                    end for
                    w_kij ← rescale(w_kij)                        ▷ moet sommeren tot 1
                    node_score ← node_score + w_ki · dot_product(w_kij, f_kij)
                end for
                node_scores.add(node_score)
            end if
        end for
    end for
    return sum(−log(node_scores))
end function
```

Bovenstaand algoritme geeft in pseudocode de manier waarop een oplossing een geëvalueerd wordt aan de hand van het preferentiemodel. Het algoritme gebruikt dezelfde datastructuren en naamgeving als het trainingsalgoritme A.2. Hier zijn er nog enkele particulariteiten die we moeten vermelden.

- Het algoritme is volledig gebaseerd op de objectieffunctie van het preferentiemodel in 6.11 en bepaalt dus een metriek voor de waarschijnlijkheid van een oplossing X.
- De functie $get\_attributes(X, P, k)$ berekent de attribuutwaarden voor knoop k in pad P in oplossing X. De attributen zijn gerangschikt volgens index i en worden opgeslagen in een lijst.

- De functie $get\_component\_weight(j, X.get\_context)$ berekent het gewicht van de j'de component aan de hand van de context van de nieuwe oplossing. Dit is mogelijk omdat het trainingsalgoritme ervoor zorgt dat elke component aan een context gekoppeld is. Het gewicht kan bijvoorbeeld bepaald worden door de Jaccard-similariteit tussen de knopenset van de historische oplossing en de nieuwe oplossing. De verkregen gewichten worden opgeslagen in de lijst w_{kij} omdat deze later nog herschaald moeten worden om te sommeren tot één. De gewichten kunnen enkel positieve waarden aannemen.

- De functie $get\_component\_probability(j, attributes[i]))$ berekent de bilateraal dalende probabiliteit van de j'de component door de i'de attribuutwaarde in te vullen in de vergelijking. Hiervoor werden de parameters x_{opt}, x_{min} en x_{max} bepaald in de trainingsstap. De uitvoerwaarden van deze functie worden opgeslagen in de lijst f_{kij}.

- Om het nut van de functie $rescale(w_{kij})$ te begrijpen, kijken we terug naar vergelijking 6.7. Deze vergelijking stelt de voorwaarde dat de gewichten positief zijn en sommeren tot één. De herschalende functie brengt de voorwaarde in orde door de elementen van de gewichtenlijst te transformeren aan de hand van formule A.1.

- Wanneer het algoritme voor een bepaalde knoop in de oplossing alle componenten van een attribuut heeft overlopen en wanneer de gewichten herschaald zijn, kan de attribuutprobabiliteit worden berekend aan de hand van vergelijking 6.7. Vervolgens wordt deze attribuutprobabiliteit vermenigvuldigd met het gewicht van het attribuut en opgeteld bij de additieve score voor de knoop zoals in vergelijking 6.6. De probabiliteit voor de knoop in de oplossing wordt vervolgens opgeslagen in de lijst $node\_scores$.

- De evaluatiescore van een oplossing X wordt bepaald door de som van het negatieve logaritme van de knoopprobabiliteiten. Deze probabiliteiten zijn na afloop van alle lussen terug te vinden in de lijst $node\_scores$.

$$w_i \leftarrow \frac{w_i}{\sum\limits_{w_j \in list} w_j} \tag{A.1}$$

A.4 Perturbatiemechanisme voor clustering

Algorithm 5 Perturbatiemechanisme voor clusters

```
function PERTURB(X, P, k_i)                    ▷ oplossing X, pad P, knoop k_i
    C ← get_cluster(P)                          ▷ zet pad om in cluster
    V ← get_stops(X)                            ▷ alle stops in probleem X
    perturbations ← list()
    for k ∈ V \ C do:
        perturbations.append(C ∪ {k})
    end for
    for k ∈ C \ {k_i} do:
        perturbations.append(C ∩ {k})
    end for
    return perturbations
end function
```

De functie *get_cluster* zet eerst het gegeven pad P om in de corresponderende cluster C. Daarna worden er twee soorten perturbaties op deze cluster uitgevoerd. De eerste perturbatie voegt telkens één knoop toe aan C, die tot een ander pad dan P behoort in X. De tweede perturbatie verwijdert telkens één knoop uit C, behalve knoop k_i waarvoor de attributen berekend moeten worden. Deze twee mechanismen zorgen ervoor dat de gegenereerde clusters exact één knoop verschillen met de historische clusters waardoor de gegenereerde clusters goed lijken op de historische oplossing.

A.5 Algoritme voor clustering

Algorithm 6 Algoritme voor clustering gebaseerd op *iterated local search*

Input: Knopen V, historische oplossingen H, het preferentiemodel voor clustering M en de probleemcontext Z

```
function FIND BEST CLUSTERS(V, H, M, Z)
    C* ← initialize(V, H)
    C* ← local_search(C*)
    while not converged(C*) do
        C' ← mutate(C*)
        C' ← local_search(C')
        if M.evaluate(C', Z) < M.evaluate(C*, Z) then
            C* ← C'
        end if
    end while
    return clusters
end function
```

- De functie *initialize* genereert de eerste oplossing. Het algoritme bekomt deze oplossing door eerst de CVRP oplossing met de transitieprobabiliteiten te berekenen en daaruit de clusters te halen.
- De functie *mutate* introduceert een perturbatie in twee willekeurig gekozen clusters van de oplossing. De perturbatie wisselt twee stops in de eerste cluster met één stop in de tweede cluster uit. Het is dus een $'twee - voor - één'$ uitwisseling. Dit perturbatiemechanisme introduceert redelijk wat variatie waardoor het algoritme uit de lokale optima zou moeten kunnen geraken.
- De functie *local_search* voert een lokale zoekprocedure uit op de oplossing. De lokale zoekprocedure mag enkel operaties gebruiken die de perturbatie niet kunnen annuleren. Anders zou deze procedure meestal terugvallen op het lokaal optimum waar we ons eerder in bevonden. De lokale zoekprocedure voert uitwisseling van één knoop tussen twee clusters uit die de beste objectieve waarde geeft.

A.6 Algoritme voor routing

Algorithm 7 Genetisch algoritme voor het vinden van het beste pad door een gegeven cluster

Input: Cluster C, het preferentiemodel voor routing M en de probleemcontext Z

function FIND BEST PATH(C, M, Z)
  $population \leftarrow initialize\_population(C)$
  while not $converged(population)$ **do**
    $selected\_parents \leftarrow k\_tournament(population)$
    $children \leftarrow PMX\_crossover(selected\_parents)$
    $children \leftarrow swap\_mutation(children)$
    $population \leftarrow (\lambda + \mu)\_elimination(population, children, M, Z)$
  end while
  return $get\_best\_individual(population)$
end function

- De populatiegrootte λ is gelimiteerd tot 25 individuen. Elk individu is een pad door alle knopen van de gegeven cluster C. In $initialize\_population(C)$ wordt de populatie geïnitialiseerd door 25 willekeurige permutaties van de elementen van C te genereren. De willekeur bij het genereren van de individuen zorgt voor voldoende diversiteit binnen de populatie.

- Het genetische algoritme vervangt de populatie een aantal keer door een nieuwe generatie totdat het convergentiecriterium bereikt is. Een nieuwe generatie wordt bepaald door de stappen selectie, kruising, mutatie en eliminatie sequentieel uit te voeren. Bij het opstellen van een nieuwe generatie is het aantal nakomelingen μ altijd 50.

- De *k-tournament*-operator wordt gebruikt om 25 paren uit de populatie te selecteren. Het selectieproces bestaat uit twee stappen. De operator kiest uit drie willekeurig geselecteerde individuen uit de populatie het individu met de beste objectieve score. Hierdoor verloopt de selectie enerzijds willekeurig. Anderzijds probeert *k-tournament* ook kwaliteitsvolle ouders te selecteren.

- De geselecteerde paren vormen in de kruisingsstap elk twee nakomelingen door *Partially Mapped Crossover* (PMX) uit te voeren. Het toepassen van kruising resulteert totaal in 50 nakomelingen. Deze operator selecteert eerst twee willekeurige indices in het pad van beide ouders. Om een nakomeling te maken, vervangt het stuk pad tussen deze indices van de ene ouder het overeenkomstige stuk pad van de andere ouder. Vervolgens worden enkele knopen buiten het gekopieerde deel aangepast om gedupliceerde knopen te elimineren en zo een nieuw pad te bekomen die alle knopen van de cluster bevat. De kruissingsstap heeft het doel om de goede eigenschappen van beide ouders door te geven aan het nageslacht. [12, 8]

- Mutatie wordt op elke nakomeling met een kans van 30% toegepast en heeft het doel om variatie in de populatie te brengen. Hierdoor bekijkt het algoritme een groter deel van de zoekruimte. De mutatie-operator verwisselt twee willekeurige knopen in het pad.
- In de eliminatiestap wordt de populatie vervangen door een nieuwe generatie. De $(\lambda + \mu)$-operator houdt uit de verzameling van de originele populatie en de nakomelingen samen de 25 individuen met de beste objectieve score over .
- De functie $converged(population)$ bepaalt of het algoritme geconvergeerd is. Convergentie wordt bereikt wanneer alle individuen in de populatie identiek zijn of wanneer de gemiddelde objectieve waarde van de populatie voor drie iteraties lang onveranderd is gebleven.
- Alle individuen krijgen een objectieve score die bepaald wordt door het preferentiemodel.

A.7 Padreconstructie

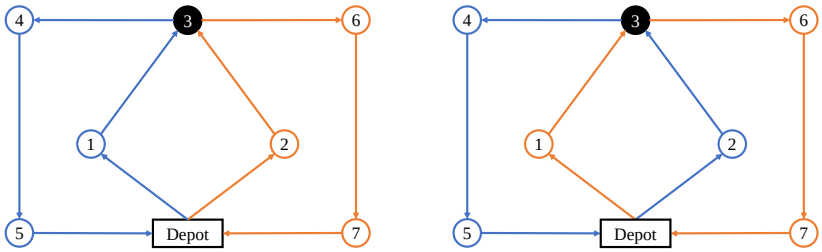
Algorithm 8 Padreconstructie uit een grafe

```
function FIND PATHS(G, D, c, n) ▷ grafe G, vraag D, capaciteit c, bestuurders n
    P_c ← {[0]}                                              ▷ paden in constructie
    P ← {}                                                   ▷ geldige paden
    while P_c ≠ ∅ do
        for p ∈ P_c do
            P_c ← P_c \ {p}
            k ← get_last_node(p)                             ▷ laatste knoop in p
            K' ← get_next_nodes(G, k)      ▷ knopen in K' bereikt vanuit k in G
            for k' ∈ K' do
                p' ← p.add(k')                               ▷ voeg k toe op het eind van p
                if k' = 0 then
                    P ← P ∪ {p'}                             ▷ voeg p' toe aan P'
                else
                    if is_valid_path(p', c, D) then          ▷ check geldigheid van p'
                        P_c ← P_c ∪ {p'}
                    end if
                end if
            end for
        end for
    end while
    P ← filter(P, n)                                         ▷ houd n paden over
    return P
end function
```

Om de paden te bepalen uit de verbindingsmatrix, werd het bovenstaande algoritme ontwikkeld. Het algoritme neemt als argument de grafe op met alle knopen en bogen in de verbindingsmatrix. Dit zijn enkele belangrijke algoritmische details.

- Het algoritme werkt constructief en voegt in elke iteratie één knoop toe aan alle paden in constructie. De knoop die wordt toegevoegd moet direct bereikbaar zijn vanuit het laatste punt van het pad in constructie. Het startpunt van alle paden is het depot (knoop 0).
- De functie *is_valid_path* checkt de geldigheid van de geconstrueerde paden. In geldige paden komt een knoop maximaal éénmaal voor. De gesommeerde vraag van de knopen in een pad mag de de capaciteit niet overschrijden.
- Omdat een knoop slechts éénmaal kan voorkomen in een pad in constructie zijn er nooit lussen en loopt het algoritme gegarandeerd ten einde.
- Een pad begint en eindigt in het depot. Wanneer de volgende knoop van een pad in constructie k' het depot is, dan is het pad compleet en geldig.

- Na de *while* lus is het mogelijk dat het aantal geldige paden $\#(P)$ groter is dan het aantal bestuurders n. De functie $filter$ heeft de taak om n paden over te houden. Deze functie moet ervoor zorgen dat alle gekozen bogen in de verbindingsmatrix ook in de overblijvende paden zitten. Deze stap kan paden behouden die in werkelijkheid niet genomen werden. Hierdoor bestaat de mogelijkheid dat het preferentiemodel getraind wordt op verkeerde data en bij gevolg minder accuraat wordt. Figuur A.1 toont dit probleem grafisch met een voorbeeld waarin het algoritme de werkelijke paden niet met zekerheid kan bepalen.



(A) Optie 1 (B) Optie 2

FIGUUR A.1: Probleemgeval voor padreconstructie

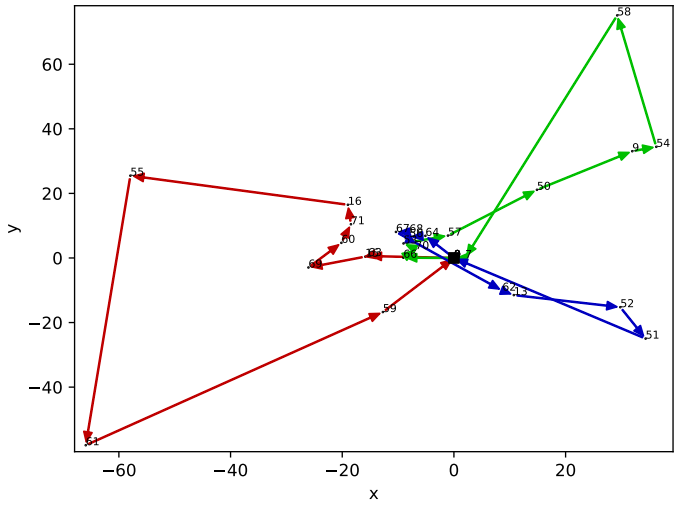
Bijlage B

Historisch voorbeeld

Tabel B.1 geeft een historisch voorbeeld uit de dataset. Het probleem heeft drie bestuurders, een capaciteit van dertien eenheden per wagen en speelt zich af op een zondag. Er zijn in totaal 27 klanten, elk met hun eigen vraag. De oplossing bestaat uit drie paden die rondom het centraal depot gelegen zijn. In de oplossing heeft elke bestuurder duidelijk een eigen regio. In dit probleem kan men opmerken dat de dataset niet volledig foutloos is. De oplossing laat immers toe dat de capaciteit van de wagens overschreden wordt. Deze constraint moet in enkele experimenten dus gerelaxeerd worden door de capaciteit bijvoorbeeld te verhogen.

Probleem 100	
Aantal bestuurders : 3	
Capaciteit : 13	
Weekdag : 0 (Zondag)	
Knoop	**Vraag**
0	0
7	1
9	1
13	1
15	1
16	1
50	2
51	2
52	2
53	2
54	2
55	2
56	4
57	2
58	2
59	2
60	4
61	2
62	2
63	3
64	2
66	2
67	2
68	3
69	2
70	2
71	4

Oplossing



Tabel B.1: Historisch voorbeeld

Bijlage C

Experimentele resultaten

C.1 Clustering

Alle code is geschreven voor Python 3.9.11. Voor het MIP probleem werd de Gurobi 9.5.1 *solver* gebruikt op de standaard instelling. De experimenten voor clustering werden uitgevoerd op een ASUS Vivobook Pro X580VD systeem met een Intel Core i7-7700HQ processor met een kloksnelheid van 2.80GHz en 8GB RAM-geheugen.

Bij het genereren van de initiële clusters werd bij het berekenen van de transitieprobabiliteitsmatrix *Laplace smoothing* toegepast met $\lambda = 0.0001$ en werden de voorbeelden exponentieel gewogen met $\alpha = 0.7$ [2]. Voor de evaluatie van de oplossingen gebruikt het preferentiemodel een uniform gewicht voor elke attribuutprobabiliteit en een exponentiële piekvorm met $p = 0.1$. De lokale clusters in het preferentiemodel bevatten drie knopen.

Legende:

Naam	**Betekenis**
Id	Probleemnummer van het historisch voorbeeld (geordend volgens datum)
V_{CVRP}	De historische oplossing schendt de CVRP constraints. Er zijn meerdere bestuurders die in dezelfde knoop passeren.
V_Q	De historische oplossing schendt de CVRP constraints. Er zijn bestuurders met een grotere lading dan de toegelaten capaciteit.
$HIST_{score}$	De score van het preferentiemodel voor de historische oplossing
$PRED_{score}$	De score van het preferentiemodel van de voorspelling
RD	Absoluut *route difference*
$RD(\%)$	Procentueel *route difference*
$Time(s)$	Predictietijd in seconden
τ	Id begin training set
ρ	Id einde training set en begin test set
ω	Id einde test set

Id	V_{CVRP}	V_Q	$HIST_{score}$	$PRED_{score}$	RD	$RD(\%)$	$Time(s)$
91	1	0	34.8164	35.9167	7	0.179487	4.98105
92	1	0	42.5496	39.4338	9	0.257143	4.19477
93	0	1	30.7837	39.7857	9	0.346154	9.10367
94	0	0	38.4777	33.9475	1	0.030303	5.18512
95	1	0	53.5043	64.6924	11	0.305556	3.81584
96	1	0	63.7749	53.6715	11	0.297297	3.11268
97	1	0	44.2379	69.5767	16	0.444444	3.46669
98	1	0	34.9684	55.6916	13	0.351351	4.57778
99	1	0	39.5132	40.6565	7	0.2	2.15427
100	0	1	34.9267	39.6589	9	0.346154	15.4009
101	0	0	38.3036	40.6284	12	0.352941	6.75095
102							
103	1	0	54.3503	51.202	9	0.243243	3.13362
104	1	0	53.8775	65.6034	11	0.289474	3.86768
105	1	0	44.4285	54.5179	13	0.333333	7.08808
106	1	0	34.7807	47.0412	7	0.2	4.71242
107	0	1	33.4084	39.5477	9	0.346154	24.7948
108	0	0	38.0869	46.7473	11	0.323529	7.11601
109	1	0	50.3263	59.9916	14	0.378378	4.25762
110	1	0	43.3058	47.2853	8	0.216216	4.15391
111	0	0	35.9405	29.7881	7	0.212121	5.95109
112							
113	1	0	40.9731	56.0219	9	0.243243	4.54884
114	1	0	40.5082	47.3801	9	0.257143	3.04388
115	1	0	33.8295	45.8232	7	0.2	3.91955
116	1	0	32.1708	42.9292	7	0.2	2.85537
117	0	1	32.6444	38.344	5	0.192308	20.1841
118	1	0	38.0307	46.7497	7	0.205882	2.77157
119							
120	1	0	37.1618	49.1006	9	0.257143	3.38791
121	1	0	34.4865	39.3367	7	0.189189	5.49232
122	1	0	32.8633	36.3249	6	0.176471	4.04321
123	0	1	31.8824	33.3812	9	0.346154	20.7874
124	0	0	33.1858	44.2783	12	0.375	8.60201
125	1	0	32.854	54.0115	13	0.393939	3.47974

Tabel C.1: Resultaten van clustering voor *data drift* ($\tau = 0, \rho = 90, \omega = 125$)

Id	V_{CVRP}	V_Q	$HIST_{score}$	$PRED_{score}$	RD	$RD(\%)$	$Time(s)$
161	0	0	24.1711	24.4408	4	0.173913	2.1089
162	0	0	36.9756	46.608	11	0.407407	0.819797
163	0	0	35.1307	33.2223	4	0.148148	1.14692
164	0	0	32.105	32.105	0	0	1.42616
165	0	0	18.4663	32.8204	5	0.3125	4.87101
166	0	0	29.5114	34.7213	10	0.4	1.81019
167	0	0	37.5038	28.0692	9	0.36	1.81918
168	1	0	32.6144	35.6209	5	0.2	1.49999
169	0	0	27.739	27.739	0	0	1.24467
170	0	0	25.503	30.1655	1	0.037037	1.50596
171	0	0	24.0569	24.0569	0	0	1.26858
172	0	0	18.028	18.8043	4	0.25	5.56911
173	1	0	41.8865	46.7518	12	0.413793	2.50732
174	0	0	30.4604	31.5418	9	0.36	1.60871
175	1	0	30.0828	35.2166	5	0.185185	1.0841
176	0	0	25.1766	22.3545	2	0.0833333	1.11801
177	0	0	31.6605	28.6953	5	0.185185	1.79323
178	0	0	20.5178	25.3837	2	0.08	1.2457
179	0	0	17.7822	23.565	5	0.3125	6.24035
180	1	0	50.6818	41.4135	12	0.4	2.84138
181	0	0	27.5065	26.3877	5	0.217391	1.72738
182	0	0	32.714	31.7972	1	0.04	1.7194
183	0	0	28.7747	25.1397	1	0.0434783	2.13233
184	0	0	33.2093	39.9348	3	0.115385	2.46142
185	0	0	26.7794	27.9677	1	0.04	1.49398
186	0	0	17.2383	20.4233	4	0.25	7.07509
187	0	0	33.113	40.3474	14	0.518519	2.78352
188	0	0	23.4134	26.9878	4	0.173913	2.40856
189	0	0	33.44	31.3061	3	0.111111	2.11136
190	0	0	27.9007	25.6941	2	0.0833333	2.12033
191	0	0	30.1263	29.4097	1	0.0384615	2.19414
192	0	0	26.8192	25.601	2	0.08	1.87499
193	0	0	16.9571	20.1854	4	0.25	8.54021
194	1	0	42.4763	50.1752	10	0.333333	3.78591
195	0	0	30.0348	32.0607	8	0.32	2.63496
196	0	0	27.0611	26.931	2	0.0769231	3.17849
197	0	0	24.5953	24.5715	3	0.125	2.50234
198	0	0	31.429	34.3596	5	0.185185	2.90423
199	0	0	24.099	25.5596	3	0.12	4.59971
200	0	0	16.7562	20.2021	4	0.25	9.5794

TABEL C.2: Resultaten van clustering na *data drift* ($\tau = 125, \rho = 160, \omega = 200$)

C.2 Routing

Alle code is geschreven voor Python 3.9.11. De experimenten voor routing werden uitgevoerd op een Departementale KU Leuven Computer met een Intel Core i5-7500 processor met een kloksnelheid van 3.40GHz en 8GB RAM-geheugen. Instanties 130-165 werden gebruikt als trainingdata en vervolgens werden instanties 165-200 incrementeel geëvalueerd.

Het routingalgoritme neemt als input de clusters die corresponderen met de paden van de historische oplossing. Vervolgens zoekt het routingalgoritme voor elke historische cluster het meest geprefereerde pad. Deze testen staan dus volledig los van de clustertesten. Alle oplossing werden gezocht aan de hand van het genetisch algoritme uit A.6 en het preferentiemodel voor clustering uit hoofdstuk 8. Onderstaande tabellen C.3, C.4, C.5 en C.6 geven de scores van het preferentiemodel voor de historische oplossingen en de best gevonden oplossingen, de accuraatheid gemeten aan de hand van *arc difference* en de tijd die het algoritme nodig had om de oplossing te vinden. Elke tabel werd gegenereerd met een andere soort piek in de bilateraal dalende probabiliteitscomponenten van het preferentiemodel. De gekozen pieken zijn respectievelijk lineair, ellipsvormig, exponentieel met $p = 0.1$ en exponentieel met $p = 0.01$. Bij het bepalen van de attribuutwaarden werd de lokale omgeving gedefinieerd als het segment dat tussen twee knopen voor en twee knopen na de stop ligt.

Legende:

Naam	**Betekenis**
Id	Probleemnummer van het historisch voorbeeld (geordend volgens datum)
$HIST_{score}$	De score van het preferentiemodel voor de historische oplossing
$PRED_{score}$	De score van het preferentiemodel van de voorspelling
AD	Absoluut *arc difference*
$AD(\%)$	Procentueel *arc difference*
$Time(s)$	Predictietijd in seconden
τ	Id begin training set
ρ	Id einde training set en begin test set
ω	Id einde test set

Id	$HIST_{score}$	$PRED_{score}$	AD	$AD(\%)$	$Time(s)$
165	14.2801	14.2801	0	0	60.5532
166	45.798	36.8421	7	0.21875	10.1477
167	48.7385	39.9955	7	0.21875	42.4084
168	43.2715	37.1359	3	0.0909091	36.7761
169	98.5367	98.5367	0	0	15.0903
170	35.6487	35.6487	0	0	16.8982
171	33.8186	33.0576	3	0.0909091	12.1341
172	13.6611	13.6611	0	0	95.5325
173	53.871	46.5888	13	0.351351	67.2473
174	36.5326	34.3452	7	0.21875	49.1945
175	37.5029	35.8771	3	0.0833333	55.413
176	32.3348	32.3348	0	0	15.2852
177	37.6838	33.2833	3	0.0882353	57.7108
178	31.2636	30.2571	3	0.0909091	22.3655
179	13.3386	13.3386	0	0	99.0438
180	60.8756	48.4356	11	0.289474	81.5279
181	39.0449	32.5902	7	0.241379	45.0445
182	41.3625	39.4251	6	0.1875	147.013
183	36.1326	34.146	3	0.103448	62.0295
184	36.326	33.3155	7	0.21875	96.4893
185	39.1725	38.5606	3	0.0909091	40.3265
186	13.3718	20.237	3	0.166667	129.99
187	41.2416	37.6103	7	0.205882	27.9459
188	32.6013	31.65	3	0.103448	48.3051
189	42.0524	41.4855	3	0.0882353	120.078
190	33.9177	33.9177	0	0	94.6386
191	32.1185	31.8162	3	0.0909091	75.9869
192	35.0553	35.0553	0	0	48.5163
193	13.4922	13.4922	0	0	175.276
194	54.3553	47.929	10	0.263158	132.249
195	33.8108	31.966	3	0.09375	92.6756
196	38.615	38.615	0	0	78.3465
197	30.3731	30.3731	0	0	76.8063
198	32.4618	33.9261	6	0.176471	122.431
199	32.4237	32.4237	0	0	75.1955

TABEL C.3: Resultaten van routing ($lineair, \tau = 130, \rho = 165, \omega = 200$)

Id	$HIST_{score}$	$PRED_{score}$	AD	$AD(\%)$	$Time(s)$
165	13.898	13.898	0	0	97.3273
166	44.6613	35.9605	7	0.21875	13.0977
167	46.63	38.5961	7	0.21875	55.0473
168	41.4708	35.7469	3	0.0909091	31.0375
169	97.6887	97.6758	3	0.0909091	11.381
170	34.5954	34.5954	0	0	21.6866
171	32.8757	32.1851	3	0.0909091	12.7721
172	13.2788	13.2788	0	0	124.429
173	52.6124	45.1747	13	0.351351	78.3602
174	35.6353	33.388	7	0.21875	72.6401
175	36.6528	35.0235	3	0.0833333	39.2507
176	31.4361	31.4361	0	0	15.7054
177	36.1371	33.6266	6	0.176471	71.5742
178	30.5089	29.5569	3	0.0909091	18.8381
179	12.9788	12.9788	0	0	141.687
180	58.3319	47.0965	11	0.289474	66.4486
181	37.2492	31.3722	4	0.137931	36.5568
182	39.528	37.3971	6	0.1875	68.1145
183	34.6664	32.7745	3	0.103448	65.2478
184	34.7117	31.8805	7	0.21875	89.7763
185	37.9077	37.2809	3	0.0909091	37.6317
186	13.0078	22.6115	3	0.166667	107.041
187	39.9025	36.4927	7	0.205882	21.9214
188	31.4958	30.4686	3	0.103448	46.7611
189	40.6473	40.0819	3	0.0882353	113.988
190	32.7104	32.7104	0	0	68.6448
191	31.1796	30.8901	3	0.0909091	74.2395
192	33.9855	33.9855	0	0	39.0727
193	13.1341	13.1341	0	0	132.048
194	115.035	46.7461	10	0.263158	99.3561
195	32.8898	31.0247	3	0.09375	92.8058
196	37.3176	37.3176	0	0	82.7472
197	29.3562	29.3562	0	0	70.4524
198	31.3843	31.3065	3	0.0882353	95.6036
199	31.5012	31.5012	0	0	46.5518

Tabel C.4: Resultaten van routing ($ellipsvormig, \tau = 130, \rho = 165, \omega = 200$)

Id	$HIST_{score}$	$PRED_{score}$	AD	$AD(\%)$	$Time(s)$
165	14.2824	14.2824	0	0	93.1533
166	43.6229	36.4544	7	0.21875	12.9124
167	47.3173	39.7003	7	0.21875	43.9935
168	42.3918	37.1993	3	0.0909091	29.7488
169	98.1354	98.1354	0	0	9.99086
170	35.5332	35.5332	0	0	22.0374
171	33.49	32.747	3	0.0909091	11.7164
172	13.6617	13.6617	0	0	80.1979
173	52.1626	46.2268	13	0.351351	76.4339
174	36.0145	34.0129	7	0.21875	47.1749
175	37.1661	35.5513	3	0.0833333	34.7008
176	31.8463	31.8463	0	0	20.8424
177	37.1786	33.3663	3	0.0882353	71.6444
178	30.9123	29.9675	3	0.0909091	18.3337
179	13.35	13.35	0	0	165.867
180	59.2492	47.9227	11	0.289474	80.3898
181	37.5456	32.3712	7	0.241379	35.9733
182	40.6737	39.0153	6	0.1875	49.0416
183	35.7739	36.5073	5	0.172414	67.5773
184	36.1251	34.2687	4	0.125	148.637
185	38.4917	37.9189	3	0.0909091	49.3455
186	13.3636	23.1914	3	0.166667	140.461
187	40.313	37.2909	7	0.205882	28.488
188	31.9856	31.2646	3	0.103448	38.8406
189	41.5384	42.2373	6	0.176471	80.3852
190	33.668	33.668	0	0	75.5968
191	31.4034	31.1672	3	0.0909091	65.7294
192	34.6754	34.6754	0	0	51.7882
193	13.4595	23.3941	3	0.166667	133.525
194	50.4419	47.1312	10	0.263158	118.196
195	32.962	31.463	3	0.09375	110.035
196	38.3161	38.3161	0	0	84.4495
197	30.1958	30.1958	0	0	62.9565
198	31.9323	31.8809	3	0.0882353	127.375
199	32.0911	32.0911	0	0	57.5445

Tabel C.5: Resultaten van routing
($exponentieel, p = 0.1, \tau = 130, \rho = 165, \omega = 200$)

Id	$HIST_{score}$	$PRED_{score}$	AD	$AD(\%)$	$Time(s)$
165	15.199	15.199	0	0	122.63
166	49.4025	38.8736	7	0.21875	12.831
167	56.636	43.7981	7	0.21875	65.4618
168	50.0455	41.8844	3	0.0909091	36.5128
169	100.927	100.927	0	0	17.2602
170	39.0383	39.0383	0	0	12.9031
171	36.5483	35.5937	3	0.0909091	19.0794
172	14.5246	14.5246	0	0	116.247
173	58.1005	50.037	12	0.324324	88.2192
174	38.87	38.87	0	0	66.2978
175	39.8995	38.3163	3	0.0833333	38.9307
176	34.9542	34.9542	0	0	24.4173
177	43.0688	38.1226	6	0.176471	66.0883
178	33.4945	32.3471	3	0.0909091	18.8138
179	14.1808	14.1808	0	0	167.186
180	70.6712	52.8447	12	0.315789	66.2496
181	43.8078	36.0075	7	0.241379	47.6792
182	46.1509	44.3093	6	0.1875	89.9518
183	39.8309	37.813	3	0.103448	117.07
184	41.8077	37.0109	7	0.21875	102.45
185	42.4941	42.1556	3	0.0909091	55.323
186	14.1912	14.1912	0	0	193.007
187	44.7798	40.5397	7	0.205882	27.0817
188	35.5031	34.747	3	0.103448	79.056
189	45.8332	45.203	6	0.176471	100.547
190	37.0443	37.0443	0	0	92.4747
191	34.0991	37.9342	6	0.181818	85.7492
192	37.82	37.82	0	0	71.4964
193	14.2954	14.2954	0	0	385.683
194	58.3692	51.4175	10	0.263158	106.328
195	35.9336	34.15	3	0.09375	79.361
196	42.1768	42.1768	0	0	105.282
197	33.0127	33.0127	0	0	116.667
198	34.6477	34.5946	3	0.0882353	152.506
199	34.8286	34.8286	0	0	68.0958

Tabel C.6: Resultaten van routing
$(exponentieel, p = 0.01, \tau = 130, \rho = 165, \omega = 200)$

Bijlage D

Poster

Onderstaande poster geeft een grafische weergave van de *workflow* van het *capacitated vehicle routing* systeem tijdens de ontwikkeling van dit werk in de maand *April 2022*. Het werk was nog niet compleet en beschrijft een preferentiemodel op die werkzaam is op het hoogste abstractieniveau. Daarom geeft de poster enkele de attributen van een oplossing in zijn geheel en perturbeert het voorbeeld zowel paden als clusters. Zowel het nieuwe *conjoint model* uit 6.2.2 als de twee-fasen aanpak uit 5.2 zijn in deze poster zijn niet zichtbaar. De attributen in dit abstractieniveau bieden niet de nodige resolutie voor het bepalen van de voorkeuren waardoor de tekst van dit onderzoek zich enkel richt tot het laagste abstractieniveau.

KU LEUVEN
RESEARCH & DEVELOPMENT

Context-aware preference learning of daily route planners: A Conjoint Analysis approach

Achilles Demey

Introduction

For many decades, businesses have been busy creating daily tours for their drivers to bring goods to their customers. We can observe from historical trajectories that tours that have been created in the past are often reused or slightly modified older plans because customers and their demands tend to be similar across instances. The tours that have been created in the past were not only determined by minimizing an objective cost, such as distance. Implicit preferences of drivers that are hard to formalize have also been considered, such as customer familiarity or fairness constraints.

Problem Statement

The thesis subject involves learning preferences of a daily route planner based on historical data of a company. The preferences will be made explicit by using a technique based on conjoint analysis. Ultimately, the preferences will be used to construct new routes.

Goals and Motivation

Previous research has been done on catching the planners' preferences by estimating a probabilistic transition matrix instead of an objective cost matrix and using that matrix in a MIP solver. We take this one step further by looking at features on the level of a solution, such as geometric properties, and assigning utility scores to the values that the features can take.

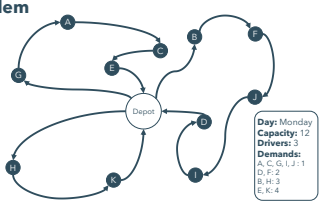
Capacitated Vehicle Routing Problem

Problem

- Central depot
- Fleet of vehicles with uniform capacity Q
- Customers with a demand q_i
- Transition cost matrix C

Objectives

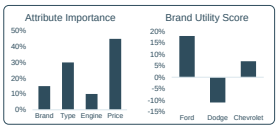
- ✓ Assign a sequence of customers to each truck
- ✓ Each customer served by only one vehicle
- ✓ Minimize the total routing cost



Conjoint Analysis

A statistical market research method used to understand how customers value different features of a product. It assumes that any product can be broken down into a set of attributes that ultimately impact the perceived value of the product. The conjoint model estimates the relative importance between attributes and a assigns a utility score to the levels of each attribute. Conjoint analysis is conducted using surveys that ask respondents to choose between competing sets of features.

Brand	Ford	Chevy	Dodge
Type	SUV	Truck	Truck
Engine	V6, 3.6L	V8, 3.6L	V8, 4.2L
Price	$36,599	$42,299	$42,299
	Selected	Select	Select



The Solution Framework

Input and Data Analysis

Historical Contexts
Day of the week
Customers
Demands
Number of vehicles
Vehicle capacity

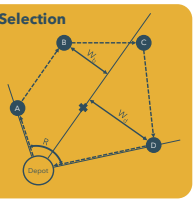
Historical Trajectories
Incidence matrix

Distances
Objective Cost Matrix

Attribute Selection

Geometric
- ✓ Distance
- ✓ Tour span (R)
- ✓ Tour width (W)
- ✓ Center of gravity (X)
- ✓ Max distance from depot

Fairness
- ✓ Customers per driver
- ✓ Quantity delivered per driver



Solver

CVRP Context → Route Plan

Heuristic Solver

Find a route plan for a given context that maximizes the utilities. Since the fitness of a route is measured on the level of a solution, the solver is based on a heuristic.

Iterated Local Search
Genetic Algorithm

Conjoint Analysis

Given a historical solution and its corresponding context, create a pseudorandom solution for that context by making small perturbations in the original solution by e.g. switching arcs or reversing the visiting order of a tour. Create a profile card by calculating and discretising the attribute values.

Historical Random Profile Generation

Historical Solution #54

Profile #54.20	
Context: Monday, 3 drivers	
Attribute	Level
Average tour distance	100 km
Average tour width	50 km
Average tour span	90°
$\sigma_{customers}$	1

Profile Choice Task

Choose the best profile from of a set of profiles that were generated from the same historical instance. The fitness of a profile can be estimated in multiple different ways:

Total distance
Similarity to historical solution
Total arc probability

Conjoint Model

Using the count of the attribute levels in the responses as utility measure has been proven not to be robust in practice. Consequently the parameters of a conjoint model are estimated instead.

Multinomial Logit Model (Hierarchical Bayes)

Contextual Part-Worth Utility Estimations

Context: Monday, 3 drivers		
Attribute	Level	Utility
Average tour distance (22.6%)	50 km	1.41
	100 km	4.35
	150 km	-8.12
Average tour width (14.5%)	40 km	15.80
	50 km	4.09
	60 km	-12.11
Average tour span (31.7%)	10°	-1.11
	45°	4.05
	90°	10.8
	180°	3.02
St. Dev. of customers per tour (31.2 %)	1	-2.04
	2	4.65
	3	9.52
	4	-14.01

Promotor: Tias Guns
Supervisor: Victor Bucarey

Bijlage E

Fiche Masterproef

LEARNING DRIVER PREFERENCES THROUGH CONJOINT ANALYSIS

About

Vehicle routing problems (VRP) is constrained by the limited number of vehicles, the capacity of each delivery vehicle, and the scheduling horizon within which all deliveries have to be made. The objective, often implicitly, can include a wide range of company goals including reducing operational costs, minimizing fuel consumption and carbon emissions, as well as optimizing driver familiarity with the routes and maximizing fairness by assigning tours of similar length and/or time duration to the drivers. Daily plans are often created in a route optimization software that is capable of producing plans that are optimal in terms of route length and travel time. We have observed, however, that in practice, route planners usually modify the result given by the software, or simply pull out, modify, and reuse an old plan that has been used and known to work in the past. The planners, by performing these modifications, are essentially optimizing with their own set of objectives and personal preferences.

These preferences are often subjective and sometimes delicate to formalize in constraints. Some examples are: where the best places for lunch breaks are, which stops are best served earlier or later, and which drivers (tours) are more flexible in receiving more stops. Failure to capture such human aspects is often a source of frustration for both drivers and planners, and a cause for reluctance to use the optimisation software. Furthermore, planners may find it easier or more effective to manually change a previous solution than to provide or update the detailed information in the system. Being able to automatically capture such preferences without the need to formalize them can hence lead to a wider acceptance and better use of optimisation systems.

Aim

The aim of this research is to investigate drivers and planners preferences for vehicle routing problems through conjoint analysis. Conjoint analysis queries the user to get the most preferred option between 2 while the preferences are learnt in the process. The goal is to use these learned subjective preferences in addition to a cost-based objective criterion in vehicle routing systems. This is an alternative to the practice of distinctively formulating a custom VRP for every company with its own routing requirements. Instead, we assume the presence of past vehicle routing solutions over similar sets of customers, and learn to make similar choices.

Starting papers:

"Learn-n-Route: Learning implicit preferences for vehicle routing"[2]
"Vehicle routing by learning from historical solutions"[4]
"Thirty years of conjoint analysis: Reflections and prospects"[9]

Promotor: Prof. dr. T. Guns

Daily advisor: Dr. V. Bucarey

Bibliografie

[1] J. Caceres Cruz, P. Arias, D. Guimarans, D. Riera, and A. Juan. Rich vehicle routing problem: Survey. *ACM Computing Surveys*, 47:1–28, 12 2014.

[2] R. Canoy, V. Bucarey, J. Mandi, and T. Guns. Learn-n-route: Learning implicit preferences for vehicle routing. 2021.

[3] R. Canoy, V. Bucarey, Y. Molenbruch, M. Mulamba, J. Mandi, and T. Guns. Probability estimation and structured output prediction for learning preferences in last mile delivery. *CoRR*, abs/2201.10269, 2022.

[4] R. Canoy and T. Guns. Vehicle routing by learning from historical solutions. 2019.

[5] V. Ceikute and C. S. Jensen. Routing service quality – local driver behavior versus routing services. In *2013 IEEE 14th International Conference on Mobile Data Management*, volume 1, pages 97–106, 2013.

[6] V. Chahar, S. Katoch, and S. Chauhan. A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, 80, 02 2021.

[7] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[8] K. Deep and H. Adane. Variant of partially mapped crossover for the travelling salesman problem. *IJCOPI*, 3:47–69, 01 2012.

[9] P. Green, A. Krieger, and Y. Wind. Thirty years of conjoint analysis: Reflections and prospects. *Interfaces*, 31:S56–S73, 06 2001.

[10] P. E. Green, A. M. Krieger, and P. Bansal. Completely unacceptable levels in conjoint analysis: A cautionary note. *Journal of Marketing Research*, 25(3):293–300, 1988.

[11] A. Hannachi. *Principal Coordinates or Multidimensional Scaling*, pages 201–217. Springer International Publishing, Cham, 2021.

[12] P. Kora and P. Yadlapalli. Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications*, 162:34–36, 03 2017.

[13] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[14] G. Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics (NRL)*, 54(8):811–819, 2007.

[15] G. Laporte, S. Ropke, and T. Vidal. *Chapter 4: Heuristics for the Vehicle Routing Problem*, pages 87–116. 11 2014.

[16] J. Mandi and R. Canoy. CP2021-Data-Driven-VRP. https://github.com/JayMan91/CP2021-Data-Driven-VRP, 2021. Published on Github.

[17] J. Mandi, R. Canoy, V. Bucarey, and T. Guns. Data Driven VRP: A Neural Network Model to Learn Hidden Preferences for VRP. In L. D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[18] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[19] C. Nilsson. Heuristics for the traveling salesman problem. 01 2003.

[20] J.-Y. Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63:337–370, 1996.

[21] D. Raghavarao, J. Wiley, and P. Chitturi. *Choice-Based Conjoint Analysis: Models and Designs*. 08 2010.

[22] V. R. Rao. *Applied Conjoint Analysis*. Springer, 2014 edition, 2014.

[23] T. Stützle and R. Ruiz. *Iterated Local Search*, pages 579–605. Springer International Publishing, Cham, 2018.

[24] S. Wahyuningsih, D. Satyananda, and D. Hasanah. Implementations of tsp-vrp variants for distribution problem. 12:723–732, 01 2016.

[25] W. Zhang. Depth-first branch-and-bound versus local search: A case study. 04 2000.