ERASMUS UNIVERSITEIT ROTTERDAM

Erasmus School of Economics

Bachelorscriptie econometrie

Selectie methodes van het Evolutionaire Algoritme voor het dynamisch voertuigrouteringsprobleem

Auteur:

Jip Ridderbos (499784)

Begeleider:

Ymro Hoogendoorn

Tweede beoordelaar:

Paul Bouman

Samenvatting

Door de ontwikkeling in de technologie kan er steeds meer rekening worden gehouden met onbekende verzoeken van nieuwe klanten tijdens het rijden van routes van voertuigen. Het vinden van de optimale route voor de nieuwe klanten wordt het dynamisch voertuigrouteringsprobleem genoemd. In deze thesis is er onderzocht wat de beste wachtstrategie is voor voertuigen. Waarbij de wachtstrategie zodanig is gekozen dat zoveel mogelijk nieuwe klanten kunnen worden geholpen. Hierbij wordt er gekeken naar simpele heuristische wachtstrategieën en wachtstrategieën gegenereerd door een Evolutionaire Algoritme met verschillende startpopulaties, selectiemethodes en mutatietypes. Bij de selectiemethodes is er gekeken naar de roulette selectiemethode, de toernooiselectiemethode en aanpassingen op deze selectiemethodes. De toepassingen hiervan is nuttig voor de hele transportindustrie. Door verschillende wachtstrategieën te generen en te testen met verschillende datasets in Java is in dit onderzoek aangetoond dat op basis van de gebruikte datasets de *Variabele* wachtstrategie overal het beste presteert. Bij de verschillende selectie, startpopulaties en mutaties hangt de best presterende wachtstrategie af van de gekozen dataset. Daarnaast is er gemiddeld een klein verschil tussen de verschillende selectiemethodes bij de Evolutionaire Algoritmes met een mutatiestap. Als er geen mutatiestap is dan presteert het toenooiselectiemethode beter dan de andere selectie methodes.

Sleutelwoorden: *DVRP, EA, wachtstrategieën, selectie methodes*

4 juli 2021

1 Introductie

In Nederland worden steeds meer pakketten aan huis bezorgd. In 2020 heeft deze opgaande trend zich doorgezet en is het bezorgen van pakketten alleen maar toegenomen, zo werden in het vierde kwartaal van 2020 ongeveer 30 procent meer pakjes verstuurd dan een jaar eerder. PostNL verwacht dat de komende jaren deze groei zal doorzetten, dit komt omdat de mensen gewend zijn geraakt aan het gemak van online bestellen (RTL-nieuws, 2021). Tegelijkertijd staat de logistieke sector door toenemende complexiteit en stijgende kosten onder druk (M.R.J. Kindt, 2020). Bovendien verwachten klanten steeds snellere en flexibele levertijden. Het belang van voertuigenrouteringen wordt daarom steeds groter. Een voertuigenroutering is een planning van een vloot voertuigen om een gegeven set klanten te bezoeken. Een voorbeeld hiervan is het handelsreizigersprobleem (Laporte, 1990). Een optimale voertuigenroutering kan immers antwoord geven voor vervoerders op vragen omtrent het efficiënt plannen van routes en het bepalen van de optimale leverdatums. Bij een statische voertuigroutering zijn alle locaties waar de voertuigen langs gaan van tevoren bekend. Elk voertuig heeft een beperkte reistijd en elke klant heeft een bepaalde plek in het serviceregio. Een uitbreiding van voertuigroutering is een dynamische voertuigroutering. Hierbij kunnen extra klanten worden toegevoegd aan de route van de voertuigen als ze het voertuig al onderweg is (Pillac, 2013). In deze thesis wordt dit dynamische voertuigrouteringsprobleem ('Vehicle Routeing Problem') onderzocht.

Doordat er tegenwoordig steeds meer rekening kan worden gehouden met onbekende verzoeken van nieuwe klanten tijdens de routes van de voertuigen (Branke, 2005) neemt het belang van dynamische voertuigrouteringprobleem steeds meer toe in de transportindustrie. . Een voorbeeld van een bedrijf dat deze wijze van dynamische voertuigenroutering toepast is het bedrijf Inzamelhelden (Inzamelhelden, 2019). Dit is een bedrijf dat vuilnis gescheiden ophaalt op verzoek van particulieren en ondernemers in Delft. Deze verzoeken zijn meestal bekend voordat de vuilnisman begint. Echter, ophaalverzoeken kunnen ook binnenkomen als de vuilnismannen al aan hun route zijn begonnen. Naast inzamelhelden zijn er nog vele andere bedrijven in de transport industrie die willen anticiperen op verzoeken van klanten als de voertuigen al zijn begonnen aan hun routes. Door te anticiperen op deze verzoeken wordt de flexibiliteit van verzoeken gewaarborgd. Vervoersbedrijven kunnen anticiperen door voertuigen op specifieke locaties te laten wachten. Deze locaties kunnen bijvoorbeeld het depot zijn of een locatie op de route. Branke (2000) zegt dat door te anticiperen op verwachtingen de flexibiliteit van de routes gewaarborgd kan worden. In deze thesis wordt er onderzocht wat de beste locatie is om voertuigen te laten wachten. Deze locaties moeten worden gekozen op een manier waarbij de kans wordt gemaximaliseerd dat zoveel mogelijk nieuwe klanten kunnen worden bereikt.

In deze thesis wordt er onderzoek gedaan naar het dynamische voertuigrouteringsprobleem door te onderzoeken wat de beste locatie is om voertuigen te laten wachten (de zogenaamde wachtstrategieën) als één klant wordt toegevoegd, nadat de voertuigen het depot hebben verlaten. Hierbij is de plek en de aankomsttijd van de nieuwe klant aan de start van de route nog niet bekend. De plek wordt bekend op het moment dat tijdens de route het verzoek van de nieuwe klant binnenkomt. Daarnaast is de aankomsttijd

van de voertuigen niet van tevoren bekend. Verder moet een voertuig altijd het depot op tijd kunnen bereiken. Dit houdt in dat als het reizen van en naar de nieuwe klant te veel tijd kost, de locatie van de nieuwe klant niet kan worden opgenomen in de route van het voertuig.

Het doel van de thesis is om te onderzoeken wat de beste locatie of locaties zijn om voertuigen te laten wachten. Hierbij worden de locaties zodanig gekozen dat zoveel mogelijk nieuwe klanten kunnen worden geholpen. Voor het uitwerken van de thesis wordt gebruik gemaakt van het analysemodel van Branke (2005). In het onderzoek van Branke (2000) wordt het dynamische voertuigroutering toegepast. Branke (2000) concludeert in zijn onderzoek dat door te anticiperen op verwachtingen de flexibiliteit van de routes gewaarborgd kan worden. Om het dynamische voertuigrouteringsprobleem (VRP) op te lossen wordt gelijk aan het onderzoek van Branke (2005) gebruik gemaakt van heuristieken. In deze thesis kijken we naar zes verschillende heuristieken, deze zes komen uit Branke (2005). Vervolgens wordt er in deze scriptie ook gekeken naar de locaties van de nieuwe klant en hoe deze verdeeld zijn over het serviceregio. Volgens Branke (2005) is het VRP probleem NP-hard en kan het vinden van de optimale oplossing onpraktisch zijn voor een VRP instantie van redelijke omvang, vanwege de enorm grote rekenkracht die vereist is. Daarom wordt er in deze scriptie gebruik gemaakt van een Evolutionair Algoritme (EA). Om te kijken hoe goed de oplossingen van de EA zijn worden ze vergeleken met de oplossing van een aantal simpele deterministische heuristieken. Bij de EA's wordt er gekeken naar de twee verschillende startpopulaties, dezelfde als in Branke (2005), de vier verschillende selectie methodes en wordt er naar de mutatiestap gekeken.

Na het kijken naar de bovengenoemde dingen, is er gebleken dat de beste wachtstrategie, voor het dynamische voertuigrouteringsprobleem met het toevoegen van één nieuwe klant, *Variabele* optimaal is. Daarnaast is er gebleken dat de beste selectiemethode verschilt per dataset. Daarnaast hebben de gemiddeldes een klein verschil met elkaar. Ook is er gekekeken naar de beste sigma voor de normaal verdeling bij de mutatiestap. Hieruit is gekomen dat geen mutatiestap, sigma van nul, het beste resultaat geeft bij de toernooiselectiemethode.

In het vervolg van de thesis wordt als eerste in Sectie 2 de relevante literatuur gepresenteerd en aangegeven hoe deze literatuur voor het oplossen van de vraagstelling van deze thesis is gebruikt. Vervolgens wordt in Sectie 3 het probleem beschreven. Daarna wordt in Sectie 4 de methodologie beschreven, waarbij in Sectie 4.1 de verschillende simpele heuristische wachtstrategieën uitgelegd worden. Daarna wordt het proces van een EA uitgelegd en de verschillende soorten selectie methodes van de EA's in Sectie 4.2. Na de uitleg over de verschillende wachtstrategieën wordt er in Sectie 5 de resultaten gepresenteerd en worden deze vergeleken met het onderzoek van Branke (2005). Als laatst wordt in Sectie 6 de conclusie uiteengezet en worden er een aantal vervolg onderzoeken aanbevolen.

2 Literatuuroverzicht

Het probleem dat in deze thesis centraal staat wordt in de literatuur beschreven als het voertuigrouteringsprobleem (VRP). Dit is een combinatorisch optimalisatieprobleem waarbij vanuit verschillende perspectieven onderzoek wordt gedaan naar mogelijke oplossingen voor dit optimalisatieprobleem. Zelfs vanuit het biologische perspectief (Marinakis, 2010) waarbij via vergelijking van het paringsproces van honingbijen is gezocht naar nieuwe hybride algoritmes heeft een bijdrage geleverd aan het VRP. In deze thesis staat het toevoegen van een nieuwe klant centraal.

In de literatuur wordt een onderscheid gemaakt tussen statische en dynamische VRP. Zo hebben de papers van Psaraftis (1995) en van Psaraftis (1988) dit verschil beschreven. Ook recentelijk heeft Pillac (2013) het verschil tussen statische en dynamische VRP beschreven. Volgens de hiervoor genoemde papers is het grootste verschil tussen de dynamische en statische VRP dat, bij statische VRP nadat de voertuigen zijn vertrokken er geen extra verzoeken meer kunnen komen, terwijl dit bij het dynamische VRP wel mogelijk is. Bij het oplossen van het dynamische VRP kan onderscheid gemaakt worden in de situatie waarbij de route eerst wordt gemaakt en bij het toevoegen van een nieuwe klant het hele probleem weer opnieuw geoptimaliseerd wordt en de nieuwe routes worden berekend en bereden. Daarnaast is er de situatie waarbij alleen wordt gekeken naar wat de volgende klant is waar het voertuig heen moet gaan. Hierbij wordt er nadat een voertuig bij een klant is geweest, berekend wat de volgende optimale klant is en gaat het voertuig hierheen. In deze variant is aan het begin niet de hele route van de voertuigen bekend, maar alleen de klanten waar de voertuigen als eerst heen gaan. Dit probleem kan daarom volgens de classificatie van Pillac (2013) worden geclassificeerd als een dynamisch VRP probleem waarbij uit wordt gegaan van de situatie dat de routes van tevoren al bekend zijn.

Naast het onderscheid in statische en dynamische VRP wordt in literatuur ook veel geschreven over de wachtstrategieën als onderdeel van het oplossen van het optmaliseringsprobleem. Branke (2005) heeft bijvoorbeeld een dynamische VRP onderzocht, waarbij er wordt gekeken naar het Evolutonaire Algoritme met twee verschillende startpoppulaties. Deze worden vervolgens nog vergeleken met zes verschillende wachtstrategie. Het doel van Branke (2005) is of het introduceren van wachtstrategieën zorgt voor het verhogen van de kans dat nieuwe klanten gedurende een route kunnen worden bereikt en daarmee geholpen. In deze thesis wordt de onderzoeksopzet van Branke (2005) gebruikt en aangevuld.

Bent (2007) heeft ook onderzoek gedaan naar wachtstrategieën bij een dynamisch VRP. Het doel van Bent (2007) is om het aantal klanten, die worden geholpen, te maximaliseren door gebruik te maken van wachtstrategieën en verplaatsingstrategieën. Hierdoor kunnen voertuigen ofwel wachten bij de huidige positie ofwel zich verplaatsen naar willekeurige plekken. Na dit te hebben onderzocht, concluderen ze dat door gebruik te maken van wachtstrategieën en verplaatsingstrategieën er meer klanten kunnen worden geholpen. Het effect van de strategieën is groter als het VRP erg dynamisch is en er veel klanten op late tijdstippen een verzoek indienen. Dus door gebruik te maken van de strategieën kunnen er meer klanten worden geholpen.

Naast Branke (2005) en Bent (2007) wordt er nog veel onderzoek gedaan naar het nut van wachtstrategieën

bij het dynamisch VRP. Zo kijkt Pureza (2008) bijvoorbeeld naar twee strategieën voor het dynamische ophaal- en bezorgprobleem om de kwaliteit van de antwoorden te verbeteren. Ze maken ten eerste gebruik van een wachtstrategie en daarnaast een bufferstrategie. Bij een bufferstrategie stelt de toewijzing van niet-dringende nieuwe aanvragen uit naar de volgende routeplanning. Door het gebruik te maken van de twee strategieën konden er meer klanten worden geholpen, terwijl het aantal routes werd verminderd en de rijafstand niet aanzienlijk is toegenomen ten opzichte van de oude oplossing. In deze thesis wordt er niet gekeken naar bufferstrategieën en alleen naar wachtstrategieën.

Recentelijk heeft Park (2021) een dynamische VRP onderzocht, waarbij een wachtstrategie en een hybride genetisch algoritme wordt gebruikt. De paper richt zich op het minimaliseren van de totale kosten. Doordat er nieuwe klanten komen, terwijl de voertuigen al aan het rijden zijn, wordt er gebruik gemaakt van een wachtstrategie. Deze wachtstrategie houdt in dat de oude route wordt gereden totdat de optimale omleidingindicator wordt bereikt. Als deze omleidingindicator wordt bereikt dan wordt er een nieuwe route berekend, waarna de voertuigen hun weg vervolgen volgens de nieuwe route. Door de omleiding indicator te veranderen, verandert de klanttevredenheid en verandert het aantal omleidingen, wat de totale kosten beïnvloed. In het onderzoek van Park (2021) wordt er rekening gehouden met de mogelijkheid van een ophaal- of bezorgservice. In deze thesis is gekozen voor maar één soort service, waarbij er geen rekening wordt gehouden met de servicetijd. Daarnaast is het doel van Park (2021) dat de totale kosten geminimaliseerd worden. In deze thesis en in Branke (2005) is het doel niet kostenminimalisatie, maar het maximaliseren van de kans dat de nieuwe klant geholpen kan worden.

Eén van deze manieren die wordt gebruikt in deze thesis om wachtstrategieën te genereren is het gebruik van een Evolutonaire Algoritme (EA). Het EA wordt vaak behandeld in de literatuur voor niet alleen het genereren van wachtstrategieën. In het boek van Davis (1991) wordt uitgelegd wat het EA is en hoe ze kunnen worden gebruikt om echte problemen op te lossen. Lawrence Davis legt in het begin van het boek uit wat de hoe de EA traditioneel is ontworpen en geïmplementeerd en laat zien hoe de basistechniek kan worden toegepast op een heel eenvoudig numeriek optimalisatieprobleem. De basis werkt met een poel van oplossingen die vervolgens gecombineerd en gemuteerd naar nieuwe oplossingen. De combinatie wordt gedaan door de selectie van twee initiële oplossingen. Vervolgens worden de basistechniek wordt vervolgens op een aantal manieren gewijzigd en verfijnd, waarbij de effecten van elke wijziging wordt gemeten door vergelijkingen met de prestaties van het origineel. In het vervolg van het boek worden er door meerdere auteurs uitgelegd hoe zij het EA hebben gebruikt op echte problemen in de wereld.

Zo heeft Weise (2009) onderzocht of het gebruik van een EA ervoor zorgt dat de benutting van middelen optimaal gebeurd en de afgelegde afstand minimaliseert van de vrachtvervoerders in de echte wereld. Hiervoor hebben ze data uit de echte wereld gebruikt. Uit de resultaten is geconcludeerd dat ze substantiële verbeteringen verkrijgen in vergelijking met de oorspronkelijke plannen voor de vrachtvervoerders.

Eén van de stappen van een EA is de selectiestap. In deze thesis worden er verschillende selectiestappen behandeld en gekeken wat de beste is, waardoor de kans dat een extra klant kan worden geholpen zo hoog mogelijk is. Zo hebben Chudasama (2011) en Pandey (2016) gekeken wat de beste selectiemethode is voor het handelsreizigersprobleem (‘Travelling Salesman Problem’). Chudasama (2011) heeft gekeken

naar drie selectiemethodes, namelijk de roulette, Elitisme en toernooiselectie. In dit onderzoek wordt geconstateerd dat in de eerste generaties alle drie de selectiemethodes hetzelfde presteren en in de laatste generatie de Elitisme selectiemethode het best presteert, ten opzichte van het roulette selectiemethode en toernooiselectiemethode. In dit onderzoek wordt aangetoond dat de beste oplossingen met behulp van de Elitisme selectiemethode worden gevonden. Pandey (2016) heeft ook gekeken naar drie selectiemethodes, waarvan twee dezelfde als Chudasama (2011). Deze selectiemethodes die worden behandeld zijn de roulette selectiemethode, toernooiselectiemethode en ranking selectiemethode. De ranking selectiemethode gaf het beste resultaat in termen van afstand van het handelsreizigersprobleem. Ze hebben naast dat ook geconcludeerd dat de ranking en roulette selectiemethode vergelijkbaar zijn in hun resultaten en dat ze aan elkaar kunnen worden gekoppeld.

3 Probleembeschrijving

In dit deel wordt het probleem uit Branke (2005) uitgebreid beschreven. Het voertuigrouteringprobleem (VRP) is dynamisch omdat er één enkele nieuw verzoek ingepland moet worden tijdens het uitvoeren van de routes. De locatie van dit verzoek is uniform verdeeld binnen het rechthoekige serviceregio, wat gelegen is in het Euclidische vlak. De gegeven klanten zijn $0, 1, 2, ..., n$ waarbij klant 0 het depot is en d_{ij} voor $0 \leq i < j \leq n$ de Euclidische afstand is tussen de klant i en j. Alle voertuigen krijgen een route toegewezen uit R, die bestaat uit een volgorde van klanten die worden bezocht, beginnend en eindigend bij het depot. Alle m voertuigen vertrekken op tijdstip 0 uit het depot en moeten terug zijn uiterlijk op een gegeven tijdstip $T > 0$. Zoals in Branke (2005) wordt ervan uitgegaan dat de voertuigen direct van de ene klant naar de andere gaan met een constante snelheid en de constante snelheid gedefinieerd is als één afstandseenheid per tijdseenheid.

In het probleem is er geen servicetijd bij de klant net zoals bij Branke (2005). Dus als een voertuig bij een plek is gekomen kan die daar direct weg als dat noodzakelijk is. De duur van een route r uit R is t_r, dit is de tijd die een voertuig nodig heeft om alle klanten te bezoeken in route r. Waarbij $s_r = T - t_r > 0$ de speling is de tijd die over is nadat een voertuig direct naar elke klant is geweest in route r. Deze tijd kan gebruikt worden om te wachten of naar de nieuwe klant te gaan. Elk voertuig volgt de volgorde in de route die is gegeven, tenzij het voertuig de nieuwe klant gaat helpen tussendoor. Als het helpen van de de nieuwe klant door meerdere voertuigen kan worden gedaan, gaat het voertuig dat kleinste omleiding moet nemen naar de nieuwe klant. Als er niet genoeg tijd is om de klant te helpen, wordt het verzoek geweigerd.

Een wachtstrategie voor route r is het toewijzen van wachttijden aan bepaalde plekken in r, inclusief het depot, voor de voertuigen, die langs deze plekken komen. Hierbij heeft elke wachtstrategie een restrictie, dat de som van de wachttijden kleiner moet zijn dan speling die over is. Dus als w_r de totale wachttijd is van route r geldt dat $s_r = T - r_t - w_r \geq 0$. In deze thesis staat centraal het zoeken naar de optimale wachtstrategie zodat de kans dat de nieuwe klant wordt geholpen gemaximaliseerd wordt.

4 Methodologie

In dit deel wordt uitgelegd welke simpele heuristische wachtstrategieën en de wachtstrategieën, die zijn gegenereerd door het EA, er gaan gebruikt worden om het probleem te onderzoeken. Hierbij wordt er gekeken naar wachtstrategieën uit Branke (2005) en gekeken naar het EA met verschillende selectie methodes en mutaties.

4.1 Simpele heuristische wachtstrategieën

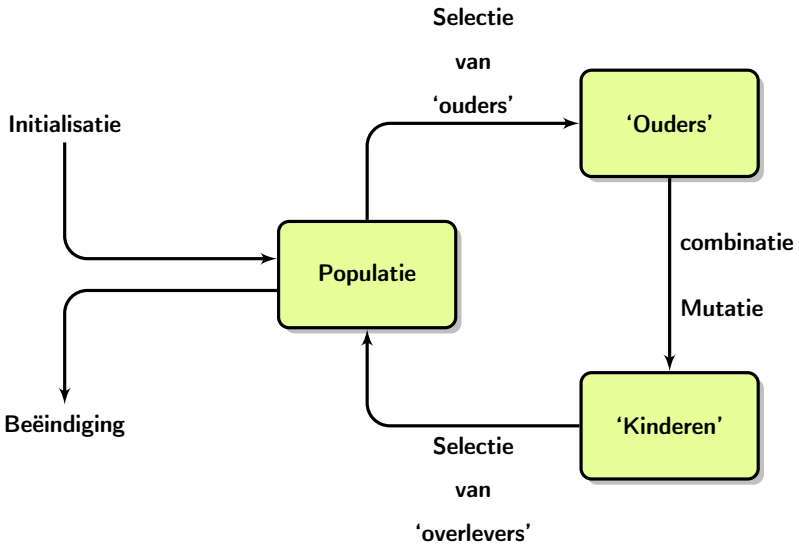
In Branke (2005) worden zes verschillende soorten simpele heuristische wachtstrategieën gebruikt. Deze zes simpele heuristische wachtstrategieën worden hieronder besproken.

- *NietWachten*, hierbij wordt er bij geen één plek gewacht.
- *Depot*, hierbij wordt er bij het depot zo lang mogelijk gewacht voordat de voertuigen beginnen met rijden.
- *MaxAfstand*, hierbij wordt bij de plek, die het verst van het depot afligt, zo lang mogelijk gewacht voordat de voertuigen verder rijden.
- *Locatie*, hierbij wordt er bij elke plek even lang gewacht. Dit houdt in dat voor elke route r met n_r aantal klanten, de wachttijd op elke plek gelijk is aan w_r/n_r.
- *Afstand*, hierbij wordt bij elke plek in verhouding van de afgelegen weg van de vorige klant naar de huidige klant gewacht waarbij $s_r = w_r$. Dus voor elke route $r = (0, k_1, k_2, ..., k_{n_r}, 0)$ waarbij k_i staat voor de klant die wordt bezocht op plek i in de route. De wachttijd van klant i is gelijk aan $s_r \cdot d_{k_i,k_{(i-1)}} / \sum_{j=1}^{n_r} d_{k_j,k_{(j-1)}}$
- *Variabele*, hierbij wachten de voertuig in route r niet totdat de tijd om de resterende afstand naar het depot te rijden gelijk is aan s_r. Vervolgens zijn de beschikbare wachttijd s_r over de overgebleven klanten in verhouding tot de resterende afstanden.

De bovengenoemde wachtstrategieën worden allemaal geanalyseerd om te kijken wat de beste wachtstrategie is. Naast de wachtstrategieën wordt er deze thesis ook gekeken naar twee Evolutionaire Algoritmes (EA) met verschillende selectiemethodes en wordt er ook nog gekeken naar de mutatie stap van de EA, de EA wordt in de volgende subsectie uitgelegd.

4.2 Evolutionaire Algoritmes

In deze thesis wordt ook gebruik gemaakt van een EA. Een EA is een iteratieve stochastische zoekmethode gebaseerd op de principes van natuurlijke evolutie. Zo heeft elke EA een startpopulatie waarbij een aantal individuen worden gecombineerd en de beste individuen uit de vorige populatie een nieuwe generatie vormen. Daarnaast vindt worden vindt er een mutatie plaats bij de nieuwe individuen. Een globale werking van de EA wordt hieronder in Figuur 1 gepresenteerd.



Figuur 1: Globale werking EA

In Figuur 1 hierboven wordt het globale proces van een EA geïllustreerd. Het proces begint bij de initialisatie, waarin een beginpopulatie wordt geconstrueerd. Vervolgens vindt er een selectie plaats waarbij individuen gecombineerd worden, dit zijn veelal de individuen die het beste presteren ten opzichte van de andere individuen. Vervolgens vindt de combinatie plaats en worden er ook mutaties uitgevoerd. Dan worden een aantal individuen gekozen en worden deze geselecteerde individuen toegevoegd aan de nieuwe populatie (de overlevers) en ontstaat een nieuwe generatie. Deze nieuwe generatie bestaat uit de beste van de vorige populatie en de overlevers van de combinatie en mutatie. Dit proces blijft zich herhalen tot een stopcriterium wordt bereikt. Door het proces een aantal keer te herhalen en gebruik te maken van de reproductie van goede oplossingen, komt er uiteindelijk een betere oplossing voor het probleem.

In deze thesis wordt er naast de zes simpele heuristische wachtstrategieën uit sectie 4.1, ook gekeken naar de wachtstrategieën gemaakt aan de hand van het EA. Deze wachtstrategieën verschillen aan de hand van aantal dingen, namelijk de startpopulatie, selectie van ouders en de mutatiestap. Deze verschillen zullen hieronder uitgelegd worden. De wachtstrategieën worden gerepresenteerd als een string van waardes, waarvan de waardes de percentuele wachttijd is van de speling bij de bepaalde klant per route. Voor een probleem met m routes en n klanten heeft een wachtstrategie $(n + 2 \cdot m)$ waardes voor de wachttijd per klant, wachttijd voor het depot aan de start van de route en de wachttijd voor het depot aan het eind van de route. Elke waarde bestaat uit een deel van de speling per route, s_r, dat wordt gebruikt om te wachten bij een bepaalde klant. De som van waarde van één route is gelijk aan één.

In deze thesis wordt de EA uitgevoerd met een populatiegrootte van 100 en voor 100 generaties. Per generatie worden er 99 kinderen gemaakt en wordt de oude populatie daardoor vervangen afgezien de beste oplossing, de generatie bestaat uit 99 kinderen en één beste wachtstrategie van de vorige generatie.

Startpopulatie. Zoals hierboven uitgelegd is heeft elke EA een startpopulatie. In deze thesis zijn er twee verschillende startpopulaties dezelfde startpopulaties die zijn gebruikt door Branke (2005). Namelijk EA1 en EA2, waarbij EA1 bestaat uit 100 wachtstrategieën die willekeurig zijn gegenereerd. EA2 bestaat uit 94 wachtstrategieën die willekeurig zijn gegenereerd en de 6 simpele heuristische wachtstrategieën.

Tussentijdse verwachtingen wachtstrategie. Om te kijken welke wachtstrategie tussentijds de beste verwachting heeft om één nieuwe klant te helpen, worden er 100 willekeurige klanten gegenereerd en wordt er één voor één gekeken of deze kunnen worden geholpen als ze worden toegevoegd aan de route. Het percentage van de klanten die konden worden geholpen is de schatting van hoe goed de oplossing is. Om er voor te zorgen dat de vergelijking van de verschillende wachtstrategieën per generatie eerlijk gaat, word elke generatie getest met dezelfde set van klanten.

Selectie van 'ouders'. Om te bepalen wat de twee ouders worden moet er een selectie plaats vinden. In deze thesis zijn de selectiemethodes gebaseerd op de roulette selectiemethode en de toernooiselectie-methode. Bij het toernooiselectiemethode worden de 100 wachtstrategieën van de generatie willekeurig verdeeld over twee groepen en vervolgens wordt van beide groepen de wachtstrategie, die de beste tussentijdse verwachting heeft ten aanzien van de andere wachtstrategieën, de ouders, dit wordt Tour1 genoemd in deze thesis (Blickle, 1995). Naast deze selectie methode wordt er ook gekeken naar een variant op Tour1, namelijk Tour2. De variatie hierop is dat, na elke tien keer in één generatie de toernooiselectie te doen, de wachtstrategie die de beste tussentijdse verwachting heeft tegenover de rest van wachtstrategieën, niet meer geselecteerd kan worden als ouder. Dus na 10, 20, 30, 40, 50, 60, 70, 80 en 90 keer toernooiselectie te doen wordt elke keer de best presteerde wachtstrategie eruit gehaald.

De roulette selectie methode houd in dat hoe beter de individu, de kans om ouder te worden groter is (Lipowski, 2012). Hoe goed een individu is wordt aan de hand van een bepaalde fitness bepaald. Dus als de fitness hoger is, is de kans om ouder te worden groter. Er wordt namelijk bij roulette selectie slechts één ouder per keer gekozen. In deze thesis worden er twee verschillende fitness gebruikt, de een op de tussentijdse verwachtingen van de wachtstrategie (Roul1) en de ander op positie (Roul2). Met de positie wordt er bedoelt de plek van de wachtstrategie als alle tussentijdse verwachtingen op een rij staan, van best presterende naar slechts presterende. De formules die hiervoor gebruikt is als volgt.

$$totalFitness = \sum fitness \tag{1}$$

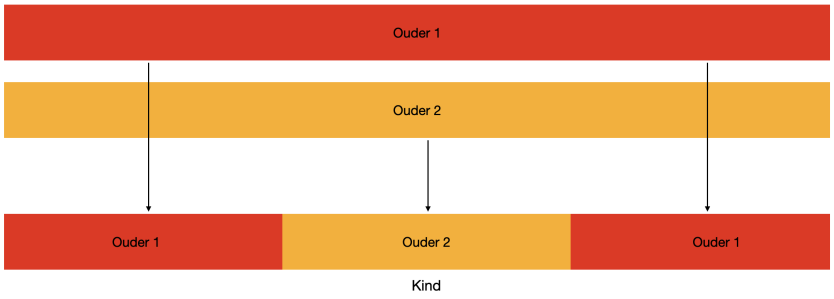
$$percentage = fitness/totalFitness \tag{2}$$

$$fitnessR1 = tussentijdse\ verwachtingen \tag{3}$$

$$fitnessR2 = 1/positie \tag{4}$$

De eerste formule geeft aan hoe de totale fitness is berekent. Vervolgens geeft de tweede formule aan wat de kans is voor een bepaalde wachtstrategie om gekozen te worden als ouder. De derde formule geeft aan wat de fitness is bij Roul1. De laatste, vierde, formule geeft aan wat de fitness is bij Roul2.

Combinatie. Door de combinatie stap tussen twee ouders, krijgt men een nieuw individu, een kind. Net zoals in Branke (2005) wordt er om de combinatie te doen een willekeurige reeks van de reële waarden in de wachtstrategie van de ene ouder geselecteerd en vervangt de overeenkomstige reeks in de wachtstrategie van de andere ouder om één kind te creëren. In Figuur 2 is de globale werking van de combinatiestap te zien.



Figuur 2: Combinatiestap EA

Er is dus te zien in Figuur 2 dat van de twee geselecteerde ouders Dus als de ene ouder bestaat uit de string (0.5 0.3 0.1 0.1) en de andere ouder uit (0.25 0.25 0.25 0.25), kan er een combinatie ontstaan van (0.25 0.3 0.1 0.25).

Mutatie. Nadat de combinatiestap is geweest vindt er een mutatie plaats. Deze mutatie houdt in dat elke waarde, in wachtstrategie van het kind, er een normaal verdeelde getal bij op wordt geteld, net zoals in Branke (2005). In deze thesis wordt er bij de antwoorden gebruik gemaakt van een normale verdeling met het gemiddelde van 0 en een sigma van 0.1. Door de combinatie en mutatie stap kan het zo zijn dat de som van waardes niet gelijk zijn aan één per route, dus worden alles waardes uiteindelijk achteraf genormaliseerd per route.

5 Resultaten

In dit deel worden de resultaten van de vergelijking van de verschillende wachtstrategie gepresenteerd. Daarnaast worden de resultaten met de resultaten van Branke (2005) vergeleken en wordt er gekeken naar de beste significantie voor de normaal verdeling in de mutatie van de EA's.

Het voertuigrouteringprobleem (VRP) in deze thesis bestaat uit van tevoren gegenereerde routes R uit Beasley's OR bibliotheek (Beasley, 1990), namelijk c50, c75, c100, c100b, c120, c150, en c199. Voor elke route is de optimale oplossing gebruikt waar geen service tijd in rekening wordt gebracht. De heuristieken en EA's, die zijn gebruikt om de wachtstrategieën te genereren, zijn gecodeerd in Java gebruikmakend van IDE Eclipse 2018-12 (versie 4.10) en gerund op een Laptop met 2.6 GHz 6-Core Intel Core i7 processor en een 16 GB RAM. De codes die zijn geschreven om de resultaten te generen zijn te vinden in de appendix Listing 1 en 2.

Om alle verschillende wachtstrategieën te kunnen vergelijken worden er 1000 nieuwe klanten gegenereerd, die uniform verdeeld binnen het serviceregio liggen en elke klant heeft een uniforme verdeelde aankomsttijd tussen $[0, T]$, waarbij T T de maximale reistijd is voor een instantie, die is gedefinieerd al de tijd die nodig is voor de de langste rit in de instantie. Daarnaast is het serviceregio gedefinieerd als de rechthoek, die wordt begrensd door de maximale en minimale waarden van de x- en de y-coördinaten van alle gegeven locaties van klanten. Vervolgens wordt er per wachtstrategie gekeken wat het aantal klanten zijn die kunnen worden geholpen, als één klant wordt toegevoegd. Het testen wordt 20 keer herhaald per wachtstrategie. Daarnaast is in deze thesis elke afstand, die berekend is, afgerond naar gehele getallen.

5.1 Simpele heuristieken

Het effect van het gebruiken van de verschillende wachtstrategieën, bij het toevoegen van een extra klant in de VRP, is te zien in Tabel 1. Deze tabel laat het gemiddelde en de standaard deviatie, tussen insluitingstekens, van het aantal klanten die succesvol kan worden toegevoegd aan het VRP, van de 1000 nieuwe klanten, per dataset. Tabel 2 laat het relatieve percentage ten opzichte van het *NietWachten* van alle datasets. Daarnaast laat Tabel 2 ook nog het gemiddelde relatieve percentage van alle datasets.

Uit Tabel 1 en 2 kan worden gezien dat de strategie om bij het depot te wachten, de laagste aantal klanten kan worden toegevoegd ten opzichte van alle andere strategieën. Vervolgens is de kans dat één klant kan worden toegevoegd het hoogst bij de *variabele* strategie. Branke (2005) concludeerde dat de *variabele* strategie de optimale wachtstrategie is wanneer er twee voertuigen betrokken zijn bij een dynamische VRP. Voor elke andere heuristieken zijn er datasets waar de wachtstrategie beter presteren dan andere en bij sommige datasets ook slechter presteren dan andere. Een voorbeeld hiervan is *NietWachten* strategie. Deze heeft de hoogste kans bij dataset c120 en presteert laag bijvoorbeeld bij dataset c100. Dus het ligt aan de dataset welke wachtstrategie het best presteert ten aanzien van alle andere wachtstrategieën.

Tabel 1: Aantal klanten die kan worden geholpen voor verschillende heuristieken en alle testinstanties.

Wachtstrategie	c50	c75	c100	c100b	c120	c150	c199
NietWachten	599.1 (16.66)	644.95 (15.0)	757.1 (12.15)	568.9 (11.31)	629.9 (12.14)	650.25 (13.61)	657.55 (12.39)
Depot	132.25 (8.57)	451.0 (13.29)	640.65 (17.1)	392.75 (17.06)	377.85 (16.41)	488.55 (15.91)	573.2 (15.59)
MaxAfstand	483.45 (12.3)	640.6 (12.62)	751.15 (11.1)	578.6 (16.41)	576.8 (15.60)	652.0 (12.0)	687.2 (16.82)
Locatie	476.75 (16.44)	639.85 (13.7)	765.1 (14.3)	557.35 (18.5)	548.8 (15.17)	623.7 (14.62)	629.25 (15.59)
Afstand	497.55 (15.03)	639.95 (13.4)	761.15 (14.10)	561.8 (14.09)	537.65 (15.43)	631.35 (7.7)	642.3 (15.65)
Variabele	602.05 (10.99)	667.0 (14.32)	761.9 (17.97)	572.9 (15.20)	627.35 (16.49)	668.95 (15.30)	671.55 (12.72)
EA1 Roul1	474.25 (20.97)	625.15 (18.18)	763.65 (11.73)	553.05 (20.97)	541.8 (19.6)	626.1 (17.36)	634.45 (13.3)
EA1 Roul2	468.3 (12.9)	628.9 (24.09)	764.4 (12.6)	551.30 (16.6)	536 (18.12)	628.95 (21.92)	626.8 (19.15)
EA1 Toer1	268.6 (23.10)	629.6 (17.4)	763.2 (12.9)	556.85 (13.9)	543.95 (15.43)	625.75 (12.5)	634.9 (17.0)
EA1 Toer2	472.15(17.91)	630 (14,89)	765.1 (15.31)	551.7 (13.9)	541.1 (15.82)	618.95 (19.69)	636.5(15.94)
EA2 Roul1	465.3 (14.9)	626.8 (15.53)	761.7 (13.31)	545.6 (13.51)	539.05 (13.35)	624.15 (16.90)	632.1 (15.30)
EA2 Roul2	467.85 (18.2)	622.05 (14.2)	767.55 (14.2)	558.65 (13.44)	537.65 (19.05)	624.1 (21.19)	630.35 (14.80)
EA2 Toer1	468.7 (18.35)	629.6 (16.69)	762.75 (14.55)	550.95 (13.9)	543.75 (12.84)	623.25 (20.30)	629.8 (20.62)
EA2 Toer2	467.75 (14.05)	629.8 (16.17)	765.6 (13.35)	548.75 (16.2)	540.3 (16.66)	625.15 (22.45)	636.5 (13.93)

Tabel 2: Relatieve percentage tenopzichte van *NietWachten* strategie

Wachtstrategie	c50	c75	c100	c100b	c120	c150	c199	Gemiddelde
NietWachten	100%	100%	100%	100%	100%	100%	100%	100%
Depot	22.1%	69.9%	84.6%	69.0%	60.0%	75.1%	87.2%	66.8%
MaxAfstand	80.7%	99.3%	99.2%	101.7%	91.6%	100.3%	104.5%	96.8%
Locatie	79.6%	99.2%	101.1%	98.0%	87.1%	95.9%	95.7%	93.8%
Afstand	83.0%	99.2%	100.5%	98.8%	85.4%	97.1%	97.7%	94.5%
Variabele	100.4%	103.4%	100.6%	100.7%	99.6%	102.9%	102.1%	101.4%
EA1 Roul1	79.2%	96.9%	100.9%	97.2%	86.0%	96.3%	96.5%	93.3%
EA1 Roul2	78.2%	97.5%	101.0%	96.9%	85.1%	96.7%	95.3%	93.0%
EA1 Toer1	78.2%	97.6%	100.8%	97.9%	86.4%	96.2%	96.6%	93.4%
EA1 Toer2	78.8%	97.7%	101.1%	97.0%	85.9%	95.2%	96.8%	93.2%
EA2 Roul1	77.6%	97.2%	100.6%	95.9%	85.6%	96.0%	96.1%	92.7%
EA2 Roul2	78.1%	96.4%	101.4%	98.2%	85.4%	96.0%	95.9%	93.1%
EA2 Toer1	78.2%	97.6%	100.7%	96.8%	86.3%	95.8%	95.8%	93.0%
EA2 Toer2	78.1%	97.7%	101.1%	96.5%	85.8%	96.1%	96.8%	93.2%

5.2 EA's

Als er wordt gekeken naar welke selectie methode het best is voor de twee verschillende EA's, kan er ook worden gekeken naar Tabel 1 en 2. Hieruit is te halen dat gemiddeld met de EA1 met toernooi selectie de hoogste kans heeft om een klant te kunnen helpen van alle verschillende selectie methodes en de twee EA's. Het maximale verschil tussen de best presterende selectie methode en EA en de slechtst presterende is ongeveer 0.7%, wat relatief klein is. Dus kan er geen beste selectie methode worden gekozen voor de

twee EA's. In het algemeen kan je uit de gegevens per dataset zien dat elke methode de ene keer beter presteert dan een andere methode en de andere keer weer slechter presteert. Dus het lijkt erop dat de beste presterende selectie methode en EA van de dataset afhangt.

Een oorzaak waarom alle verschillende selectie methodes van de verschillende EA's dicht bij elkaar in de buurt zitten, kan komen doordat de aantal herhalingen laag zijn, namelijk 20 keer. Als deze hoger zijn kunnen de verschillen wel duidelijker zijn. Bij het verhogen van de herhalingen gaat echter de rekentijd van de code omhoog. Een andere oorzaak hiervoor zou zijn dat alle afstanden zijn afgerond. Daarnaast zou een andere oorzaak zijn de mutatie met de normaal verdeling. Als de normaal verdeling een te hoge sigma heeft kan het gebeuren dat alle resultaten uit de EA's willekeurig worden gemaakt. Doordat het willekeurig normale getal, die er bij een waarde optelt tijdens de mutatie, in verhouding groter is dan de waarde, de waardes van de nieuwe wachtstrategie normaal verdeeld worden. Het gevolg hiervan is dat na alle verschillende selectie methodes uiteindelijk de wachtstrategie normaal verdeeld raakt. Hierdoor hebben alle verschillende resultaten van de EA's een verschil dat niet substantieel is om het verschil te detecteren.

Naast de sigma van de normale verdeling, het afronden van afstanden en het lage aantal herhalingen, kan het ook aan het aantal klanten liggen om de wachtstrategie tijdens het proces van een EA de prestatie te schatten. Dit aantal ligt op 100 klanten, zo kan het zijn dat met deze 100 klanten dat een wachtstrategie hier goed presteert, alleen als deze wachtstrategie als oplossing komt voor deze bepaalde EA en wordt gekeken hoeveel van de 1000 gegenereerde klanten geholpen wordt, lager uitvalt. Dit komt omdat er met willekeurigheid wordt gewerkt. Er wordt namelijk verwacht dat de beste wachtstrategie steeds naar de volgende generatie gaat. Dit zou bij EA2 de wachtstrategie *Variabele* zijn, echter is er te zien in Tabel 1 en 2 dat de waardes van EA2 bij elke dataset lager zijn dan die van *Variabele*, behalve die van dataset c100, hierbij zijn de waardes van de beide toernooi selectie en de waarde van roulette selectie met de fitness gebaseerde op de positie hoger dan de waarde van *Variabele*. De oorzaak waarom de waardes van EA2 niet altijd hoger zijn dan de beste van de zes standaard wachtstrategieën ligt bij de de 100 klanten die worden gebruikt om de verwachte prestatie te berekenen. Het kan zo zijn dat de verwachting van de prestatie van de wachtstategie anders is dan de uiteindelijke test met 1000 klanten.

5.3 Vergelijking met Branke (2005)

In vergelijking met het onderzoek van Branke (2005) laten de resultaten uit Tabel 1 en 2 een aantal verschillen zien. Deze verschillen betreffen: andere percentages, andere gemiddelden en andere standaarddeviaties. Een van de oorzaken is het afronden van afstanden. In deze thesis is elke afstand die is berekend afgerond naar gehele getallen. Hierdoor is het mogelijk dat de resultaten verschillen. De maximale reistijd T kan hierdoor groter of kleiner zijn uitgevallen, waardoor s_r groter of kleiner was per route. Hierdoor kunnen er andere resultaten ontstaan en de standaarddeviatie groter wordt. Dit komt omdat als er niet werd afgerond de antwoorden exacter uitkomen, waardoor de deviatie minder wordt.

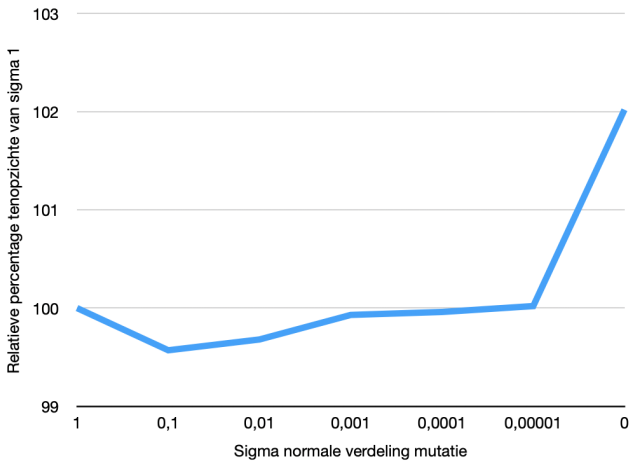
Een andere oorzaak voor het verschil tussen dit onderzoek en het onderzoek van Branke (2005) is het lage aantal herhalingen van 20 met onbekende random seed. Hierdoor zijn er andere resultaten gegenereerd,

door de lage aantal herhalingen in combinatie van een onbekende random seed, als bijvoorbeeld er meer herhalingen plaatsvinden wordt de standaarddeviatie lager en de resultaten exacter.

Voor de twee EA's zijn er in deze thesis vier verschillende soorten selecties onderzocht. In de twee EA's van Branke (2005) is er gebruik gemaakt van een onbekende selectie methode. Hierdoor zullen de resultaten verschillen. Naast de onbekende selectie methode wordt er in Branke (2005) niet aangegeven welke sigma ze gebruiken voor de normaal verdeling bij mutatie van de EA's. Bij de resultaten uit Tabel 1 en 2 heeft de normaal verdeling een sigma van 0.1 bij de mutatie van de kinderen. Doordat de sigma van Branke (2005) onbekend is verschillen de resultaten uit Tabel 1 en 2 met elkaar.

5.4 Verschillende sigma's normale verdelingen

Zoals hierboven aangegeven zou één van de oorzaken van de verschillen tussen Branke (2005) en de resultaten in Tabel 1 en 2 de sigma zijn van de normaal verdeling bij de mutatie stap van de EA's. Daarom is er gekeken naar de resultaten van de best presteerde selectie methode van EA2 met verschillende sigma's bij de verschillende datasets. De waarde hiervan zijn te vinden in de Appendix in Tabel 4. Als er wordt gekeken naar gemiddeld beste sigma is dat die van 0. Dit is te zien in Figuur 3, hieruit is te zien dat als er van sigma 1 naar sigma van 0.1 gaat de waardes, hoe goed de wachtstrategie presteert, gemiddeld afnemen. Vervolgens als de sigma nog lager wordt de waardes gemiddeld stijgen. De waardes met sigma 0 zijn hoger dan de waardes met een sigma van 1.



Figuur 3: Gemiddelde percentage per sigma van EA2

Het is mogelijk dat met een sigma tussen de dan 0.00001 en de 0 de waardes gemiddeld nog hoger worden. Echter is dit nog niet onderzocht. Uit de Tabel 4 in de Appendix is te zien dat niet bij elke data set de waardes met sigma 0 het hoogst is van de waardes. Dus de tot nu toe beste sigma is bij elke dataset verschillend, maar in het algemeen is sigma van 0 het best. Dus een EA2 zonder mutatiestap geeft het beste resultaten onder het EA. Deze waardes zijn niet hoger dan de waardes van de *NietWachten* strategie uit Tabel 1 en 2. De verwachting is als er meer herhalingen plaatsvinden dat de verschillen beter te zien zijn en een beter schatting kan worden gedaan voor een goede sigma.

Zoals hierboven aangegeven en te zien in Figuur 3 is te zien dat een sigma van 0 het beste resultaat geeft bij EA2. Als er wordt gekeken naar het verschil in prestaties tussen EA1 en EA2 is op te merken dat EA2 beter presteert zonder mutatiestap, een sigma van 0. In Tabel 3 zijn de waardes te zien per selectie methode van EA1 en EA2 van de testinstantie c199.

Tabel 3: Waardes c199 met sigma 0 voor normale verdeling

	EA1	EA2
Roul1	634.2 (12.57)	629.3 (13.74)
Roul2	632.2 (16.72)	642.35 (12.94)
Toer1	633.85 (13.30)	651.15 (16.39)
Toer2	630.55 (15.36)	645.3 (19.67)

Bij een normale verdeling met een sigma van 0 en EA2 geeft de toernooiselectiemethode het beste resultaat. Zo is dat te zien in Tabel 3 . Hierin is te zien dat Toer1 het beste resultaat geeft bij EA2. Het verschil tussen EA1 en EA2 ligt aan de startpopulatie, want bij EA2 worden de simpele heutistische wachtstrategieën meegenomen, dit gebeurt niet bij EA1. De verwachting is wel dat dit dan in de buurt van de waardes van de strategie *Variabele*. Echter, is dit niet zo. De verklaring hiervoor is dat de tussentijdse verwachtingen maar kijken naar 100 klanten, zoals eerder genoemd.

6 Conclusie

In deze thesis is er onderzocht wat de beste wachtstrategie is voor voertuigen. Waarbij de wachtstrategie zodanig is gekozen dat zoveel mogelijk nieuwe klanten kunnen worden geholpen. In deze thesis is gekeken naar zes verschillende wachtstrategie, twee verschillende evolutionair algoritmes met vier verschillende selectie methodes en verschillende soorten sigma's voor de mutatie van de kinderen in de EA's. De prestaties van de verschillende manieren zijn berekend in Java door te kijken hoeveel van 1000 gegenereerde klanten, één voor één, kunnen worden toegevoegd.

Uit de resultaten is gebleken dat de wachtstrategie *Variabele* in dit onderzoek gemiddeld de hoogste kans heeft om een nieuwe klant te kunnen helpen. Bij de wachtstrategie *Variabele* wachtend de voertuigen in de route niet totdat de tijd om de resterende afstand naar het depot te rijden gelijk is aan speling. Vervolgens is de beschikbare wachttijd over de overgebleven klanten in verhouding tot de resterende afstanden verdeeld. Daarnaast is er geconcludeerd dat er niet één goede selectiemethode is over de verschillende EA's. Dit wordt veroorzaakt door het weinige aantal herhalingen, 20 herhalingen, een niet optimale sigma voor de normaal verdeling in de mutatie stap van de EA's, tussendoor afronden en nog nader onbepaalde redenen. Ook kan er geconcludeerd worden dat voor elke dataset een andere sigma het best resultaat geeft voor de EA2, maar gemiddeld een sigma van 0 met een toernooiselectiemethode het beste resultaat geeft. Dit houdt in dat er geen mutatiestap het best is om het beste resultaat onder de EA's te krijgen.

Voor vervolgonderzoek zou er kunnen worden gekeken naar verschillende soorten mutaties, ten opzichte van de de mutatie met de normaal verdeling. Daarnaast kan er verder worden gekeken naar wat de optimale sigma is voor de normaal verdeling bij de mutatie, door niet de afstanden af te ronden, door meer dan twintig herhalingen te doen. Ook kan er worden gekeken naar een meer efficiënte manier om de EA te runnen, waardoor de tussentijdse prestatie verwachtingen beter zijn en de uiteindelijke prestaties ook beter worden.

Literatuur

Beasley, J. E. (1990). Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, 41(11):1069–1072.

Bent, Russell en Van Hentenryck, P. (2007). Waiting and relocation strategies in online stochastic vehicle routing. In *IJCAI*, volume 7, pages 1816–1821.

Blickle, Tobias en Thiele, L. (1995). A mathematical analysis of tournament selection. In *ICGA*, volume 95, pages 9–15. Citeseer.

Branke, Jürgen en Mattfeld, D. C. (2000). Anticipation in dynamic optimization: The scheduling case. In *International Conference on Parallel Problem Solving from Nature*, pages 253–262. Springer.

Branke, Jürgen en Middendorf, M. e. N. G. e. D. M. (2005). Waiting strategies for dynamic vehicle routing. *Transportation science*, 39(3):298–312.

Chudasama, Chetan en Shah, S. e. P. M. (2011). Comparison of parents selection methods of genetic algorithm for tsp. In *International Conference on Computer Communication and Networks CSI-COMNET-2011, Proceedings*, pages 85–87.

Davis, L. (1991). Handbook of genetic algorithms.

Inzamelhelden (2019). Inzamelhelden inzamelhelden.

Laporte, Gilbert en Martello, S. (1990). The selective travelling salesman problem. *Discrete applied mathematics*, 26(2-3):193–207.

Lipowski, Adam en Lipowska, D. (2012). Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196.

Marinakis, Yannis en Marinaki, M. e. D. G. (2010). Honey bees mating optimization algorithm for large scale vehicle routing problems. *Natural Computing*, 9(1):5–27.

M.R.J. Kindt, S.J. van der Meulen, T. G. (2020). Nederland van gateway naar logistieke hub bouwen aan een grenzeloos netwerk.

Pandey, H. M. (2016). Performance evaluation of selection methods of genetic algorithm and network security concerns. *Procedia Computer Science*, 78:13–18.

Park, Hyungbin en Son, D. e. K. B. e. J. B. (2021). Waiting strategy for the vehicle routing problem with simultaneous pickup and delivery using genetic algorithm. *Expert Systems with Applications*, 165:113959.

Pillac, Victor en Gendreau, M. e. G. C. e. M. A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11.

Psaraftis, H. N. (1988). Dynamic vehicle routing problems. *Vehicle routing: Methods and studies*, 16:223–248.

Psaraftis, H. N. (1995). Dynamic vehicle routing: Status and prospects. *Annals of operations research*, 61(1):143–164.

Pureza, Vitória en Laporte, G. (2008). Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows. *INFOR: Information Systems and Operational Research*, 46(3):165–175.

RTL-nieuws (2021). Postnl bezorgt recordaantal pakketten door corona.

Weise, Thomas en Podlich, A. e. G. C. (2009). Solving real-world vehicle routing problems with evolutionary algorithms. In *Natural intelligence for scheduling, planning and packing problems*, pages 29–53. Springer.

7 Appendix

Tabel 4: De hoogste waarde van EA2 met verschillende sigma's

Sigma	c50	c50	c75	c75	c100	c100	c100b	c100b	c120	c120	c150	c150	c199	c199	Gemiddelde stijging
1	471.25	1	637.2	1	762.9	1	556.5	1	548.95	1	633.85	1	635.75	1	1
0.1	471.25	1	629.8	0.988	762.9	1	558.65	1.004	543.75	0.991	625.15	0.986	636.5	1.001	0.996
0.01	473.4	1.005	627.65	0.985	767.85	1.006	561.4	1.009	544.6	0.992	622.8	0.983	634.6	0.998	0.997
0.001	476.4	1.0109	627.65	0.985	767.95	1.007	557.2	1.001	549.55	1.001	625.0	0.986	638.5	1.004	0.999
0.0001	483.3	1.0256	629.1	0.987	767.85	1.006	552.35	0.993	546.35	0.995	630.05	0.994	633.2	0.996	1.000
0.00001	485.4	1.030	632.1	0.992	764.55	1.002	552.75	0.993	545.4	0.994	625.5	0.987	638.05	1.004	1.000
0	493.2	1.047	640.3	1.005	764.0	1.001	570.35	1.025	562.9	1.025	642.8	1.014	651.15	1.024	1.020

Listing 1: Main code

```
import java.io.File;
import java.io.FileNotFoundException;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Random;

public class main {
        private static int maximumx = Integer.MIN_VALUE;
        private static int minimumx = Integer.MAX_VALUE;
        private static int maximumy = Integer.MIN_VALUE;
        private static int minimumy = Integer.MAX_VALUE;
        private static int[] antwoordenlijstMax;

        public static void main(String[] args) throws FileNotFoundException {
                Random rnd = new Random(134535);
//              int klanten = 50;
//              int voertuigen = 5;
//              File bestand = new File("c50.txt");
//              int[] voerPerRout = { 11, 9, 11, 9, 10 };
//              int[][] locaties = gegevens(klanten, voertuigen, bestand);
//              int[][] routes = { { 0, 12, 37, 44, 15, 45, 33, 39, 10, 49, 5, 46, 0 },
//                              { 0, 18, 13, 41, 40, 19, 42, 17, 4, 47, 0, 0, 0 }, { 0, 32, 1, 22, 20, 35, 36, 3, 28, 31, 26, 8, 0 },
//                              { 0, 6, 14, 25, 24, 43, 7, 23, 48, 27, 0, 0, 0 }, { 0, 38, 9, 30, 34, 50, 16, 21, 29, 2, 11, 0, 0 } };

//              int klanten = 75;
//              int voertuigen = 10;
//              File bestand = new File("c75.txt");
//              int[] voerPerRout = { 7, 8, 5, 9, 10, 6, 6, 6, 9, 9 };
//              int[][] locaties = gegevens(klanten, voertuigen, bestand);
//              int[][] routes = { { 0, 68, 2, 28, 61, 21, 74, 30, 0, 0, 0, 0 }, { 0, 6, 33, 63, 23, 56, 24, 49, 16, 0, 0, 0 },
//                              { 0, 67, 46, 34, 4, 75, 0, 0, 0, 0, 0, 0 }, { 0, 51, 73, 1, 43, 41, 42, 64, 22, 62, 0, 0 },
//                              { 0, 27, 37, 20, 70, 60, 71, 69, 36, 47, 48, 0 }, { 0, 58, 10, 38, 65, 66, 53, 0, 0, 0, 0, 0 },
//                              { 0, 26, 12, 39, 9, 40, 17, 0, 0, 0, 0, 0 }, { 0, 7, 11, 59, 14, 35, 8, 0, 0, 0, 0, 0 },
//                              { 0, 45, 29, 5, 15, 57, 13, 54, 19, 52, 0, 0 }, { 0, 3, 44, 32, 50, 18, 55, 25, 31, 72, 0, 0 } };

                int klanten = 100;
                int voertuigen = 8;
                File bestand = new File("c100.txt");

                int[] voerPerRout = { 14, 14, 12, 16, 12, 13, 5, 14 };
                int[][] locaties = gegevens(klanten, voertuigen, bestand);

                int[][] routes = { { 0, 52, 7, 82, 48, 19, 11, 64, 49, 36, 47, 46, 8, 83, 18, 0, 0, 0 },
                                { 0, 76, 77, 3, 79, 78, 34, 35, 65, 71, 9, 51, 81, 33, 50, 0, 0, 0 },
                                { 0, 53, 26, 4, 25, 55, 54, 24, 29, 68, 80, 12, 28, 0, 0, 0, 0, 0 },
                                { 0, 58, 2, 57, 15, 43, 42, 14, 44, 38, 86, 16, 91, 100, 37, 9, 92, 0 },
                                { 0, 21, 72, 75, 56, 39, 67, 23, 41, 22, 74, 73, 40, 0, 0, 0, 0, 0 },
                                { 0, 89, 60, 5, 84, 45, 17, 61, 85, 93, 59, 99, 96, 6, 0, 0, 0, 0 },
                                { 0, 13, 87, 97, 95, 94, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                                { 0, 27, 69, 1, 70, 30, 20, 66, 32, 90, 63, 10, 62, 88, 31, 0, 0, 0 } };

//              int klanten = 100; //verander in gegevens de spazie naar twee spazies
//              int voertuigen = 10;
//
//              File bestand = new File("c100b.txt");
//              int[] voerPerRout = { 9, 11, 8, 9, 10, 6, 14, 11, 9, 13 };
//              int[][] locaties = gegevens(klanten, voertuigen, bestand);
//              int[][] routes = { { 0, 10, 12, 14, 16, 15, 19, 18, 17, 13, 0, 0, 0, 0, 0, 0 },
//                              { 0, 75, 1, 2, 4, 6, 9, 11, 8, 7, 3, 5, 0, 0, 0, 0 },
//                              { 0, 57, 59, 60, 58, 56, 53, 54, 55, 0, 0, 0, 0, 0, 0, 0 },
//                              { 0, 32, 33, 31, 35, 37, 38, 39, 36, 34, 0, 0, 0, 0, 0, 0 },
//                              { 0, 90, 87, 86, 83, 82, 84, 85, 88, 89, 91, 0, 0, 0, 0, 0 },
//                              { 0, 67, 65, 63, 74, 62, 66, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
//                              { 0, 81, 78, 76, 71, 70, 73, 77, 79, 80, 72, 61, 64, 68, 69, 0 },
//                              { 0, 20, 24, 25, 27, 29, 30, 28, 26, 23, 22, 21, 0, 0, 0, 0 },
//                              { 0, 98, 96, 95, 94, 92, 93, 97, 100, 99, 0, 0, 0, 0, 0, 0 },
//                              { 0, 43, 42, 41, 40, 44, 46, 45, 48, 51, 50, 52, 49, 47, 0, 0 }, };
//
//              int klanten = 120; // verander naar dubbele spazie
//              int voertuigen = 7;
```

```
75  //
76  //              File bestand = new File("c120.txt");
77  //              int[] voerPerRout = { 21, 16, 18, 16, 16, 17, 16 };
78  //              int[][] locaties = gegevens(klanten, voertuigen, bestand);
79  //              int[][] routes = {
80  //                              { 0, 109, 21, 20, 23, 26, 28, 32, 35, 29, 36, 34, 31, 30, 33, 27, 24, 22, 25, 19, 16, 17, 0 },
81  //                              { 0, 106, 73, 76, 68, 77, 79, 80, 78, 75, 72, 74, 71, 70, 69, 67, 107, 0, 0, 0, 0, 0, 0 },
82  //                              { 0, 119, 81, 112, 84, 117, 113, 83, 108, 118, 18, 114, 90, 91, 89, 85, 86, 111, 82, 0, 0, 0, 0 },
83  //                              { 0, 100, 52, 54, 57, 59, 65, 61, 62, 64, 66, 63, 60, 56, 58, 55, 53, 0, 0, 0, 0, 0, 0 },
84  //                              { 0, 8, 12, 13, 14, 15, 11, 10, 9, 7, 6, 5, 4, 3, 1, 2, 88, 0, 0, 0, 0, 0, 0 },
85  //                              { 0, 87, 92, 93, 96, 94, 97, 115, 110, 98, 116, 103, 104, 99, 101, 102, 105, 120, 0, 0, 0, 0, 0 },
86  //                              { 0, 95, 37, 38, 39, 42, 41, 44, 47, 46, 49, 50, 51, 48, 45, 43, 40, 0, 0, 0, 0, 0, 0 }
87  //              };
88  //
89  //              int klanten = 150;
90  //              int voertuigen = 12;
91  //
92  //              File bestand = new File("c150.txt");
93  //              int[] voerPerRout = {16,15,13,12,13,15,11,13,3,13,15,11 };
94  //              int[][] locaties= gegevens(klanten,voertuigen,bestand);
95  //              int[][] routes= {
96  //              {0,69 ,101 , 70 , 30 , 20 ,128,  66 , 71 , 65 ,136 , 35 ,135 , 34 , 78 ,129 , 79,0},
97  //              {0,137 ,  2 ,115 ,145 , 41 , 22 ,133 , 23 , 56 , 75 , 74 , 72,  73 , 21,  40,0,0},
98  //              {0,138, 109,  54, 130 , 55 , 25 , 67 , 39 ,139 ,  4 ,110 ,149,  26,0,0,0,0},
99  //              {0,132 ,  1 ,122,  51 ,103,   9 ,120 , 81 , 33 ,102 , 50 ,111,0,0,0,0,0},
100 //              {0,27,  31 , 10, 108, 131,  32,  90,  63 ,126,  62, 148,  88 ,127,0,0,0,0},
101 //              {0,18 ,114,  46, 124,  47,  36, 143,  49,  64,  11, 107 , 19 ,123 ,  7 ,146,0,0},
102 //              {0,89 ,118 , 60 , 83 ,125 , 45 ,  8,  48,  82, 106 , 52,0,0,0,0,0,0},
103 //              {0,147 ,  5,  84,  17, 113,  86 ,140 , 38 , 43 , 15 , 57 ,144 , 58,0,0,0,0},
104 //              {0,105,  53 ,112,0,0,0,0,0,0,0,0,0,0,0,0},
105 //              {0,28 , 76 ,116 , 77,   3, 121,  29 , 24 ,134 , 80 ,150 , 68 , 12,0,0,0,0},
106 //              {0,6 , 96,  99, 104,  61,  16 ,141 , 44 ,119 , 14 ,142 , 42,  87 , 97, 117,0,0},
107 //              {0,94 , 59,  93,  85,  91, 100 , 37 , 98 , 92 , 95 , 13,0,0,0,0,0,0}
108 //      };
109 //
110 //              int klanten = 199;
111 //              int voertuigen = 17;
112 //
113 //              File bestand = new File("c199.txt");
114 //              int[] voerPerRout = { 15, 14, 14, 14, 14, 13, 13, 13, 13, 12, 11, 11, 11, 10, 10, 10, 1 };
115 //              int[][] locaties = gegevens(klanten, voertuigen, bestand);
116 //              int[][] routes = { { 0, 89, 166, 60, 84, 17, 113, 86, 140, 38, 14, 43, 15, 57, 144, 137, 0 },
117 //                              { 0, 59, 151, 92, 37, 98, 100, 193, 91, 85, 93, 104, 99, 96, 6, 0, 0 },
118 //                              { 0, 82, 46, 124, 168, 47, 36, 143, 49, 64, 11, 175, 107, 19, 123, 0, 0 },
119 //                              { 0, 101, 122, 20, 188, 66, 71, 65, 136, 35, 135, 164, 34, 78, 50, 0, 0 },
120 //                              { 0, 195, 134, 163, 24, 29, 121, 169, 129, 79, 185, 158, 3, 77, 184, 0, 0 },
121 //                              { 0, 2, 178, 115, 145, 41, 22, 133, 75, 74, 171, 73, 21, 105, 0, 0, 0 },
122 //                              { 0, 54, 130, 165, 55, 25, 170, 67, 23, 186, 56, 197, 72, 40, 0, 0, 0 },
123 //                              { 0, 189, 10, 108, 90, 126, 63, 181, 32, 131, 160, 128, 30, 70, 0, 0, 0 },
124 //                              { 0, 5, 173, 61, 16, 141, 191, 44, 119, 192, 142, 42, 172, 87, 0, 0, 0 },
125 //                              { 0, 146, 88, 148, 159, 62, 182, 48, 7, 194, 106, 153, 52, 0, 0, 0, 0 },
126 //                              { 0, 26, 149, 179, 155, 4, 139, 187, 39, 110, 198, 180, 0, 0, 0, 0, 0 },
127 //                              { 0, 76, 196, 116, 68, 80, 150, 177, 109, 12, 138, 154, 0, 0, 0, 0, 0 },
128 //                              { 0, 27, 167, 127, 190, 31, 162, 69, 132, 176, 111, 28, 0, 0, 0, 0, 0 },
129 //                              { 0, 147, 118, 83, 199, 125, 45, 174, 8, 114, 18, 0, 0, 0, 0, 0, 0 },
130 //                              { 0, 1, 51, 103, 161, 9, 120, 81, 33, 157, 102, 0, 0, 0, 0, 0, 0 },
131 //                              { 0, 112, 183, 94, 95, 97, 117, 13, 58, 152, 53, 0, 0, 0, 0, 0, 0 },
132 //                              { 0, 156, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };
133
134             int[][] afstanden = getDistanceMatrix(locaties, klanten + 1);
135             int[] totalTimeRoute = totalTime(voertuigen, afstanden, routes, voerPerRout);
136
137             double[] wachttijden2 = nietWachtenStra(routes, voerPerRout, klanten);
138             System.out.println(Arrays.toString(wachttijden2));
139             int timeLimit = maximum(totalTimeRoute);
140 //          System.out.println(routes[0].length + "dit is het percentage");
141
142             int percentage = 0;
143             for (int i = 0; i < 1000; i++) {
144
145                     int[] extraKlant = extraKlant(rnd);
146 //                  int[] extraKlant = {52,13};
147                     int klantArrival = klantArrival(rnd, timeLimit);
148 //                  System.out.println(" dit is arrival" + klantArrival);
149                     if (basis(extraKlant, klantArrival, timeLimit, routes, afstanden, totalTimeRoute, locaties, wachttijden2,
150                                     voerPerRout)) {
151                             percentage++;
```

```
152                         }
153
154                 }
155                 System.out.println(percentage + "dit is het percentage");
156 //              if (nietWachten(extraKlant, klantArrival, timeLimit, routes, afstanden, totalTimeRoute)) {
157 //                      System.out.println("Ik ben jip ridderbos");
158 //              }
159
160                 System.out.println();
161                 System.out.println("Ik ben jip ridderbos");
162                 System.out.println(Arrays.toString(routes[0]));
163                 int[] arrivalTime = arrivalTime(timeLimit, rnd);
164                 int[][] nieuweKlant = honderdExtraKlant(timeLimit, rnd);
165 //              for (int i = 0; i < 100; i++) {
166 //                      System.out.println(nieuweKlant[i][0]+ "  " +nieuweKlant[i][1]);
167 //              }
168
169                 ArrayList<Klant> lijst = StartGeneratie(voerPerRout, klanten, rnd);
170
171 //              ArrayList<Klant> lijst = StartGeneratieEA2(routes, afstanden, totalTimeRoute, voerPerRout, klanten);
172
173                 for (Klant klant : lijst) {
174                         int getal = percentage(arrivalTime, nieuweKlant, timeLimit, routes, afstanden, totalTimeRoute, locaties,
175                                         klant.wachttijd, voerPerRout);
176 //                      System.out.println(klant.geholpen);
177                         klant.setGeholpen(getal);
178
179                 }
180                 System.out.println("dit is gesoorteerd");
181                 Collections.sort(lijst, new SortbyGeholpen());
182
183                 ArrayList<Klant> nieuweGen = evolutionAlgoritme(lijst, voerPerRout, timeLimit, routes, afstanden,
184                                 totalTimeRoute, locaties, rnd);
185                 for (int i = 0; i < 100; i++) {
186                         System.out.println(i);
187                         nieuweGen = evolutionAlgoritme(nieuweGen, voerPerRout, timeLimit, routes, afstanden, totalTimeRoute,
188                                         locaties, rnd);
189
190                 }
191                 double[] wachtGen = nieuweGen.get(0).wachttijd;
192                 int uitslag = 0;
193
194                 for (int i = 0; i < 1000; i++) {
195
196                         int[] extraKlant1 = extraKlant(rnd);
197 //                      int[] extraKlant = {52,13};
198                         int klantArrival1 = klantArrival(rnd, timeLimit);
199 //                      System.out.println(" dit is arrival" + klantArrival);
200                         if (basis(extraKlant1, klantArrival1, timeLimit, routes, afstanden, totalTimeRoute, locaties, wachtGen,
201                                         voerPerRout)) {
202                                 uitslag++;
203                         }
204
205                 }
206
207                 System.out.println("dit is uitslag gen : " + uitslag + " size " + wachtGen.length);
208                 double x = 0;
209                 for (int i = 0; i < 16; i++) {
210                         x = x + wachtGen[i];
211                 }
212
213                 System.out.println(x);
214                 System.out.println(Arrays.toString(wachtGen));
215                 System.out.println(percentage + "dit is het percentage");
216                 System.out.println(Arrays.toString(totalTimeRoute) + "dit is het time");
217                 System.out.println(19 % 10);
218
219                 int breedVraag = 20;
220                 int[] antwoordenlijst1 = new int[breedVraag];
221                 int[] antwoordenlijst2 = new int[breedVraag];
222                 int[] antwoordenlijstR1 = new int[breedVraag];
223                 int[] antwoordenlijstR2 = new int[breedVraag];
224                 int[] antwoordenlijstNiet = new int[breedVraag];
225                 int[] antwoordenlijstDepot = new int[breedVraag];
226                 int[] antwoordenlijstMax = new int[breedVraag];
227                 int[] antwoordenlijstLoc = new int[breedVraag];
228                 int[] antwoordenlijstDist = new int[breedVraag];
```

```
229                 int[] antwoordenlijstVari = new int[breedVraag];
230
231                 double[] antwoordWachtNiet = nietWachtenStra(routes, voerPerRout, klanten);
232                 double[] antwoordWachtDepot = depotStra(routes, voerPerRout, klanten);
233                 double[] antwoordWachtMax = maxDistStra(routes, afstanden, voerPerRout, klanten);
234                 double[] antwoordWachtLoc = locationStra(routes, voerPerRout, klanten);
235                 double[] antwoordWachtDist = distanceStra(routes, afstanden, totalTimeRoute, voerPerRout, klanten);
236                 double[] antwoordWachtVari = variableStra(routes, afstanden, totalTimeRoute, voerPerRout, klanten);
237
238
239
240                 for (int i = 0; i < breedVraag; i++) {
241                         Random rnd1 = new Random(i);
242   //                    ArrayList<Klant> lijst1 = StartGeneratie(voerPerRout, klanten, rnd1);
243
244                         ArrayList<Klant> lijst1 = StartGeneratieEA2(routes, afstanden, totalTimeRoute, voerPerRout, klanten,rnd1);
245                         int[] arrivalTime1 = arrivalTime(timeLimit, rnd1);
246                         int[][] nieuweKlant1 = honderdExtraKlant(timeLimit, rnd1);
247
248                         for (Klant klant : lijst1) {
249                                 int getal = percentage(arrivalTime1, nieuweKlant1, timeLimit, routes, afstanden, totalTimeRoute, locaties,
250                                                 klant.wachttijd, voerPerRout);
251   //                            System.out.println(klant.geholpen);
252                                 klant.setGeholpen(getal);
253   //                            System.out.println(klant.geholpen);
254
255                         }
256                         System.out.println("dit is gesoorteerd");
257                         Collections.sort(lijst1, new SortbyGeholpen());
258                         ArrayList<Klant> GenRoul1 = evolutionAlgoritmeTournament1(lijst1, voerPerRout, timeLimit, routes, afstanden,
259                                         totalTimeRoute, locaties, rnd1);
260                         ArrayList<Klant> GenRoul2 = evolutionAlgoritmeTournament2(lijst1, voerPerRout, timeLimit, routes, afstanden,
261                                         totalTimeRoute, locaties, rnd1);
262                         ArrayList<Klant> Gen1 = evolutionAlgoritme(lijst1, voerPerRout, timeLimit, routes, afstanden,
263                                         totalTimeRoute, locaties, rnd1);
264                         ArrayList<Klant> Gen2 = evolutionAlgoritme2(lijst1, voerPerRout, timeLimit, routes, afstanden,
265                                         totalTimeRoute, locaties, rnd1);
266                         for (int j = 0; j < 100; j++) {
267
268                                 Gen1 = evolutionAlgoritme(Gen1, voerPerRout, timeLimit, routes, afstanden, totalTimeRoute,
269                                                 locaties, rnd1);
270                                 Gen2 = evolutionAlgoritme2(Gen2, voerPerRout, timeLimit, routes, afstanden, totalTimeRoute,
271                                                 locaties, rnd1);
272                                 GenRoul2 = evolutionAlgoritmeTournament2(GenRoul2, voerPerRout, timeLimit, routes, afstanden,
273                                                 totalTimeRoute, locaties, rnd1);
274                                 GenRoul1 = evolutionAlgoritmeTournament1(GenRoul1, voerPerRout, timeLimit, routes, afstanden,
275                                                 totalTimeRoute, locaties, rnd1);
276
277                         }
278                         double[] antwoordWacht1 = Gen1.get(0).wachttijd;
279                         double[] antwoordWacht2 = Gen2.get(0).wachttijd;
280                         double[] antwoordWachtR1 = GenRoul1.get(0).wachttijd;
281                         double[] antwoordWachtR2 = GenRoul2.get(0).wachttijd;
282
283
284
285                         antwoordenlijst1[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWacht1,
286                                         voerPerRout, rnd1);
287                         antwoordenlijst2[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWacht2,
288                                         voerPerRout, rnd1);
289                         antwoordenlijstR1[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWachtR1,
290                                         voerPerRout, rnd1);
291                         antwoordenlijstR2[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWachtR2,
292                                         voerPerRout, rnd1);
293                         antwoordenlijstNiet[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWachtNiet,
294                                         voerPerRout, rnd1);
295                         antwoordenlijstDepot[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWachtDepot,
296                                         voerPerRout, rnd1);
297                         antwoordenlijstMax[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWachtMax,
298                                         voerPerRout, rnd1);
299                         antwoordenlijstLoc[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWachtLoc,
300                                         voerPerRout, rnd1);
301                         antwoordenlijstDist[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWachtDist,
302                                         voerPerRout, rnd1);
303                         antwoordenlijstVari[i] = antwoord(timeLimit, routes, afstanden, totalTimeRoute, locaties, antwoordWachtVari,
304                                         voerPerRout, rnd1);
305                 }
```

```
306                 double mean1 = mean(antwoordenlijst1);
307                 double standaardDef1 = getStd(antwoordenlijst1, mean1);
308                 System.out.println("dit is mean " + mean1 + " (" + standaardDef1 + "  gen1");
309 
310                 double mean2 = mean(antwoordenlijst2);
311                 double standaardDef2 = getStd(antwoordenlijst2, mean2);
312                 System.out.println("dit is mean " + mean2 + " (" + standaardDef2 + "  gen2");
313 
314                 double meanR1 = mean(antwoordenlijstR1);
315                 double standaardDefR1 = getStd(antwoordenlijstR1, meanR1);
316                 System.out.println("dit is mean " + meanR1 + " (" + standaardDefR1 + "  genR1");
317 
318                 double meanR2 = mean(antwoordenlijstR2);
319                 double standaardDefR2 = getStd(antwoordenlijstR2, meanR2);
320                 System.out.println("dit is mean " + meanR2 + " (" + standaardDefR2 + "  genR2");
321 
322                 double meanNiet = mean(antwoordenlijstNiet);
323                 double standaardDefNiet = getStd(antwoordenlijstNiet, meanNiet);
324                 System.out.println("dit is mean " + meanNiet + " (" + standaardDefNiet + "  NietWAchten");
325 
326                 double meanDepot = mean(antwoordenlijstDepot);
327                 double standaardDefDepot = getStd(antwoordenlijstDepot, meanDepot);
328                 System.out.println("dit is mean " + meanDepot + " (" + standaardDefDepot + "  Depot");
329 
330                 double meanMax = mean(antwoordenlijstMax);
331                 double standaardDefMax = getStd(antwoordenlijstMax, meanMax);
332                 System.out.println("dit is mean " + meanMax + " (" + standaardDefMax+ "  MAx distance");
333 
334                 double meanLoc = mean(antwoordenlijstLoc);
335                 double standaardDefLoc = getStd(antwoordenlijstLoc, meanLoc);
336                 System.out.println("dit is mean " + meanLoc + " (" + standaardDefLoc + "  locatie");
337 
338                 double meanDist = mean(antwoordenlijstDist);
339                 double standaardDefDist = getStd(antwoordenlijstDist, meanDist);
340                 System.out.println("dit is mean " + meanDist + " (" + standaardDefDist + "  distance");
341 
342                 double meanVari = mean(antwoordenlijstVari);
343                 double standaardDefVari = getStd(antwoordenlijstVari, meanVari);
344                 System.out.println("dit is mean " + meanVari + " (" + standaardDefVari + "  Variable");
345 
346         }
347 
348         public static double mean(int[] lijst) {
349 
350                 int totaal = 0;
351                 for (int getal : lijst) {
352                         totaal = totaal + getal;
353                 }
354                 double dTotaal = totaal;
355                 double tussen = lijst.length;
356                 return dTotaal / tussen;
357         }
358 
359         public static double getStd(int[] lijst, double mean) {
360 
361                 double temp = 0;
362                 for (int a : lijst) {
363                         double b = a;
364                         temp += (b - mean) * (b - mean);
365                 }
366                 double tussen = lijst.length;
367                 return Math.sqrt(temp / (tussen - 1));
368         }
369 
370         public static int antwoord(int timeLimit, int[][] routes, int[][] afstanden, int[] totalTimeRoute, int[][] locaties,
371                         double[] wachtStrategie, int[] voerPerRout, Random rnd) {
372                 int uitslag = 0;
373 
374                 for (int i = 0; i < 1000; i++) {
375 
376                         int[] extraKlant3 = extraKlant(rnd);
377 //                      int[] extraKlant = {52,13};
378                         int klantArrival3 = klantArrival(rnd, timeLimit);
379 //                      System.out.println(" dit is arrival" + klantArrival);
380                         if (basis(extraKlant3, klantArrival3, timeLimit, routes, afstanden, totalTimeRoute, locaties,
381                                         wachtStrategie, voerPerRout)) {
382                                 uitslag++;
```

```
383                         }
384
385                 }
386
387                 return uitslag;
388         }
389
390         public static int percentage(int[] arrivalTime, int[][] nieuweKlant, int timeLimit, int[][] routes,
391                         int[][] afstanden, int[] totalTimeRoute, int[][] locaties, double[] wachtStrategie, int[] voerPerRout) {
392                 int som = 0;
393                 for (int i = 0; i < 100; i++) {
394 //                      System.out.println(" dit is arrival" + klantArrival);
395                         if (basis(nieuweKlant[i], arrivalTime[i], timeLimit, routes, afstanden, totalTimeRoute, locaties,
396                                         wachtStrategie, voerPerRout)) {
397                                 som++;
398                         }
399
400                 }
401
402                 return som;
403         }
404
405         public static ArrayList<Klant> evolutionAlgoritme(ArrayList<Klant> generatie, int[] voerPerRout, int timeLimit,
406                         int[][] routes, int[][] afstanden, int[] totalTimeRoute, int[][] locaties, Random rnd) {
407
408                 int totFitness = fitness(generatie);
409                 ArrayList<Klant> nieuwGeneratie = new ArrayList<>();
410                 nieuwGeneratie.add(generatie.get(0));
411                 int[] arrivalTime = arrivalTime(timeLimit, rnd);
412                 int[][] nieuweKlant = honderdExtraKlant(timeLimit, rnd);
413
414                 for (int i = 0; i < 99; i++) {
415 //                      int roulIndex1 = roulette2(generatie, rnd);
416 //                      int roulIndex2 = roulette2(generatie, rnd);
417 //                      while (roulIndex1 == roulIndex2) {
418 //                              roulIndex2 = roulette(generatie, totFitness, rnd);
419 //                      }
420                         int roulIndex1 = roulette(generatie, totFitness, rnd);
421                         int roulIndex2 = roulette(generatie, totFitness, rnd);
422                         while (roulIndex1 == roulIndex2) {
423                                 roulIndex2 = roulette(generatie, totFitness, rnd);
424                         }
425                         double[] kindWacht = kinderenMaken(generatie.get(roulIndex1).wachttijd, generatie.get(roulIndex2).wachttijd,
426                                         voerPerRout, rnd);
427                         int percentageKind = percentage(arrivalTime, nieuweKlant, timeLimit, routes, afstanden, totalTimeRoute,
428                                         locaties, kindWacht, voerPerRout);
429                         Klant kind = new Klant(kindWacht, percentageKind);
430                         nieuwGeneratie.add(kind);
431                 }
432                 Collections.sort(nieuwGeneratie, new SortbyGeholpen());
433
434                 return nieuwGeneratie;
435         }
436
437         public static ArrayList<Klant> evolutionAlgoritme2(ArrayList<Klant> generatie, int[] voerPerRout, int timeLimit,
438                         int[][] routes, int[][] afstanden, int[] totalTimeRoute, int[][] locaties, Random rnd) {
439
440                 int totFitness = fitness(generatie);
441                 ArrayList<Klant> nieuwGeneratie = new ArrayList<>();
442                 nieuwGeneratie.add(generatie.get(0));
443                 int[] arrivalTime = arrivalTime(timeLimit, rnd);
444                 int[][] nieuweKlant = honderdExtraKlant(timeLimit, rnd);
445
446                 for (int i = 0; i < 99; i++) {
447                         int roulIndex1 = roulette2(generatie, rnd);
448                         int roulIndex2 = roulette2(generatie, rnd);
449                         while (roulIndex1 == roulIndex2) {
450                                 roulIndex2 = roulette(generatie, totFitness, rnd);
451                         }
452
453                         double[] kindWacht = kinderenMaken(generatie.get(roulIndex1).wachttijd, generatie.get(roulIndex2).wachttijd,
454                                         voerPerRout, rnd);
455                         int percentageKind = percentage(arrivalTime, nieuweKlant, timeLimit, routes, afstanden, totalTimeRoute,
456                                         locaties, kindWacht, voerPerRout);
457                         Klant kind = new Klant(kindWacht, percentageKind);
458                         nieuwGeneratie.add(kind);
459                 }
```

```
460                 Collections.sort(nieuwGeneratie, new SortbyGeholpen());
461
462                 return nieuwGeneratie;
463         }
464
465
466         public static ArrayList<Klant> evolutionAlgoritmeTournament1(ArrayList<Klant> generatie, int[] voerPerRout,
467                         int timeLimit, int[][] routes, int[][] afstanden, int[] totalTimeRoute, int[][] locaties, Random rnd) {
468
469                 ArrayList<Klant> nieuwGeneratie = new ArrayList<>();
470                 nieuwGeneratie.add(generatie.get(0));
471                 int[] arrivalTime = arrivalTime(timeLimit, rnd);
472                 int[][] nieuweKlant = honderdExtraKlant(timeLimit, rnd);
473
474                 for (int i = 1; i < 100; i++) {
475                         ArrayList<Klant> ouders = tournament1(generatie, rnd);
476
477                         double[] kindWacht = kinderenMaken(ouders.get(0).wachttijd, ouders.get(1).wachttijd, voerPerRout, rnd);
478                         int percentageKind = percentage(arrivalTime, nieuweKlant, timeLimit, routes, afstanden, totalTimeRoute,
479                                         locaties, kindWacht, voerPerRout);
480                         Klant kind = new Klant(kindWacht, percentageKind);
481                         nieuwGeneratie.add(kind);
482                 }
483                 Collections.sort(nieuwGeneratie, new SortbyGeholpen());
484
485                 return nieuwGeneratie;
486         }
487         public static ArrayList<Klant> evolutionAlgoritmeTournament2(ArrayList<Klant> generatie, int[] voerPerRout,
488                         int timeLimit, int[][] routes, int[][] afstanden, int[] totalTimeRoute, int[][] locaties, Random rnd) {
489
490                 ArrayList<Klant> nieuwGeneratie = new ArrayList<>();
491                 nieuwGeneratie.add(generatie.get(0));
492                 int[] arrivalTime = arrivalTime(timeLimit, rnd);
493                 int[][] nieuweKlant = honderdExtraKlant(timeLimit, rnd);
494
495                 for (int i = 1; i < 100; i++) {
496                         if (i % 10 == 0) {
497                                 generatie.remove(0);
498                         }
499
500                         ArrayList<Klant> ouders = tournament1(generatie, rnd);
501
502                         double[] kindWacht = kinderenMaken(ouders.get(0).wachttijd, ouders.get(1).wachttijd, voerPerRout, rnd);
503                         int percentageKind = percentage(arrivalTime, nieuweKlant, timeLimit, routes, afstanden, totalTimeRoute,
504                                         locaties, kindWacht, voerPerRout);
505                         Klant kind = new Klant(kindWacht, percentageKind);
506                         nieuwGeneratie.add(kind);
507                 }
508                 Collections.sort(nieuwGeneratie, new SortbyGeholpen());
509
510                 return nieuwGeneratie;
511         }
512
513         public static int fitness(ArrayList<Klant> generatie) {
514                 int som = 0;
515                 for (Klant klant : generatie) {
516                         som = som + klant.geholpen;
517                 }
518                 return som;
519         }
520
521         public static double fitness2() {
522                 double harmonische = 0.0;
523                 for (int i1 = 1; i1 <= 100; i1++) {
524                         double tussen1 = i1;
525                         double tussen = (1 / tussen1);
526                         harmonische = harmonische + tussen;
527                 }
528                 return harmonische;
529         }
530
531         public static ArrayList<Klant> tournament1(ArrayList<Klant> generatie, Random rnd) {
532
533                 ArrayList<Klant> tour1 = new ArrayList<>();
534                 ArrayList<Klant> tour2 = new ArrayList<>();
535                 ArrayList<Klant> ouders = new ArrayList<>();
536                 for (Klant ouder : generatie) {
```

```
537
538                         if (rnd.nextDouble() <= 0.5) {
539                                 tour1.add(ouder);
540                         } else {
541                                 tour2.add(ouder);
542                         }
543                 }
544
545                 Collections.sort(tour1, new SortbyGeholpen());
546                 Collections.sort(tour2, new SortbyGeholpen());
547
548                 ouders.add(tour1.get(0));
549                 ouders.add(tour2.get(0));
550
551                 return ouders;
552
553         }
554
555         public static int roulette(ArrayList<Klant> generatie, int totFitness, Random rnd) {
556
557                 int p = rnd.nextInt(totFitness);
558                 int index = 0;
559                 for (int i = 0; i < generatie.size(); i++) {
560                         if (p <= 0) {
561                                 index = i;
562                                 break;
563                         } else {
564                                 p = p - generatie.get(i).geholpen;
565
566                         }
567
568                 }
569                 return index;
570
571 //              Random rnd = new Random();
572 //              double per = rnd.nextDouble();
573 //              double p = 0;
574 //
575 //              int index = 0;
576 //              for (int i = 0; i < generatie.size(); i++) {
577 //                      double tussen1= generatie.get(i).geholpen;
578 //                      double tussen2= totFitness;
579 //                      double percen= tussen1/tussen2;
580 //                      p = p + percen;
581 //                      if( p<= per) {
582 //                              index =i;
583 //                              break;
584 //                      }
585 //
586 //
587 //
588 //
589 //              }
590 //              return index;
591         }
592
593         public static int roulette2(ArrayList<Klant> generatie, Random rnd) {
594
595                 double totFitness = fitness2();
596                 double p = rnd.nextDouble() * totFitness;
597                 int index = 0;
598                 for (int i = 0; i < generatie.size(); i++) {
599                         if (p <= 0) {
600                                 index = i;
601                                 break;
602                         } else {
603                                 double tussen = i;
604                                 double tussen2 = 1 / tussen;
605                                 p = p - tussen2;
606
607                         }
608
609                 }
610                 return index;
611         }
612
613         public static double[] kinderenMaken(double[] element1, double[] element2, int[] voerPerRout, Random rnd) {
```

```
614
615                 int getal1 = rnd.nextInt(element1.length);
616                 int getal2 = rnd.nextInt(element1.length);
617                 if (getal1 <= getal2) {
618                         for (int i = getal1; i <= getal2; i++) {
619                                 element1[i] = element2[i];
620                         }
621
622                 } else {
623                         for (int i = getal2; i <= getal1; i++) {
624                                 element1[i] = element2[i];
625                         }
626                 }
627                 for (int i = 0; i < element1.length; i++) {
628                         element1[i] = element1[i] +( rnd.nextGaussian()*0.01);
629                 }
630                 return normaliseren(voerPerRout, element1);
631         }
632
633         public static ArrayList<Klant> StartGeneratieEA2(int[][] routes, int[][] afstanden, int[] totalTimeRoute,
634                         int[] voerPerRout, int klanten, Random rnd) {
635                 ArrayList<Klant> lijst = new ArrayList<>();
636
637                 int waarde = (voerPerRout.length * 2) + klanten;
638                 double[] array1 = nietWachtenStra(routes, voerPerRout, klanten);
639                 Klant klant1 = new Klant(array1, 0);
640                 lijst.add(klant1);
641
642                 double[] array2 = depotStra(routes, voerPerRout, klanten);
643                 Klant klant2 = new Klant(array2, 0);
644                 lijst.add(klant2);
645
646                 double[] array3 = maxDistStra(routes, afstanden, voerPerRout, klanten);
647                 Klant klant3 = new Klant(array3, 0);
648                 lijst.add(klant3);
649
650                 double[] array4 = locationStra(routes, voerPerRout, klanten);
651                 Klant klant4 = new Klant(array4, 0);
652                 lijst.add(klant4);
653
654                 double[] array5 = distanceStra(routes, afstanden, totalTimeRoute, voerPerRout, klanten);
655                 Klant klant5 = new Klant(array5, 0);
656                 lijst.add(klant5);
657
658                 double[] array6 = variableStra(routes, afstanden, totalTimeRoute, voerPerRout, klanten);
659                 Klant klant6 = new Klant(array6, 0);
660                 lijst.add(klant6);
661
662                 for (int i = 0; i < 94; i++) {
663
664                         double[] array = new double[waarde];
665                         for (int j = 0; j < waarde; j++) {
666                                 array[j] = rnd.nextDouble();
667                         }
668                         double[] tussen = normaliseren(voerPerRout, array);
669                         int geholpen = 0;
670                         Klant klant = new Klant(tussen, geholpen);
671                         lijst.add(klant);
672                 }
673
674                 return lijst;
675         }
676
677         public static ArrayList<Klant> StartGeneratie(int[] voerPerRout, int klanten, Random rnd) {
678                 ArrayList<Klant> lijst = new ArrayList<>();
679
680                 int waarde = (voerPerRout.length * 2) + klanten;
681                 for (int i = 0; i < 100; i++) {
682
683                         double[] array = new double[waarde];
684                         for (int j = 0; j < waarde; j++) {
685                                 array[j] = rnd.nextDouble();
686                         }
687                         double[] tussen = normaliseren(voerPerRout, array);
688                         int geholpen = 0;
689                         Klant klant = new Klant(tussen, geholpen);
690                         lijst.add(klant);
```

```
691                 }
692
693                 return lijst;
694         }
695
696         public static double[] normaliseren(int[] voerPerRout, double[] lijst) {
697
698                 int index = 0;
699                 for (int i = 0; i < voerPerRout.length; i++) {
700                         double som = 0;
701
702                         for (int j = 0; j < voerPerRout[i] + 2; j++) {
703
704                                 som = som + Math.abs(lijst[index + j]);
705                         }
706                         for (int j = 0; j < voerPerRout[i] + 2; j++) {
707                                 double tussen = Math.abs(lijst[index + j]);
708                                 lijst[index + j] = tussen / som;
709                         }
710                         index = index + voerPerRout[i] + 2;
711                 }
712                 return lijst;
713         }
714
715         public static boolean basis(int[] extraKlant, int klantArrival, int timeLimit, int[][] routes, int[][] afstanden,
716                         int[] totalTimeRoute, int[][] locaties, double[] wachtStrategie, int[] voerPerRout) {
717                 int waarde = 0;
718                 for (int i = 0; i < routes.length; i++) {
719                         int slackTime = timeLimit - totalTimeRoute[i];
720                         int tijdenlijk = 0;
721                         int time = 0;
722                         int j = 0;
723                         while (time <= timeLimit && j <= routes[0].length) {
724                                 if (slackTime == 0) {
725                                         break;
726                                 } else if (j != 0 && routes[i][j] == 0) {
727                                         slackTime = timeLimit - klantArrival;
728                                         if (toevoegenEind(extraKlant, locaties, slackTime)) {
729                                                 return true;
730                                         } else {
731                                                 break;
732                                         }
733
734                                 } else {
735                                         if (wachtStrategie[waarde + j] != 0) {
736 //                                              System.out.println("dit is wacht: " + wachtStrategie[i][j] + " dit plaats:  " + j);
737                                                 tijdenlijk = (int) (time + wachtStrategie[waarde + j] * slackTime);
738
739                                                 if (klantArrival <= tijdenlijk) {
740                                                         slackTime = slackTime - (klantArrival - time);
741                                                         if (toevoegen(routes[i], extraKlant, slackTime, j, locaties, afstanden)) {
742
743                                                                 return true;
744                                                         } else {
745                                                                 break;
746                                                         }
747
748                                                 }
749                                                 time = tijdenlijk;
750                                                 slackTime = (int) (slackTime - wachtStrategie[waarde + j] * slackTime);
751
752                                         }
753
754                                         int begin = routes[i][j];
755                                         int eind = routes[i][j + 1];
756                                         int afstand = afstanden[begin][eind];
757                                         tijdenlijk = time + afstand;
758 //                                      System.out.println("dit is de tijd" + tijdenlijk);
759
760                                         if (klantArrival <= tijdenlijk) {
761                                                 // kijken of het kan worden toegevoegd tussen de routes of eind van de route
762                                                 int presentTime = klantArrival - time;
763 //                                              System.out.println("dit is presentTime " +  presentTime);
764 //                                              System.out.println("dit is begin "+ Arrays.toString(locaties[begin] )+ "eind " +
        Arrays.toString(locaties[eind] ));
765                                                 int[] positieVoertuig = positieRoute(locaties[begin], locaties[eind], presentTime, afstand);
766
```

```
767                                             int detourBegin = detour(positieVoertuig, locaties[eind], extraKlant);
768   //                                        System.out.println();
769   //                                        System.out.println("dit is detour eerste "+ detourBegin );
770   //                                        System.out.println("dit is voertuig locatie "+ Arrays.toString(positieVoertuig));
771                                             if (detourBegin <= slackTime + afstanden[begin][eind] - presentTime) {
772                                                     return true;
773
774                                             }
775                                             if (toevoegen(routes[i], extraKlant, slackTime, j + 1, locaties, afstanden)) {
776
777                                                     return true;
778                                             } else {
779                                                     break;
780                                             }
781
782                                     }
783
784                             }
785                             time = tijdenlijk;
786                             j++;
787                     }
788                     waarde = waarde + voerPerRout[i] + 2;
789             }
790
791             return false;
792
793     }
794
795     public static double[] nietWachtenStra(int[][] routes, int[] voerPerRout, int klanten) {
796             double[] wachttijden = new double[(routes.length * 2) + klanten];
797             int waarde = 0;
798
799             for (int i = 0; i < routes.length; i++) {
800                     wachttijden[waarde + voerPerRout[i] + 1] = 1;
801                     waarde = waarde + voerPerRout[i] + 2;
802             }
803
804   //        double[][] wachttijden = new double[routes.length][routes[0].length];
805   //        for (int i = 0; i < routes.length; i++) {
806   //
807   //                wachttijden[i][routes[0].length - 1] = 1; // niet wachten
808   //        }
809
810             return wachttijden;
811
812     }
813
814     public static double[] depotStra(int[][] routes, int[] voerPerRout, int klanten) {
815             double[] wachttijden = new double[(routes.length * 2) + klanten];
816             int waarde = 0;
817
818             for (int i = 0; i < routes.length; i++) {
819                     wachttijden[waarde] = 1;
820                     waarde = waarde + voerPerRout[i] + 2;
821             }
822   //        double[][] wachttijden = new double[routes.length][routes[0].length];
823   //        for (int i = 0; i < routes.length; i++) {
824   //
825   //                wachttijden[i][0] = 1; // niet wachten
826   //        }
827   //
828             return wachttijden;
829
830     }
831
832     public static double[] maxDistStra(int[][] routes, int[][] afstanden, int[] voerPerRout, int klanten) {
833             double[] wachttijden = new double[(routes.length * 2) + klanten];
834             int waarde = 0;
835             for (int i = 0; i < routes.length; i++) {
836                     int max = 0;
837                     int plek = 0;
838                     for (int j = 0; j < routes[0].length; j++) {
839                             if (max < afstanden[0][routes[i][j]]) {
840                                     max = afstanden[0][routes[i][j]];
841                                     plek = j;
842                             }
843
```

```
844                         }
845
846                         wachttijden[waarde + plek] = 1; // niet wachten
847                         waarde = waarde + voerPerRout[i] + 2;
848                 }
849
850                 return wachttijden;
851
852         }
853
854         public static double[] locationStra(int[][] routes, int[] voerPerRoute, int klanten) {
855                 double[] wachttijden = new double[(routes.length * 2) + klanten];
856                 int waarde = 0;
857                 for (int i = 0; i < routes.length; i++) {
858                         for (int j = 0; j <= voerPerRoute[i]; j++) {
859                                 double tussen2 = voerPerRoute[i];
860                                 double tussen = (1 / tussen2);
861                                 wachttijden[waarde + j] = tussen;
862
863                         }
864                         waarde = waarde + voerPerRoute[i] + 2;
865                 }
866
867                 return wachttijden;
868
869         }
870
871         public static double[] distanceStra(int[][] routes, int[][] afstanden, int[] totalTimeRoute, int[] voerPerRoute,
872                         int klanten) {
873                 double[] wachttijden = new double[(routes.length * 2) + klanten];
874                 int waarde = 0;
875                 for (int i = 0; i < routes.length; i++) {
876                         int index = 0;
877                         for (int j = 1; j <= voerPerRoute[i]; j++) {
878                                 int begin = routes[i][index];
879                                 int eind = routes[i][j];
880
881                                 double tussen = afstanden[begin][eind];
882                                 double tussen2 = totalTimeRoute[i];
883                                 wachttijden[waarde + j] = tussen / tussen2;
884                                 index++;
885                         }
886                         waarde = waarde + voerPerRoute[i] + 2;
887                 }
888
889                 return wachttijden;
890
891         }
892
893         public static double[] variableStra(int[][] routes, int[][] afstanden, int[] totalTimeRoute, int[] voerPerRoute,
894                         int klanten) {
895                 double[] wachttijden = new double[(routes.length * 2) + klanten];
896                 int waarde = 0;
897                 int timeLimit = maximum(totalTimeRoute);
898                 for (int i = 0; i < routes.length; i++) {
899
900                         int time = 0;
901                         int slack = timeLimit - totalTimeRoute[i];
902                         int omslag = totalTimeRoute[i] - slack;
903                         boolean bool = false;
904
905                         for (int j = 0; j <= voerPerRoute[i]; j++) {
906
907                                 int begin = routes[i][j];
908                                 int eind = routes[i][j + 1];
909                                 int tijdenlijk = time + afstanden[begin][eind];
910
911                                 if (tijdenlijk >= omslag && slack != 0) {
912                                         double slack1 = slack;
913                                         if (bool == false) {
914                                                 bool = true;
915                                                 double tussenAf = tijdenlijk - omslag;
916                                                 wachttijden[waarde + j + 1] = tussenAf / slack1;
917
918                                         } else {
919                                                 double tussen1 = afstanden[begin][eind];
920                                                 wachttijden[waarde + j + 1] = tussen1 / slack1;
```

```
921                                     }
922
923                             }
924                             time = time + afstanden[begin][eind];
925
926                     }
927                     waarde = waarde + voerPerRoute[i] + 2;
928             }
929
930             return wachttijden;
931     }
932
933     public static boolean toevoegenEind(int[] extraKlant, int[][] locaties, int slackTime) {
934             if (detour(locaties[0], locaties[0], extraKlant) <= slackTime) {
935                     return true;
936             } else {
937                     return false;
938             }
939     }
940
941     public static boolean toevoegen(int[] route, int[] extraKlant, int slackTime, int plek, int[][] locaties,
942                     int[][] afstand) {
943             while (plek < route.length) {
944                     if (plek != 0 && route[plek] == 0) {
945                             if (toevoegenEind(extraKlant, locaties, slackTime)) {
946                                     return true;
947                             } else {
948                                     return false;
949                             }
950                     } else {
951                             int begin = route[plek];
952                             int eind = route[plek + 1];
953                             int detourAfstand = detour(locaties[begin], locaties[eind], extraKlant);
954                             if (detourAfstand <= slackTime + afstand[begin][eind]) {
955                                     return true;
956                             }
957                     }
958                     plek++;
959
960             }
961             return false;
962     }
963
964     public static int[] totalTime(int voertuigen, int[][] afstanden, int[][] routes, int[] voerPerRout) {
965             int[] totaal = new int[voertuigen];
966
967             for (int i = 0; i < voertuigen; i++) {
968                     int som = 0;
969                     for (int j = 0; j < voerPerRout[i] + 1; j++) {
970
971                             int begin = routes[i][j];
972                             int eind = routes[i][j + 1];
973 //                          System.out.println(begin);
974 //                          System.out.println(eind);
975
976                             System.out.print(som + "  ");
977                             som += afstanden[begin][eind];
978                     }
979                     System.out.print(som + "  ");
980                     System.out.println();
981                     totaal[i] = som;
982             }
983
984             return totaal;
985     }
986
987     public static int maximum(int[] array) {
988             int max = 0;
989             for (int nummer : array) {
990                     max = Math.max(max, nummer);
991             }
992             return max;
993     }
994
995     public static int[][] getDistanceMatrix(int[][] xy, int n) {
996
997             int[][] distances = new int[n][n];
```

```
998                 for (int i = 0; i < n; i++) {
999                         for (int j = 0; j < n; j++) {
1000
1001                                 distances[i][j] = (int) Math
1002                                                 .round(Math.sqrt(Math.pow(xy[i][0] - xy[j][0], 2) + Math.pow(xy[i][1] - xy[j][1], 2)));
1003
1004                         }
1005                 }
1006                 return distances;
1007         }
1008
1009         public static int[] positieRoute(int[] locatie1, int[] locatie2, int tijd, int afstand) {
1010                 double xPos = (locatie2[0] - locatie1[0]);
1011                 double yPos = (locatie2[1] - locatie1[1]);
1012                 xPos = xPos / afstand;
1013                 yPos = yPos / afstand;
1014 //              System.out.println("dit is afstand " + afstand);
1015 //              System.out.println("dit is xPos" + xPos + " dit is ypos " + yPos);
1016                 int[] answer = new int[2];
1017                 answer[0] = locatie1[0] + (int) Math.round(xPos * tijd);
1018                 answer[1] = locatie1[1] + (int) Math.round(yPos * tijd);
1019
1020                 return answer;
1021         }
1022
1023         public static int detour(int[] locatie1, int[] locatie2, int[] detour) {
1024                 int detourAfstand = 0;
1025                 detourAfstand = (int) Math
1026                                 .round(Math.sqrt(Math.pow(locatie1[0] - detour[0], 2) + Math.pow(locatie1[1] - detour[1], 2))
1027                                                 + Math.sqrt(Math.pow(detour[0] - locatie2[0], 2) + Math.pow(detour[1] - locatie2[1], 2)));
1028                 return detourAfstand;
1029         }
1030
1031         public static int[][] gegevens(int klanten, int voertuigen, File bestand) throws FileNotFoundException {
1032                 int[][] locaties = new int[klanten + 1][2];
1033
1034                 Scanner scan = new Scanner(bestand);
1035                 int getal = 0;
1036
1037                 while (scan.hasNextLine()) {
1038                         String line = scan.nextLine();
1039 //                      String[] deel = line.split("  ");
1040                         String[] deel = line.split(" ");
1041 //                      if(getal==0) {
1042 //                              klanten = Integer. parseInt (deel[0]);
1043 //
1044 //                              getal++; }else
1045                         if (getal == 2) {
1046 //                              System.out.println(deel[0]);
1047 //                              System.out.println(deel[1]);
1048
1049                                 locaties[getal - 2][0] = Integer.parseInt(deel[0]);
1050                                 locaties[getal - 2][1] = Integer.parseInt(deel[1]);
1051                                 minimumx = Math.min(minimumx, Integer.parseInt(deel[0]));
1052                                 minimumy = Math.min(minimumy, Integer.parseInt(deel[1]));
1053                                 maximumx = Math.max(maximumx, Integer.parseInt(deel[0]));
1054                                 maximumy = Math.max(maximumy, Integer.parseInt(deel[1]));
1055                         } else if (getal > 2 && getal < klanten + 3) { // dit veranderen per ding
1056
1057                                 locaties[getal - 2][0] = Integer.parseInt(deel[1]);
1058                                 locaties[getal - 2][1] = Integer.parseInt(deel[2]);
1059                                 minimumx = Math.min(minimumx, Integer.parseInt(deel[1]));
1060                                 minimumy = Math.min(minimumy, Integer.parseInt(deel[2]));
1061                                 maximumx = Math.max(maximumx, Integer.parseInt(deel[1]));
1062                                 maximumy = Math.max(maximumy, Integer.parseInt(deel[2]));
1063
1064                         }
1065                         getal++;
1066
1067                 }
1068                 scan.close();
1069                 return locaties;
1070
1071         }
1072
1073         public static int klantArrival(Random rnd, int timeLimit) {
1074                 return rnd.nextInt(timeLimit + 1);
```

```
1075            }
1076
1077            public static int[] arrivalTime(int timeLimit, Random rnd) {
1078                    int[] arrivalTime = new int[100];
1079
1080                    for (int i = 0; i < 100; i++) {
1081                            arrivalTime[i] = rnd.nextInt(timeLimit + 1);
1082    //                      System.out.println(arrivalTime[i]);
1083                    }
1084                    return arrivalTime;
1085            }
1086
1087            public static int[] extraKlant(Random rnd) {
1088                    int[] klant = new int[2];
1089
1090                    klant[0] = rnd.nextInt(maximumx - minimumx + 1) + minimumx;
1091                    klant[1] = rnd.nextInt(maximumy - minimumy + 1) + minimumy;
1092                    return klant;
1093            }
1094
1095            public static int[][] honderdExtraKlant(int timeLimit, Random rnd) {
1096
1097                    int[][] nieuweKlant = new int[100][2];
1098                    for (int i = 0; i < 100; i++) {
1099
1100                            int x = rnd.nextInt(maximumx - minimumx + 1) + minimumx;
1101                            int y = rnd.nextInt(maximumy - minimumy + 1) + minimumy;
1102                            nieuweKlant[i][0] = x;
1103                            nieuweKlant[i][1] = y;
1104                    }
1105
1106                    return nieuweKlant;
1107
1108            }
1109
1110    }
```

Listing 2: Klant code

```
 1
 2   import java.util.Comparator;
 3
 4   public class Klant{
 5           double[] wachttijd;
 6           int geholpen;
 7           public Klant(double[] wachttijd, int geholpen) {
 8                   this.wachttijd= wachttijd;
 9                   this.geholpen= geholpen;
10           }
11
12           public double[] krijgWacht() {
13                   return this.wachttijd;
14           }
15
16           public void setWacht(double[] set) {
17                   this.wachttijd= set;
18
19           }
20
21           public void setGeholpen(int nieuw) {
22                   this.geholpen = nieuw;
23           }
24
25   }
26   class SortbyGeholpen implements Comparator<Klant> {
27       // Used for sorting in ascending order of
28       // roll number
29       public int compare(Klant a, Klant b)
30       {
31           return -(a.geholpen - b.geholpen);
32       }
33   }
```