ORTEC

Roosters genereren in HARMONY: De eerste stap van het huidige algoritme

Stageverslag van Laurenz Eveleens
Bedrijfswiskunde & Informatica
Vrije Universiteit Amsterdam
augustus 2005

Executive summary

This master thesis deals with generating rosters. At ORTEC rosters are generated in three steps. Step 1 is the bottleneck method, step 2 is a genetic algorithm and step 3 is local optimization. Four new methods have been developed to improve step 1. These are bottleneck with local optimization, stochastic bottleneck, random and abstract roster.

The bottleneck method is the best of the methods if the user wants to make a roster without using the genetic algorithm. The quality of the roster made with the bottleneck with local optimization is of better quality then the rosters of the bottleneck method. The time it takes to make the rosters however is too long to be of use.

If the user wants to make rosters using the genetic algorithm it is recommended to give the user two options. The stochastic bottleneck method is recommended if the user has a limited amount of time. If the amount of time used by the algorithm is less important to the user the random method should be used.

To make rosters with the abstract roster method is not recommended at this time. Before this method is able to make better rosters the method needs to be developed further. The expectation is that this will decrease the amount of time needed enormously. It should be possible to reach and surpass the level of quality of the random method.

Management samenvatting

In dit afstudeerverslag hebben we gekeken naar het genereren van roosters. Bij ORTEC gebeurt dit in drie stappen. Stap 1 is de bottleneck methode, stap 2 een genetisch algoritme en stap 3 is locale optimalisatie. Voor stap 1 zijn vier nieuwe methoden ontwikkeld. Dit zijn de bottleneck met locale optimalisatie, stochastische bottleneck, volledig willekeurig en abstract rooster.

Als de gebruiker een rooster wil maken zonder stappen 2 en 3 uit te voeren, is de bottleneck methode het meest geschikt. Hoewel de kwaliteit van het rooster gemaakt met de bottleneck methode met locale optimalisatie beter is, duurt het maken van dit rooster te lang om gebruikt te kunnen worden.

Als de gebruiker een rooster wil maken door gebruik te maken van het genetisch algoritme is het aan te bevelen om de gebruiker twee mogelijkheden te geven. Heeft de gebruiker een beperkte hoeveelheid tijd dan is het aan te raden om de stochastische bottleneck methode te gebruiken. Als de tijd die nodig is voor het maken van een rooster minder belangrijk is dan valt de volledig willekeurige methode aan te bevelen.

Het maken van roosters met de abstracte rooster methode moet momenteel niet worden gebruikt. Voordat met deze methode betere roosters kunnen worden gemaakt zal de methode verder ontwikkeld moeten worden. De verwachting is dat hierdoor de benodigde tijd sterk kan worden teruggedrongen. Voor de kwaliteit moet het mogelijk zijn het kwaliteitsniveau van de volledig willekeurig methode te kunnen overtreffen.

Voorwoord

Dit verslag vormt de afsluiting van mijn stage bij ORTEC die ik in januari 2005 begon. De stage is een onderdeel van de studie Bedrijfswiskunde en Informatica aan de Vrije Universiteit in Amsterdam. Deze stage vormt tevens de afsluiting van bovengenoemde studie.

Bij deze wil ik ook een aantal mensen hartelijk danken voor hun hulp. Zonder hen was dit stageverslag niet tot stand gekomen.

- Laurens Fijn van Draat. Hij heeft mij van vanuit ORTEC begeleid. Hij heeft meegedacht over het probleem, de oplossing, het programmeren en het schrijven van het stageverslag. Daarnaast heeft hij me geholpen binnen ORTEC wegwijs te maken.
- Sandjai Bhulai. Mijn begeleider vanuit de Vrije Universiteit. Ook hij heeft meegedacht over het probleem en de mogelijke manieren om verbetering te vinden. Ook heeft hij waardevolle op- en aanmerkingen gegeven op het stageverslag.

Daarnaast bedank ik ORTEC voor het bieden van deze stage. Ook de collega's bij ORTEC OSD wil ik enorm bedanken voor de leuke tijd die ik er heb mogen meemaken. De sfeer is open, waardoor het als nieuwkomer gemakkelijk is je thuis te voelen binnen de afdeling. De behulpzaamheid van iedereen heb ik zeer gewaardeerd.

Daarnaast bedank ik mijn ouders die het voor mij mogelijk hebben gemaakt om deze studie te doen en die mij alle steun hebben gegeven die ik nodig had tijdens mijn stage. Ook mijn broer, zus, hun aanhang en mijn vrienden wil ik bedanken voor hun steun en motivatie.

Laurenz Eveleens, augustus 2005

Index

1. Inleiding

De opdracht van de stage heeft betrekking op het maken van personeelsroosters. ORTEC heeft een software product genaamd HARMONY dat het mogelijk maakt roosters te genereren. Hiervoor gebruikt het programma een genetisch algoritme. Het genetisch algoritme heeft een aantal beginroosters nodig als startpunt. De indruk bij ORTEC is dat de kwaliteit van het uiteindelijke rooster kan verbeteren door met betere beginroosters het genetisch algoritme te starten. Doel van de stage is te onderzoeken of er betere beginroosters te maken zijn en of dit de uiteindelijke kwaliteit van het rooster kan verbeteren. In dit stageverslag staat een uitvoerige beschrijving van de uitwerking van deze opdracht. Hieronder staat beschreven hoe het verslag is opgebouwd.

Het hoofdstuk 'Stageplek: ORTEC en HARMONY' bevat een korte beschrijving van het bedrijf ORTEC. Ook komt het programma HARMONY, ontwikkeld door ORTEC, aan bod. Dit programma vormt de basis voor de opdracht en de stage.

Het hoofdstuk 'Opdrachtomschrijving' behandelt de probleemomschrijving die in samenwerking met ORTEC is geformuleerd. Doel van de stage is te onderzoeken of betere beginroosters te maken zijn waarmee de kwaliteit of benodigde tijd voor het maken van een rooster verbeterd kan worden.

Het hoofdstuk 'Huidige planautomaat' legt de werking uit van de planautomaat waar HARMONY gebruik van maakt. De eerste versie van de planautomaat, maar ook de aanpassingen die er in de loop van de tijd aan zijn gemaakt, komen aan bod.

Het hoofdstuk 'Nieuwe methoden' beschrijft de aanpassingen die ik voorstel om een antwoord te vinden op de probleemomschrijving. Hierbij zal ik ook een afweging geven van de voor- en nadelen van de voorgestelde veranderingen. Eén van de voorgestelde veranderingen is een abstract rooster te maken.

Het hoofdstuk 'Maken van abstract rooster' behandelt een aantal methoden uit de literatuur waarmee een abstract rooster te maken is. Bij elke methode is aangegeven hoe de methode toe te passen is om dit te doen. Daarna volgt een afweging van de voor- en nadelen van de verschillende methoden. Ook motiveer ik mijn keuze voor het gebruiken van een genetisch algoritme om het abstracte rooster mee te maken.

Het hoofdstuk 'Implementatie abstract rooster' bespreekt de uitwerking van het genetisch algoritme in detail. Verschillende mutatie/recombinatie en selectie methoden, die tijdens het programmeren zijn gebruikt, komen aan bod. Ook vermeld ik ad hoc resultaten, die tijdens het maken naar voren kwamen.

Het hoofdstuk 'Test opzet' geeft een wiskundige beschrijving van de tests die zijn gedaan. Het beschrijft hoe de data op een statistisch verantwoorde manier is geanalyseerd.

Hoofdstukken 'Abstract rooster instellingen testresultaten', 'Abstract rooster inplannen testresultaten' en 'Beginroosters maken testresultaten' beschrijven de uitkomsten van de tests die zijn gedaan naar aanleiding van de toetsen uit het hoofdstuk 'Testopzet'.

Het hoofdstuk 'Conclusies' vat de opdracht, de tests en de uitkomsten samen en vermeldt de belangrijkste conclusies die hieruit zijn te trekken. Ook geeft het aanbevelingen op welke terreinen meer winst te halen valt.

In de bijlagen is meer informatie te vinden over ORTEC, HARMONY en het genetisch algoritme. Ook is er een bijlage gewijd aan de testinstellingen waarmee de data is verzameld. Deze hoofdstukken zijn bedoeld voor die lezers die onbekend zijn met / meer willen weten over één van deze onderwerpen.

2. Stageplek: ORTEC en HARMONY

Dit hoofdstuk geeft een globale indruk van het bedrijf ORTEC waar ik de stage heb gelopen. In het bijzonder komt het product HARMONY aan bod, die de basis vormt voor dit stageverslag. Voor meer informatie over ORTEC of HARMONY verwijs ik naar de bijlage 'Stageplek ORTEC' en de website van ORTEC: http://www.ortec.nl.

2.1 ORTEC

ORTEC is op 1 april 1981 opgericht door vijf vrienden met als doel econometrie in het bedrijfsleven toe te passen op consultancy basis. Wat aanvankelijk op kleine schaal begon, is uitgegroeid tot een bedrijf met ongeveer 350 medewerkers. Het houdt zich bezig met allerlei projecten die in de kern iets te maken hebben met planning, simulatie of optimalisatie. De kernbezigheden omvatten consultancy, software pakketverkoop, maatwerk en opleidingen. De doelgroep waar ORTEC voor werkt is zeer gevarieerd. Zo behoren ziekenhuizen, olieraffinaderijen, vliegvelden, verzekeringsmaatschappijen, pensioenfondsen, overheid, taxicentrales, banken en spoorwegen tot de clientèle.

In de loop der jaren heeft ORTEC een groot aantal softwarepakketten gebouwd die het verkoopt en onderhoudt. De belangrijkste producten zijn:

- HARMONY, een planning softwarepakket voor workforce management.
- PLANWISE, ontwikkeld voor productie planning, materiaal management, task scheduling en voorraadbeheer.
- ORION, een planningstool voor order genereren en voorraadbeheer.
- SHORTREC, een routeplanningsysteem voor transporten.
- SHORTROUTE, een fleet track & trace systeem.

Mijn stage heb ik gedaan op de afdeling ORTEC Software Development. Daar heb ik me bezig gehouden met het product HARMONY waarover hieronder meer uitleg volgt.

2.2 HARMONY

HARMONY is een geavanceerd planningssysteem voor dienstroostering. HARMONY onderscheidt zich door de mogelijkheid dienstroosters automatisch te genereren. Dit betekent dat een algoritme achter HARMONY de diensten, die ingeroosterd moeten worden, toekent aan de beschikbare medewerkers. Hierbij houdt het systeem rekening met de (complexe) regels uit de arbeidstijdenwet, met individuele wensen van medewerkers en met de kwalificaties die nodig zijn om specifieke diensten uit te voeren. Door zogenoemde roostercriteria kan de gebruiker de planautomaat van HARMONY sturen. De planautomaten die HARMONY aan de gebruiker ter beschikking stelt zijn:

- Inplannen dienstreeksen
- Inplannen open diensten
- Rooster genereren

De drie genoemde planautomaten kunnen apart aangeroepen worden, maar zowel 'Inplannen dienstreeksen' als 'Inplannen open diensten' zijn niet bedoeld om een volledig rooster mee te genereren. Hiervoor is (zoals de naam al aangeeft) 'Rooster genereren' bedoeld. Rooster genereren werkt in twee fases.

In de eerste fase wordt een genetisch algoritme uitgevoerd. De functie hiervan is vooral het vinden van globaal optimale roosters. Dit algoritme maakt gebruik van zowel 'inplannen dienstreeksen' als 'inplannen open diensten'.

De tweede fase is een verbeter fase en is vooral gericht op het lokaal optimaliseren van de oplossing. Het beste in de eerste fase gevonden rooster wordt grondig onderzocht op mogelijke verbeteringen. Als in een slag een verbetering wordt gevonden, wordt hetzelfde nog eens uitgevoerd. Deze fase kan daardoor lang duren, ook in verhouding tot de eerste fase. De verbeteringen die zo gevonden worden, kunnen echter aanzienlijk zijn.

3. Opdrachtomschrijving

Hieronder volgt de opdrachtomschrijving die de aanleiding vormt voor dit stageverslag. Daarnaast zal dit hoofdstuk aangeven hoe de resultaten van de opdracht te meten en vergelijken zijn.

Inleiding

ORTEC maakt en verkoopt het product HARMONY. Dit programma helpt bij het roosteren van medewerkers. Het maakt planningen, houdt de uren van medewerkers bij en controleert of de gemaakte planning voldoet aan de arbeidsvoorwaarden. Eén van de onderdelen van het programma is de planautomaat.

Deze planautomaat plant diensten voor een vastgestelde roosterperiode in bij beschikbare medewerkers. Niet alleen houdt deze planautomaat rekening met regelgeving zoals arbeidstijdenwet en CAO afspraken, maar ook met andere criteria. Zo kunnen medewerkers aangeven op welke dagen ze vrij willen hebben, zodat de planautomaat die dag voor de medewerker vrij probeert te houden.

Definitie Het *inplannen van diensten* of ook wel *inroosteren* betekent het toewijzen van diensten aan medewerkers. Een dienst A op maandag inplannen bij medewerker Jan betekent dat medewerker Jan op maandag dienst A moet uitvoeren.

De planautomaat onderneemt de volgende stappen om een rooster te genereren:

1. Bepaal een aantal beginroosters.
2. Doorloop een genetisch algoritme met deze beginroosters als eerste populatie.
3. Optimaliseer (locaal) het best gevonden rooster.

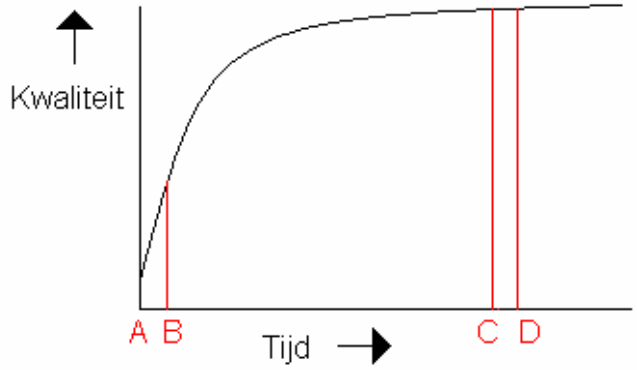
Definitie Een *beginrooster* is een rooster dat wordt gemaakt om het genetisch algoritme mee te beginnen. Dit rooster kan ook worden gebruikt door de klant zonder het algoritme uit te voeren.

Definitie Een *eindrooster* is een rooster dat ontstaat na het uitvoeren van het genetisch algoritme en mogelijk de locale optimalisatie.

De stappen 2 en 3 zijn recentelijk onderzocht.

Anytime behaviour

Uit het onderzoek van de stappen 2 en 3 is de indruk ontstaan dat de kwaliteit van de beginroosters grote invloed heeft op het eindresultaat, zeker als de beschikbare tijd beperkt is. Het 'anytime behaviour' van het genetisch algoritme geeft hier een verklaring voor. De kwaliteit van de best gevonden oplossing van een genetisch algoritme volgt vaak de volgende curve:

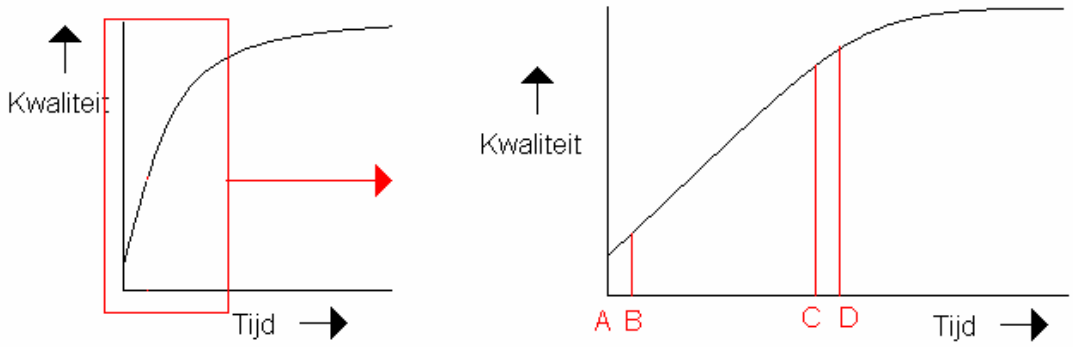


Figuur 1: Anytime behaviour van genetisch algoritme

In Figuur 1 is te zien dat op twee mogelijke manieren winst is te boeken.

1. Als we het genetisch algoritme beginnen met een kwalitatief beter beginrooster B kunnen we een eindrooster D vinden als we dezelfde tijd gebruiken die nodig is om beginnend met rooster A rooster C te vinden. In dit geval hebben we een rooster gevonden in dezelfde tijd van een betere kwaliteit. In dit geval kunnen we kwaliteitswinst boeken.
2. Ook kunnen we beginnend met rooster B rooster C vinden. Dit neemt minder tijd in beslag dan het vinden van rooster C beginnend met rooster A. In dit geval kunnen we tijdwinst behalen.

Als er minder tijd beschikbaar is, is het kwaliteitsverschil tussen punten C en D groter dan in Figuur 1. Dit betekent dat meer kwaliteitswinst kan worden behaald zoals in Figuur 2 is te zien.



Figuur 2: Anytime behaviour, eerste deel vergroot

Afgezien van de kwaliteit- en tijdwinst die kan worden behaald, is de kwaliteit van het beginrooster erg belangrijk. Reden hiervoor is dat de gebruiker kan besluiten het genetisch algoritme niet uit te voeren. Dat houdt in dat het beginrooster het definitieve rooster is. Het is vanuit ORTEC dus interessant te kijken of deze roosters verbeterd kunnen worden, niet alleen zodat deze gebruikt kunnen worden voor het uitvoeren van het genetisch algoritme.

Opdrachtomschrijving

We hebben gezien dat we in theorie tijd en kwaliteit kunnen winnen door geschikte beginroosters te gebruiken. Ook is de kwaliteit van het beginrooster belangrijk omdat het als definitief rooster gebruikt kan worden. Het doel van de stage is dan ook:

> **Onderzoek of een methode kan worden gevonden die betere beginroosters maakt dan de huidige methode.**

Om te kunnen bepalen of een beginrooster beter is dan een ander beginrooster is een aantal criteria mogelijk. Op elk van deze punten kunnen we de roosters met elkaar vergelijken.

1. De tijd die nodig is om een beginrooster te maken.
2. De tijd die gebruikt is voor het uitvoeren van het genetisch algoritme, gebruik makende van een beginrooster.
3. De tijd die nodig is voor het uitvoeren van locale optimalisatie.
4. De totale tijd nodig voor het maken van het uiteindelijke rooster.
5. De kwaliteit van het gemaakte beginrooster.
6. De kwaliteit van het rooster na het uitvoeren van het genetisch algoritme en locale optimalisatie.

In de eerste plaats bepaalt het aantal diensten dat in het rooster staat ingepland de kwaliteit van het rooster. Hoe meer diensten zijn ingepland, hoe hoger de kwaliteit van het rooster. Daarnaast bepaalt het aantal randvoorwaarden waar het rooster niet aan voldoet de kwaliteit.

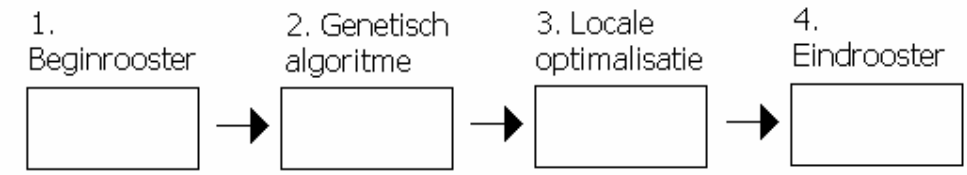
De belangrijkste nieuw te onderzoeken methode is de abstracte rooster methode. Het idee voor deze methode is vanuit ORTEC bedacht om betere roosters mee te maken. Een belangrijk deel van dit verslag zal daarom gaan over deze methode.

Alvorens te beginnen met het bespreken van nieuwe methoden, zal het volgende hoofdstuk eerst de werking van de huidige planautomaat toelichten. Deze planautomaat is de aanleiding voor de nieuwe methoden die het daaropvolgende hoofdstuk 'Nieuwe methoden' bespreekt.

4. Huidige planautomaat

Dit hoofdstuk gaat over de manier waarop HARMONY roosters genereert. De basis voor de huidige planautomaat is ontwikkeld door W. Winkelhuijzen en staat beschreven in 'Dienstroostergeneratie bij de Luchtverkeersleiding Nederland' ([1] Winkelhuijzen, 1999). Deze scriptie is tot stand gekomen tijdens diens stage bij ORTEC in 1999. Doel van die stage was een methode te ontwikkelen waarmee dienstroosters automatisch te genereren zijn.

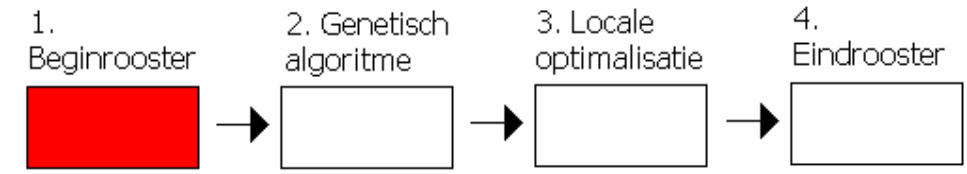
In het artikel stelt Winkelhuijzen voor een genetisch algoritme te gebruiken. Schematisch is de methode voor het genereren van de roosters als volgt weer te geven:



Figuur 3: Stappen voor het genereren van een rooster

Hieronder volgt een beschrijving van elk van de stappen.

4.1 Bottleneck methode



Figuur 4: Stap 1 beginrooster

Om een genetisch algoritme te starten is een aantal beginroosters noodzakelijk. Daarom beschrijft de scriptie een drietal methoden om deze te maken. Eén van deze methoden is de bottleneck methode. Naast de bottleneck methode bespreekt het artikel ook de methoden 'volledig willekeurig' en 'beperkt willekeurig'. Het artikel acht beide methoden minder geschikt om beginroosters mee te maken dan de bottleneck methode. Reden hiervoor is dat de kwaliteit van deze methoden niet voldoende is, ook al maken deze methoden in minder tijd een beginoplossing. In de afweging kwaliteit tegen snelheid geeft Winkelhuijzen de voorkeur aan kwaliteit. Op beide methoden zullen we hier verder niet ingaan.

Hieronder volgt de beschrijving van de bottleneck methode. De tekst is grotendeels afkomstig uit ([1] Winkelhuijzen, 1999). Op een aantal punten is de tekst samengevat of gewijzigd ter verduidelijking. De plaatjes zijn toegevoegd ter verduidelijking van de tekst.

Om een initiële oplossing te genereren gaat de bottleneck methode als volgt te werk. Allereerst wordt voor elke dienst die nog niet is ingepland een score berekend. Deze score hangt af van een aantal factoren, te weten:

- De dienst is wel / geen nachtdienst.
- De datum waarop de dienst valt, indien de dienst een nachtdienst is.
- De dienst is wel / geen weekenddienst.
- Het aantal personen waarbij de dienst niet in het rooster past zonder te overlappen met andere uit te voeren diensten of afwezigheid.
- De dienst is wel / geen reservedienst.

Deze factoren bepalen de moeilijkheid van de dienst. Hoe hoger de score voor een bepaalde dienst hoe lastiger deze is in te plannen in het rooster. Bij punt vier wordt overigens ook rekening gehouden met de benodigde kwalificaties die personen moeten bezitten om de dienst uit te kunnen voeren. Nadat voor elke dienst de score is bepaald wordt de lijst van diensten gesorteerd op deze score waarbij de moeilijkste dienst boven aan de lijst komt te staan. Indien diensten door deze regels een gelijke score opleveren worden de diensten willekeurig gesorteerd. Vervolgens wordt deze lijst één voor één doorlopen waarbij geprobeerd wordt de dienst aan een medewerker toe te wijzen. Dit gaat net zo lang door tot de lijst leeg is.

Het toewijzen van een dienst aan een medewerker gebeurt als volgt. Voor elke dienst wordt er voor de medewerkers een score berekend. Dit gebeurt alleen voor die medewerkers die de dienst kunnen uitvoeren. Dat wil zeggen dat ze de benodigde kwalificaties voor de dienst bezitten en geen overlap van de dienst met andere diensten in hun rooster hebben. De score bepaalt bij welke medewerker deze dienst het beste kan worden toegewezen. De score wordt berekend voor het rooster waarbij de dienst fictief is toegevoegd en wordt bepaald door de regels en criteria die in HARMONY zijn ingesteld. Bij het bepalen van de score wordt vooruitgekeken of series kunnen worden afgemaakt. Is dit het geval, dan levert het toevoegen van bijvoorbeeld een dienst die een serie begint geen negatieve score op.

Definitie Alles wat zegt dat iets wel of niet mag (door de planautomaat) noemen we een *regel*. De planautomaat mag geen rooster genereren waarin de regels overtreden worden. Regels zijn dus zogenaamde harde voorwaarden, voorwaarden waar altijd aan voldaan moet worden.

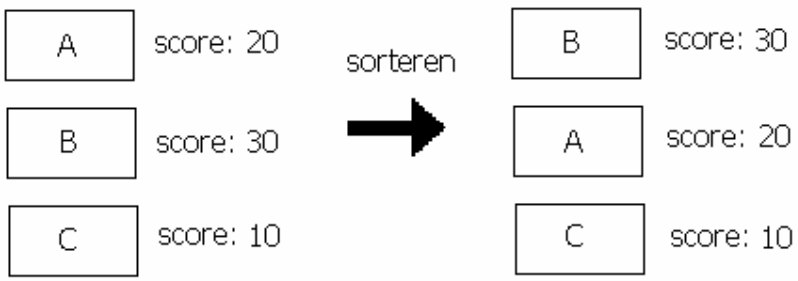
Definitie De *criteria* beschrijven de kwaliteit van het rooster. De criteria zijn ook omschreven in termen van wat wel of niet mag, maar nu is de overtreding echter toegestaan, op straffe van een belasting. Criteria zijn daarom zachte voorwaarden, omdat hier niet altijd aan voldaan hoeft te worden.

Definitie Een *serie* bestaat uit meerdere diensten die op een aantal opeenvolgende dagen aan een medewerker zijn toegewezen. Een serie is geëindigd als tussen twee diensten een grotere rusttijd zit dan is ingesteld door de gebruiker. Dit kan bijvoorbeeld zijn als er meer dan 24 uur rust tussen twee diensten zit.

Na het berekenen van de scores worden de medewerkers gesorteerd waarbij de aantrekkelijkste medewerkers boven aan de lijst komen te staan. Vervolgens wordt de lijst doorlopen waarbij voor elke medewerker gekeken wordt of de oplossing een toelaatbaar rooster geeft volgens de ingestelde regels. Zodra een medewerker gevonden is die hieraan voldoet is de toewijzing afgerond en wordt geprobeerd de volgende dienst toe te wijzen. Als de toewijzing bij geen enkele medewerker een toelaatbaar rooster oplevert wordt de dienst uit de lijst verwijderd en gaat het algoritme verder.

Voorbeeld

Stel dat we drie diensten A, B en C hebben op één bepaalde dag. We willen die toewijzen aan vier medewerkers: Jan, Piet, Klaas en Peter. Allereerst bepalen we voor elke dienst de score.

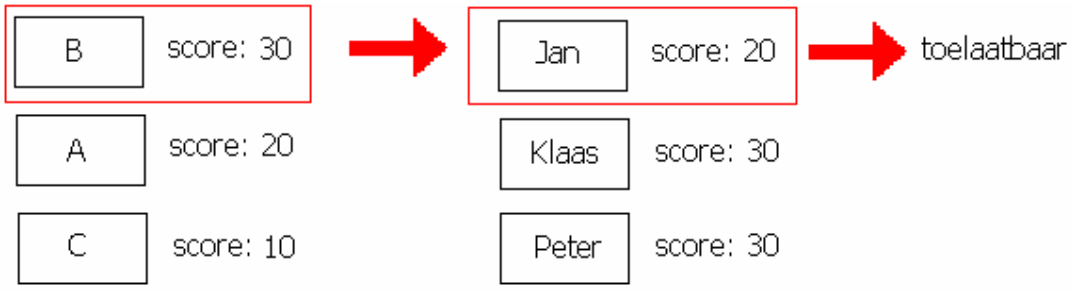


Figuur 5: Bepalen en sorteren van scores van diensten

Vervolgens bepalen we de score van elke medewerker als deze dienst B uitvoert.

Figuur 6: Ontoelaatbare toewijzing van een dienst aan de beste medewerker

Omdat deze toewijzing niet toegelaten is, proberen we de volgende medewerker.



Figuur 7: Toelaatbare toewijzing van een dienst aan de beste medewerker

Deze oplossing is toegelaten en medewerker Jan krijgt dienst B in zijn rooster. Ditzelfde proces herhalen we tot ook diensten A en C zijn toegewezen aan een medewerker of overblijven, omdat het toewijzen aan elke willekeurig overgebleven medewerker een niet toelaatbare oplossing geeft.

Meerdere roosters

De bottleneck methode is een deterministische methode. Elke keer zal hetzelfde rooster ontstaan bij het creëren van een rooster voor een bepaalde periode. Alleen als diensten dezelfde score hebben kan een afwijking plaatsvinden. In dat geval worden de diensten namelijk willekeurig gesorteerd. Voor het genetisch algoritme is het vaak nodig een beginpopulatie met meer dan één rooster te vullen. Zodra meerdere roosters gewenst zijn voldoet de bottleneck methode niet omdat maar één rooster gemaakt kan worden. Om toch met deze methode verschillende beginroosters te kunnen maken zijn een aantal methoden bedacht.

Methode 1

De eerste methode splitst de diensten op in meerdere delen na het uitvoeren van stap 1. Deze delen vormen vervolgens weer een lijst door ze in verschillende volgorden samen te voegen. Voor elk van de lijsten die op deze manier zijn gemaakt wordt het algoritme uitgevoerd. Op deze manier krijgen we verschillende roosters.

Voorbeeld

Stel dat we na het uitvoeren van stap 1 de lijst in drie delen opsplitsen, A, B en C. Deze delen voegen we samen tot de nieuwe lijsten (A, B, C), (A, C, B), (B, A, C) en (B, C, A). Voor elk van deze lijsten voeren we stap 2 en 3 uit van het algoritme. Op deze manier krijgen we vier verschillende beginroosters.

Methode 2

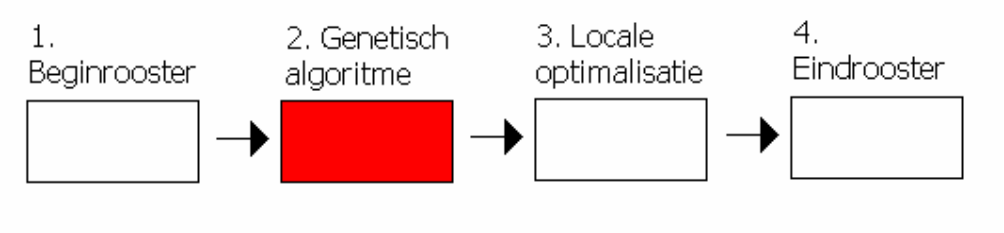
De huidige methode die is geïmplementeerd splitst de groep medewerkers willekeurig in tweeën. Elke medewerker komt met kans 1/2 in een groep. Voor elk van deze groepen afzonderlijk wordt een rooster gegenereerd. Voor elke groep kunnen alle diensten worden ingepland. Door het opsplitsen veranderen de medewerkers die in aanmerking komen voor een dienst. De diensten

worden hierdoor bij andere medewerkers ingepland. Op deze manier ontstaan verschillende roosters. Het is overigens mogelijk dat dezelfde roosters ontstaan als toevallig dezelfde groepen medewerkers worden gemaakt.

Voorbeeld

Stel we hebben 4 medewerkers: Jan, Piet, Klaas en Peter. Voor het maken van het eerste rooster worden de medewerkers willekeurig gesplitst in groep 1 met medewerker Jan en groep 2 met medewerker Piet, Klaas en Peter. Voor elk van deze groepen wordt een rooster gemaakt. Bij elkaar levert dit het eerste rooster. Voor het tweede rooster wordt de groep gesplitst in groep 1 met medewerker Piet en Klaas en groep 2 met medewerkers Jan en Peter. Het samenvoegen van deze roosters levert het tweede rooster op.

4.2 Genetisch algoritme



Figuur 8: Stap 2 Genetisch Algoritme

Na het creëren van de benodigde beginroosters volgt de uitvoering van het genetisch algoritme. Elke generatie past een aantal mutatie- en cross-overoperatoren toe die hieronder kort staan beschreven.

Mutatie operatoren

Voor het genetisch algoritme is een drietal mutatieoperatoren ontwikkeld. Deze operatoren zijn 'Muteer Korte Series', 'Muteer Bruto Werktijd' en 'Muteer Willekeurig Blok In Rooster'.

Muteer korte series

De operator 'Muteer Korte Series' plant series uit het rooster, waarbij weinig diensten achter elkaar staan. Deze diensten worden vervolgens samen met de overige nog in te plannen diensten met behulp van de bottleneck methode ingepland.

Definitie Het *uitplannen van diensten* betekent het ongedaan maken van toewijzingen van diensten aan medewerkers. Als dienst A op maandag bij medewerker Jan staat ingepland en deze wordt uitgepland, dan betekent het dat medewerker Jan op maandag niet meer dienst A in zijn rooster heeft staan.

Muteer bruto werktijd

Operator 'Muteer Bruto Werktijd' plant de diensten uit van de medewerker die het kortste werkt. Dat wil zeggen die medewerker die het grootste verschil heeft tussen de tijd die hij maximaal mag werken en de tijd waar hij momenteel voor staat ingepland. Ook plant de operator de diensten van de medewerker die het langste werkt uit. Deze medewerker heeft dus het kleinste verschil tussen de tijd waarvoor hij staat ingepland en de maximale tijd die hij in die periode mag werken. Vervolgens worden deze diensten samen met de overige nog in te plannen diensten met behulp van de bottleneck methode ingepland.

Muteer willekeurig blok in rooster

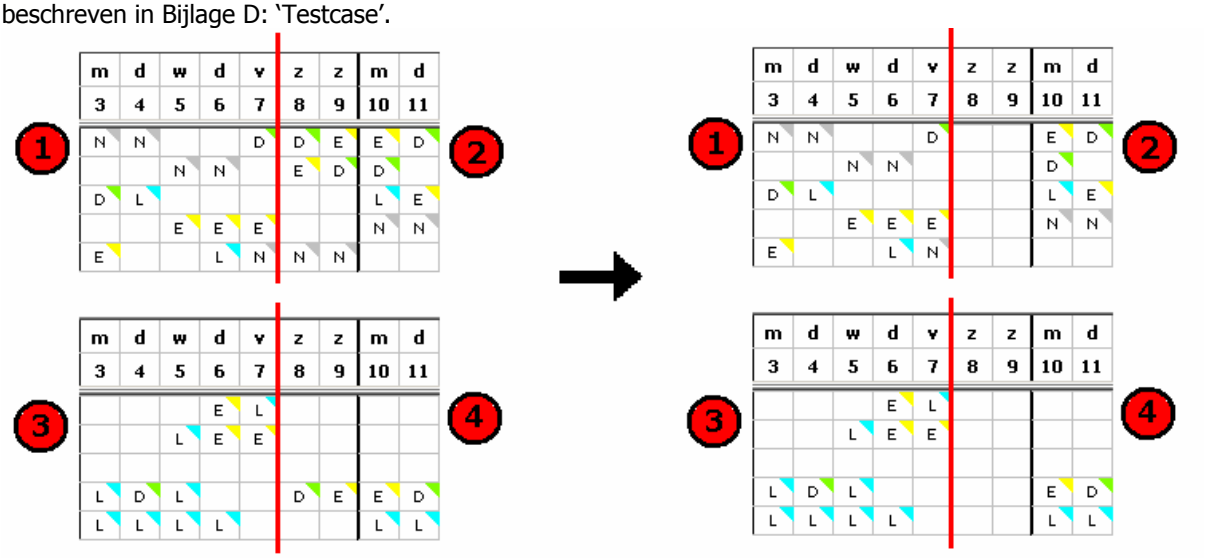
Operator 'Muteer Willekeurig Blok In Rooster' selecteert een willekeurig aantal medewerkers en een willekeurige periode en plant alle diensten van die periode bij die medewerkers uit. Vervolgens worden deze diensten samen met alle overige nog in te plannen diensten met behulp van de bottleneck methode opnieuw ingepland.

Cross-overoperatoren

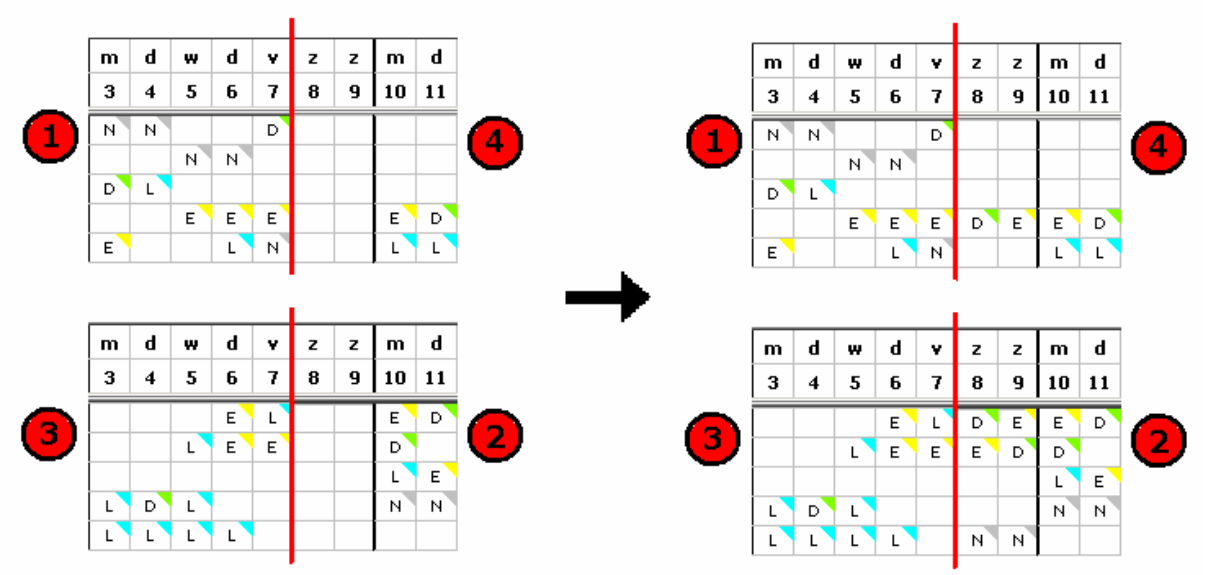
Voor twee verschillende roosters A en B wordt een dag x gekozen waarop ze worden 'doorgeknipt'. Dit levert feitelijk vier roosters op, te weten rooster A vóór dag x (1), rooster A ná dag x (2), rooster B vóór dag x (3) en rooster B ná dag x (4). Van rooster 2 en 4 worden de eerste twee dagen uitgepland. Vervolgens worden roosters 1 en 4 en rooster 2 en 3 aan elkaar geplakt. Daarna worden alle nog in te plannen diensten van de ontstane roosters ingepland met de bottleneck methode.

Voorbeeld

In dit voorbeeld worden vier typen diensten gebruikt in het rooster. Dit zijn dezelfde diensten als beschreven in Bijlage D: 'Testcase'.



Figuur 9: Opknippen van roosters, uitplannen van eerste 2 dagen na opknippen

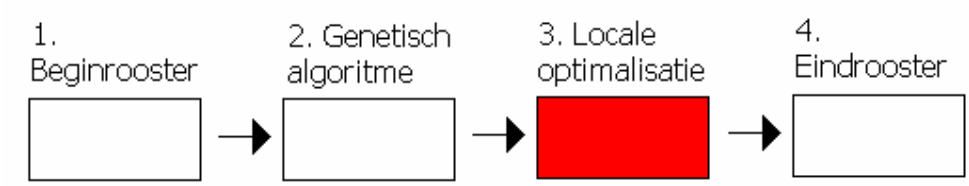


Figuur 10: Samenplakken van roosters en inplannen van nog in te plannen diensten

Operator kansen

Elke operator heeft een kans om children te mogen maken voor de volgende generatie. Deze kans wordt bepaald door het succes dat de operator heeft. Het percentage succesvolle toepassingen dient als deze kans. Ook is er een minimumkans ingesteld waaronder de kans op toepassen niet kan komen. Dit zorgt ervoor dat operatoren, die in het begin geen succes hebben, in de rest van het algoritme nog steeds worden toegepast en mogelijk voor succes kunnen zorgen.

4.3 Locale optimalisatie



Als uiteindelijk het maximale aantal generaties is bereikt of de populatie is geconvergeerd, kan er nog locale optimalisatie worden toegepast als daar nog tijd voor is. In de scriptie wordt als locale optimalisatie 1-opt en 2-opt voorgesteld. Later is hier nog de methode 'verbeter slechtste medewerker' aan toegevoegd.

1-opt en 2-opt

Hieronder volgt de beschrijving van de 1-opt en 2-opt methode zoals die in HARMONY worden toegepast. De tekst is een samenvatting van een gedeelte uit ([1], Winkelhuizen, 1999).

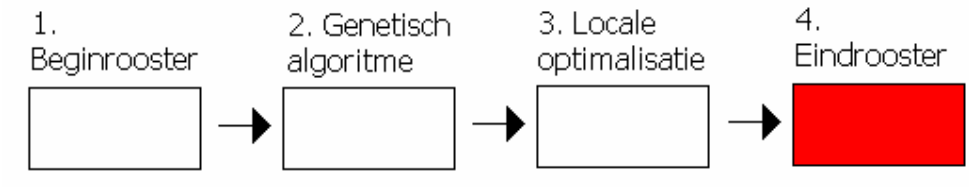
De 1-opt methode probeert elke dienst te verwisselen met een andere dienst zolang ze niet allebei aan dezelfde medewerker zijn toegewezen en niet allebei nog moeten worden ingepland. Als het rooster met de verwisseling van de diensten een hogere score oplevert, wordt deze behouden. Is dit niet het geval wordt de verwisseling ongedaan gemaakt en wordt voor een ander tweetal diensten gekeken of een verwisseling verbetering van de score oplevert. Dit gaat door tot het maximale aantal verwisselingen is bereikt of alle verwisselingen zijn uitgeprobeerd.

De 2-opt methode probeert twee verschillende diensten van een medewerker te wisselen met nog in te plannen diensten of twee diensten van een andere medewerker. Voor elke verwisseling wordt gekeken of dit geen overtredingen van regels geeft. Is dit niet het geval wordt gekeken of de score van het rooster verbetert. Als de score verbetert, wordt de wisseling behouden. Verbetert de score niet wordt de wisseling ongedaan gemaakt. Dit proces herhaalt zich tot alle mogelijkheden zijn geprobeerd of het maximale aantal verwisselingen is bereikt.

Verbeter slechtste medewerker

Later is als laatste stap nog de methode 'verbeter slechtste medewerker' toegevoegd. Zolang na het uitvoeren van het genetisch algoritme en het uitvoeren van 1-opt en 2-opt nog tijd over is, zullen de diensten van de medewerker met de slechtste score worden uitgepland. Vervolgens worden deze diensten met de bottleneck methode weer ingepland. Dit gebeurt net zo lang tot de beschikbare tijd op is.

4.4 Opmerkingen huidige planautomaat



Hierboven is beschreven hoe het huidige algoritme in zijn werk gaat. Met dit algoritme is het mogelijk roosters te maken voor zeer verschillende omstandigheden. De gebruiker heeft grote vrijheid in het instellen van de zwaarte van regels en criteria. Een aantal opmerkingen zijn echter te maken over de manier waarop deze methode het eindrooster genereert.

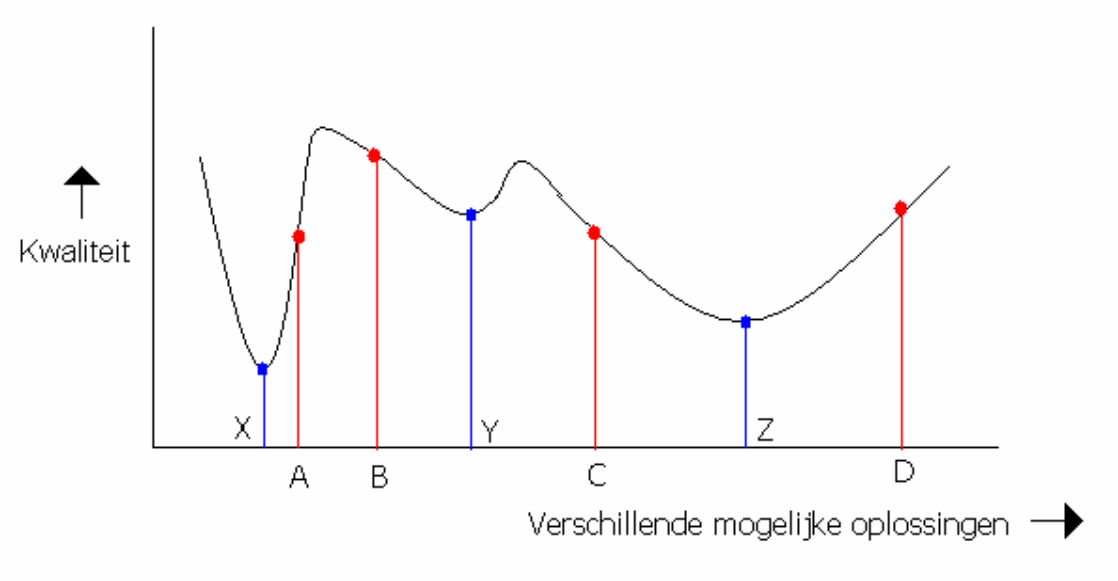
Oplossingsruimte

In 'Personnel Scheduling' ([2] Van der Put, 2005) komt een testcase aan bod waarbij de door het genetisch algoritme gevonden rooster met de hand aanzienlijk verbeterd kan worden. De oplossing gevonden door het algoritme is niet het best mogelijke rooster, maar een locaal optimum.

Definitie De *mate van convergentie* van een genetisch algoritme geeft aan hoe snel het algoritme naar een bepaalde oplossing convergeert.

Definitie *Convergeren* houdt in dat de individuen van het algoritme bij een (locaal) optimum samenkomen.

Dat een genetisch algoritme convergeert in een locaal optimum is niet verwonderlijk. Denk bijvoorbeeld in dat de kwaliteit van alle mogelijke oplossingen er als volgt uitziet:



Figuur 11: 2-dimensionale weergave van een mogelijke oplossingsruimte

Als een populatie bestaat uit de individuen B, C en D dan zal het genetisch algoritme naar alle waarschijnlijkheid convergeren naar het locale optimum Z. Individu B zal door mutaties convergeren naar Y, maar door cross-over uiteindelijk naar Z gestuurd worden. Het algoritme combineert de eigenschappen van die roosters, waardoor de children, door invloed van de individuen C en D, richting Z zullen veranderen.

Op een aantal manieren kan geprobeerd worden het algoritme naar het globale optimum te laten convergeren in plaats van een locale optimum.

In de eerste plaats kunnen in het genetisch algoritme methoden worden toegepast die ervoor zorgen dat de oplossingsruimte beter wordt doorzocht. Dit kan bijvoorbeeld door een geschikte keuze van mutatie- en cross-overoperatoren. Daarbij is het belangrijk dat de operatoren niet gehinderd worden in het veranderen van de individuen. In het huidige algoritme is dit wel het geval. Zo is het niet mogelijk de ingestelde regels te overtreden. Hierdoor wordt het doorzoeken van de oplossingsruimte bemoeilijkt. In 'Personnel Scheduling' ([2] Van der Put, 2005) is daarom de methode Tabu Search onderzocht om te kijken wat het tijdelijk toestaan van overtredingen van regels voor gevolgen heeft. Uit het onderzoek bleek dat het algoritme in staat is de oplossingsruimte beter te doorzoeken en te convergeren naar een beter optimum dan voorheen. De tijd die de methode nodig heeft om deze roosters te vinden is echter relatief groot.

Daarnaast kan het helpen het genetisch algoritme te starten met zeer verschillende, maar kwalitatief gelijke beginroosters. Door bijvoorbeeld te starten met de individuen A, C en D zal het genetisch algoritme een groter deel van de oplossingsruimte doorzoeken dan wanneer enkel met individuen C en D wordt begonnen. Hierbij is het belangrijk dat de individuen veel van elkaar verschillen. Als de beginroosters allemaal op punt C zouden lijken, zal het genetisch algoritme maar een klein deel van de gehele ruimte doorzoeken.

Als laatste is het mogelijk het genetisch algoritme te starten met beginoplossingen die in de buurt van het globale optimum liggen. Als de beginpopulatie gevuld is met individuen die lijken op A dan is de kans groot dat het algoritme convergeert naar X.

Het vermoeden bij ORTEC bestaat dat de huidige beginroosters niet in de buurt van het globale optimum liggen. Reden hiervoor kan zijn dat de bottleneck methode een gretige heuristiek is. De moeilijkst in te plannen dienst wordt op de meest gunstige plek voor die dienst neergezet. Hierdoor is het echter mogelijk dat er voor andere diensten problemen ontstaan. De structuur van het rooster zal mogelijk door deze manier van inplannen niet lijken op de optimale structuur. Het aanpassen van de structuur is moeilijk omdat hier vaak overtredingen van regels voor nodig zijn.

Daarnaast zijn de oplossingen die gevonden worden met de bottleneck planner kwalitatief zeer verschillend. Als een aantal rooster wordt gemaakt is het eerst gemaakte rooster beduidend beter dan de rest van de roosters. Hierdoor zal het genetisch algoritme de goede eigenschappen van het beste rooster overnemen in al zijn individuen, waardoor het te snel convergeert naar het locale optimum in de buurt van dat ene goede individu.

Controle regels en criteria

Tijdens het uitvoeren van het genetisch algoritme moet een groot aantal regels en criteria meermalen gecontroleerd worden. Het controleren van deze regels en criteria neemt veel tijd in beslag. Na het maken van meerdere roosters met het genetisch algoritme bleek dat tijdens het uitvoeren van het algoritme gemiddeld 83% van de tijd zit in dit controleren. Hierdoor is het belangrijk het aantal individuen dat per generatie gebruikt wordt en het aantal generaties dat wordt uitgevoerd zo klein mogelijk te houden. Beide brengen immers extra controles met zich mee wat veel tijd kost.

Locale optimalisatie

De locale optimalisatie aan het einde van het genetisch algoritme levert grote verbeteringen op in de kwaliteit van het rooster. Tijdens het uitvoeren van de tests in hoofdstuk 11: 'Beginroosters maken testresultaten' is uit de runs van de huidige methode gebleken dat de belasting van het rooster door deze stap gemiddeld met 26% verbetert.

5. Nieuwe methoden

In dit hoofdstuk wordt de aanpak uiteengezet voor de opdracht. Het doel was

> **Onderzoek of een methode kan worden gevonden die betere beginroosters maakt dan de huidige methode.**

In het vorige hoofdstuk hebben we gezien hoe het huidige algoritme te werk gaat. Daarbij werd opgemerkt dat de eindkwaliteit van het rooster vaak met de hand verbeterd kan worden. Reden hiervoor is dat het algoritme naar een locaal optimum convergeert. Dit kan worden tegengegaan door:

- Tijdelijke schendingen toe te staan, waardoor de oplossingsruimte beter kan worden doorzocht.
- Te beginnen met zeer verschillende beginroosters.
- Door te beginnen met kwalitatief goede roosters die in de buurt van het globale optimum liggen.

Daarnaast viel op dat:

- Het genetisch algoritme gemiddeld 83% van de tijd bezig is met het controleren van de regels en criteria.
- De locale optimalisatie voor een kwaliteitsverbetering van gemiddeld 26% zorgt.

Deze aandachtspunten geven aanleiding tot een aantal aanpassingen om beginroosters te maken. Elk van deze methoden probeert één of meerdere van bovenstaande opmerkingen te gebruiken om daarmee te zorgen voor tijd- en/of kwaliteitsverbetering.

5.1 Abstract rooster

De belangrijkste methode die we willen onderzoeken is de abstracte rooster methode. De methode moet het mogelijk maken die deelruimte van de oplossingsruimte te vinden die een goede, zo niet de beste oplossing bevat. De methode moet dit bereiken door zich te richten op de structuur van het rooster, maar bereikt dit ook door de mogelijkheid om regels te overtreden. Het genetisch algoritme kan dit niet zoals we hebben gezien en is hierdoor beperkt in het zoeken door de oplossingsruimte.

Ook moet de methode de totale tijd die nodig is voor het controleren van de regels en criteria reduceren. De methode moet er voor zorgen dat het genetisch algoritme minder generaties nodig heeft om de oplossing te vinden, wat tijdwinst oplevert.

Abstract rooster concept

Het concept van de abstract rooster methode is het oorspronkelijke roosterprobleem te vereenvoudigen. In plaats van een dienst toe te wijzen aan een medewerker willen we eerst bepalen of een medewerker wel of niet moet werken op een dag. Hoe laat de medewerker op die dag moet werken, waar hij moet zijn of welke dienst hij moet doen is niet van belang. Door deze vereenvoudiging ziet het rooster er als volgt uit:

Medewerker	Maandag	Dinsdag	Woensdag	Donderdag	Vrijdag	Zaterdag	Zondag
Jan	1	1	0	0	1	1	1
Klaas	1	1	1	1	0	1	1
Piet	0	0	1	1	1	0	0

Met een 1 geven we aan dat de medewerker een dienst heeft op die dag. 0 geeft aan dat de medewerker geen dienst heeft. De weergave maakt het mogelijk gericht te kijken naar de structuur van het rooster zonder rekening te houden met alle details.

Door de vereenvoudiging is het probleem gereduceerd tot het toewijzen van 0-en en 1-en aan medewerkers dusdanig dat we de bezettingseis kunnen halen en zo goed mogelijk kunnen voldoen

aan de regels en criteria. Het vereenvoudigen houdt echter ook in dat bepaalde regels en criteria niet meer goed kunnen worden gecontroleerd. Neem bijvoorbeeld een regel als:

Minimaal *2* vrije weekenden van *60* uren per *4* weken, omvattende *zaterdag 1:00* uur tot *maandag 4:00* uur.

Deze regel is nu moeilijk te controleren. We weten immers niet wat voor een dienst de medewerker uitvoert. Zo heeft medewerker Piet op vrijdag een dienst. Als dit een nachtdienst is, is het goed mogelijk dat deze dienst na zaterdag 1 uur 's morgens eindigt. In dit geval heeft medewerker Piet geen vrij weekend. Heeft hij echter op vrijdag een dagdienst dan is zijn weekend wel vrij. Hierdoor is het mogelijk dat het abstracte rooster vrije weekenden toekent die er in werkelijkheid niet zijn. Er gaat dus kwaliteit verloren tijdens het vereenvoudigen van het probleem.

Definitie Het aantal diensten dat in een rooster is ingepland en de belasting die ontstaat door het overtreden van de criteria vormen samen de *kwaliteit* van een rooster.

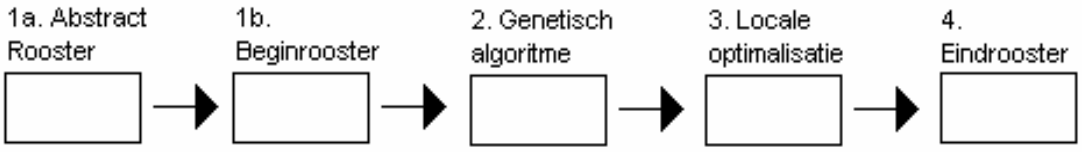
De huidige weergave is echter net te beperkt om goed te kunnen kijken naar de structuur. Dit heeft als oorzaak dat veel regels en criteria speciaal zijn toegespitst op nachtdiensten. Deze diensten zijn zeer belangrijk voor de maximale lengte van series en de benodigde rust van medewerkers. De nachtdiensten hebben hierdoor grote invloed op de structuur van het rooster. Hierdoor is het verstandig het vereenvoudigde probleem uit te breiden door onderscheid te maken tussen gewone diensten en nachtdiensten. Het vereenvoudigde rooster ziet er dan als volgt uit:

Medewerker	Maandag	Dinsdag	Woensdag	Donderdag	Vrijdag	Zaterdag	Zondag
Jan	N	N	-	-	D	D	D
Klaas	D	D	D	D	-	N	N
Piet	-	-	N	N	N	-	-

Hierbij staat de N voor een nachtdienst en de D voor een gewone dienst. Het concept van het abstracte rooster is het oorspronkelijke probleem te vereenvoudigen door eerst te kijken naar de structuur van het rooster. Dit doen we door aan te geven of een medewerker op een dag een nachtdienst, een gewone dienst of geen dienst heeft.

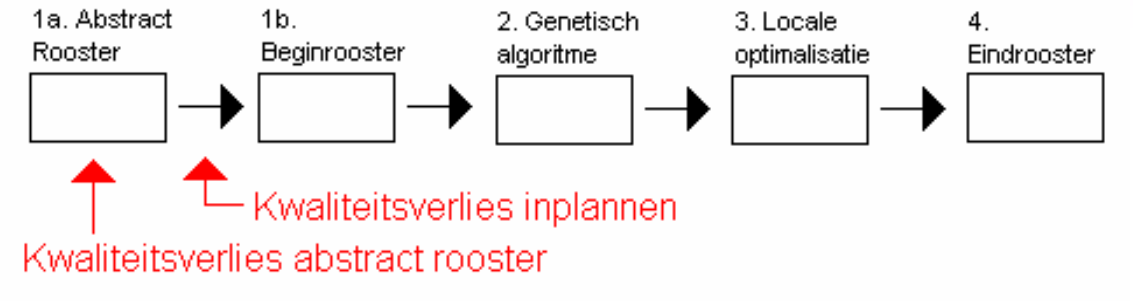
Als eenmaal een goed abstract rooster is gemaakt moet vervolgens de echte toewijzing van diensten aan medewerkers plaatsvinden. Hierbij wijzen we alleen diensten toe aan medewerkers als de medewerker ook een gewone dienst of een nachtdienst in het abstracte rooster toegewezen heeft gekregen. Als een medewerker geen dienst heeft staan in het abstracte rooster dan kan hij bij de uiteindelijke toewijzing ook geen dienst krijgen. Bij het toewijzen is het mogelijk dat kwaliteit verloren gaat als het niet mogelijk blijkt een dienst toe te kennen aan een medewerker terwijl dit bij het abstracte rooster wel is aangegeven.

Schematisch ziet het algoritme er met het gebruik van het abstracte rooster als volgt uit:



Figuur 12: Schematische weergave maken van een rooster met abstract rooster

Kwaliteitsverlies



Figuur 13: Kwaliteitsverlies in het proces

Door het vereenvoudigen van het probleem ontstaat op twee plaatsen kwaliteitsverlies. In de eerste plaats kan kwaliteit verloren gaan bij het maken van het abstracte rooster. Deze kwaliteit verliezen we als we niet de optimale oplossing voor het vereenvoudigde probleem weten te vinden. Dit verlies noemen we kwaliteitsverlies abstract rooster.

Definitie *Kwaliteitsverlies abstract rooster* is de kwaliteit die we verliezen als we niet het optimale abstracte rooster kunnen vinden.

Daarnaast kan kwaliteit verloren gaan bij de vertaling van het abstracte rooster naar een echt rooster. Als eenmaal het abstracte rooster is gemaakt, zullen op de plekken waar het abstracte rooster een dienst heeft ingepland een echte dienst moeten worden toegekend. Omdat de regels en criteria niet altijd goed te controleren zijn in het vereenvoudigde probleem is het mogelijk dat het inplannen van een echte dienst in het abstracte rooster niet is toegestaan. Dit verlies noemen we kwaliteitsverlies inplannen.

Definitie *Kwaliteitsverlies inplannen* is de kwaliteit die we verliezen bij het toewijzen van diensten aan medewerkers volgens het abstracte rooster.

Controle van regels en criteria

We hebben gezien dat de regels en criteria in het vereenvoudigde probleem niet altijd goed te controleren zijn. Dit kan tot kwaliteitsverlies inplannen leiden. Voor de regels en criteria die niet goed gecontroleerd kunnen worden, zal moeten worden gekeken hoe het verlies te beperken valt. Er zijn telkens drie mogelijkheden voor het controleren van zo een regel of criteria:

1. We controleren de regel of het criterium niet. Dit kan bijvoorbeeld zijn omdat de regel niet interessant is voor de structuur van het rooster of omdat de controle niet kan worden uitgevoerd. Door een regel of criterium niet te controleren krijgen we het grootst mogelijke kwaliteitsverlies inplannen voor deze regel. Voordeel is echter wel dat het zoeken naar een goede structuur minder tijd in beslag zal nemen dan wanneer de regel of het criterium wel gecontroleerd wordt.
2. We controleren de regel of het criterium, maar zijn minder streng. Zo kunnen we bijvoorbeeld controleren of iemand twee vrije weekenden heeft, waarbij we alleen kijken of een medewerker zaterdag en zondag geen dienst heeft. We houden dan geen rekening met de eis dat het weekend 60 uur moet bevatten. Kwaliteitsverlies inplannen kan nog steeds optreden, maar zal minder groot zijn dan in het eerste geval.
3. We controleren de regel of het criterium strenger. Zo kunnen we bijvoorbeeld bij de controle op vrije weekenden niet alleen zaterdag en zondag bekijken, maar ook maandag. In dat geval heeft iemand een vrij weekend als op geen van de drie dagen een dienst staat gepland. Hierdoor voorkomen we kwaliteitsverlies inplannen, maar tegelijk vergroten we het kwaliteitsverlies abstract rooster. Door de regel of het criterium strenger te controleren verkleinen we het aantal mogelijke oplossingen. Het optimale abstracte rooster is hierdoor moeilijker of helemaal niet meer te vinden.

Per regel en criterium zal gekeken moeten worden welke van de drie opties het beste werkt.

Voor- en nadelen abstract rooster

Door de vereenvoudiging van het probleem is het mogelijk te kijken naar alleen de structuur van het rooster, zonder alle details in ogenschouw te nemen. Hierdoor kan de structuur van het beginrooster verbeterd worden, met als gevolg dat het genetisch algoritme in staat is betere oplossingen te vinden. Het toestaan van overtredingen van regels kan helpen bij het vinden van die structuur. De methode waarmee het abstracte rooster wordt gemaakt, is hierdoor in staat de oplossingsruimte beter te doorzoeken dan het genetisch algoritme.

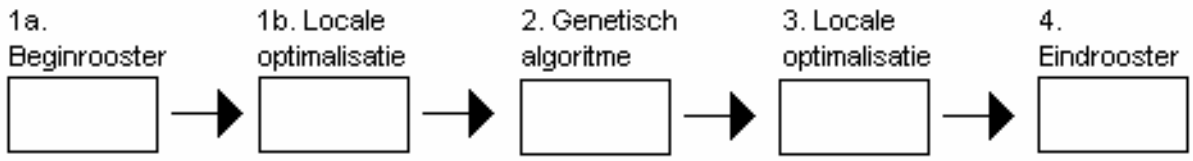
Het niet controleren van bepaalde regels en criteria moet ervoor zorgen dat het zoeken naar de structuur niet teveel tijd in beslag hoeft te nemen. Ook zorgt het ervoor dat andere details naast de structuur niet worden bekeken. Daar staat echter tegenover dat er kwaliteitsverlies kan optreden zowel bij het inplannen als bij het vinden van het beste abstracte rooster. Belangrijk is dit verlies zo klein mogelijk te houden.

5.2 Bottleneck methode met locale optimalisatie

Een andere aanpassing die we willen bekijken is de gemaakte beginroosters optimaliseren met locale optimalisatie. Het toepassen van 1-opt en 2-opt op het rooster gemaakt door het genetisch algoritme zorgt voor grote verbeteringen. Dergelijke verbeteringen zullen ook van toepassing zijn bij de beginroosters. De kwaliteit van het beginrooster zal hierdoor toenemen. Interessant is te kijken of het genetisch algoritme hier ook van profiteert.

Het maken van het beginrooster zal door het toepassen van locale optimalisatie wel meer tijd in beslag nemen. Een deel van die tijd zal wellicht kunnen worden teruggewonnen bij de uitvoering van het genetisch algoritme en de locale optimalisatie op het eind. Reden hiervoor is dat de beginoplossing van betere kwaliteit is, waardoor de vervolgstappen minder rekentijd nodig hebben om deze oplossing verder te verbeteren.

Schematisch ziet het algoritme er met de voorgestelde verandering als volgt uit:



Figuur 14: Schematische weergave bottleneck methode met locale optimalisatie

5.3 Stochastische bottleneck methode

We hebben opgemerkt dat als de bottleneck methode meerdere roosters moet genereren, dat de kwaliteit van deze rooster zeer uiteenloopt. Dit zorgt ervoor dat het genetisch algoritme snel convergeert naar een oplossing die dicht bij het beste beginrooster ligt.

Te bestuderen is wat het resultaat is als de bottleneck methode stochastisch is gemaakt. Dit kunnen we doen door na het berekenen van de score van de diensten, niet de dienst met de hoogste score uit de gesorteerde lijst te selecteren, maar met een bepaalde kans deze te selecteren. Dit maakt het mogelijk dat diensten met een minder hoge score eerder worden geselecteerd en ingepland. Hierdoor zal het gevonden rooster bij elke uitvoering van de stochastische bottleneck methode verschillen van de volgende.

5.4 Volledig willekeurig

Om te bekijken wat de invloed is van de kwaliteit van het beginrooster op de kwaliteit van het eindrooster onderzoeken we ook de methode volledig willekeurig. Deze methode wijst willekeurig diensten toe aan een medewerker om zo een rooster te maken. Van deze roosters weten we dat er geen structuur in zit en dat het kwalitatief slechtere beginroosters zijn dan bij de bottleneck

methode. Hierdoor kunnen we zien hoeveel invloed de kwaliteit van de beginroosters heeft op de tijd en/of de kwaliteit.

Ook zorgt deze methode voor verschillende, maar kwalitatief gelijke beginroosters. Omdat de methode elke keer roosters willekeurig maakt zullen de roosters veel van elkaar verschillen. Maar ook zal het niet gebeuren dat het eerst gemaakt rooster altijd beter is dan de andere zoals bij de bottleneck methode het geval is. De roosters zijn kwalitatief gelijk.

5.5 Samenvatting

In dit hoofdstuk hebben we een drietal methoden voorgesteld waarmee we beginroosters willen maken. Elk van deze methoden moet de effecten van één of meerdere nadelen van de huidige methode verminderen.

De bottleneck methode met locale optimalisatie voert na het maken van de beginroosters met de bottleneck methode locale optimalisatie uit. Hierdoor moet de structuur van het beginrooster verbeterd worden. Door een betere structuur kan de kwaliteit van het rooster na het uitvoeren van het genetisch algoritme verbeteren.

De stochastische bottleneck methode maakt de bottleneck methode stochastisch zodat de kwaliteit van de roosters die met de methode worden gemaakt dichter bij elkaar liggen. Ook dit heeft als doel de eindkwaliteit van het rooster te verbeteren.

Het abstracte rooster moet het mogelijk maken de structuur van de beginoplossingen te verbeteren. Door overtredingen toe te staan is het mogelijk de oplossingsruimte beter te doorzoeken dan het genetisch algoritme momenteel kan. Door niet alle regels en criteria te controleren moet de tijd die nodig is voor dit zoeken beperkt blijven. In het volgende hoofdstuk zal worden toegelicht hoe een abstract rooster kan worden gemaakt.

6. Maken van abstract rooster

In het voorgaande hoofdstuk hebben we een aantal methoden besproken waarmee we de nadelen van de huidige methode hopen op te vangen. Eén van de voorgestelde methoden was het abstracte rooster. In dit hoofdstuk zullen twee methoden aan bod komen waarmee we abstracte roosters kunnen maken. Deze methoden zijn: genetisch algoritme en kolomgeneratie. De voor- en nadelen van de methoden komen aan bod. Op basis daarvan maken we een keuze welke methode meest geschikt is.

6.1 Genetisch algoritme

De eerste methode die we bespreken is het genetisch algoritme. Het maken van een abstract rooster is een vereenvoudiging van het oorspronkelijke probleem dat momenteel ook met een genetisch algoritme wordt opgelost. Hieronder bespreken we per component van het genetisch algoritme hoe het kan worden aangepast voor het maken van het abstracte rooster. In Bijlage C: 'Genetisch algoritme' is meer informatie te vinden over de algemene componenten.

Individuen

Een individu is een rooster dat aan elke medewerker op een dag een abstract dienst toewijst. Met een abstracte dienst bedoelen we een dienst die alleen bestaat voor het abstracte rooster. Een abstracte dienst kan 3 soorten diensten bevatten: geen dienst, een dagdienst of een nachtdienst. Deze diensten worden niet daadwerkelijk toegewezen aan de medewerker, zoals dat in HARMONY gebruikelijk is, maar staan fictief bij de medewerker. Een mogelijke implementatie van een rooster is een matrix van medewerkers en dagen met 0, 1 en 2-en, waarbij 0 staat voor geen dienst, 1 voor een normale dienst en 2 voor een nachtdienst.

Fitness function

De kwaliteit van een individu wordt bepaald door 3 factoren:

1. De score van het overtreden van regels. Per regel kan gecontroleerd worden of het individu deze wel of niet overtreedt. Vervolgens wordt aan het overtreden van een regel een gewicht gehangen.
2. De score van het overtreden van criteria. Voor elk criterium kan gecontroleerd worden of het individu deze overtreedt. In HARMONY wordt tevens een gewicht gehangen aan het overtreden van zo een criterium. Deze score wordt in HARMONY belasting genoemd.
3. Het verschil tussen het aantal ingeplande diensten in het rooster en de bezettingseis van het rooster. Per dag kan gekeken worden hoeveel diensten er niet zijn ingepland. Per afwijking kan opnieuw een gewicht worden bepaald voor het verschil.

Elk van deze factoren geeft een score. Door deze scores bij elkaar te tellen krijgen we een fitness function. Per onderdeel moeten we de gewichten instellen om te zorgen dat de juiste factoren de goede nadruk krijgen. In dit geval willen we voorkomen dat we roosters krijgen waarin regels worden overtreden. Als we immers ons abstract rooster moeten vertalen naar een echt rooster dan weten we zeker dat een aantal diensten niet kunnen worden ingepland. Deze factor krijgt dan ook het grootste gewicht. Daarna is het van belang dat we zo veel mogelijk diensten inplannen. Minst belangrijk is het halen van de criteria.

Definitie *Score abstract rooster* is de waarde van de fitness function. De score is de som van de overtredingen van de regels keer een gewicht plus de score voor het aantal nog in te plannen diensten keer een gewicht plus de score voor het overtreden van de criteria keer een gewicht.

Het instellen van de gewichten op deze manier komt overeen met de wijze waarop het in het oorspronkelijke probleem is aangepakt. Ook daar wordt eerst geprobeerd zo veel mogelijk diensten in te plannen. Bij gelijke diensten begint pas de belasting, de overtredingen van de criteria, een rol te spelen. In het oorspronkelijke probleem worden individuen die de regels overtreden echter niet gezien als mogelijke oplossingen. Bij het abstracte rooster vergroten we dus het aantal mogelijke oplossingen door dit wel toe te laten. Door de gewichten te veranderen in de laatste fase van het

maken van het abstracte rooster kan worden voorkomen dat het uiteindelijke abstracte rooster overtredingen van regels heeft.

Parent selection

De individuen die voor nieuwe oplossingen moeten zorgen kunnen worden gekozen aan de hand van de fitness van de individuen. Mogelijkheden zijn onder andere tournament selection of elitisme.

De methoden die gebruikt worden voor het oorspronkelijke probleem kunnen ook worden toegepast voor het abstracte rooster. De selectiemethoden die daar worden gebruikt zijn recentelijk onderzocht in ([2] Van der Put, 2005). Hierin kwam naar voren dat tournament selectie als parent selection methode de beste resultaten opleverde.

Bij tournament selectie worden k individuen willekeurig geselecteerd. De beste van deze individuen wint het toernooi en wordt daarmee een parent. Dit proces herhaalt zich tot er μ parents zijn geselecteerd. De kans dat een individu wordt gekozen als parent hangt van een aantal dingen af.

- De ranking van het individu in de population.
- De toernooi grootte k. Hoe groter het toernooi, hoe groter de kans dat de betere individuen in het toernooi komen en gekozen worden als winnaar.
- Of individuen met of zonder teruglegging worden gekozen. In het laatste geval is het niet mogelijk de slechtste k-1 individuen te selecteren als ouder. Met teruglegging is dit wel mogelijk bij een gunstige trekking.

In ([2] Van der Put, 2005) is het effect van tournament selectie op het huidige genetisch algoritme onderzocht. Bij die tournament selectie is gebruikt gemaakt van een aantal geselecteerde individuen k van 2 en 6. De selectie is met teruglegging uitgevoerd. Uit het onderzoek bleek dat een tournament grootte van 6 meest geschikt was.

Recombinatie

Mutatie

Individuen kunnen op verschillende manieren muteren. We kunnen willekeurig abstracte diensten veranderen. Ook kunnen we die diensten aanpassen die een regel of criterium overtreden.

Cross-over

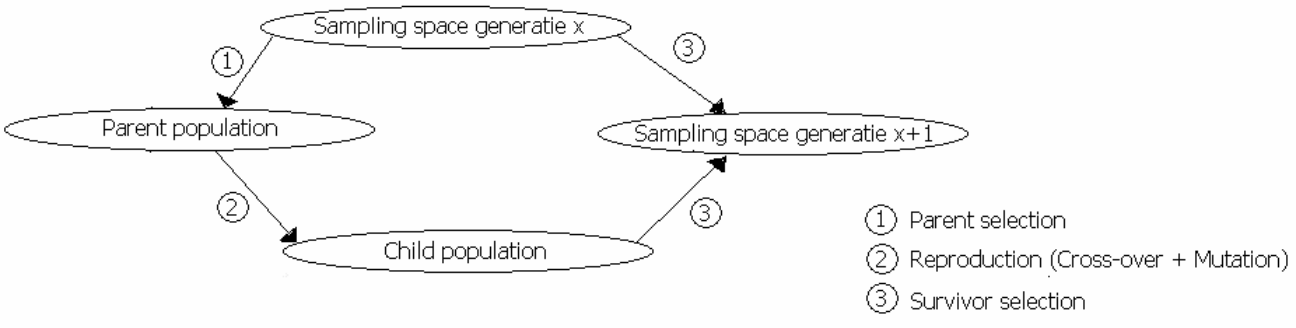
Ook voor cross-over zijn er veel manieren om individuen met elkaar te combineren. Dagen of medewerkers kunnen bijvoorbeeld worden verwisseld.

Survivor selection

Voor survivor selection zijn opnieuw tal van mogelijkheden. Ook survivor selection is onderzocht in ([2] Van der Put, 2005). Hierin kwam naar voren dat sampling space elitisme de beste resultaten gaf.

Definitie Een *sampling space* is een populatie waaruit de volgende generatie wordt bepaald. Een sampling space wordt in ons geval gevormd door individuen van de child population en de voorgaande sampling space.

Bij sampling space elitisme wordt de sampling space voor de volgende generatie gevormd door alle N individuen uit de child population te laten overleven en de E beste individuen uit de vorige sampling space. Uit deze nieuwe sampling space worden vervolgens weer de nieuwe parents gekozen. Schematisch ziet sampling space elitisme er zo uit:



Figuur 15: Sampling space elitisme

6.2 Kolomgeneratie

Een andere mogelijkheid is het gebruik van kolomgeneratie[1]. De formulering van het abstracte rooster kan worden omgezet naar een LP probleem. Omdat we gebruik zullen maken van meerdere doelstellingen wordt dit vaak Goal Programming genoemd. Hieronder zal ik een beschrijving geven van een mogelijke representatie om een abstract rooster te maken. Daarbij worden de volgende variabelen gebruikt:

t	Dit is het type dienst. De variabele is 1 als het gaat om een normale dienst, en 2 als het een nachtdienst betreft.
m	Dit is een medewerker. $m \in \{1, \ldots, \text{aantal medewerkers}\}$
E	Het aantal dagen in de roosterperiode.
d	De dag in de roosterperiode. $d \in \{1,\ldots,E\}$.
w	De dag in de week. $w \in \{1,\ldots,7\}$.
x_{tmd}	Is 1 als medewerker m ingeroosterd staat voor een dienst van het type t op dag d. Is 0 als dit niet het geval is.
v_{td}	Is het verschil tussen de bezettingeis en het aantal ingeplande diensten van type t op dag d.
C	Het totaal aantal criteria.
i	Dit is een criterium. $i \in \{1,\ldots,C\}$.
y_{id}	Is 1 als criterium i op dag d wordt overtreden, 0 anders.
c_i	De kosten voor het overtreden van criterium i.

Omdat aan de regels automatisch wordt voldaan is het doel van het lineair programmeringprobleem om het verschil tussen de bezettingeis en het aantal ingeplande diensten voor de dagen van de roosterperiode te minimaliseren

$$\text{minimaliseer } \sum_{\text{alle d}} (v_{1d} + v_{2d})$$

Daarnaast willen we onder het aantal ingeplande diensten zoveel mogelijk voldoen aan de criteria.

$$\text{minimaliseer } \sum_{\text{alle d en i}} (c_i * y_{id})$$

In HARMONY zijn er criteria die als ze overtreden worden kwadratische belasting hieraan toewijzen. Daarom kan de belasting van deze criteria niet worden overgenomen. In dat geval betekent een overtreding een normale beboeting.

We willen deze doelen minimaliseren gegeven de regels. Ook zullen we randvoorwaarden moeten maken die ervoor zorgen dat de v_{td} juist worden ingesteld en overtredingen van criteria ervoor zorgen dat de y_{id} juist worden ingesteld. Hieronder zal ik een aantal voorbeelden geven hoe regels kunnen worden meegenomen binnen het lineaire model.

[1] Voor meer informatie zie [4] H. Tijms, A. Ridder, 2002, "Mathematische Programmering"

Voorbeelden van te implementeren regels

Nachtdienst series

Indien sommige nachtdiensten na 2 uur eindigen, bedraagt het aantal achtereenvolgende nachtdiensten maximaal 3.

$\sum_{i=d}^{d+4} x_{2md} \leq 3$ voor alle d,m

Aantal nachtdiensten

Indien sommige nachtdiensten na 2 uur eindigen, bedraagt het aantal nachtdiensten per periode van 5 achtereenvolgende weken maximaal 5.

$\sum_{i=d}^{d+35} x_{2md} \leq 5$ voor alle d,m

Aantal diensten op een dag

Het maximaal aantal diensten met arbeid dat op een dag begint is 1.

$x_{1md} + x_{2md} \leq 1$ voor alle d,m

Dienst Series

Het aantal achtereenvolgende diensten is maximaal 6.

$\sum_{i=d}^{d+7} (x_{1md} + x_{2md}) \leq 6$ voor alle d,m

6.3 Voordelen en nadelen methoden

Hierboven hebben we twee methoden besproken waarmee abstracte roosters kunnen worden gemaakt. Voor het verslag heb ik mij beperkt tot deze methoden, hoewel er meer mogelijkheden zijn.

Elke methode heeft het voordeel dat het algoritme op elk willekeurig moment onderbroken kan worden zonder daarbij de tot dan toe best gevonden oplossing te verliezen. Deze eigenschap is belangrijk omdat in HARMONY de tijd altijd beperkt is. Als het algoritme onderbroken wordt, kan de best gevonden oplossing worden geretourneerd.

Voordelen kolomgeneratie

Een voordeel van kolomgeneratie is dat het in staat is de optimale oplossing te vinden. Als het algoritme de tijd krijgt, is zelfs gegarandeerd dat de optimale oplossing wordt gevonden. Dit betekent dat we het best mogelijke abstracte rooster altijd kunnen vinden met deze methode. Het kwaliteitsverlies abstract rooster is in dit geval dus 0. De enige kwaliteit die we nu verliezen zit in het inplannen van het abstracte rooster en het weglaten van regels en criteria.

Nadelen kolomgeneratie

Een nadeel is dat voor het oplossen van het lineair programmering probleem een commerciële solver nodig is, of een solver gebouwd moet worden. Het maken van een module die het probleem snel kan oplossen is niet een eenvoudige opgave. Een commerciële solver betekent echter dat een licentie vereist is.
Als een commerciële solver wordt gebruikt moet naast het programmeren van het lineair programmeringprobleem zelf, de communicatie tussen HARMONY en de solver worden geprogrammeerd. De output van HARMONY zal geschikt moeten worden gemaakt voor de input van de solver en de output van de solver moet weer worden terugvertaald naar HARMONY. Dit betekent dat er een extra stap aan programmeren bijkomt.

Daarnaast is het implementeren van regels en criteria niet eenvoudig. Het omzetten van een regel of criterium in een lineaire restrictie vereist gebruik van trucjes. Daarnaast kan het erg ingewikkeld

zijn omdat uitzonderingen op de regel zijn toegestaan. In dat geval moet een als-dan constructie lineair worden gemaakt. Zo is onderstaande regel bijzonder lastig te verwoorden in lineaire restricties.

Voorbeeld van regel met uitzonderingen

Rust na dienst, twee uitzonderingen

De minimale onafgebroken rusttijd bedraagt 11 uur in een periode van 24 achtereenvolgende uren.
De minimale onafgebroken rusttijd na afloop van een nachtdienst die eindigt na 2:00 uur bedraagt 14 uur.
De minimale rusttijd mag eenmaal in een periode van 21 maal 24 achtereenvolgende uren worden ingekort tot tenminste 8 uur en mag in die periode nog een maal worden verkort tot 10 uur.

Voordelen genetisch algoritme

Een voordeel van een genetisch algoritme is de flexibiliteit. Als het algoritme eenmaal is gemaakt, is het vrij eenvoudig veranderingen door te voeren. Door aanpassing van de instellingen kan de werking van het algoritme behoorlijk worden beïnvloed. Dit geeft controle over het proces.

Daarnaast heeft het probleem van het genereren van het abstracte rooster veel weg van het genereren van het echte rooster. Omdat voor het laatste geval al een genetisch algoritme wordt gebruikt, is het mogelijk eigenschappen van dat algoritme over te nemen in het nieuwe algoritme. Als een bepaalde selectiemethode goed werkt voor het oude algoritme, valt te verwachten dat dit ook het geval is voor het maken van het abstracte rooster.

Een ander voordeel is dat een genetisch algoritme elke keer meerdere verschillende abstracte roosters genereert. Als er meer dan één beginrooster gemaakt moet worden hoeft daarvoor het algoritme niet meerdere keren te worden gedraaid. Bovendien kan in het algoritme worden geregeld dat de verschillende abstracte roosters divers zijn. Dit betekent dat het wellicht mogelijk is nog een nadeel van de huidige methode tegen te gaan. De huidige methode heeft, zoals in hoofdstuk 4: 'Huidige planautomaat' is beschreven, moeite met het maken van meerdere gelijkwaardige roosters.

Nadelen genetisch algoritme

Dezelfde flexibiliteit die zorgt voor controle over het proces heeft als nadeel dat het genetisch algoritme erg veel parameters hebben die gevarieerd kunnen worden. Vaak is het onduidelijk wat de beste waarde van een parameter is. Om hierachter te komen zal voor veel waarden de beste configuratie achterhaald moeten worden. De werking van het algoritme wordt hierdoor sterk beïnvloed. Ook de verschillende componenten van het algoritme kunnen geheel naar eigen wens worden gemaakt. Ook hier kunnen parameters worden gebruikt die ingesteld moeten worden. De flexibiliteit zorgt er dus voor dat veel tijd gaat zitten in het vaststellen van de goede instellingen van het algoritme.

Een ander nadeel is dat het vinden van de optimale oplossing niet kan worden gegarandeerd hoe lang het algoritme ook mag draaien. Het optimale abstracte rooster hoeft dus niet te worden gevonden door het genetisch algoritme. Bij deze methode treedt dus kwaliteitsverlies abstract rooster op. Doel is dit verlies zo klein mogelijk te maken. Hoeveel kwaliteit er verloren gaat kan worden gemeten door gebruik te maken van de methode kolomgeneratie die altijd een optimale oplossing kan vinden.

Keuze voor methode

Door de beperkte beschikbare tijd tijdens de stage was het niet mogelijk beide methoden te implementeren. Daarom is de keuze gemaakt het abstracte rooster te maken met behulp van een genetisch algoritme. Vooral de mogelijkheid om informatie van het bestaande genetisch algoritme te kunnen herbruiken en de mogelijkheid meerdere roosters in één run van het algoritme te maken, gaven de doorslag te kiezen voor deze methode.

7. Implementatie abstract rooster

In het voorgaande hoofdstuk hebben we gekozen om het abstract rooster aan de hand van een genetisch algoritme te maken. In dit hoofdstuk zullen we ingaan op de implementatie van dat genetisch algoritme. Hierbij zullen we ingaan op de componenten initialisatie, parent selectie, recombinatie, mutatie, survivor selectie en de fitness function. Voor meer informatie over het genetisch algoritme zie Bijlage C: 'Basisprincipes genetisch algoritme'. Tijdens het implementeren van het algoritme vielen een aantal zaken op. Hoewel deze niet zijn getest zijn ze hieronder ook vermeld, omdat ze belangrijk waren voor de vorming van het uiteindelijke algoritme.

Ook bespreek ik welke methode gebruikt kunnen worden voor het inplannen van diensten in het abstracte rooster om het uiteindelijke beginrooster te vormen.

7.1 Initialisatie

Om het genetisch algoritme te starten is een aantal beginroosters nodig. Een aantal methodes is geprobeerd.

Methode 1: Willekeurig

Wijs willekeurige abstracte diensten toe aan medewerkers zonder daarbij rekening te houden met bezettingseisen. Hierdoor is het mogelijk dat er meer diensten ingepland worden dan nodig zijn. Dit betekent ook dat het mogelijk moet zijn via mutatie of cross-over diensten uit het rooster te verwijderen.

Definitie Het *toewijzen van een abstracte dienst aan een medewerker* is het veranderen van die abstracte dienst van de medewerker in geen dienst, normale dienst of nacht dienst. Hoewel de diensten niet echt worden toegewezen, hebben we daar voor de eenvoud wel over.

Methode 2: Willekeurig, rekening houdend met de bezettingeis

Wijs willekeurige abstracte diensten toe aan medewerkers. De kans op dit toewijzen hangt af van de bezettingseis van een type dienst. Tijdens het implementeren is gekozen voor de volgende kans:

$$Kans = \frac{bezettingseis\ van\ een\ type\ dienst}{bezettingseis\ van\ alle\ typen\ diensten}$$

Voorbeeld

Stel dat de bezettingseis van een normale dienst op een dag 8 is en van een nachtdienst 2. De kans dat een normale dienst bij een medewerker op die dag wordt ingepland is gelijk aan 8/10 en de kans op een nachtdienst is 2/10.

Door het invoeren van deze kans zal het aantal diensten dat wordt ingepland dichter bij de bezettingeis liggen dan bij methode 1 het geval is.

Methode 3: Plan bezetting, zonder teruglegging

Wijs een abstracte dienst op een dag toe aan een medewerker als nog geen abstracte dienst is ingeroosterd op die dag. Doe dit net zo lang totdat alle abstracte diensten zijn ingeroosterd of als aan alle medewerkers al een abstracte dienst is toegewezen. Op deze manier wordt de bezettingseis willekeurig ingepland in het abstracte rooster als er daarvoor genoeg werknemers zijn.

Methode 4: Plan bezetting, met teruglegging

Wijs een abstracte dienst op een dag toe aan een medewerker. Als een medewerker al een abstracte dienst heeft dan wordt deze dienst overschreven. Doe dit net zo lang totdat alle abstracte diensten één keer in het rooster zijn gezet. Opnieuw wordt geprobeerd de bezettingeis in te plannen. Doordat diensten kunnen worden overschreven zal dit niet altijd helemaal gebeuren. Het abstracte beginrooster bevat dus altijd minder of precies evenveel diensten als de bezettingseis.

Opmerkingen initialisatie

Methode 1 is de eerste methode die is geïmplementeerd. Het aantal diensten dat hiermee wordt ingepland, ligt echter ver van de bezettingeis af. Hierdoor heeft het genetisch algoritme veel generaties nodig om het aantal ingeplande diensten in de buurt van de bezettingeis te krijgen. Methode 2 moet een oplossing geven voor dit probleem. De resultaten met deze methode zijn echter niet veel beter.

Om toch meer in de buurt van de bezettingeis te komen is methode 3 ingevoerd. Bij deze methode ontstaat een probleem als de bezettingseis groter is dan het aantal beschikbare medewerkers kunnen uitvoeren. Deze situatie komt bij meerdere klanten van HARMONY voor. In dat geval zullen alle medewerkers op elke dag een abstracte dienst krijgen toegewezen. Hierdoor ontstaan veel overtredingen van regels.

Om dit probleem tegen te gaan is uiteindelijk methode 4 gemaakt. Bij deze methode ligt het aantal ingeplande diensten in de buurt van de bezettingseis, maar die zal nooit helemaal worden gehaald. Dit laat nog wat ruimte over in het rooster. Deze methode blijft wel veel overtredingen van regels houden, omdat we daar nog geen rekening mee houden.

Het rooster, dat in HARMONY voor de te plannen periode staat ingepland, wordt meegenomen als beginrooster. Dit is gedaan zodat dat rooster kan worden gebruikt om verder mee te werken.

In de testset is dit initiële rooster altijd leeg. Omdat de roosters die door bovenstaande methode gemaakt worden veel regels overtreden en omdat het overtreden van regels een zwaardere boete geeft dan het niet inplannen van diensten, is het lege rooster altijd het beste rooster na de initialisatie. Hierdoor blijf het genetisch algoritme veel tijd nodig hebben voordat het in de buurt van de bezettingseis komt.

Om dit tegen te gaan is geprobeerd dit eerste rooster niet mee te nemen in de beginpopulatie. Hoewel het aantal ingeplande diensten in het begin dichter in de buurt van de bezettingeis ligt heeft het algoritme ditmaal veel tijd nodig om alle regels die worden overtreden op te lossen. Qua tijd was er nauwelijks verschil merkbaar tussen beiden methoden. Het lege rooster wordt daarom toch steeds meegenomen.

7.2 Parent Selection

In ([2] Van der Put, 2005) is een aantal verschillende parent selectie methoden onderzocht voor het oorspronkelijke genetisch algoritme. Aangezien het genetisch algoritme voor het abstracte rooster eenzelfde structuur heeft als het oorspronkelijke probleem, nemen we aan dat die tests representatief zijn voor ons probleem. Als beste parent selectie kwam in de scriptie tournament selectie naar voren, zoals ook beschreven staat in het hoofdstuk 6: 'Maken van abstract rooster'. We voeren de tournament selectie uit met teruglegging, zodat ook het slechtste individu de kans heeft gekozen te worden. Het aantal deelnemers dat aan het toernooi deelneemt wordt besproken in het hoofdstuk 9: 'Abstract rooster instellingen testresultaten'.

7.3 Mutatie

De methoden die zijn geïmplementeerd om individuen willekeurig te veranderen zijn de volgende:

Methode 1: Muteer willekeurige dienst

Pak met een kans σ een willekeurige dienst op een dag en zet die dienst neer bij een medewerker die nog geen dienst heeft.

Methode 2: Voeg toe/verwijder dienst

Pak met een kans σ een dag. Bekijk of het aantal ingeplande diensten op die dag afwijkt van de bezetting en verwijder een dienst / voeg een dienst toe al naar gelang er te weinig / te veel zijn ingepland.

Methode 3: Verwissel diensten

Haal de dagen op met diensten die een overtreding van een regel of criterium veroorzaken. Verander een percentage σ van deze dagen. Voor elke gekozen dag kiezen we een medewerker uit waarvan we een dienst gaan aanpassen. Met 75% kans is dit de medewerker met een dienst met de hoogste belasting, met 15% kans de medewerker met een dienst met de één na hoogste belasting en met 10% kans een medewerker met een dienst met de twee na hoogste belasting. Voor deze medewerker bepalen we willekeurig of een dienst op dezelfde dag of een dag ervoor of erna wordt aangepast. Deze percentages zijn tijdens het implementeren van het algoritme tot stand gekomen. Deze waarden bleken de beste te zijn van de waarden die zijn geprobeerd door 'trial and error'.

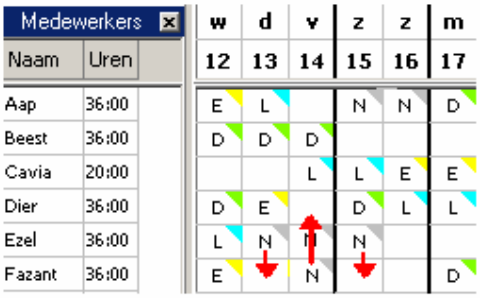
Voorbeeld

Medewerkers		w	d	v	z	z	m
Naam	Uren	12	13	14	15	16	17
Aap	36:00	E	L		N	N	D
Beest	36:00	D	D	D			
Cavia	20:00			L	L	E	E
Dier	36:00	D	E		D	L	L
Ezel	36:00	L	N	N	N		
Fazant	36:00	E		N			D

Figuur 16: Mutatievoorbeeld 1

We zien in deze situatie dat de dienst van medewerker Fazant op vrijdag de regel overtreedt, dat series nachtdiensten minimaal lengte 2 moeten hebben. Aangezien dit de enige overtreding is op die dag wordt gekozen om iets te doen op vrijdag voor medewerker Fazant.

Op vrijdag kunnen we drie dingen doen om het probleem op te lossen. Of we verplaatsen de nachtdienst naar een andere medewerker, of verplaatsen een nachtdienst van donderdag naar medewerker Fazant of we verplaatsen een nachtdienst van zaterdag naar medewerker Fazant. In alle gevallen is de overtreding van Fazant verholpen.



Figuur 17: Mutatievoorbeeld 2

Als voor een dag is gekozen zoeken we een willekeurige medewerker waarmee we diensten gaan ruilen. Als de dienst die we moeten aanpassen van het type 'geen dienst' is, dan wordt met 50% kans een medewerker gekozen die een dienst heeft van het type nacht dienst en met 50% een medewerker met een gewone dienst. Is de aan te passen dienst een normale / nacht dienst dan wordt met 80% kans een medewerker gekozen met een dienst van het type 'geen dienst' en 20% van het type nacht / normale dienst. Ook hier zijn de percentages door 'trial and error' tot stand gekomen.

Methode 4: Verwijder overtreding van regels

Pak alle dagen die diensten hebben die zorgen voor een overtreding van een regel. Kies willekeurig maximaal 2 van deze dagen uit. Verwijder voor die dagen een dienst die een overtreding van een regel veroorzaakt.

Evolueren van σ

In een aantal van de hierboven genoemde mutaties komt een parameter σ voor. Deze parameters evolueren tijdens het uitvoeren van het algoritme mee op de manier voorgesteld in ([3] A. Eiben, 2003). Voordeel hiervan is dat er niet één vaste waarde moet worden bepaald voor deze kansen. Het probleem met één vaste waarde is vaak dat de beste waarde verschilt per generatie. Zo kan een kleine waarde van de parameter goed zijn in de eerste generaties en juist een hoge waarde in de laatste generaties. Om hier mee om te kunnen gaan evolueert de parameter mee, zodat de waarde elke generatie anders is.

Dit evolueren gebeurt op de volgende manier. Elk individu heeft de waarde van de parameters waarmee ze zijn gemaakt. De beste individuen zullen overleven en daarmee ook de parameter waarden waarmee ze zijn gemaakt. Deze waarden leverden schijnbaar het beste resultaat. Vervolgens worden deze waarden gebruikt en willekeurig aangepast om weer nieuwe individuen mee te maken.

Een oplossing bestaat dus uit een abstract rooster en een parameter σ. Voordat een mutatie wordt uitgevoerd wordt de parameter aangepast op de volgende manier:

$$\sigma = \sigma * e^{\tau * N(0,1)}$$

τ heet ook wel de 'learning rate'. De waarde wordt vooraf als constante gekozen. Vaak is dit $\tau = \frac{1}{\sqrt{n}}$ met n het aantal individuen in de sampling space. N(0,1) is een trekking uit de standaard normale verdeling. Om te voorkomen dat de waarden voor σ te klein worden is er een minimumwaarde voor de parameter. Mocht σ kleiner worden dan deze waarde, dan wordt σ die waarde. Als σ groter wordt dan 1 wordt hij gelijk gesteld aan 1. De mutatie wordt vervolgens met deze nieuwe parameter uitgevoerd.

Opmerkingen mutatie

Methode 1 was de eerst geïmplementeerde methode. Het bleek tijdens het uitvoeren van tests dat deze methode te willekeurig was. Naarmate de kwaliteit van de roosters toenam, nam het aantal diensten dat nog kon worden verbeterd af. De kans dat de diensten die nog verbetering konden opleveren ook werden gekozen werd steeds kleiner. Het gevolg was dat generaties lang niks gebeurde.

De overige 3 methoden zijn momenteel allemaal geïmplementeerd en worden tijdens het uitvoeren van het algoritme toegepast. Op dit moment wordt er niet gekozen tussen de 3 methoden. Elke parent wordt door elke mutatieoperator gemuteerd.

7.4 Recombinatie

Methode 1: wissel medewerkers

Pak met vaste kans een medewerker en wissel de diensten van twee roosters voor die medewerker.

Methode 2: wissel dagen

Pak met een vaste kans een dag en wissel de dagen tussen twee roosters.

Opmerkingen recombinatie

Ook bij de recombinatie is geen keuze gemaakt tussen de twee methoden. Alle parents worden paarsgewijs door beide operatoren gerecombineerd. Dit heeft tot gevolg dat als 10 parents zijn geselecteerd er 40 children worden gemaakt. 30 children worden gemaakt door de mutaties en de overige 10 door recombinatie.

7.5 Survivor selection

In ([2] Van der Put, 2005) is een aantal verschillende survivor selectie methoden onderzocht voor het oorspronkelijke genetisch algoritme. Aangezien het genetisch algoritme voor het abstracte rooster eenzelfde structuur heeft als het oorspronkelijke probleem, nemen we aan dat die tests

representatief zijn voor ons probleem. Als beste survivor selectie kwam in de scriptie sampling space elitisme naar voren, zoals ook beschreven staat in het hoofdstuk 6: 'Maken van abstract rooster'.

Opmerkingen survivor selection

Op dit moment wordt bij de survivor selectie nog geen rekening gehouden met de diversiteit van de populatie. Met diversiteit bedoelen we het verschil tussen de roosters. Hoe meer de roosters van elkaar verschillen hoe meer divers de populatie is.

7.6 Fitnessfunctie

De fitness functie is geïmplementeerd op de wijze zoals in het hoofdstuk 6: 'Maken van abstract rooster' staat beschreven. De scores voor het aantal regels dat wordt overtreden, het aantal nog in te plannen diensten en de criteria die worden overtreden, worden vermenigvuldigd met een gewicht dat ingesteld kan worden. Hierbij wordt het grootste gewicht gelegd bij het overtreden van de regels, daarna bij de nog in te plannen diensten en als laatste bij het overtreden van de criteria.

Ook is de mogelijkheid geïmplementeerd om in de score de afstand tussen individuen mee te nemen. Hierdoor kan de diversiteit van de verschillende individuen in de populatie bevorderd worden. Op deze manier blijven meer verschillende roosters in de populatie bestaan.

Locale minima

Tijdens de implementatie bleek dat het algoritme vaak in locale minima blijft steken. Hierbij gaat het om het aantal ingeplande diensten dat voor meerdere generaties gelijk blijft. Om het algoritme sneller meer diensten te laten inplannen is een reïnitialisatie van de gewichten toegevoegd. Als er in een x aantal generaties geen nieuwe diensten zijn ingepland, dan worden de gewichten die horen bij het overtreden van regels en criteria en de nog in te plannen diensten aangepast. De gewichten worden zo aangepast dat het inplannen van diensten belangrijker wordt dan het voorkomen van overtredingen van regels. De gewichten worden na het uitvoeren van een paar generaties weer teruggezet op de oude waarde. Op deze manier zullen de individuen in die generaties diensten gaan inplannen ook al geeft dat overtredingen. Na de generaties met gewijzigde gewichten moeten die overtredingen weer worden opgelost. Soms zal dit gebeuren door de diensten uit te plannen, maar vaak kan ook een andere oplossing worden gevonden. Op deze manier kan het algoritme sneller meer diensten inplannen.

7.7 Regels en criteria

Hieronder volgt een beschrijving van de regels en criteria waar het abstracte rooster op controleert als deze door de gebruiker zijn ingesteld. Per regel zal worden aangegeven hoe de regel wordt gecontroleerd en of dit strenger of minder streng wordt gedaan. Staat er niks vermeld, dan is de regel hetzelfde geïmplementeerd. Waarden die kunnen worden ingesteld in HARMONY zijn cursief en onderstreept.

Regels

Werkuren en beschikbaarheid

Een medewerker mag in een roosterperiode niet meer dan *4* uren werken boven het aantal uren dat hij of zij beschikbaar is voor de betreffende roostergroep.

In het abstracte rooster is niet bekend hoe lang een dienst precies duurt. Daarom is voor de implementatie van deze regel gekozen om voor een nachtdienst de gemiddelde werkduur van alle nachtdiensten te rekenen. Voor een dagdienst wordt de gemiddelde werkduur van alle dagdiensten genomen.

Aantal vrije weekenden

Minimaal *2* vrije weekenden van *60* uren per *5* weken, omvattende *zaterdag* *0:00* tot *maandag* *4:00* uur. Ook geldig voor bereikbaarheidsdiensten. Ook geldig voor diensten van het soort Consignatie, Aanwezigheid, Piket.

De implementatie van deze regel is minder streng dan de oorspronkelijke. De regel controleert of zaterdag en zondag een dienst bevatten, maar controleert niet of voldaan wordt aan de 60 uur. Aanvankelijk was de regel strenger geïmplementeerd, maar hierdoor bleek in de testcase dat veel diensten overbleven omdat weekenden onterecht als werkweekenden werden gezien, terwijl ze eigenlijk vrij waren.

Arbeidstijd per periode (dag)

De maximale arbeidstijd per week bedraagt gemiddeld *36* uur in een periode van *13* achtereenvolgende weken.

Bij deze regel wordt ook gebruik gemaakt van de gemiddelde werkduur van de dag- en nachtdiensten net als bij 'werkuren en beschikbaarheid'.

Nachtdienst series

Indien sommige nachtdiensten na *2:00* uur eindigen, bedraagt het aantal achtereenvolgende nachtdiensten maximaal *3*.

De regel is strenger geïmplementeerd. In het abstracte rooster geldt deze regel voor alle nachtdiensten. In het echte rooster hoeft dit niet het geval te zijn. Als een nachtdienst vóór die tijd eindigt geldt de regel niet.

Aantal nachtdiensten

Indien sommige nachtdiensten na *2:00* uur eindigen, bedraagt het aantal nachtdiensten per periode van *5* achtereenvolgende weken maximaal *3*.

Voor deze regel geldt hetzelfde als voor de regel 'Nachtdienst series'. De regels is dus strenger geïmplementeerd.

Rust na serie nachtdiensten

Na het beëindigen van een reeks van minimaal *1* achtereenvolgende nachtdienst moet een onafgebroken rust van tenminste *42* uur worden genoten.

De implementatie van deze regels is strenger. Van alle nachtdiensten wordt de tijd waarop de laatste nachtdienst eindigt bewaard. Ook wordt voor elk type abstract dienst bijgehouden op welke tijd de vroegste dienst begint. Bij de tijd van de laatste nachtdienst wordt de 42 uur opgeteld. Vervolgens wordt voor de abstracte diensten van de volgende dagen gekeken of het vroegst mogelijke begintijdstip van het type dienst dat staat ingepland op een dag voldoet aan de 42 uur rust.

Aantal diensten op een dag

Het maximaal aantal diensten met arbeid dat op een dag begint is *1*.

Dienst Series

Het aantal achtereenvolgende diensten is maximaal *6*.

Criteria

Minimale/Maximale aantal diensten

Instellingen voor het totaal aantal diensten in een periode van *5* weken:

- Minimaal: *0*
- Gewenst: *2*
- Maximaal: *3*

❖ Dit criterium is geldig voor *alle dagen van de week*.
❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling *DNSTVRZ Test* met de namen *N*.
❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste *0* en niet meer dan *48* uur per week.
❖ Gewicht = *1000.*

De namen van alle diensten worden opgeslagen in het abstracte rooster. Per type weet het abstracte rooster welke namen bij dit type horen. Als een criterium geldt voor een dienst met een bepaalde naam, wordt gekeken welk abstract dienst type bij deze naam hoort. Vervolgens worden voor alle abstracte diensten van dat(die) type(s) het criterium gecontroleerd. Dit criterium wordt dus strenger gecontroleerd.

Compleet Weekend

Voor diensten zoals hieronder geselecteerd, beginnend tussen *vrijdag 22:00* uur en *maandag 0:00* uur dient te gelden: of geen diensten, of tenminste 2 diensten.

- Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling *DNSTVRZ_Test* met de namen *D, E, L, N*.
- Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste *0* en niet meer dan *48* uur per week.
- Gewicht = *1000*.

Met behulp van de vroegste begintijd van een type abstract dienst wordt gekeken of het type binnen de ingestelde periode valt. Voor het selecteren van de diensten op basis van namen wordt hetzelfde gedaan als hierboven. Ook deze regel wordt dus strenger geïmplementeerd.

Aantal dagen rust na een dienst

De rust na een serie diensten die eindigt met een dienst zoals hieronder geselecteerd, bedraagt ten minste *2* dagen.

- Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de diensten verzameling *DNSTVRZ_Test* met de namen *D, E, L*.
- Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste *0* en niet meer dan *48* uur per week.
- Gewicht = *100*.

Deze regel is strenger geïmplementeerd om dezelfde reden als 'Minimale/maximale aantal diensten'.

Aantal dagen rust voor een dienst

De rust voor een serie diensten die begint met een dienst zoals hieronder geselecteerd, bedraagt tenminste *1* dag.

- Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen *N*.
- Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste *0* en niet meer dan *48* uur per week.
- Gewicht = *100*.

Deze regel is strenger geïmplementeerd om dezelfde reden als 'Minimale/maximale aantal diensten'.

Geen losse diensten

Vermijd het inplannen van losse diensten met arbeid.

- Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste *0* en niet meer dan *48* uur per week.
- Gewicht = *1000*.

In het abstracte rooster is sprake van een losse dienst als de abstracte diensten ervóór en erna van het type 'geen dienst' zijn. In HARMONY is een dienst een losse dienst als 24 uur voor het begin van de dienst of 24 uur na het einde van de dienst geen andere dienst wordt gedaan. De implementatie van deze regel is voor abstracte diensten van het type dagdienst strenger. Bij een dagdienst wordt er namelijk aan beide kanten rekening gehouden met meer dan 24 uur. Voor het type nachtdienst wordt vóór het uitvoeren van de dienst met meer dan 24 uur rekening gehouden. Na de nachtdienst kan dit echter minder dan 24 uur zijn, omdat de nachtdienst deels op de volgende dag wordt gedaan.

Wensen

Wel/niet werken in een periode

Medewerker *Jansen* heeft in de periode van *07-12-2004 7:00* uur tot *09-12-2004 22:00* uur als wens: *Niet werken*

De implementatie van deze wens is strenger gemaakt. Van elk type abstracte dienst weten wat het vroegste begintijdstip is en het laatste eindtijdstip. Voor elke abstracte dienst wordt gekeken of de begin- of eindtijd van dat type mogelijk overlapt met de ingestelde periode.

7.8 Inplannen van diensten in abstract rooster

Als laatste stap moeten de diensten in het abstracte rooster worden gezet, waarbij rekening wordt gehouden met alle regels en criteria die zijn ingesteld door de gebruiker. Hierbij is het belangrijk dat we het kwaliteitsverlies inplannen zo klein mogelijk houden. Een aantal methoden worden voorgesteld om de diensten in te plannen.

Methode 1: Bottleneck methode

Het inplannen van diensten in een abstract rooster gebeurd met behulp van de bottleneck methode. Eerst worden alle niet vastgezette diensten uitgepland. Vervolgens wordt een abstract rooster gegenereerd met het genetisch algoritme. Daarna wordt de bottleneck methode aangeroepen.

Zoals we in hoofdstuk 4: 'Huidige planautomaat' hadden gezien, werkt het toewijzen van een dienst aan een medewerker bij de bottleneck methode als volgt: Voor elke dienst wordt er voor de medewerkers een score berekend. Dit gebeurt alleen voor die medewerkers die de dienst kunnen uitvoeren. Dat wil zeggen dat ze de benodigde kwalificaties voor de dienst bezitten en geen overlap van de dienst met andere diensten in hun rooster hebben. De score bepaalt bij welke medewerker deze dienst het beste kan worden toegewezen. De score wordt berekend voor het rooster waarbij de dienst fictief is toegevoegd en wordt bepaald door de regels en criteria die in HARMONY zijn ingesteld. Bij het bepalen van de score wordt vooruitgekeken of series kunnen worden afgemaakt. Is dit het geval, dan levert het toevoegen van bijvoorbeeld een dienst die een serie begint geen negatieve score op.

Na het berekenen van de scores worden de medewerkers gesorteerd waarbij de aantrekkelijkste medewerkers boven aan de lijst komen te staan. Vervolgens wordt de lijst doorlopen waarbij voor elke medewerker gekeken wordt of de oplossing een toelaatbaar rooster geeft volgens de ingestelde regels. Zodra een medewerker gevonden is die hieraan voldoet is de toewijzing afgerond en wordt geprobeerd de volgende dienst toe te wijzen. Als de toewijzing bij geen enkele medewerker een toelaatbaar rooster oplevert wordt de dienst uit de lijst verwijderd en gaat het algoritme verder.

Voor het inplannen van diensten in het abstracte rooster is een aanpassing gemaakt voor het berekenen van de score van de medewerkers. Deze score wordt alleen berekend als in het abstracte rooster ook een dienst van dat type bij die medewerker staat ingepland. Anders wordt er geen score berekend en zal de bottleneck methode de dienst ook niet bij de medewerker zetten.

Methode 2: Stochastische bottleneck methode

Ook de stochastische bottleneck methode kan gebruikt worden voor het inplannen van de diensten in het abstracte rooster. Deze methode zou geen wezenlijk andere resultaten moeten halen dan de eerste methode.

Methode 3: Matching

Een andere methode is de matching methode. De matching methode plant diensten optimaal in voor één dag. Deze methode gaat ervan uit dat de andere dagen staan ingepland zoals ze moeten zijn. Vervolgens probeert de methode de diensten op één dag zo neer te zetten dat de belasting op die dag minimaal is. Deze methode kijkt dus niet vooruit, maar neemt de beslissingen alleen voor die dag. Dit is echter geen probleem met het abstracte rooster. Dat geeft immers aan op welke plekken de matching methode de diensten van die dag mag inplannen. Door te beginnen op de eerste dag van de roosterperiode en dan voor elke dag de matching methode aan te roepen kan het abstracte rooster worden ingevuld.

Opmerkingen

Bij het inplannen van diensten met de bottleneck methode blijft het probleem bestaan dat niet gekeken wordt of het verstandig is een dienst ergens neer te zetten. Er wordt geen rekening gehouden met de consequenties. Door een dienst in het abstracte rooster neer te zetten kunnen daardoor andere diensten niet meer worden ingepland. Hoewel het effect minder is doordat de bottleneck methode de diensten maar op specifieke plaatsen mag zetten, kan dit ervoor zorgen dat er meer kwaliteit verloren gaat dan gewenst. De matching methode moet minder last hebben van dit probleem. De rekentijd die nodig is voor de matching methode is waarschijnlijk wel groter dan bij de bottleneck methode.

8. Test opzet

Dit hoofdstuk beschrijft welke tests we willen uitvoeren en waarom. De tests kunnen we opsplitsen in drie categorieën. In de eerste plaats willen we tests uitvoeren die de invloed van de parameter settings op de werking van het genetisch algoritme bekijken. Daarnaast zijn er tests die een vergelijking maken tussen de verschillende methoden voor het inplannen van een abstract rooster. Als laatste komen de tests aan bod waarmee we een vergelijking willen trekken tussen de methoden waarmee we beginroosters maken.

8.1 Abstract rooster instellingen

Elk genetisch algoritme heeft instellingen die de werking van het algoritme beïnvloeden. Deze instellingen kwamen in hoofdstuk 7: 'Implementatie' aan bod. Voor een aantal parameters willen we bekijken wat hun invloed is op de werking van het algoritme. De volgende instellingen willen we in de tests belichten:

- Sampling space grootte
- Parent population grootte
- Tournament grootte
- Aantal generaties

Voor elk van deze instellingen proberen we een aantal verschillende waarden. Deze waarden vergelijken we door te kijken naar vier dingen:

1. Het gemiddelde aantal nog in te plannen diensten
2. De gemiddelde belasting
3. De gemiddelde kwaliteit per generatie
4. De gemiddelde benodigde tijd

Op basis van deze gegevens kunnen we bekijken welke waarde het beste is. De tests zullen we ceteris paribus uitvoeren. Dat wil zeggen dat alle instellingen gelijk blijven op de te bekijken instelling na. Voor deze tests maken we telkens 20 abstracte roosters per waarde van een instelling. De tests voeren we uit op de testcase beschreven in Bijlage D: 'Testcase'. Hieronder zal worden aangegeven wat het doel van de tests is.

Sampling space grootte

Het doel van deze test is tweeledig. In de eerste plaats is het interessant te bekijken wat de invloed van de sampling space grootte is op de benodigde tijd. Het aantal individuen in de sampling space zou moeten zorgen dat meer tijd nodig is voor het initialiseren en het sorteren van de populatie. Dit eerste omdat meer individuen moeten worden aangemaakt en het tweede omdat de te sorteren populatie groter is.

In de tweede plaats is de vraag wat er verandert in de kwaliteit van het abstracte rooster bij verschillende waarden. Een grotere sampling space betekent dat meer individuen overleven. Hierdoor bestaat de kans dat relatief slechtere individuen langer overleven. De grootte is hierdoor van invloed op de mate van convergentie van het algoritme. Een kleinere sampling space zou ertoe moeten leiden dat het algoritme sneller convergeert dan een grote sampling space. Bij een kleine sampling space mogen alleen de beste individuen zorgen voor children, terwijl bij een grote sampling space de slechtere individuen ook mogen reproduceren. Door alleen de beste individuen te laten reproduceren zal eerder convergentie optreden, omdat impulsen van de slechtere individuen niet mogelijk zijn.

Parent population grootte

In de huidige implementatie van het abstracte rooster wordt elke parent in de parent population gebruikt door elke mutatie- en cross-overoperator. Dit betekent dat de child population altijd vier keer zo groot is als de parent population. De grootte van laatstgenoemde population is dus zeer

bepalend voor de tijd die nodig is voor het maken en controleren van de children. Hoe groter de population hoe meer tijd we nodig zullen hebben.

Daarnaast heeft de parent population grootte effect op de kwaliteit van het abstracte rooster. Hoe meer children worden gemaakt, hoe groter de kans een goed individu te vinden. Het aantal children neemt toe waardoor, bij gelijkblijvende sampling space grootte, de individuen die overleven gemiddeld een hogere kwaliteit zullen hebben. Een grotere parent population moet dus een betere kwaliteit abstract rooster opleveren.

Tournament grootte

De tournament grootte heeft invloed op de mate van convergentie en daarmee de kwaliteit van het rooster. Het aantal parents dat we selecteren bepaalt namelijk de concurrentie die ontstaat tussen de individuen. Als we meer parents gebruiken voor het toernooi is de kans groter dat de betere individuen geselecteerd zullen worden. Hoe groter het aantal parents hoe beter de individuen in de parent population gemiddeld zullen zijn. Dit kan de convergentie versnellen.

Aantal generaties

Het aantal uit te voeren generaties is belangrijk voor de tijd. Het uitvoeren van één generatie kost een bepaalde hoeveelheid tijd. De totale tijd die we nodig hebben is daarom direct gerelateerd aan het aantal generaties dat we kunnen uitvoeren.

Maar het aantal generaties is ook van belang voor de kwaliteit. Het algoritme moet voldoende generaties kunnen uitvoeren om te convergeren naar een goede oplossing. Stoppen we het algoritme te vroeg, dan heeft het algoritme wellicht nog geen goede oplossing gevonden. Daar staat echter tegenover dat, door het anytime behaviour (zie hoofdstuk 3, paragraaf 'Anytime behaviour') van genetisch algoritmen, het uitvoeren van onnodig veel generaties de kwaliteit nauwelijks verbetert. Deze tijd kunnen we ons dan beter besparen.

8.2 Abstract rooster inplannen

In hoofdstuk 7: 'Implementatie' kwamen drie methoden aan bod waarmee we diensten aan medewerkers kunnen toewijzen volgens het abstracte rooster. Deze methoden waren:

1. Bottleneck methode
2. Stochastische bottleneck methode
3. Matching methode

We willen onderzoeken bij welke van deze methoden het kwaliteitsverlies inplannen zo klein mogelijk is. Dit kwaliteitsverlies kan opgedeeld worden in twee delen. In de eerste plaats is het verschil tussen het aantal nog in te plannen diensten belangrijk. Daarnaast is het verschil in belasting tussen het abstracte rooster en het uiteindelijke beginrooster van belang. Het aantal nog in te plannen diensten is hierbij het belangrijkste.

Net als bij de abstract rooster instellingen vergelijken we de methoden door te kijken naar het gemiddelde aantal nog in te plannen diensten, de gemiddelde belasting, de gemiddelde kwaliteit per generatie en de gemiddelde benodigde tijd.

De tests zullen we ceteris paribus uitvoeren. Dat wil zeggen dat we alleen de methode voor het inplannen van het abstracte rooster veranderen. Uitzondering is dat de random seed wel telkens wordt veranderd. Voor elke methode worden telkens 20 nieuwe abstracte roosters gemaakt die worden ingepland. Omdat niet telkens dezelfde roosters worden gebruikt zit in de resultaten een afwijking die ontstaat door de verschillen de roosters gemaakt voor elke methode. De tests voeren we uit op de testcase beschreven in Bijlage D: 'Testcase'.

Naast het bekijken van het gemiddelde kan waar nodig gebruik worden gemaakt van de standaarddeviatie. Daarnaast kan waar nodig gebruik worden gemaakt van de Wilcoxon toets om uitspraken te kunnen doen welke methode het best gebruikt kan worden.

Wilcoxon toets

De Wilcoxon toets voor twee steekproeven is een verdelingsvrije toets waarmee we kunnen toetsen of de medianen van de steekproeven van elkaar verschillen. Daarbij kunnen we ook toetsen of de ene mediaan groter is dan de andere. Dit betekent dat als we willen weten of één methode meer diensten inplant dan een andere methode, we dit kunnen toetsen met waarnemingen van het aantal nog in te plannen diensten. Voor meer informatie over de Wilcoxon toets verwijs ik naar ([10] De Gunst, 2001).

Wiskundig geformuleerd voeren we een Wilcoxon toets als volgt uit:
Laat $t_i^k; i = 1, \ldots, I$ realisaties zijn van een stochast voor model k (k=1,..,K). Neem vervolgens aan dat t_i^1 een verdeling A heeft, t_i^2 verdeling B, etc. Neem aan dat we willen toetsen of de mediaan van één van de verdelingen kleiner is dan die van een andere verdeling. In dat geval maken we gebruik van de volgende hypothese:

$$H_0 := A \leq B$$
$$H_1 := A > B$$

H_0 wordt de nulhypothese genoemd en H_1 de alternatieve hypothese. Deze nulhypothese kunnen we aanpassen naar wat we willen testen. Zo kunnen we als ook A = B, met alternatieve hypothese A ≠ B, of A ≥ B, met alternatieve hypothese A ≤ B, gebruiken. Het resultaat van de toets is een p-waarde.

Definitie De *p-waarde* heet ook wel overschrijdingskans. Als voor een toetsingsgrootheid T de waarde t is waargenomen is de overschrijdingskans gelijk aan $P_{H_o}(T \geq t)$. In woorden is het de kans dat de toetsingsgrootheid een waarde groter dan t heeft onder de nulhypothese. Is de kans kleiner of gelijk aan een gekozen grenswaarde verwerpen we de nulhypothese (zie: [10] J. Oosterhoff, 2001).

Als bij het uitvoeren van een test de Wilcoxon toets is toegepast, zal worden aangegeven wat de nulhypothese is die is gebruikt.

8.3 Beginroosters maken

De verschillende methoden voor het maken van de beginroosters willen we ook met elkaar vergelijken. De methoden, zoals in hoofdstuk 5: 'Nieuwe methoden' beschreven, zijn:

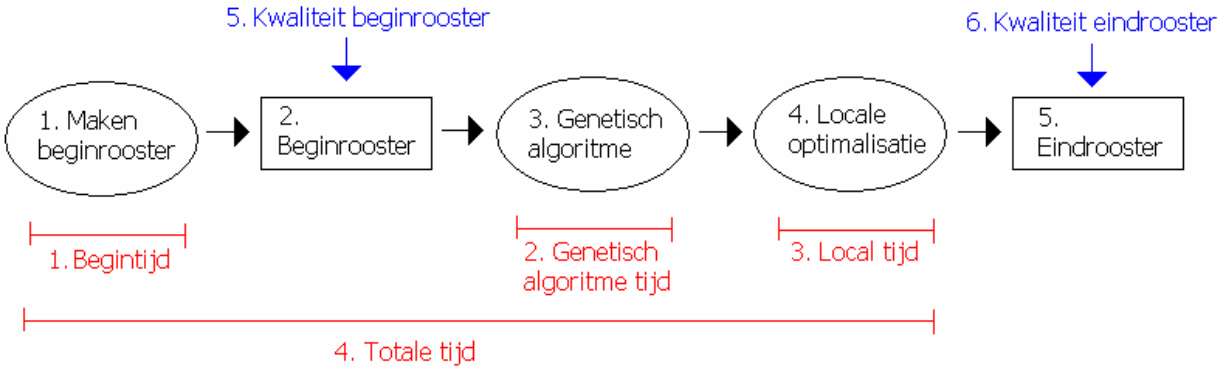
- Bottleneck methode
- Stochastische bottleneck methode
- Bottleneck methode met locale optimalisatie
- Abstract rooster
- Volledig willekeurige roosters

In hoofdstuk 3: 'Opdrachtomschrijving' bespraken we dat we de methoden op een aantal punten met elkaar kunnen vergelijken.

1. De tijd die nodig is om een beginrooster te maken.
2. De tijd die gebruikt is voor het uitvoeren van het genetisch algoritme, gebruik makende van een beginrooster.
3. De tijd die nodig is voor het uitvoeren van locale optimalisatie.
4. De totale tijd nodig voor het maken van het uiteindelijke rooster.
5. De kwaliteit van het gemaakte beginrooster.
6. De kwaliteit van het rooster na het uitvoeren van het genetisch algoritme en locale optimalisatie.

De tijd die we hier bekijken is de tijd die nodig is voor het algoritme om te convergeren naar een oplossing. De kwaliteit van deze oplossingen zal echter verschillen. Bij punt 4 zal ook gekeken worden hoeveel tijd een methode nodig heeft voor het behalen van een bepaalde kwaliteit. Op deze manier kan vergeleken worden of een methode sneller is dan een andere methode.

Schematisch gezegd kijken we naar de volgende onderdelen:



Figuur 18: De onderdelen voor het maken van een rooster

De tests zullen we ceteris paribus uitvoeren. Dat wil zeggen dat we alleen de methode voor het maken van het beginrooster veranderen. Voor deze tests maken we telkens 100 keer een eindrooster met behulp van het algoritme. De tests voeren we uit op de testcase beschreven in Bijlage D: 'Testcase'.

Naast te kijken naar de gemiddelden kunnen we ook hier de standaarddeviatie gebruiken voor het vergelijk. De Wilcoxon toets is opnieuw bruikbaar om beweringen te toetsen.

9. Abstract rooster instellingen testresultaten

In dit hoofdstuk volgt een beschrijving van de resultaten van het uitvoeren van de tests zoals beschreven in Hoofdstuk 8: 'Test opzet'. Hierbij kijken we naar de parameters:

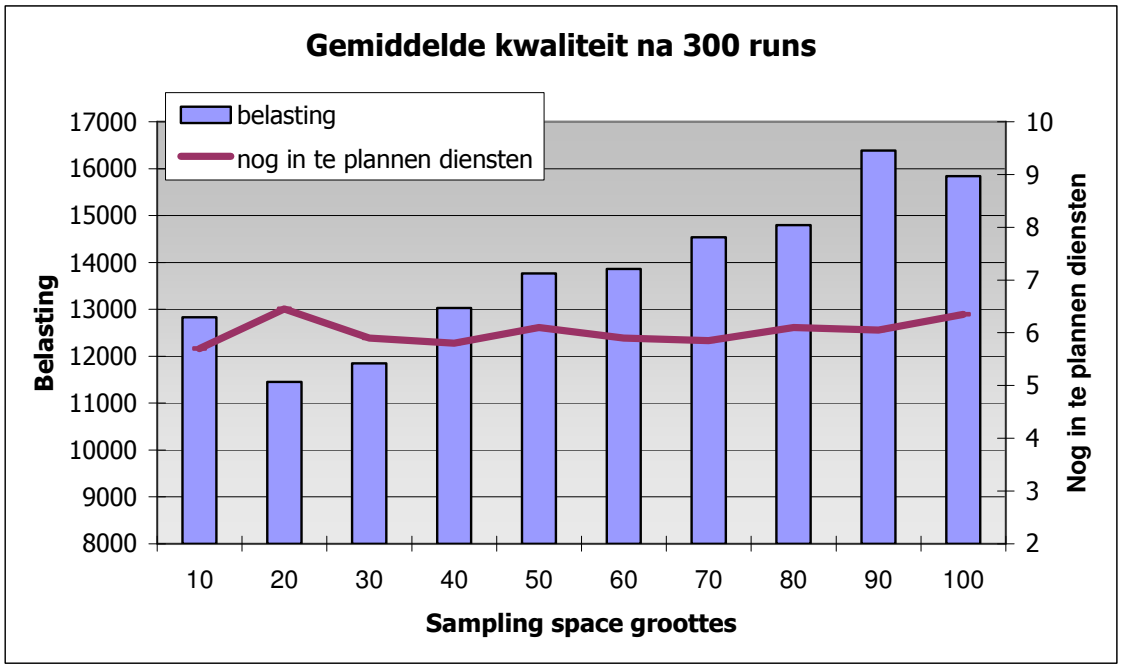
- Sampling space grootte
- Parent population grootte
- Tournament grootte
- Aantal generaties

Bij elke test zal worden aangegeven welke waarden deze parameters hebben, ook voor die parameters die niet worden getest. De basisinstellingen van de parameters zijn een sampling space grootte van 50, een parent population grootte van 50 (dit houdt in dat per generatie 200 children worden gemaakt), een tournament grootte van 4 en 300 generaties. Deze waarden zijn tijdens het implementatie door 'trial and error' tot stand gekomen.

9.1 Sampling space grootte

Voor de test van de sampling space grootte is gekozen voor de waarden 10, 20 t/m 100. Tijdens de implementatie bleek dat beste resultaten behaald konden worden in dit bereik. Daarom is voor deze waarden gekozen. Tijdens het uitvoeren van de test zijn de overige parameters ingesteld op de basisinstellingen. Voor elk van deze waarden zijn 20 runs van 300 generaties uitgevoerd.

Kwaliteit



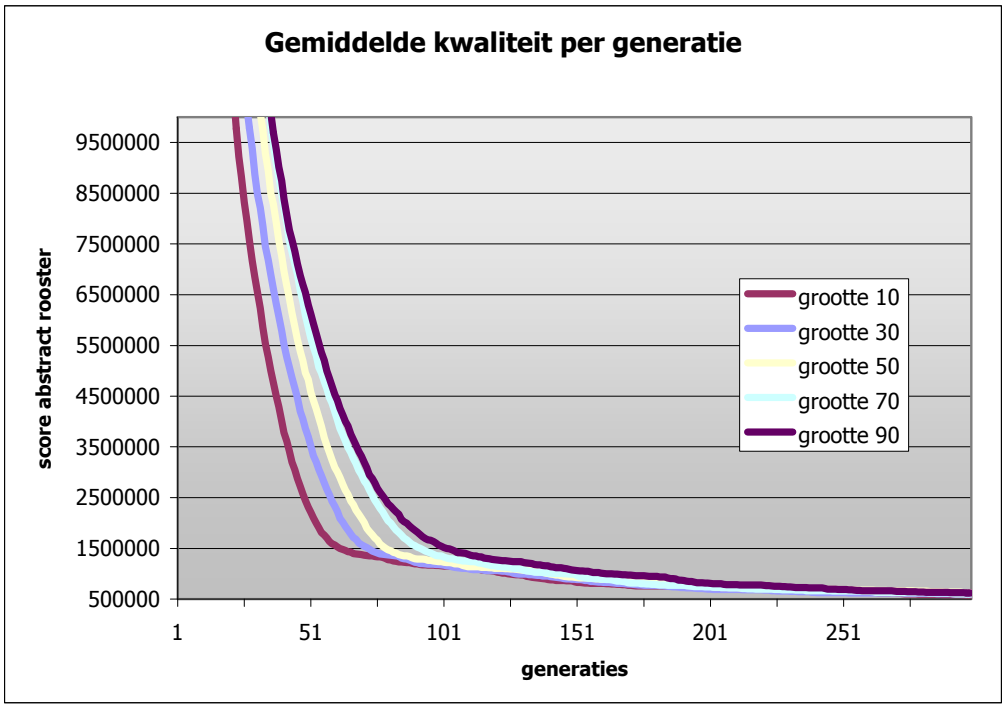
Figuur 19: Test van kwaliteit bij verschillende sampling space groottes

Als we kijken naar de belasting zien we dat hoe kleiner de sampling space hoe lager de belasting van het rooster, met uitzondering van grootte 10. Als we de standaarddeviatie in overweging nemen, die rond de 2000 ligt, kunnen we concluderen dat er sprake van een trend. Op basis daarvan is de sampling space grootte 20 meest wenselijk.

De trend kan verklaard worden doordat een beperkte sampling space een concurrentiedruk oplegt. Naarmate de sampling space groter is, hoeven individuen minder goed te zijn om te kunnen overleven. Door de sampling space te verkleinen zal een individu van steeds betere kwaliteit moeten

zijn en zal de concurrentiedruk toenemen. Gevolg is dat het algoritme in minder generaties zal convergeren naar een optimum.

Als we kijken naar de mate van convergentie wordt bovenstaande verklaring bevestigd.



Figuur 20: Gemiddelde kwaliteit per generatie voor 20 runs

Uit de grafiek[2] kunnen we aflezen dat hoe kleiner onze sampling space is hoe sneller de oplossing naar een goed rooster convergeert. Hoewel het in de grafiek niet is af te lezen hebben ook na 300 generaties de kleinere sampling spaces nog een betere kwaliteit. Per generatie loopt het verschil in kwaliteit tussen de verschillende sampling space groottes wel terug.

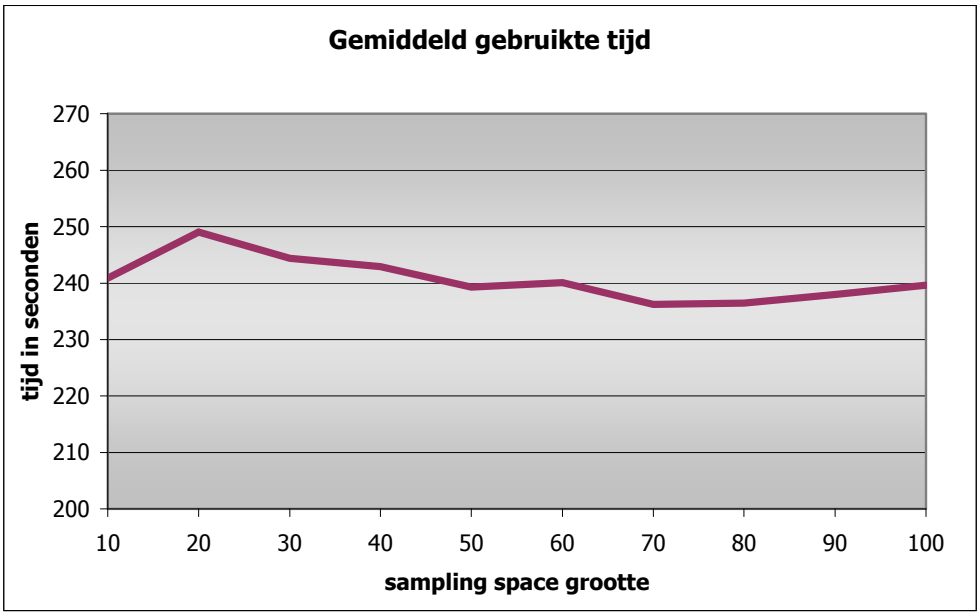
Voor het aantal nog in te plannen diensten zien we in Figuur 19 alleen een uitschieter bij grootte 20. Voor de overige waarden ligt het gemiddelde aantal nog in te plannen diensten dicht bij elkaar. Om te zien of er sprake kan zijn van een trend bekijken we de standaarddeviatie.

Sampling space grootte	10	20	30	40	50	60	70	80	90	100
Gemiddeld aantal niet ingeplande diensten	5.7	6.5	5.9	5.8	6.1	5.9	5.8	6.1	6.0	6.4
Standaarddeviatie	1.4	1.3	1.1	1.0	1.2	1.1	1.0	1.6	1.8	1.5

Als we kijken naar de standaarddeviatie is het moeilijk te zeggen of er bij het aantal niet ingeplande diensten sprake van een trend is. Het lijkt erop alsof de grootte van de sampling space geen invloed heeft op het aantal nog in te plannen diensten.

[2] Voor details over score abstract rooster zie paragraaf 6.1: Onderdeel 'Fitness function'

Tijd



Figuur 21: Gemiddelde gebruikte tijd voor het uitvoeren van 300 generaties

Als we kijken naar de tijd die nodig is voor het uitvoeren van 300 generaties bij de verschillende sampling space groottes zien we dat er nagenoeg geen verschil is tussen de testwaarden. Het lijkt alsof de grotere waarden een klein voordeel hebben ten opzichte van de kleinere waarden. Als we echter rekening houden met de standaarddeviatie van 8 seconden lijkt het verschil niet significant. De tijd voor het aanmaken van de individuen in het begin en het elke generatie sorteren van de sampling space vormt blijkbaar maar een klein aandeel in de totaal benodigde tijd. Op dit terrein kan daarom geen belangrijke tijdswinst worden geboekt.

Conclusie

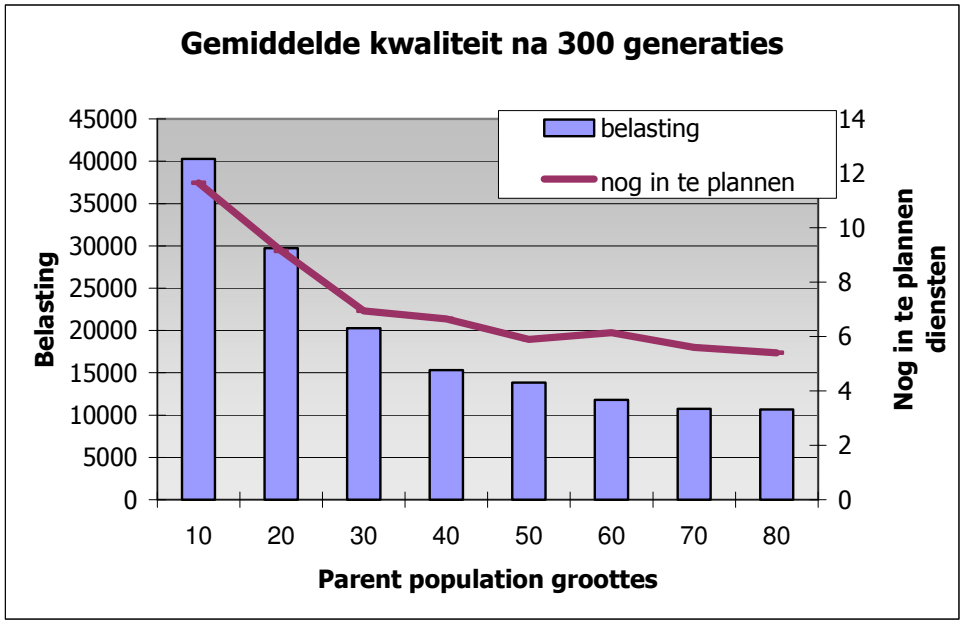
We kunnen concluderen dat een lage waarde voor de sampling space grootte kwaliteitsvoordeel oplevert. Dit komt omdat een kleinere sampling space een concurrentiedruk oplegt die ervoor zorgt dat het algoritme in minder generaties naar een goede oplossing convergeert. Hoewel het hogere aantal nog in te plannen diensten bij populatiegrootte 20 niet significant groter lijkt te zijn, kan in dit geval het beste gekozen worden voor een populatiegrootte van 30. Wat betreft de benodigde tijd is er geen verschil tussen de sampling space groottes.

9.2 Parent population grootte

Voor de test van de parent populatie grootte is gekozen voor de waarden 10, 20 t/m 80. Tijdens de implementatie bleek dat beste resultaten behaald konden worden in dit bereik. Daarom is voor deze waarden gekozen. Tijdens het uitvoeren van de test zijn de overige parameters ingesteld op de volgende waarden: sampling space grootte van 60, 300 generaties en een tournament grootte van 4. De sampling space is ingesteld op deze waarde omdat de eerste analyse uitwees dat dit de beste waarde was. Bij een nadere analyse bleek de beste waarde echter, zoals u hierboven kunt lezen, 30 te zijn. Voor elk van deze waarden zijn 20 runs van 300 generaties uitgevoerd.

Kwaliteit

Als we kijken naar de gemiddelde kwaliteit aan het einde van elke run zien we het volgende:



Figuur 22: Kwaliteit na 300 generaties bij verschillende parent population groottes

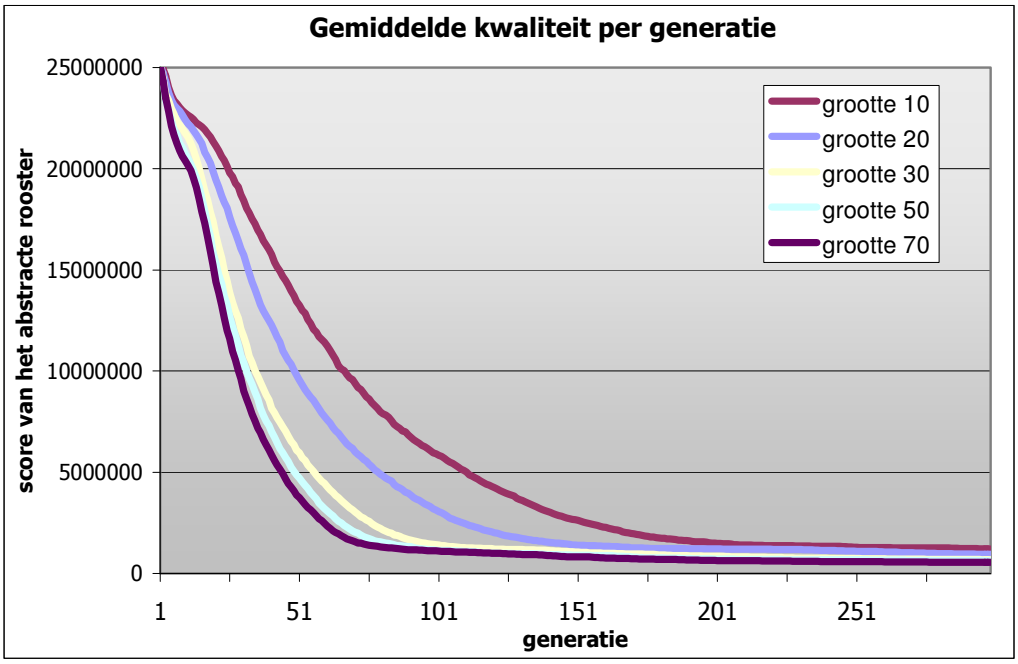
In bovenstaande figuur zien we dat de parent population invloed heeft op de kwaliteit. Voor kleinere waarden van de parent population zien we een hogere belasting én een hoger aantal nog in te plannen diensten dan bij grotere waarden van de parent population. Het verschil is met name voor de waarden 10 en 20 goed te zien. Na de 40 zwak het effect echter af en zorgt het vergroten van de population nauwelijks tot verbeteringen van de kwaliteit.

Dit effect kan verklaard worden doordat het verhogen van de parent populatie in deze implementatie leidt tot meer children. Die extra children leveren als de populatie grootte klein is, vaak goede resultaten. Naarmate er al meer children in de populatie zitten voegen de extra children steeds minder toe.

Uit deze grafiek kunnen we afleiden dat de waarde 80 de meest geschikte keuze is. Als we ook de standaarddeviatie bekijken zien we hetzelfde beeld. Het maken van zoveel mogelijk children is dus gunstig voor de kwaliteit.

Parent population size	10	20	30	40	50	60	70	80
Gemiddeld aantal niet ingeplande diensten	11.7	9.2	7.0	6.6	5.9	6.2	5.6	5.4
Standaarddeviatie	1.5	1.7	1.6	1.6	1.1	1.5	1.3	1

Kwaliteit per generatie



Figuur 23: Gemiddelde kwaliteit per generatie voor verschillende parent population groottes

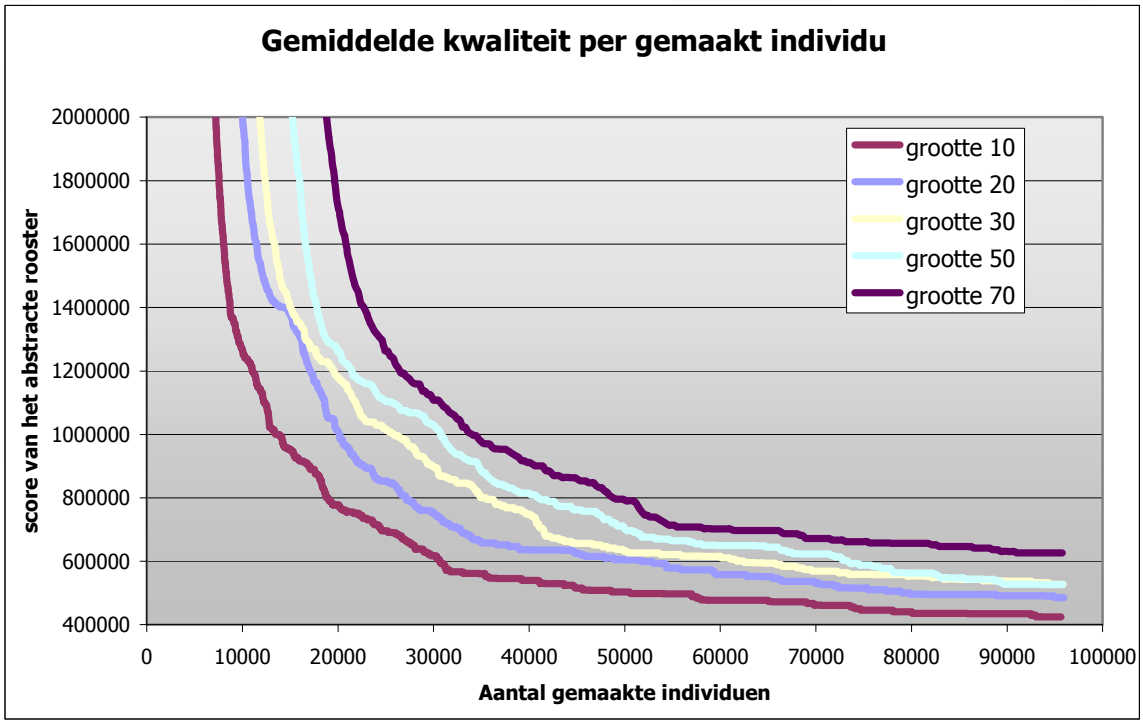
Als we kijken naar de kwaliteit per generatie zien we dat hoe groter onze parent population is hoe minder generaties nodig zijn om te convergeren naar een goede oplossing. Hiervoor geldt dezelfde verklaring. Naarmate we meer children maken hebben we elke generatie meer kans op het vinden van een beter rooster dan we al hebben gevonden.

Kwaliteit per generatie

Hierboven hebben we de parent population groottes vergeleken door het aantal generaties vast te stellen. Door het aantal generaties vast te stellen variëren we wel het aantal individuen dat wordt gemaakt. Een parent population grootte van 10 betekent dat elke generatie 40 children worden gemaakt terwijl bij een populatie grootte van 20 al 80 children worden gemaakt per generatie. Daarom stellen we hier het aantal individuen dat wordt gemaakt vast en kijken we naar de gemiddelde kwaliteit van het aantal gemaakte individuen.

Dit aantal is vastgesteld op 96000. Dit is gedaan zodat bij de parent population grootte 80 300 generaties kunnen worden uitgevoerd. Dit moet voldoende zijn om het genetisch algoritme te laten convergeren. Het aantal generaties dat nodig is voordat dit aantal individuen is gemaakt per verschillende parent population groottes staat in onderstaande tabel.

Parent population size	10	20	30	40	50	60	70	80
Aantal generaties	2400	1200	800	600	480	400	343	300



Figuur 24: kwaliteit per gemaakt individu

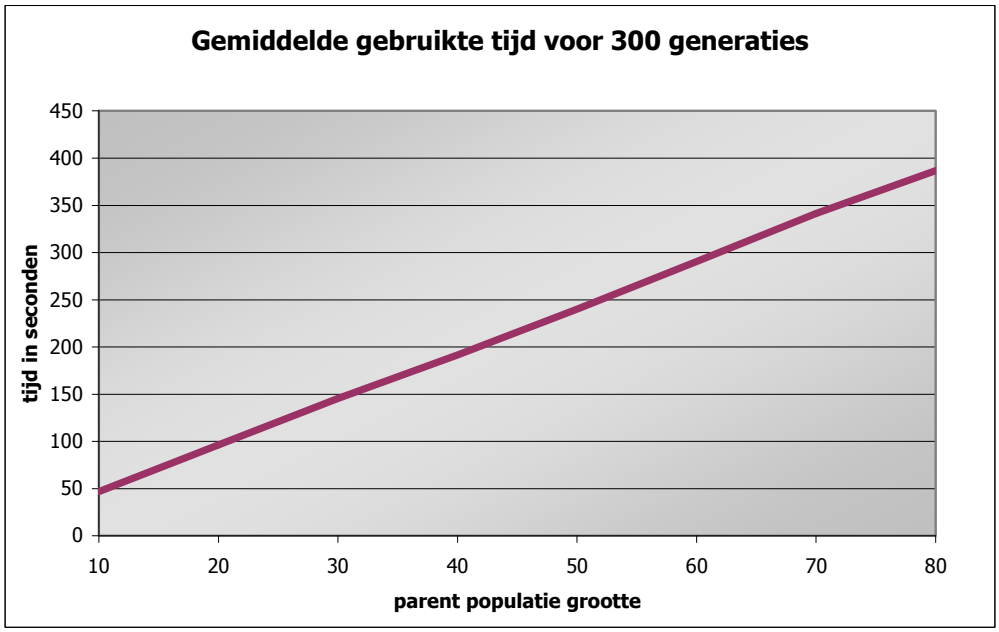
We zagen eerder al dat het maken van meer children de kwaliteit van de oplossing verhoogt. Dat effect zien we in deze grafiek terug. De gemiddelde score van het abstracte rooster neemt af naarmate we meer individuen creëren en daarmee neemt de kwaliteit van de oplossing toe.

In het figuur is ook te zien dat de oplossing bij een kleinere parent population grootte sneller convergeert. Dit is te verklaren omdat bij een kleinere population de children eerder zelf children zullen maken. Als 80 individuen zijn gemaakt dan zijn dat bij grootte 20 allemaal varianten van de beginpopulatie. Bij een parent population grootte van 10 zijn dan al 40 individuen gemaakt met behulp van de eerste 40 individuen.

Opvallend is echter dat na het maken van een groot aantal individuen er nog steeds een groot verschil is tussen de parent population groottes. Te verwachten was dat na het maken van een groot aantal individuen de oplossingen geconvergeerd moeten zijn en dus nagenoeg dezelfde kwaliteit zouden moeten hebben. Population grootte 10 laat zien dat zelfs na een zeer groot aantal generaties het algoritme nog steeds verbeteringen kan vinden. Dit geeft aan dat de mutatie- en cross-overoperatoren niet goed genoeg zijn om de juiste veranderingen door te voeren. De veranderingen die het teweegbrengt zijn elke generatie dusdanig klein dat veel generaties nodig zijn om de oplossing aanzienlijk te verbeteren.

Op basis van deze figuur lijkt een parent population van 10 het meest geschikt. Het aantal individuen dat gemaakt moet worden, en dus het aantal generaties dat het algoritme moet draaien, wordt in een volgende paragraaf besproken.

Tijd



Figuur 25: Gemiddelde benodigde tijd voor het maken uitvoeren van 300 generaties

Als we naar de grafiek kijken lijkt er een lineair verband te zijn tussen de grootte van de parent population en de benodigde rekentijd. Dit kan verklaard worden door het feit dat, wanneer we de parent population grootte verdubbelen, er dan ook twee keer zo veel children worden gemaakt. Elke child moet vervolgens op de regels en criteria worden gecontroleerd om een score te kunnen bepalen. Twee keer zoveel children betekent dus twee keer zo vaak de regels en criteria testen.

Opmerkelijk is wel dat het verdubbelen van de grootte ook nagenoeg een verdubbeling van de tijd oplevert. Dit zou betekenen dat bijna alle vereiste rekentijd zit in het controleren van regels en criteria. Bij het oorspronkelijke genetisch algoritme hadden we een dergelijk verband ook al opgemerkt. We zien hier duidelijk dat tijd kan worden bespaard door minder regels en criteria te moeten controleren.

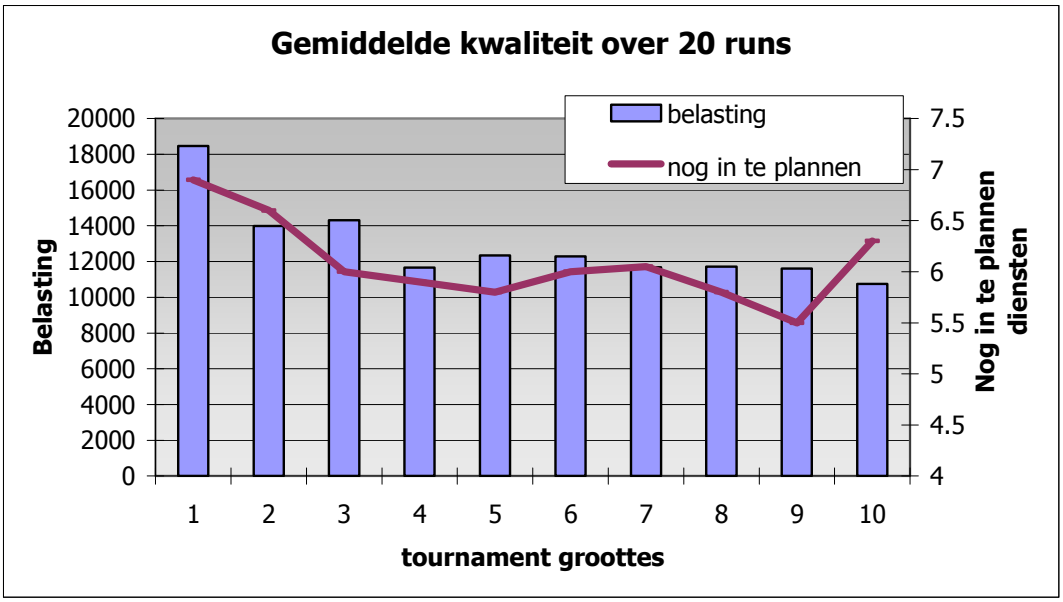
Conclusies

We hebben gezien dat de grootte van de parent population invloed heeft op zowel de kwaliteit als de tijd voor het uitvoeren van het algoritme. Een kleinere parent population convergeert sneller naar een goede oplossing dan grotere parent population groottes. Daarom is een parent population grootte van 10 het meest geschikt.

9.3 Tournament grootte

De test voor de tournament grootte omvatte 20 runs voor de waarden 1, 2 t/m 10. In ([2], Van der Put, 2005) is onderzoek gedaan voor de tournament groottes 2 en 6 voor de parent selection. Daarom is gekozen te kijken naar waarden in de buurt van deze waarden.

Kwaliteit



Figuur 26: Gemiddelde kwaliteit bij verschillende tournament groottes

Als we kijken naar de belasting lijkt het erop alsof een grotere tournament size beter is. Als we de standaarddeviatie meenemen zien we het volgende.

Tournament grootte	1	2	3	4	5	6	7	8	9	10
Gemiddelde belasting	18472	13998	14306	11663	12352	12296	11681	11711	11615	10750
Standaarddeviatie	3371	2676	4268	2590	3559	3584	3134	3798	2262	2511

Naast de gemiddelde belasting neemt ook de standaarddeviatie af bij een toenemende tournament grootte. Als we kijken naar de belasting bij grootte 10 en 1 en de standaarddeviatie meenemen, lijkt het meer dan toeval.

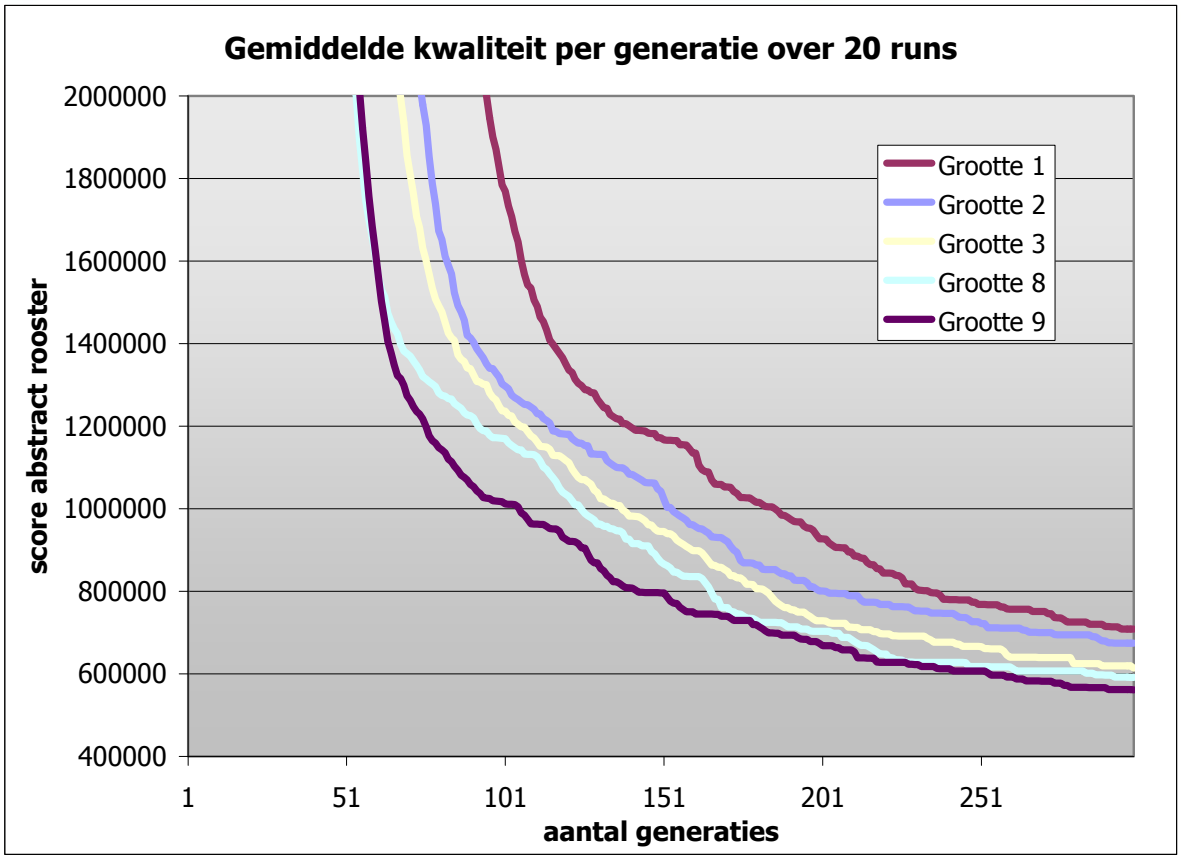
Een verklaring ligt in de concurrentie die de tournament grootte oplegt. Hoe groter het toernooi hoe kleiner de kans is voor de slechtere individuen zich voort te planten. Naarmate het toernooi groter wordt zullen de betere individuen meer children produceren. Dit resulteert in ons geval in een verbetering van de belasting.

Kijken we naar het aantal niet ingeplande diensten dan zien we een soortgelijke trend. Als we nu de standaarddeviatie bekijken zien we dat in dit geval het verband minder duidelijk is. De standaarddeviatie is relatief groot ten opzichte van het gemiddelde.

Tournament grootte	1	2	3	4	5	6	7	8	9	10
Gemiddelde belasting	6.9	6.6	6	5.9	5.8	6	6.05	5.8	5.5	6.3
Standaarddeviatie	1.8	1.3	1.4	1.1	1.5	1.5	1.4	1.1	1.2	1.4

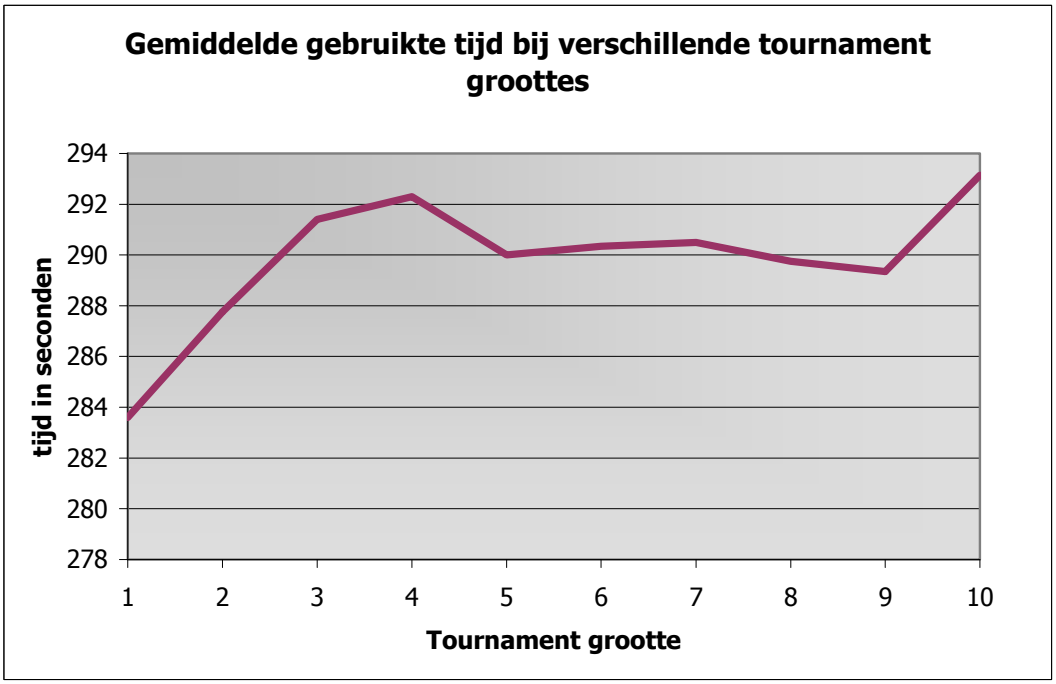
Kwaliteit per generatie

In onderstaand figuur zien we dat het verhogen van de concurrentie ervoor zorgt dat het algoritme sneller convergeert. Hoewel dit niet altijd beter hoeft te zijn, convergeert in dit geval het algoritme niet alleen sneller, maar vindt het ook betere oplossingen.



Figuur 27: Kwaliteit per generatie bij verschillende tournament groottes

Tijd



Figuur 28: Gebruikte tijd voor het uitvoeren van 300 generaties bij verschillende tournament groottes

Uit de grafiek ontstaat de indruk dat de kleinere waarden voor de tournament grootte minder tijd nodig hebben dan de grotere waarden. Kijken we echter ook naar de standaarddeviatie lijkt het verschil meer toeval te zijn.

Tournament grootte	1	2	3	4	5	6	7	8	9	10
Gemiddelde belasting	283.6	287.8	291.4	292.3	290.0	290.4	290.5	289.7	289.4	293.2
Standaarddeviatie	5.5	5.6	7.8	6.7	6.6	6.6	4.6	6.8	5.3	6.7

Conclusie

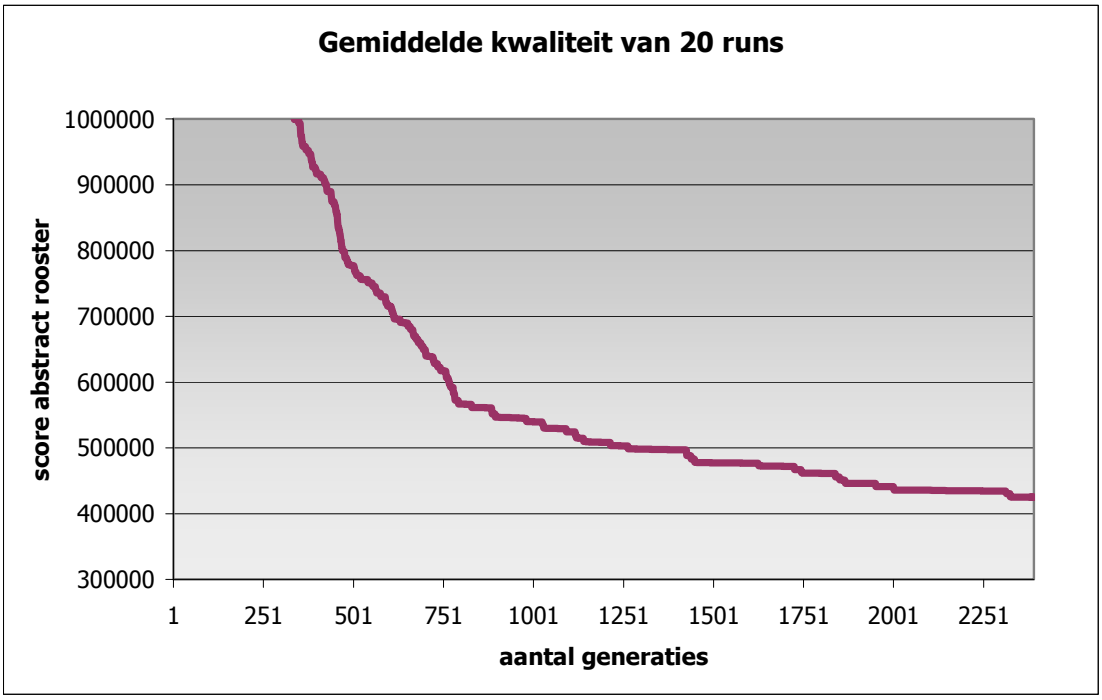
Gezien de belasting en de mate van convergentie heeft de tournament grootte invloed op de kwaliteit en de snelheid waarmee een goede oplossing wordt gevonden. Een groter tournament is in dit geval voordelig. Wat betreft de tijd die nodig is voor het uitvoeren van 300 generaties is er geen duidelijk verschil tussen de verschillende groottes. Daarom wordt gekozen voor een tournament grootte van 9.

Door een gebrek aan tijd is niet onderzocht of de kwaliteit blijft verbeteren voor grotere waarden dan 10. In ([2], Van der Put, 2005) werden de waarden 2 en 6 bekeken. Ook daar bleek de grootste tournament grootte het beste te zijn. Onduidelijk is of daar ook gekeken is naar andere waarden behalve 2 en 6.

9.4 Aantal generaties

De test voor het aantal generaties omvatte 20 runs voor de parent population grootte 10. Deze waarde is gekozen naar aanleiding van paragraaf 9.2: 'Parent population grootte' waarin we zagen dat de parent population grootte 10 de beste keuze is. De sampling space is opnieuw ingesteld op 60 en de tournament grootte op 4.

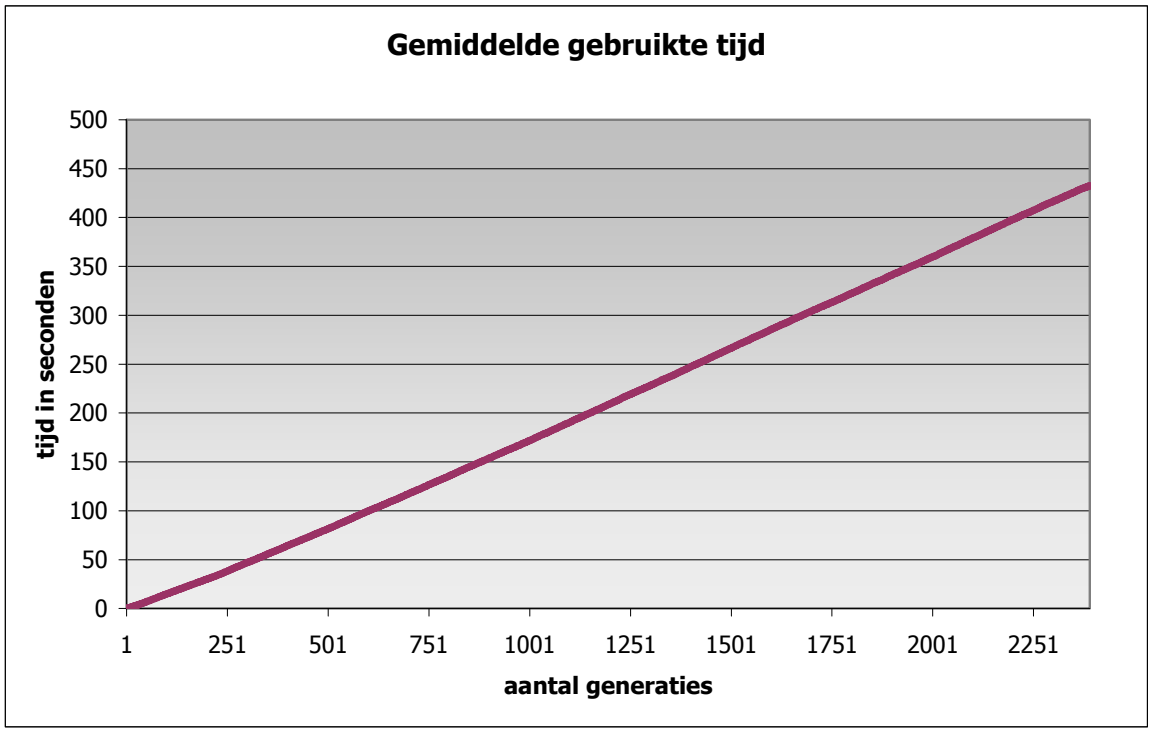
Kwaliteit



Figuur 29: Kwaliteit per generatie

In de grafiek zien we dat zelfs na 2250 generaties het algoritme bij een parent population grootte van 10 nog niet is geconvergeerd. Reden hiervoor zijn de mutatie- en cross-overoperatoren die te weinig verandering leveren. Na 1250 generaties zien we echter dat de kwaliteit steeds minder toeneemt.

Tijd



Figuur 30: Gebruikte tijd per aantal generaties

Zoals we al eerder hadden gezien is er een lineair verband tussen het aantal generaties en de benodigde tijd in seconden.

Conclusie

Gelet op de kwaliteit lijkt het uitvoeren van 1250 generaties een goede keuze. Na het uitvoeren van dit aantal generaties neemt de kwaliteit die wordt gewonnen door het uitvoeren van meer generaties steeds verder af.

9.5 Samenvatting

Uit bovenstaande tests kunnen we afleiden dat de volgende instellingen het beste zijn:

- Sampling space grootte van 30
- Parent population grootte van 10
- Tournament grootte van 9
- 1250 generaties

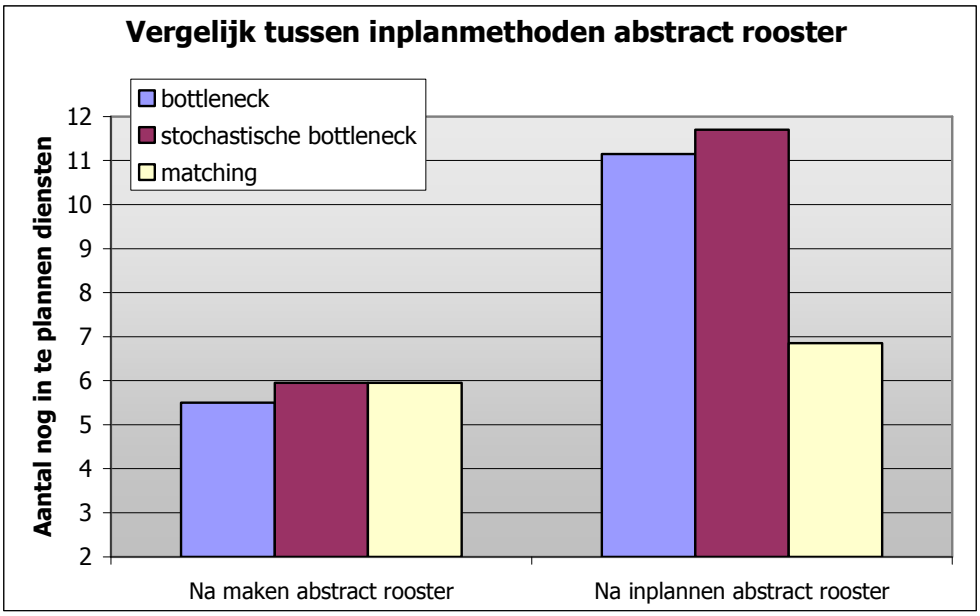
Omdat deze waarden echter in nadere analyse tot stand kwamen is in de hoofdstukken 10 en 11 gebruikt gemaakt van de instellingen: Sampling space grootte van 30, Parent population grootte van 40, Tournament grootte van 9 en 300 generaties. Gevolg hiervan is dat de tijd die gebruikt is voor het uitvoeren van het abstracte rooster hoger ligt dan nodig was met de beste parameters. In tijd zal dit ongeveer 50 seconden schelen.

10. Abstract rooster inplannen testresultaten

In dit hoofdstuk vergelijken we drie methoden waarmee het abstracte rooster kan worden ingepland. Daarbij wordt gekeken naar de in hoofdstuk 8: 'Test opzet' besproken zaken. Bij het uitvoeren van deze tests zijn telkens 20 abstracte roosters gemaakt voor elke inplanmethode. Deze roosters zijn vervolgens door de drie methoden bottleneck, stochastische bottleneck en matching ingepland. Voor het maken van het abstracte rooster zijn de instellingen gebruikt die het best bleken in hoofdstuk 9.

Kwaliteit

Als we kijken naar het verschil tussen het aantal diensten ingepland in het abstracte rooster en het rooster na het inplannen, dan krijgen we het volgende figuur.



Figuur 31: Aantal nog in te plannen diensten bij verschillende inplanmethoden

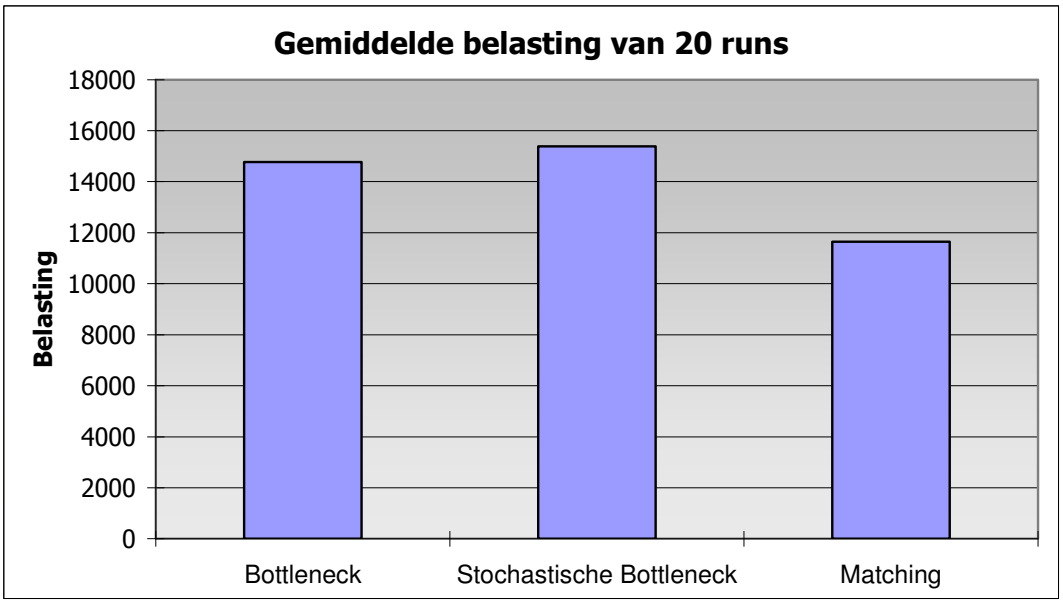
Uit deze grafiek kunnen we aflezen dat er weinig verschil is tussen de bottleneck methode en de stochastische bottleneck methode. Opvallend is ook dat beide methoden moeite hebben om de diensten van het abstracte rooster in het echte rooster in te plannen. Beiden houden gemiddeld 6 diensten meer over boven op het aantal diensten dat al overblijft. De matching methode lijkt het beduidend beter te doen. Het verschil tussen het abstracte rooster en het rooster na het inplannen bedraagt gemiddeld maar 1 dienst bij deze methode.

Om te toetsen of de matching methode daadwerkelijk beter werkt dan de andere methode passen we de Wilcoxon toets toe. We kijken of de verdelingen van het verschil van de matching methode een kleinere mediaan heeft dan de verdelingen van de verschillen van de andere methoden. De nulhypothese is dat de mediaan groter of gelijk is aan die van de bottleneck of stochastische bottleneck methode. De alternatieve hypothese is dat de mediaan van de verdeling van matching kleiner is.

Na het uitvoeren van de test krijgen we de volgende p-waarden:

Wilcoxon test	Bottleneck ≤ Stochastische bottleneck	Bottleneck ≤ Matching	Stochastische bottleneck ≤ Matching
p-waarde	0.65	0.00	0.00

De uitkomsten van de Wilcoxon test bevestigen het vermoeden dat we hadden. Vergelijken we de verdeling van matching met beide andere methoden dan mogen we bij de toetsen de nulhypothese verwerpen. Dat wil zeggen dat we mogen aannemen dat de mediaan van de verdeling van de matching methode kleiner is dan de andere methoden. Met het inplannen van de matching methode hebben we dus minder kwaliteitsverlies wat betreft het aantal nog in te plannen diensten.



Figuur 32: Gemiddelde belasting bij de drie verschillende methoden

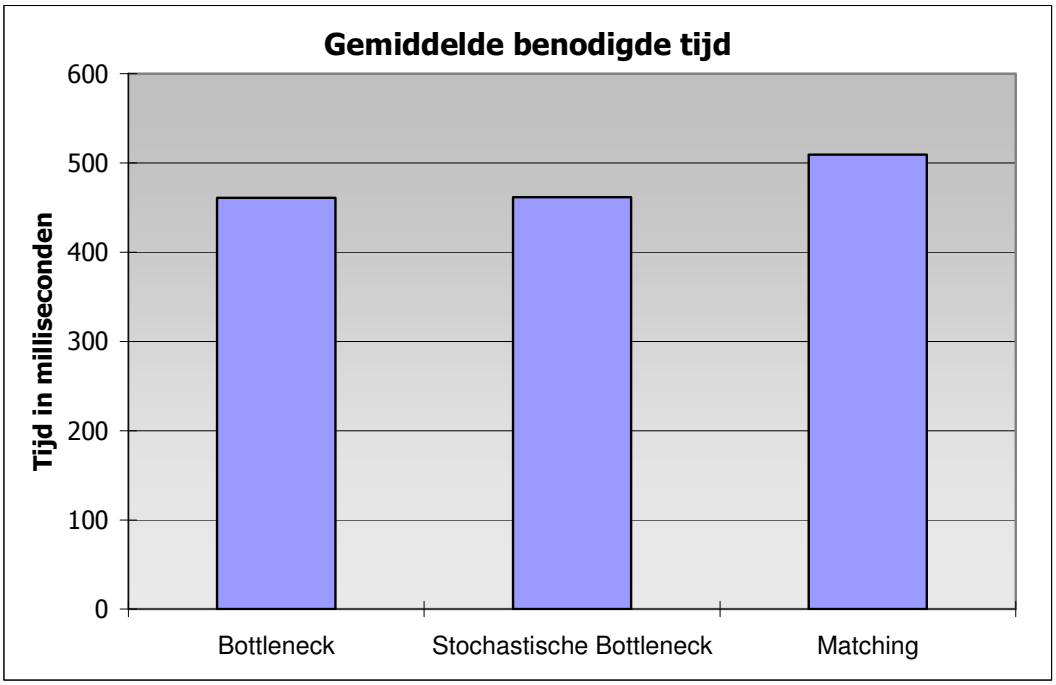
Als we kijken naar de gemiddelde belasting lijken opnieuw de bottleneck en stochastische bottleneck methode weinig voor elkaar onder te doen. De matching methode laat ook hier een beter resultaat zien.

Na het uitvoeren van de test krijgen we de volgende p-waarden:

Wilcoxon test	Bottleneck ≤ Stochastische bottleneck	Bottleneck ≤ Matching	Stochastische bottleneck ≤ Matching
p-waarde	0.69	0.00	0.00

Ook voor de belasting geldt dat we de nulhypothese mogen verwerpen voor de vergelijking Bottleneck – Matching en Stochastische bottleneck – Matching. Dat wil zeggen dat we mogen aannemen dat het rooster dat ontstaat na het inplannen van een abstract rooster met de matching methode een minder hoge belasting heeft dan wanneer dit met de bottleneck of stochastische bottleneck methode gebeurt.

Tijd



Figuur 33: Benodigde tijd voor het uitvoeren van 300 generaties

Voor de benodigde tijd zien we dat matching gemiddeld meer tijd nodig heeft voor het inplannen van het abstracte rooster. We zien dat deze methode gemiddeld 11% meer tijd kost dan de andere twee methoden.

Het maken van het abstracte rooster bij deze test nam gemiddeld 299 seconden in beslag. Het inplannen van het abstracte rooster kostte gemiddeld 510 milliseconden. Dit betekent dat het inplannen op het gehele proces nog geen 0.2% bedraagt. Hoewel de tijd voor de matching methode langer is dan met de andere methoden, weegt dit verschil op het geheel nauwelijks mee.

Conclusie

We hebben gezien dat de matching methode het best gebruikt kan worden voor het inplannen van het abstracte rooster. Van de drie methoden is de kwaliteit die we verliezen wat betreft het aantal nog in te plannen diensten en de belasting het kleinst. Dit is niet verwonderlijk aangezien deze methode het rooster dag voor dag optimaal plant. Aangezien het abstracte rooster zich bezighoudt met de globale eigenschappen kan de matching methode zo een rooster goed invullen door vooral te letten op de locale eigenschappen.

De tijd die gebruikt wordt door de matching methode is groter dan bij de andere methoden. Dit verschil is echter ondergeschikt aan de betere kwaliteit die het oplevert. Bij het inplannen van het abstracte rooster wordt voortaan de matching methode gebruikt.

11. Beginroosters maken testresultaten

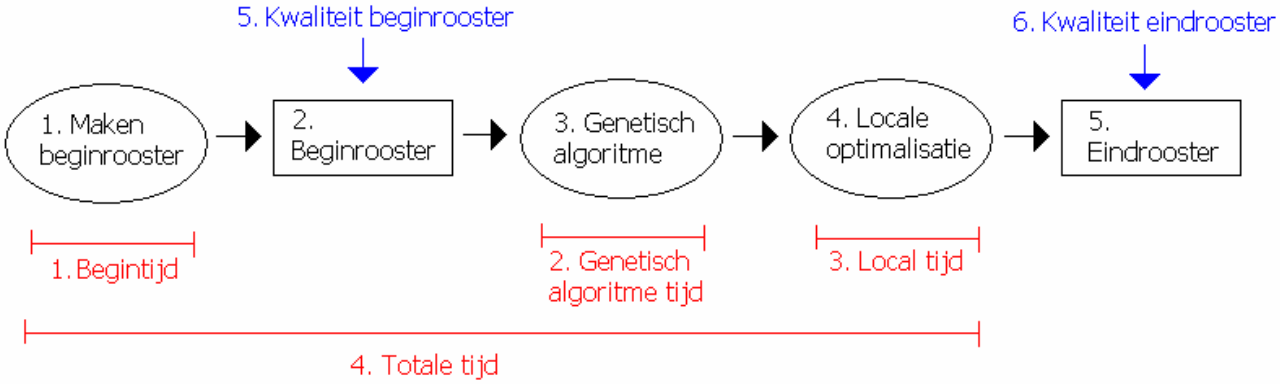
Dit hoofdstuk bespreekt de resultaten van de tests die zijn uitgevoerd voor het vergelijken van de methoden voor het maken van beginroosters. De methoden, zoals in hoofdstuk 5: 'Nieuwe methoden' beschreven, zijn:

- Bottleneck methode
- Stochastische bottleneck methode
- Bottleneck methode met locale optimalisatie
- Abstract rooster
- Volledig willekeurige roosters

In hoofdstuk 8: 'Test opzet' bespraken we dat we de methoden op een aantal punten met elkaar kunnen vergelijken.

1. De tijd die nodig is om een beginrooster te maken.
2. De tijd die gebruikt is voor het uitvoeren van het genetisch algoritme, gebruik makende van een beginrooster.
3. De tijd die nodig is voor het uitvoeren van locale optimalisatie.
4. De totale tijd nodig voor het maken van het uiteindelijke rooster.
5. De kwaliteit van het gemaakte beginrooster.
6. De kwaliteit van het rooster na het uitvoeren van het genetisch algoritme en locale optimalisatie.

Schematisch gezegd kijken we naar de volgende onderdelen:

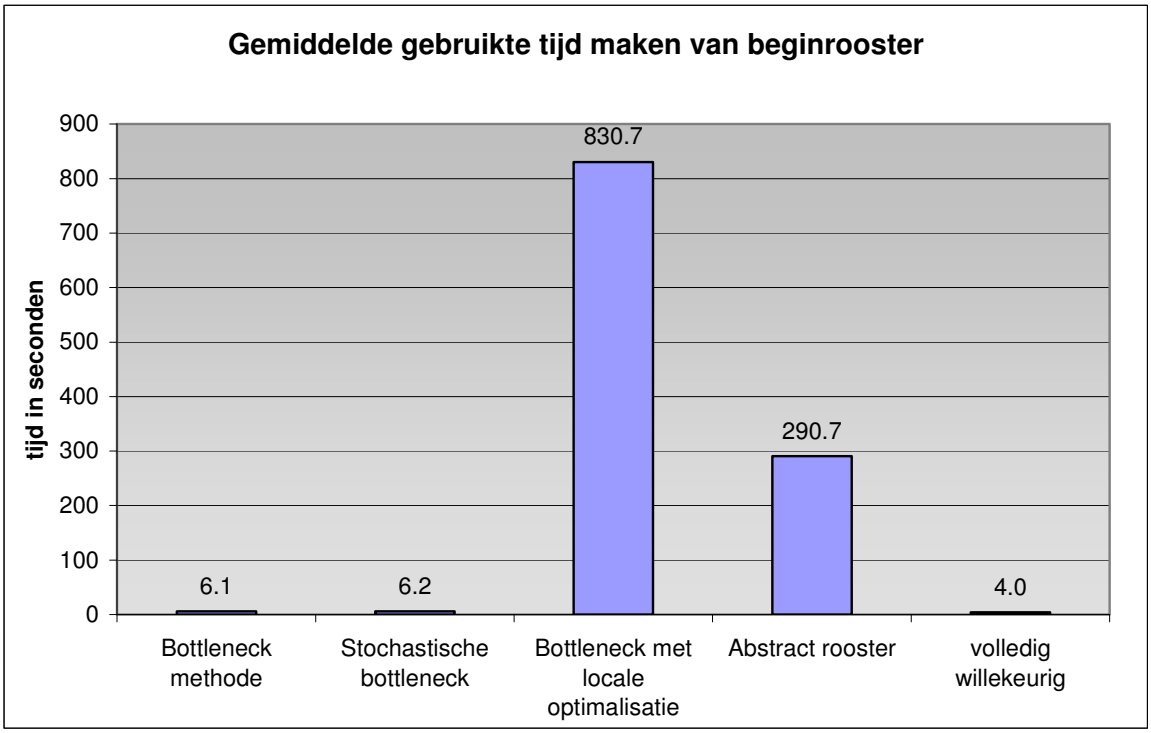


Figuur 34: De onderdelen voor het maken van een rooster

De hieronder volgende paragrafen geven een beschrijving van de resultaten gevonden bij het uitvoeren van de tests. Bij het uitvoeren van de tests zijn telkens vijf beginroosters gemaakt waar het genetisch algoritme mee start.

11.1 Tijd beginrooster

Als we van elke methode de gemiddelde benodigde tijd voor het maken van het beginrooster tegen elkaar zetten zien we het volgende figuur:



Figuur 35: gemiddelde gebruikte tijd voor het maken van een beginrooster per methode

In het figuur is op te maken dat de bottleneck met locale optimalisatie en het abstracte rooster beduidend meer tijd gebruiken dan de overige drie methoden. Op basis van Figuur 35 is de ordening tussen de methoden van snel naar langzaam:

1. Volledig willekeurig
2. Bottleneck methode / Stochastische bottleneck methode
3. Abstract rooster
4. Bottleneck methode met locale optimalisatie

Daarbij moeten we opmerken dat als maar één individueel rooster moet worden gemaakt de abstracte rooster methode het langste duurt. Bij de abstracte rooster methode duurt het maken van één rooster net zo lang als het maken van vijf roosters. Bij de andere methode zal voor elk rooster de methode opnieuw moeten worden aangeroepen. De tijd die nodig is voor het maken van vijf roosters is dan ook vijf keer zo groot. Hoewel de bottleneck methode met locale optimalisatie voor vijf roosters dus meer tijd nodig heeft zal deze methode voor het maken van één rooster minder tijd in beslag nemen.

Het verschil in tijd tussen de abstracte rooster methode en de bottleneck methode is echter geflatteerd. Door een tweetal zaken is de hoeveelheid tijd die het abstracte rooster nodig heeft om een beginrooster te maken groter dan in theorie mogelijk moet zijn. In de eerste plaats is de code van het abstracte rooster niet geoptimaliseerd wat voor de bottleneck methode wel is gebeurd. Het optimaliseren van de code kan het algoritme aanzienlijk versnellen.

Daarnaast heeft het genetisch algoritme teveel generaties nodig om voor het maken van het abstracte rooster. In paragraaf 9.4: 'Aantal generaties' zagen we dat ruim 300 generaties nodig zijn voordat het abstracte genetisch algoritme convergeert naar een oplossing. Het oorspronkelijke genetisch algoritme gebruikt ter vergelijking na de bottleneck methode gemiddeld 40 generaties. Dit

geeft aan dat een betere keuze van mutatie- en cross-overoperatoren het aantal generaties en daarmee de benodigde tijd verder kunnen terugdringen.

Door deze zaken moet het mogelijk zijn de tijd die gebruikt wordt voor het maken van het abstracte rooster drastisch te verkleinen. Het maken van het abstracte rooster zal echter wel meer tijd blijven kosten dan de bottleneck, ook als de methode geoptimaliseerd is.

Dat de bottleneck methode met locale optimalisatie voor het maken van vijf roosters de meeste tijd in beslag neemt is niet verwonderlijk. Op elk van de beginroosters wordt 1-opt en 2-opt toegepast, wat veel tijd kost. Door het uitvoeren van de optimalisatie moet echter tijd kunnen worden teruggewonnen bij het uitvoeren van het genetisch algoritme en de daaropvolgende locale optimalisatie. Beide methoden krijgen te maken met geoptimaliseerde roosters en daardoor zal er weinig aan te hoeven veranderen.

Conclusie

Op basis van de benodigde tijd voor het maken van het beginrooster kunnen we concluderen dat de volledig willekeurige methode het snelst is gevolgd door de bottleneck methode en de stochastische bottleneck methode.

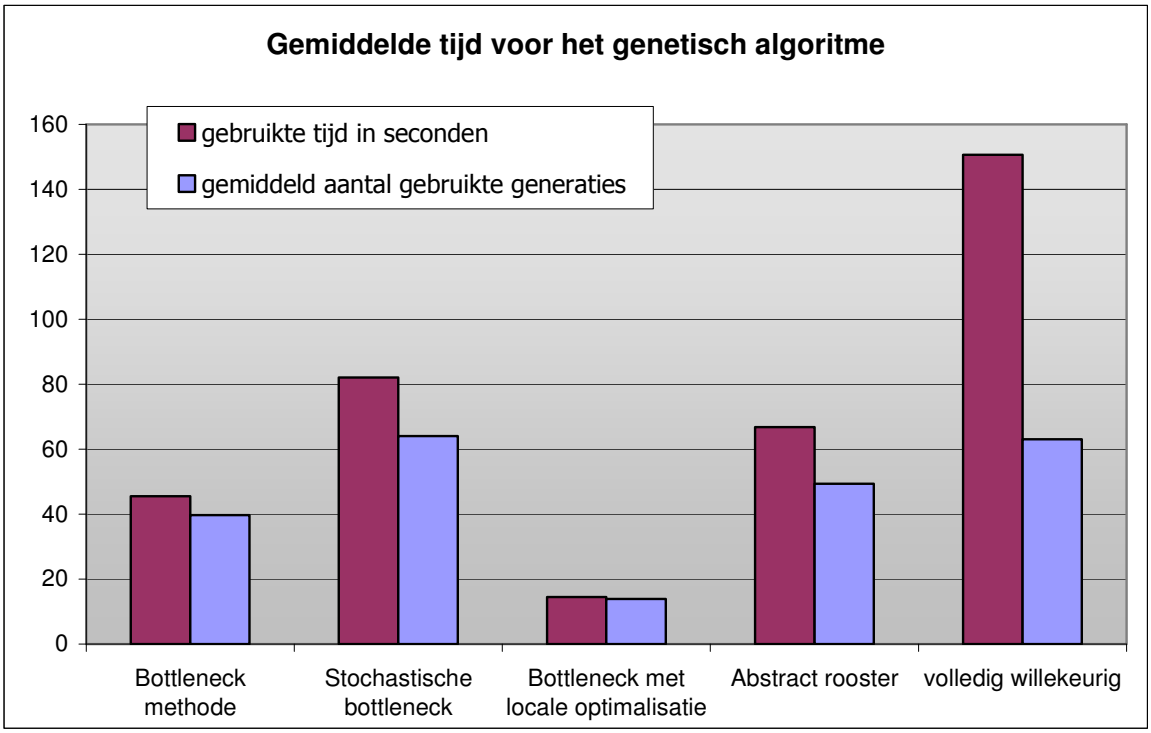
Het maken van een beginrooster door de abstracte rooster methode verliest tijd ten opzichte van de andere methode door twee dingen:

1. Niet geoptimaliseerde code.
2. Het aantal benodigde generaties van het abstracte genetisch algoritme is veel hoger dan bij het oorspronkelijke genetisch algoritme. De keuze van mutatie- en cross-overoperatoren kan hiervan de oorzaak zijn.

Ook de bottleneck methode met locale optimalisatie verliest veel tijd ten opzichte van de overige methoden. Een deel van de tijd moet echter bij het uitvoeren van het genetisch algoritme en de daaropvolgende locale optimalisatie teruggewonnen kunnen worden.

11.2 Tijd tussenrooster

De gemiddelde tijd die de methoden nodig hadden tijdens de 100 runs voor het uitvoeren van het genetisch algoritme is in onderstaande histogram weergegeven.



Figuur 36: Gemiddelde tijd nodig voor het uitvoeren van het genetisch algoritme over 100 runs

Als we de methoden ordenen van snel naar langzaam krijgen we:

1. Bottleneck met locale optimalisatie
2. Bottleneck methode
3. Abstract rooster
4. Stochastische bottleneck methode
5. Volledig willekeurig

Zoals verwacht wint de bottleneck met locale optimalisatie bij het uitvoeren van het genetisch algoritme tijd terug ten opzichte van de andere methoden. De beginroosters gemaakt met deze methode zijn geoptimaliseerd waardoor het genetisch algoritme snel kan convergeren.

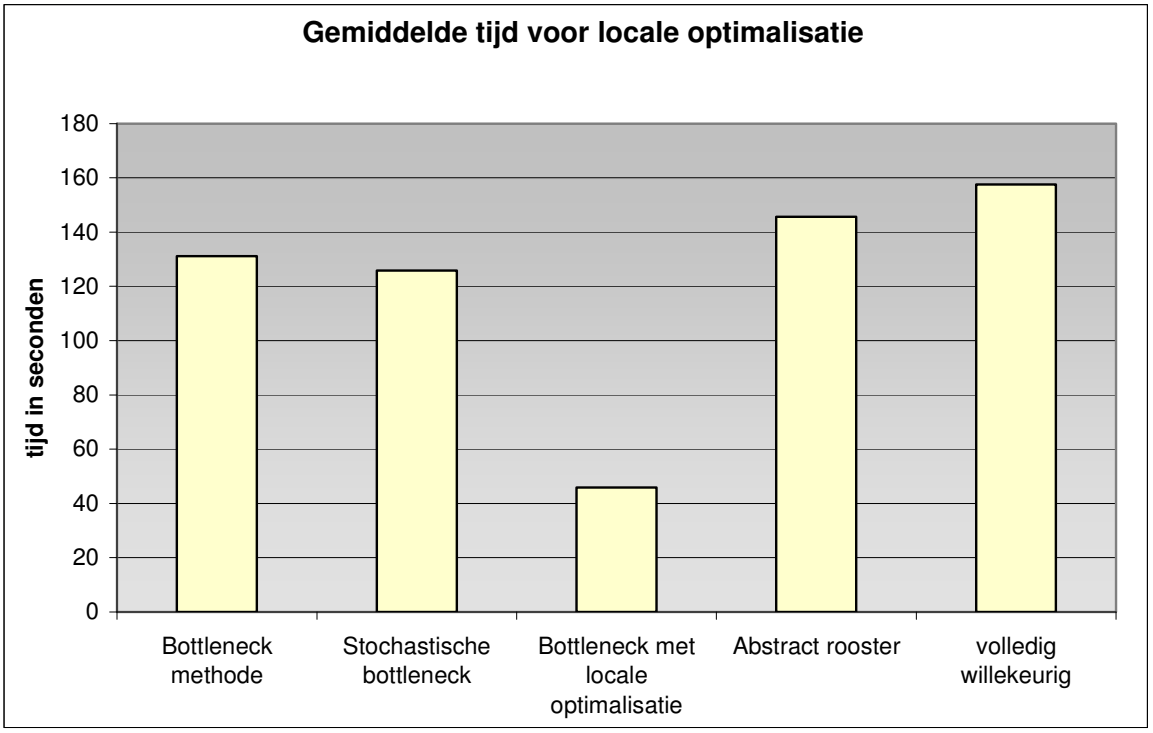
De stochastische methode en de volledig willekeurige methode verliezen tijd ten opzichte van de bottleneck methode. Omdat bij de bottleneck methode één beginrooster domineert convergeert het genetisch algoritme snel. De stochastische methode en de volledig willekeurige methode produceren meer van elkaar verschillende roosters waardoor het genetisch algoritme meer tijd nodig voor het doorzoeken van de oplossingsruimte. Het convergeren van het algoritme duurt dan ook langer. Door het doorzoeken van een groter gedeelte van de oplossingsruimte kan mogelijk een beter optimum worden gevonden, waar het algoritme naar convergeert.

Het uitvoeren van het genetisch algoritme bij de abstracte rooster methode duurt langer dan de bottleneck methode, maar korter dan de stochastische bottleneck en de volledig willekeurige methoden. Reden hiervoor is dat beginroosters in de huidige implementatie meer op elkaar lijken dan bij de andere twee methoden, waardoor het genetisch algoritme snel convergeert.

Conclusie

Kijkend naar de benodigde tijd voor het uitvoeren van het genetisch algoritme blijkt de bottleneck met locale optimalisatie de snelste methode te zijn. De stochastische bottleneck methode en de volledige willekeurige methode nemen de meeste tijd in beslag. De verschillen kunnen verklaard worden door de beginroosters. Hoe meer de beginroosters van elkaar verschillen, hoe meer tijd het genetisch algoritme kwijt is om te convergeren.

11.3 Tijd eindrooster



Figuur 37: Gemiddelde tijd gebruikt voor locale optimalisatie

In deze histogram is zien dat de tijd die nodig is voor locale optimalisatie voor de bottleneck methode met locale optimalisatie is beduidend lager dan de overige methoden. Zoals we in paragraaf 11.1: 'Tijd' hadden besproken is de structuur van de beginroosters met deze methode dusdanig dat zowel het genetisch algoritme als de locale optimalisatie op het eind maar weinig hoeven te veranderen aan het rooster.

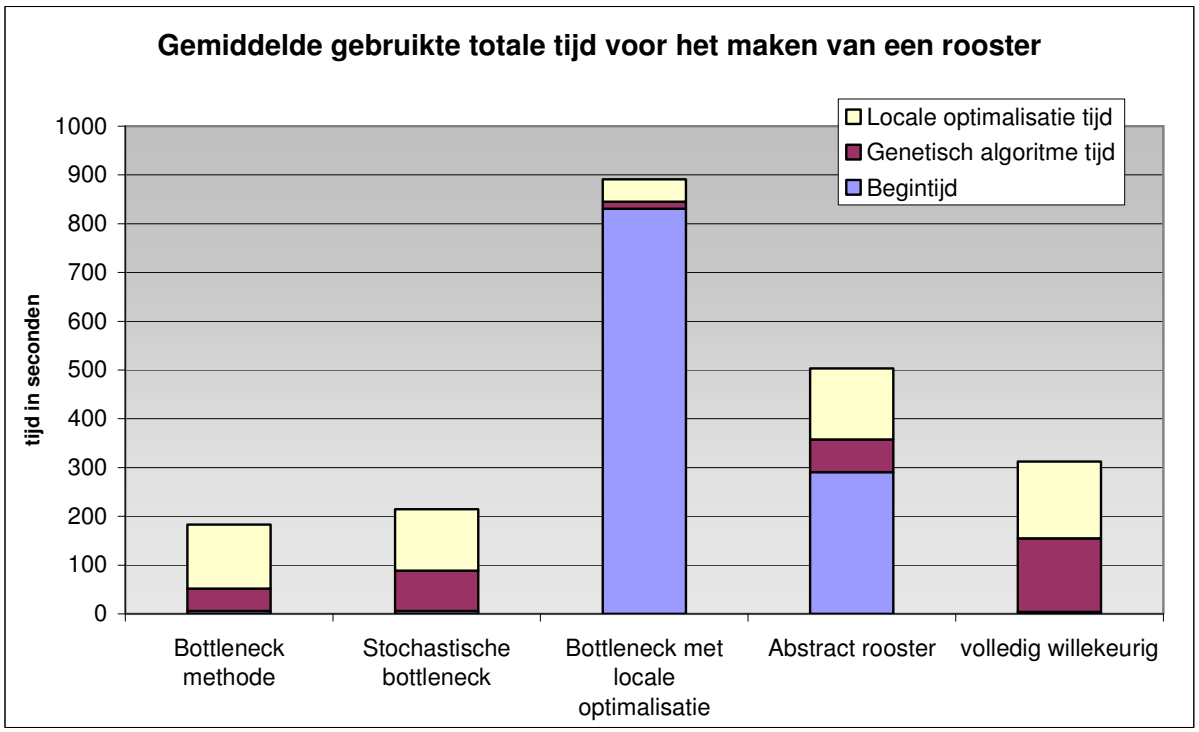
De abstracte rooster methode en de volledig willekeurige methode gebruiken beiden meer tijd dan de bottleneck methode en de stochastische bottleneck methode. Het lijkt erop alsof de locale optimalisatie meer verbeteringen kan aanbrengen bij deze methoden. Dit zou betekenen dat de kwaliteitsverbetering die we in paragraaf 11.7: 'Kwaliteit eindrooster' gaan bekijken groter moet zijn voor deze methoden dan bij de andere methoden.

Conclusie

De bottleneck methode met locale optimalisatie gebruikt van alle methoden de minste tijd voor locale optimalisatie. De bottleneck methode en stochastische bottleneck methode gebruiken beduidend meer tijd, maar zijn wel sneller dan de abstracte rooster methode en de volledig willekeurige methode.

11.4 Tijd totaal

Hierboven hebben we gekeken naar gemiddelde benodigde tijd per onderdeel. We hebben gezien dat voor bepaalde methoden het maken van het beginrooster langer duurde dan andere methoden. Daar stond tegenover dat bij het genetisch algoritme en locale optimalisatie weer tijdwinst werd geboekt. Als we kijken naar de totale tijd voor het maken van een rooster met het genetisch algoritme zien we het volgende:



Figuur 38: Gebruikte tijd voor het maken van een rooster

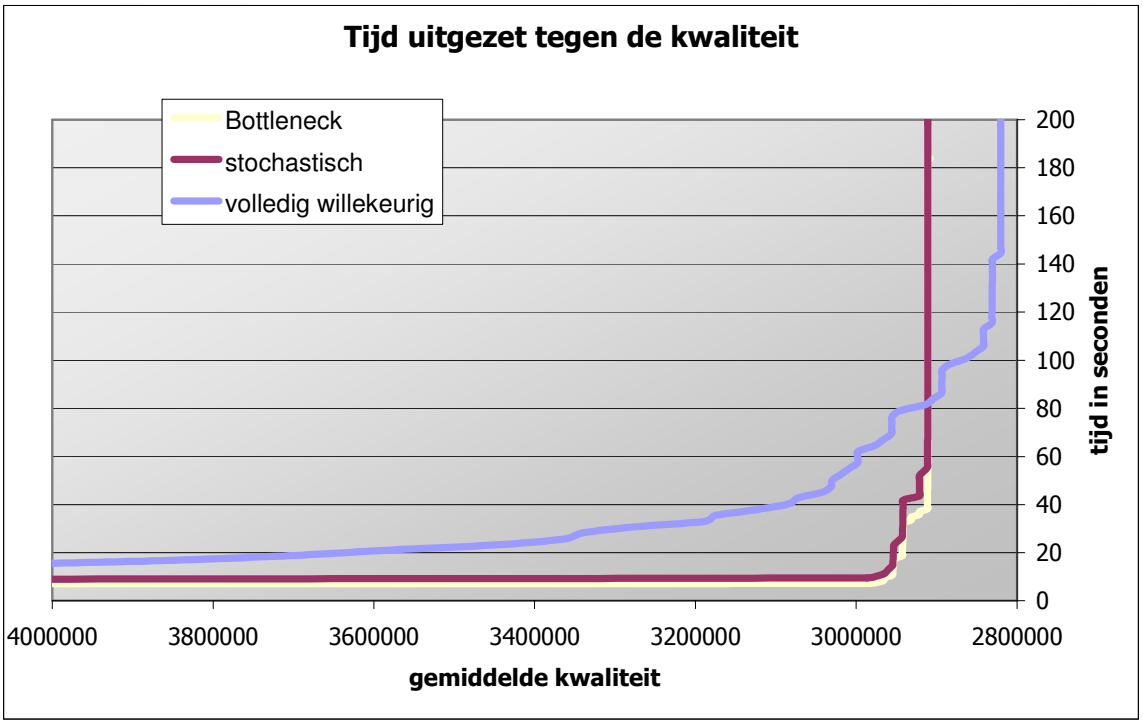
Als we een ordening maken tussen de methoden op basis van dit figuur krijgen we van snel naar langzaam:

1. Bottleneck methode
2. Stochastische bottleneck methode (gemiddeld 0,17 keer langzamer)
3. Volledig willekeurig (gemiddeld 0,70 keer langzamer)
4. Abstract rooster (gemiddeld 1,75 keer langzamer)
5. Bottleneck met locale optimalisatie (gemiddeld 3,87 keer langzamer)

Zowel de bottleneck methode met locale optimalisatie als de abstracte rooster methode verliezen het grootste gedeelte van de tijd bij het maken van het beginrooster. Bij de uitvoering van het genetisch algoritme en de locale optimalisatie wordt wel wat tijd teruggewonnen, maar dit staat niet in verhouding.

De volledig willekeurige methode en de stochastische bottleneck methode verliezen tijd bij het uitvoeren van het genetisch algoritme. Door de verschillende beginroosters duurt het langer voordat het algoritme convergeert.

Tijd uitgezet tegen de kwaliteit



Figuur 39: Tijd uitgezet tegen de kwaliteit

In dit vergelijk zijn de methode abstract rooster en bottleneck met locale optimalisatie weggelaten. In paragraaf 11.1: 'Tijd' zagen we al dat het maken van de beginroosters met deze methode al meer dan de 200 seconden duurt. Deze methoden vallen hierdoor buiten het bereik van de grafiek.

We zien dat zowel de bottleneck methode als de stochastische bottleneck methode binnen korte tijd roosters met een goede kwaliteit kunnen maken. De bottleneck is iets sneller dan de stochastische bottleneck methode. Als we de kwaliteit vastzetten zit de lijn van de bottleneck methode net onder die van de stochastische methode.

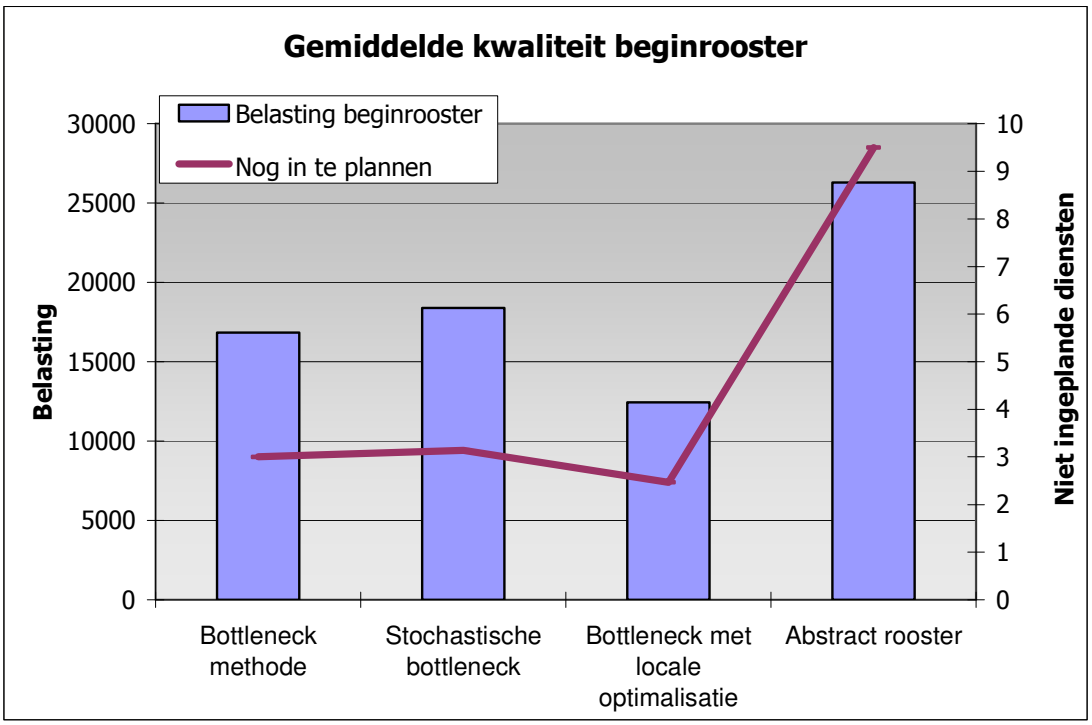
De volledig willekeurige methode heeft meer tijd nodig om dezelfde kwaliteit roosters te maken dan de andere twee methoden. Daar staat tegenover dat het erop lijkt dat met deze methode betere roosters gevonden kunnen worden. Of dit het geval is onderzoeken we in de onderstaande paragrafen.

Conclusie

Als we kijken naar de tijd die nodig is voor het maken van een rooster zien we dat de bottleneck methode het snelst is. De methoden stochastische bottleneck en volledig willekeurig verliezen tijd bij het uitvoeren van het genetisch algoritme, waar methoden abstract rooster en bottleneck met locale optimalisatie tijd verliezen bij het maken van het beginrooster.

11.5 Kwaliteit beginrooster

Onderstaand figuur geeft een weergave van de kwaliteit van het beste van de vijf gemaakte beginroosters bij verschillende methoden. De kwaliteit van de methode volledig willekeurig is achterwege gelaten. De gemiddelde waarde van deze methode ligt vele malen hoger dan de anderen.



Figuur 40: Gemiddelde kwaliteit van het beste beginrooster van de vijf die zijn gemaakt[3]

Als we een ordening maken van goed naar slecht op basis van de gemiddelde kwaliteit krijgen we:

1. Bottleneck met locale optimalisatie
2. Bottleneck methode
3. Stochastische bottleneck methode
4. Abstract rooster
5. Volledig willekeurig

De gemiddelde waarden voor het aantal nog in te plannen diensten bij de methoden bottleneck, stochastische bottleneck en bottleneck met locale optimalisatie liggen echter dicht bij elkaar. Ditzelfde kan gezegd worden voor de belasting. Om te onderzoeken of de verschillen significant zijn maken we gebruik van de Wilcoxon toets. Als stochast nemen we het aantal nog in te plannen diensten. Voor het testen van de belasting gebruiken we als stochast de belasting. Vervolgens toetsen we de volgende nulhypothesen:

1. Het aantal ingeplande diensten bij stochastische bottleneck methode ≤ het aantal ingeplande diensten bij de bottleneck methode.
2. Het aantal ingeplande diensten bij de bottleneck methode ≤ aantal ingeplande diensten bij de bottleneck met locale optimalisatie.

[3] Tussen het uitvoeren van tests voor de abstracte roosters instellingen en bovenstaande test zijn wijzigingen in de code doorgevoerd om de juiste gegevens op te slaan. Dit heeft er onbedoeld toe geleid dat de resultaten van het abstracte rooster slechter uitvallen dan in hoofdstuk 9 te zien is. De oorzaak van het probleem is niet achterhaald.

3. De belasting bij stochastische bottleneck methode ≤ de belasting bij de bottleneck methode.
4. De belasting bij de bottleneck methode ≤ de belasting bij de bottleneck met locale optimalisatie.

Uit de toetsen kwamen deze resultaten:

Wilcoxon test	Stochastische bottleneck ≤ bottleneck	Bottleneck met locale optimalisatie ≤ bottleneck
p-waarde diensten	0.00	0.00
p-waarde belasting	0.00	0.00

In alle gevallen mogen we de nulhypothesen verwerpen en aannemen dat de alternatieve hypothesen waar zijn. Dat wil zeggen dat in het beste rooster gemaakt met de bottleneck methode met locale optimalisatie gemiddeld meer diensten zijn ingepland en gemiddeld een lagere belasting heeft dan het beste rooster gemaakt met de bottleneck methode. De bottleneck methode kan op haar beurt hetzelfde zeggen ten opzichte van de stochastische bottleneck methode.

Interessant is bovendien het gemiddeld aantal in te plannen diensten van het beste rooster van alle vijf gemaakte beginroosters en het gemiddelde van de overige vier roosters. In onderstaande tabel staan deze waarden vermeld.

Aantal nog in te plannen diensten	Bottleneck methode	Stochastische bottleneck methode	Bottleneck methode met locale optimalisatie	Abstract rooster
Gemiddelde van het beste rooster	3	3.14	2.47	9.5
Standaarddeviatie van het beste rooster	0	0.40	0.50	2.38
Gemiddelde van de overige roosters	11.56	10.68	3.05	10.01
Standaarddeviatie van de overige roosters	2.50	2.21	0.11	2.39

Allereerst zien we dat het beste rooster van de bottleneck methode telkens dezelfde is, zoals we ook verwachten. In paragraaf 4.1: 'Bottleneck methode' hadden we gezien dat de methode deterministisch is. Het beste rooster dat gemaakt wordt door de stochastische bottleneck methode verschilt niet bijzonder veel van die van de bottleneck methode. Groot verschil is wel dat het gemiddelde van de overige vier roosters beter is dan bij de bottleneck methode. Hierdoor is de kans groter dat deze roosters invloed kunnen uitoefenen bij de mutatie en cross-over.

De kwaliteit van het beste rooster van het abstracte rooster blijft ver achter bij de andere methoden. Het gemiddelde van de overige vier roosters is echter nagenoeg hetzelfde als bij de stochastische bottleneck methode. Daarbij moet opgemerkt worden dat in de huidige implementatie geen rekening wordt gehouden met het verschil tussen de roosters. De kwaliteit van de overige vier rooster liggen dicht bij die van de beste omdat de roosters maar op een beperkt aantal diensten van elkaar verschillen. Dit is bij de stochastische bottleneck en de bottleneck methode niet het geval.

Conclusie

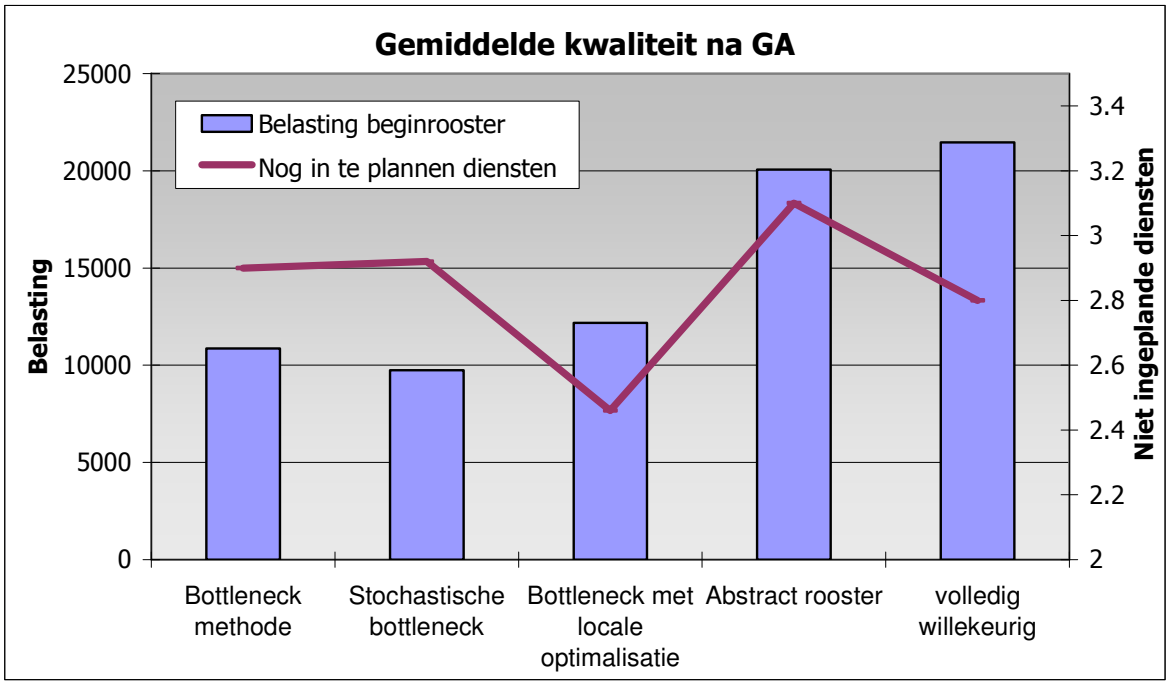
De bottleneck methode met locale optimalisatie levert de beste kwaliteit beginroosters op, gevolgd door de bottleneck methode en de stochastische bottleneck methode. De kwaliteit van de beginroosters gemaakt met de abstracte roosters en de volledig willekeurige methode blijven achter bij de rest.

Daar staat tegenover dat de gemiddelde kwaliteit van vijf gemaakte beginroosters door het abstracte rooster en de bottleneck methode nagenoeg gelijk zijn. De stochastische bottleneck methode en de bottleneck methode met locale optimalisatie hebben gemiddeld over de vijf beginroosters een betere kwaliteit dan de beginroosters gemaakt door de bottleneck methode. Ook

de variantie van het aantal nog in te plannen diensten is lager, wat aangeeft dat de kwaliteit van de oplossingen dichter bij elkaar liggen.

11.6 Kwaliteit tussenrooster

Als we kijken naar de kwaliteit na het uitvoeren van het genetisch algoritme zien we het volgende:



Figuur 41: Gemiddelde kwaliteit na het uitvoeren van het genetisch algoritme

Op basis van de grafiek lijkt de ordening van goed naar slecht als we kijken naar de gemiddelde kwaliteit na uitvoering van het genetisch algoritme als volgt te zijn:

1. Bottleneck met locale optimalisatie
2. Volledig willekeurig
3. Stochastische bottleneck methode
4. Bottleneck methode
5. Abstract rooster

Hierbij kijken we eerst naar het aantal nog in te plannen diensten en daarna naar de belasting.

Om te onderzoeken of de verschillen significant zijn maken we gebruik van de Wilcoxon toets. Als stochast nemen we het aantal nog in te plannen diensten. Vervolgens toetsen we de volgende nulhypothesen:

1. Het aantal ingeplande diensten bij de bottleneck methode ≤ het aantal ingeplande diensten bij de stochastische bottleneck methode.
2. Het aantal ingeplande diensten bij de bottleneck methode ≤ aantal ingeplande diensten bij de bottleneck met locale optimalisatie.
3. Het aantal ingeplande diensten bij de bottleneck methode ≤ aantal ingeplande diensten bij de volledig willekeurige methode.
4. Het aantal ingeplande diensten bij de abstracte rooster methode ≤ aantal ingeplande diensten bij de bottleneck methode.

Voor het testen van de belasting gebruiken we als stochast de belasting en toetsen we de nulhypothesen:

5. De belasting bij de bottleneck methode ≤ de belasting bij de stochastische bottleneck methode.
6. De belasting bij de bottleneck methode ≤ de belasting bij de bottleneck methode met locale optimalisatie.
7. De belasting bij de bottleneck methode ≤ de belasting bij de volledig willekeurige methode.
8. De belasting bij de abstracte rooster methode ≤ de belasting bij de bottleneck methode.

Uit de toetsen kwamen deze resultaten:

Wilcoxon test	Bottleneck ≤ Stochastische bottleneck	Bottleneck ≤ Bottleneck met locale optimalisatie	Bottleneck ≤ volledig willekeurig	Abstract rooster ≤ Bottleneck
p-waarde diensten	0.69	0.00	0.02	0.08
p-waarde belasting	0.00	0.57	1.00	0.00

Aan de hand van de toetsen zien we dat we nulhypothesen 1, 4, 6 en 7 niet mogen verwerpen. Dit betekent dat het aantal nog in te plannen diensten van de bottleneck methode, stochastische bottleneck methode en de abstracte rooster methode niet significant van elkaar verschillen. De hypothesen 2, 3, 5 en 8 mogen we wel verwerpen. We mogen dus aannemen dat zowel de bottleneck methode met locale optimalisatie als de volledig willekeurige methode in staat zijn minder in te plannen diensten over te houden dan de bottleneck methode. De stochastische bottleneck methode heeft bovendien een significant lagere belasting dan de bottleneck methode.

We zien dus dat zowel de stochastische bottleneck methode als de volledig willekeurige methode betere tussenroosters hebben in vergelijking met de bottleneck methode. Van beide methoden hadden we geconstateerd dat de beginoplossingen zeer van elkaar verschillen, maar kwalitatief beter op elkaar lijken dan bij de bottleneck methode. Het genetisch algoritme profiteert hier van. Het is in staat de oplossingen beter te muteren en combineren waardoor het betere roosters kan vinden.

De bottleneck methode met locale optimalisatie levert kwalitatief de beste tussenroosters. Hoewel de beginroosters niet veel van elkaar verschillen is de kwaliteit van de beginroosters wel hoog. Het genetisch algoritme is met kwalitatief betere beginroosters dus in staat betere locale minima te vinden.

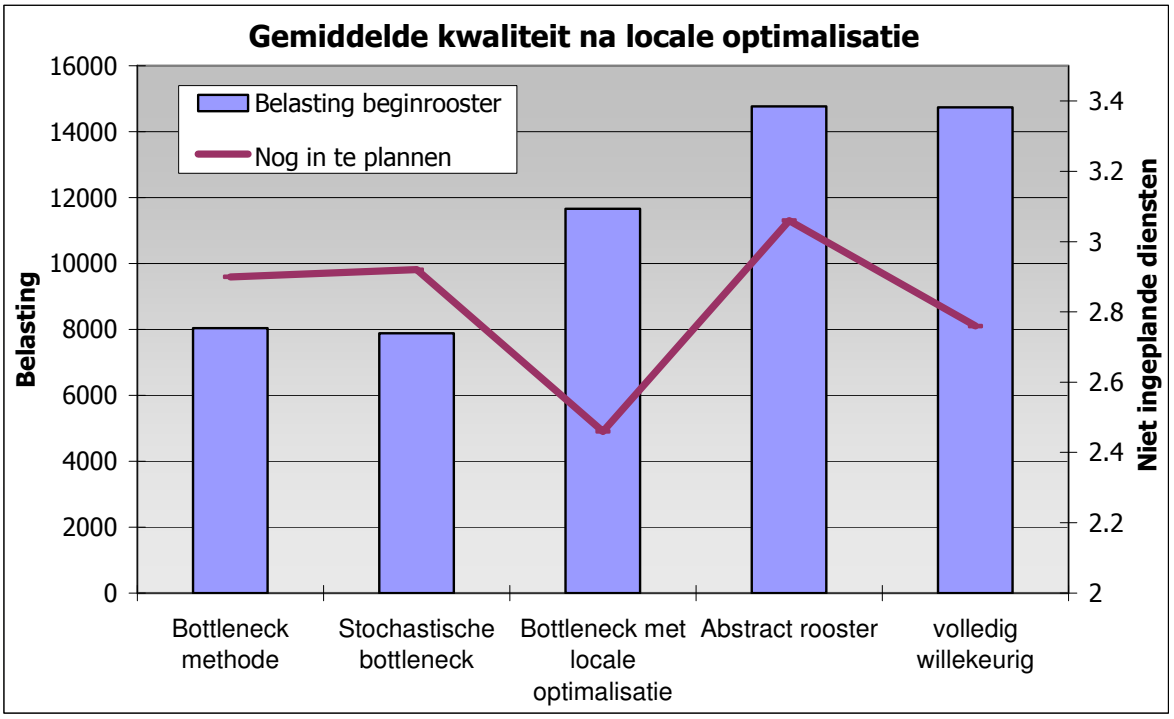
Conclusie

De bottleneck methode met locale optimalisatie maakt de kwalitatief beste tussenroosters. Ook de stochastische bottleneck methode en de volledige willekeurige methode zijn in staat betere tussenroosters te genereren. Dit kan verklaard worden doordat de beginroosters zeer van elkaar verschillen, maar kwalitatief aan elkaar gewaagd blijven.

Omdat de beginroosters bij de abstracte rooster methode erg op elkaar lijken en kwalitatief achterblijven bij de beginroosters van bijvoorbeeld de stochastische bottleneck en bottleneck methode, is ook de kwaliteit van het tussenrooster slechter dan de andere methoden.

11.7 Kwaliteit eindrooster

Tot slot kijken we naar de kwaliteit van het rooster na het uitvoeren van locale optimalisatie.



Figuur 42: Gemiddelde kwaliteit na het uitvoeren van locale optimalisatie

Als we de methoden ordenen van hoge kwaliteit naar lage kwaliteit krijgen we:

1. Bottleneck met locale optimalisatie
2. Volledig willekeurig
3. Stochastische bottleneck methode
4. Bottleneck methode
5. Abstract rooster

Kijkend naar de grafiek zien we dat er, afgezien van een verlaging van de belasting van elke methode, weinig is veranderd na het uitvoeren van het genetisch algoritme. Als we kijken naar de verbetering van de belasting die de locale optimalisatie ten opzichte van het tussenrooster heeft behaald voor elke methode heeft opgeleverd zien we:

Percentage verbetering belasting	Bottleneck	Stochastische bottleneck	Bottleneck met locale optimalisatie	Abstract rooster	volledig willekeurig
Percentage	26%	19%	4%	26%	31%

Als verwacht levert de locale optimalisatie bij de bottleneck met locale optimalisatie methode nauwelijks verbeteringen op. De verbetering bij de stochastische bottleneck methode ligt ook wat lager dan bij de bottleneck methode. De procentuele verbetering bij de volledig willekeurige methode ligt hoger zoals we gezien de tijd die gebruikt wordt voor locale optimalisatie hadden verwacht. Opvallend dat we ditzelfde niet zien bij de abstracte rooster methode.

Hoewel er geen belangrijke veranderingen lijken te zijn opgetreden toetsen we opnieuw of de verschillen tussen de methoden significant zijn. Hierbij voeren we dezelfde nulhypothesen uit als in voorgaande paragraaf.

Wilcoxon test	Bottleneck ≤ Stochastische bottleneck	Bottleneck ≤ Bottleneck met locale optimalisatie	Bottleneck ≤ volledig willekeurig	Abstract rooster ≤ Bottleneck
p-waarde diensten	0.69	0.00	0.00	0.10
p-waarde belasting	0.03	1.00	1.00	0.00

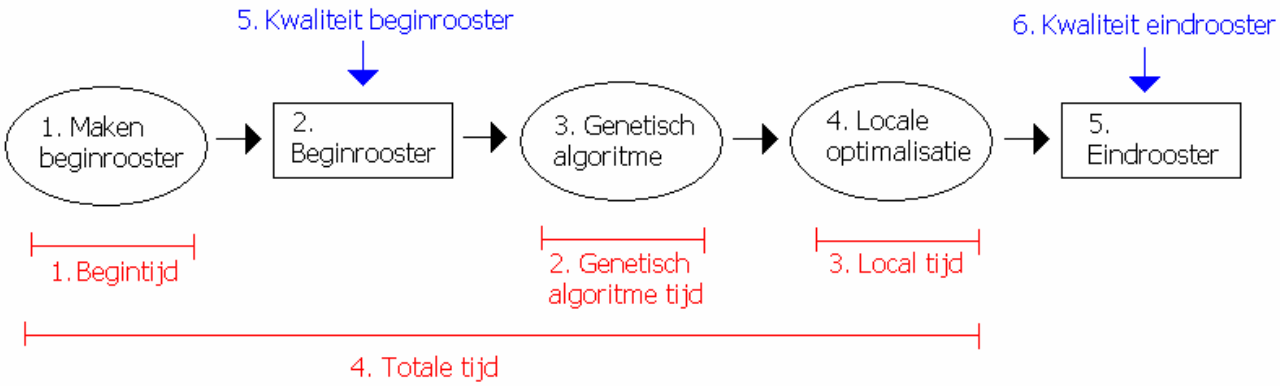
Een aantal waarden zijn iets veranderd, maar we mogen nog steeds dezelfde nulhypothese verwerpen. Dat houdt in dat de stochastische bottleneck methode, de bottleneck methode met locale optimalisatie en de volledig willekeurige methode kwalitatief betere eindroosters maken dan de bottleneck methode.

Conclusie

De kwaliteit van het eindrooster laat eenzelfde beeld zien als de kwaliteit van de tussenroosters. De locale optimalisatie verlaagt de belasting bij de bottleneck methode en het abstracte rooster met gemiddeld 26%. Bij de bottleneck methode met locale optimalisatie is dit slechts 4% en ook bij de stochastische bottleneck methode is het verschil lager. Bij de volledig willekeurige methode is de verbetering van de belasting 31%.

11.8 Samenvatting

In dit hoofdstuk hebben we vijf methoden op de aantal punten met elkaar vergeleken.



Figuur 43: De onderdelen voor het maken van een rooster

Als we kijken naar de tijd die nodig is voor het maken van een rooster is de volgorde van snel naar langzaam tussen de methoden:

1. Bottleneck methode
2. Stochastische bottleneck methode (gemiddeld 0,17 keer langzamer)
3. Volledig willekeurig (gemiddeld 0,70 keer langzamer)
4. Abstract rooster (gemiddeld 1,75 keer langzamer)
5. Bottleneck met locale optimalisatie (gemiddeld 3,87 keer langzamer)

De methoden 2 en 3 verliezen tijd bij het uitvoeren van het genetisch algoritme. De beginroosters van deze methoden verschillen meer van elkaar ten opzichte van de andere methoden, waardoor het genetisch algoritme meer tijd nodig heeft te convergeren. Methoden 4 en 5 verliezen tijd bij het maken van het beginrooster.

Als we kijken naar de kwaliteit van de beginroosters levert de bottleneck methode met locale optimalisatie de beste kwaliteit op, gevolgd door de bottleneck methode en de stochastische bottleneck methode. De kwaliteit van de beginroosters gemaakt met de abstracte roosters en de volledig willekeurige methode blijft achter bij de rest.

De bottleneck methode met locale optimalisatie maakt de kwalitatief beste eindroosters. Door de hogere kwaliteit van de beginroosters is het genetisch algoritme in staat betere locale minima te vinden. Ook de stochastische bottleneck methode en de volledige willekeurige methode zijn in staat betere tussenroosters te genereren. Dit kan verklaard worden doordat de beginroosters zeer van elkaar verschillen, maar kwalitatief aan elkaar gewaagd blijven.

12. Conclusies en aanbevelingen

Het doel van het afstudeerverslag is:

> **Onderzoek of een methode kan worden gevonden die betere beginroosters maakt dan de huidige methode.**

Eerst hebben we in hoofdstuk 4: 'Huidige planautomaat' gekeken naar de werking van het huidige algoritme. Aan de hand van de opmerkingen die we hierover konden maken hebben we een aantal nieuwe methoden ontwikkeld om beginroosters te maken zoals besproken is in hoofdstuk 5: 'Nieuwe methoden'. Vervolgens zijn de methoden aan de hand van tests met elkaar vergeleken. Uit de testresultaten zijn een aantal conclusies te trekken. Hieronder worden deze conclusies en de daarbijbehorende aanbevelingen besproken.

Bottleneck met locale optimalisatie

De bottleneck methode met locale optimalisatie voert na het maken van de beginroosters met de bottleneck methode locale optimalisatie uit. Deze methode gebruikt het feit dat locale optimalisatie zorgt voor kwaliteitsverbetering. De gedachte is dat hierdoor kwalitatief betere beginroosters kunnen worden gemaakt, wat ook kan zorgen voor betere eindroosters.

Uit de testresultaten kwam naar voren dat de kwaliteit van de beginroosters gemaakt met deze methode significant beter is dan die van de beginroosters gemaakt door de huidige methode. Maar ook de kwaliteit van het eindrooster was gemiddeld van hogere kwaliteit dan bij de bottleneck methode. Dit betekent dat deze methode kwalitatief betere begin- en eindroosters kan maken dan de huidige methode. Dat houdt in dat als het genetisch algoritme begint met kwalitatief betere beginrooster het in staat is een beter locaal optimum te vinden.

Als naar de tijd wordt gekeken is deze methode echter minder geschikt voor het maken van beginroosters dan de bottleneck methode. Het uitvoeren van locale optimalisatie zorgt ervoor dat het maken van een rooster gemiddeld 3,87 keer langer duurt dan met de bottleneck methode.

Stochastische bottleneck methode

De stochastische bottleneck methode maakt de bottleneck methode stochastisch door niet de beste dienst, maar met een kans de beste dienst eerst in te plannen. Doel hiervan is te zorgen dat de kwaliteit van verschillende beginroosters dichter bij elkaar ligt zodat we verschillende, maar kwalitatief gelijke beginroosters krijgen.

Bij het uitvoeren van de tests zagen we dat met deze methode de kwaliteit van de beginroosters slechter is dan bij de bottleneck methode. Door niet telkens de beste dienst in te plannen verliezen we kwaliteit. Daar staat tegenover dat de eindroosters die worden gemaakt met de stochastische bottleneck methode van betere kwaliteit zijn. De verklaring is dat het genetisch algoritme profijt heeft van verschillende, maar kwalitatief gelijke roosters. Hierdoor kan een groter gedeelte van de oplossingsruimte worden bekeken.

Gelet op de tijd duurt het uitvoeren van het algoritme voordat het convergeert gemiddeld 0,17 keer langer dan de bottleneck methode. Als we kijken hoe lang de stochastische bottleneck methode erover doet een bepaalde kwaliteit te behalen, hebben we gezien dat deze methode er net iets langer over doet dan de bottleneck methode.

Volledig willekeurig

De methode volledig willekeurig maakt beginroosters door willekeurige diensten toe te wijzen aan willekeurige medewerkers. Doel van deze methode is te onderzoeken of de kwaliteit van het beginrooster invloed heeft op de kwaliteit van het eindrooster. Deze methode zorgt net als de stochastische bottleneck methode voor verschillende, maar kwalitatief gelijke beginroosters.

De beginroosters gemaakt door deze methode bleken na het uitvoeren van de tests als verwacht van een zeer slechte kwaliteit. De kwaliteit van de eindroosters echter, is beter dan die van de

bottleneck methode en de stochastische bottleneck methode. Hieruit blijkt dat het uitmaakt voor de kwaliteit van het eindrooster met welke roosters het genetisch algoritme begint. Het geeft aan dat de beginroosters van een bottleneck methode in de buurt van een locaal optimum liggen, waar het genetisch algoritme niet uit kan komen. Door het genetisch algoritme een groter deel van de oplossingsruimte te laten doorzoeken kan een beter optimum worden gevonden.

De tijd die nodig is voor het convergeren van het algoritme is echter gemiddeld 0,70 keer langer. Verklaring hiervoor is dat het doorzoeken van de oplossingsruimte meer tijd in beslag neemt.

Abstract rooster

De abstract rooster methode vereenvoudigt het oorspronkelijke probleem en probeert deze eerst op te lossen. Elke dag wordt alleen gekeken of een medewerker een dienst moet doen. Welke dienst dat is, is niet van belang. Hierdoor moeten kwalitatief goede roosters gemaakt kunnen worden, waarbij gelet wordt op de structuur van het beginrooster. Ook is het met deze methode mogelijk verschillende, maar kwalitatief gelijke rooster te maken en kunnen overtredingen van regels worden gemaakt om de oplossingsruimte te doorzoeken.

De beginroosters die zijn gemaakt met de abstracte rooster methode zijn slechter dan de beginroosters van de bottleneck methode. Uit de testresultaten is gebleken dat de huidige implementatie van de methode niet goed genoeg is om de kwaliteit van de beginroosters te verhogen. Ook de kwaliteit van de eindroosters is slechter dan bij de bottleneck methode. Reden hiervoor kan zijn dat de kwaliteit van de beginroosters te laag is of dat het maken van verschillende beginroosters niet is meegenomen in de implementatie.

De tijd die nodig is voor het algoritme om te convergeren duurt gemiddeld 1,75 keer langer dan bij de bottleneck methode. Het grootste deel van dit verschil zit in de tijd die nodig is bij de abstracte rooster methode om het beginrooster te maken. Het verschil kan mogelijk worden verminderd door de code te optimaliseren. Ook heeft het genetisch algoritme voor het maken van een abstract rooster veel generaties nodig heeft om te convergeren naar een oplossing. Dit kan vermoedelijk worden teruggebracht door een betere keuze van mutatie- en cross-overoperatoren. Ook op deze manier kan het verschil in tijd worden teruggebracht.

Bij de bottleneck methode met locale optimalisatie hebben we gezien dat kwalitatief betere beginroosters een voordeel zijn voor het vinden van een beter eindrooster. Bij de stochastische bottleneck methode en volledig willekeurige methode zagen we dat de kwaliteit verder verbeterd kan worden door veel verschillende, maar kwalitatief gelijke beginroosters te gebruiken. Hoewel de abstracte rooster methode in theorie in staat is beide te doen, is gebleken dat de implementatie onvoldoende is de voordelen eruit te halen.

Vergelijk ten opzichte van de bottleneck methode	Stochastische bottleneck	Bottleneck met locale optimalisatie	Abstract rooster	volledig willekeurig
Aantal keer langzamer tot convergeren methode	0.17	3.87	1.75	0.70
Kwaliteit beginrooster	slechter	beter	slechter	slechter
Kwaliteit eindrooster	beter	beter	slechter	beter

Aanbevelingen

Als de gebruiker alleen een beginrooster wil maken is de bottleneck methode het meest geschikt. Hoewel de kwaliteit van het beginrooster gemaakt met de bottleneck methode met locale optimalisatie beter is, duurt het maken van dit rooster te lang om gebruikt te kunnen worden.

Als de gebruiker een rooster wil maken door gebruik te maken van het genetisch algoritme is het aan te bevelen om de gebruiker twee mogelijkheden te geven. Heeft de gebruiker een beperkte hoeveelheid tijd dan is het aan te raden om de stochastische bottleneck methode te gebruiken. Als de tijd die nodig is voor het maken van het rooster minder belangrijk is dan valt de volledig willekeurige methode aan te bevelen.

Het maken van beginroosters met de abstracte rooster methode moet momenteel niet worden gebruikt. Alvorens met deze methode betere begin- of eindroosters kunnen worden gemaakt zal de methode verder ontwikkeld moeten worden. Daarbij moet gedacht worden aan het optimaliseren van de code en het ontwikkelen van betere mutatie- en cross-overoperatoren. De verwachting is dat hierdoor de benodigde tijd sterk kan worden teruggedrongen. Voor de kwaliteit moet het mogelijk zijn het kwaliteitsniveau van de volledig willekeurig methode te kunnen overtreffen.

13. Bronvermelding

[1] W. Winkelhuijzen, 1999, "Dienstroostergeneratie bij de Luchtvaartverkeersleiding Nederland", scriptie bij ORTEC aan de Erasmus Universiteit.

[2] M. van der Put, 2005, "Personnel Scheduling: Generating Schedules using Meta-Heuristics", scriptie bij ORTEC aan de Technische Universiteit Delft.

[3] A. Eiben, J. Smith, 2003, "Introduction to Evolutionairy Computing", Uitgeverij: Heidelberg.

[4] H. Tijms, A. Ridder, 2002, "Mathematische Programmering", collegedictaat Vrije Universiteit.

[5] M. Kroon, 2004, "Dienstreeksen plannen in personeelsroosters", scriptie bij ORTEC aan de Universiteit Twente.

[6] R. Flik, A. Mokveld, D. Boon, I. Corver, W. Luijten, J. Winde, 2002, " ORTEC: Volgens paspoort volwassen…", biografie van ORTEC ter gelegenheid van 21-jarig bestaan.

[7] M. Sellmann, K. Zervoudakis, P.Stamatopoulos, T. Fahle, 2000, "Intergrating direct CP search and CP-based column generation for the airline crew assigment problem", *Proceedings of the 2nd International workshop on integration of AI and OR techniques in combinatorial optimization problems CP-AI-OR '00, p. 163-170.*

[8] A. Mason, M. Smith, 1998, "A nested column generator for solving rostering problems with integer programming", *International conference on optimization: Techniques and applications, p. 827-834*.

[9] M. de Gunst, A. van der Vaart, 2001, "Statistische Data Analyse", collegedictaat Vrije Universiteit.

[10] J. Oosterhoff, A. van der Vaart, 2001, "Algemene statistiek", collegedictaat Vrije Universiteit.

Bijlage A. Definities

Beginrooster

Een *beginrooster* is een rooster dat wordt gemaakt om het genetisch algoritme mee te beginnen. Dit rooster kan ook worden gebruikt door de klant zonder het algoritme uit te voeren.

Bezettingseisen

Voor een roostergroep kunnen ook *bezettingseisen* gedefinieerd worden. Hierin staat voor elke dienst per dag van de week aangegeven hoeveel van dit soort diensten er ingepland moeten worden.

Belasting

De *belasting* van een rooster is de score voor het overtreden van de ingestelde criteria.

Convergeren

Convergeren houdt in dat de individuen van het algoritme bij een (locaal) optimum samenkomen. Als een x aantal generaties geen verbetering optreedt wordt vaak aangenomen dat het algoritme is geconvergeerd.

Criterium

De *criteria* bepalen wat wel of niet mag, maar nu is de overtreding echter toegestaan, op straffe van een belasting. Criteria zijn daarom zachte voorwaarden, omdat hier niet altijd aan voldaan hoeft te worden.

Dienst

Activiteiten zijn werkzaamheden die uitgevoerd kunnen worden en een *dienst* is een verzameling van één of meerdere activiteiten die tijdens een gegeven tijdsinterval uitgevoerd moeten worden. *Bijzondere diensten*, dit zijn aparte diensten die bestaan uit activiteiten zoals allerlei soorten verlof, ziekte, onwettige afwezigheid en dergelijke. Ook roostervrij is een bijzondere dienst.

Eindrooster

Een eindrooster is een rooster dat ontstaat na het uitvoeren van het genetisch algoritme en mogelijk de locale optimalisatie.

Inplannen van diensten

Het *inplannen van diensten* of ook wel *inroosteren* betekent het toewijzen van diensten aan medewerkers. Een dienst A op maandag inplannen bij medewerker Jan betekent dat medewerker Jan op maandag dienst A moet uitvoeren.

Kwaliteit

Het aantal diensten dat in een rooster is ingepland en de belasting die ontstaat door het overtreden van de criteria vormen samen de *kwaliteit* van een rooster.

Kwaliteitsverlies inplannen

Kwaliteitsverlies inplannen is de kwaliteit die we verliezen bij het toewijzen van diensten aan medewerkers volgens het abstracte rooster.

Kwaliteitsverlies abstract rooster

Kwaliteitsverlies abstract rooster is de kwaliteit die we verliezen als we niet het optimale abstracte rooster kunnen vinden.

Dienstenverzameling

Elke roostergroep heeft zijn eigen *dienstenverzameling*, dit is een verzameling van diensten die door deze roostergroep gebruikt mogen worden. Bijzondere diensten horen hier niet bij omdat deze voor elke medewerker gebruikt mogen worden en niet alleen voor medewerkers uit een bepaalde roostergroep.

Mate van convergentie

De *mate van convergentie* van een genetisch algoritme geeft aan hoe snel het algoritme naar een bepaalde oplossing convergeert.

Medewerkers

Medewerkers zijn de mensen waarvoor de planning gemaakt wordt. In HARMONY zijn er erg veel medewerkerkenmerken op te geven, hierbij moet onder meer gedacht worden aan standaardkenmerken zoals naam, adres, telefoonnummer en geboortedatum. Verder zijn er ook meer geavanceerde kenmerken zoals *functies* en *kwalificaties*. Deze geven aan welke functies een medewerker uit mag voeren en welke kwalificaties een medewerker heeft. Dit is van belang als er voor een activiteit een bepaald kwalificatieniveau vereist is. Een verpleegkundige kan bijvoorbeeld geen activiteit uitvoeren waarvoor de kwalificatie chirurg nodig is. Binnen HARMONY heeft een kwalificatie ook nog een bepaald niveau, dat gerepresenteerd wordt door een getal. Er moet minstens aan dit niveau voldaan worden. Dit betekent dat, wanneer het kwalificatieniveau hoger is dan er vereist is, dat de betreffende medewerker deze activiteit uit mag voeren.

P-waarde

De *p-waarde* heet ook wel overschrijdingskans. Als voor een toetsingsgrootheid T de waarde t is waargenomen is de overschrijdingskans gelijk aan $P_{H_o}(T \geq t)$. In woorden is het de kans dat de toetsingsgrootheid een waarde groter dan t heeft onder de nulhypothese. Is de kans kleiner of gelijk aan een gekozen grenswaarde verwerpen we de nulhypothese (zie: [10] J.Oosterhoff, 2001).

Planningsperiode

De *planningsperiode* is de tijdsduur waarvoor een dienstrooster gemaakt moet worden. In veel gevallen is dit een maand of vier weken, maar er is binnen HARMONY geen enkele belemmering om voor een kortere of langere periode te kiezen.

Regel

Alles wat zegt dat iets wel of niet mag (door de planautomaat) noemen we een *regel*. De planautomaat mag geen rooster genereren waarin de regels overtreden worden. Regels zijn dus zogenaamde harde voorwaarden, voorwaarden waar altijd aan voldaan moet worden.

Roostergroep

Een *roostergroep* is een groep van medewerkers. Een planning wordt altijd voor één roostergroep tegelijk gemaakt. Roostergroepen zijn nuttig om mensen naar aanleiding van kenmerken in bepaalde groepen in te delen. Ook is het op die manier makkelijker om goede planningen te maken, omdat het makkelijker is om tien planningen voor 30 personen te maken, dan om één planning voor 300 personen te maken.

Sampling space

Een *sampling space* is een populatie waaruit de volgende generatie wordt bepaald. Een sampling space wordt in ons geval gevormd door individuen van de child population en de voorgaande sampling space.

Score abstract rooster

Score abstract rooster is de waarde van de fitness function. De score is de som van de overtredingen van de regels keer een gewicht plus de score voor het aantal nog in te plannen diensten keer een gewicht plus de score voor het overtreden van de criteria keer een gewicht.

Serie

Een *serie* bestaat uit meerdere diensten die op een aantal opeenvolgende dagen aan een medewerker zijn toegewezen. Een serie is geëindigd als tussen twee diensten een grotere rusttijd zit dan is ingesteld door de gebruiker. Dit kan bijvoorbeeld zijn als er meer dan 24 uur rust tussen twee diensten zit.

Toewijzen van een abstracte dienst

Het *toewijzen van een abstracte dienst aan een medewerker* is het veranderen van die abstracte dienst van de medewerker in geen dienst, normale dienst of nacht dienst. Hoewel de diensten niet echt worden toegewezen, hebben we daar voor de eenvoud wel over.

Uitplannen van diensten

Het *uitplannen van diensten* betekent het ongedaan maken van toewijzingen van diensten aan medewerkers. Als dienst A op maandag bij medewerker Jan staat ingepland en deze wordt uitgepland betekent het dat medewerker Jan op maandag niet meer dienst A in zijn rooster heeft staan.

Bijlage B. Stageplek Ortec

1. Geschiedenis van ORTEC

Op 1 april 1981 besloten vijf vrienden het bedrijf 4xB, wat staat voor Bureau voor Betere Bedrijfsbeslissingen, op te richten. De vijf, allen econometrie studenten en een aantal nog werkzaam bij de Erasmus universiteit, zagen in de praktijk weinig toepassing van econometrie. Ze besloten een bedrijf op te richten waarmee ze los van de universiteit de stof in praktijk konden brengen.

Naast het werken voor de universiteit werden op kleine schaal projecten aangenomen. Van een duidelijke visie was nog geen sprake. In 1982 veranderde de naam van 4xB in ORTEC, wat staat voor Operations Research TEChnology. Aanvankelijk werd het werk thuis gedaan en op de universiteit, later bij één van de oprichters thuis op de benedenverdieping in Rotterdam.

Met klanten als Schuitema, Rabobank en Shell begon ORTEC te groeien en in 1986 kwam het eerste strategische plan. De oprichters beseften dat ORTEC serieuze vormen begon aan te nemen en dat groei noodzakelijk was. Met een omzet van 5 ton in guldens en 5 full time medewerkers verwachtte ze in 1990 1,6 miljoen gulden om te kunnen zetten met 13 full time medewerkers. De doelstelling van de vereniging onder firma werd: "Exploitatie bedrijfseconomisch adviesbureau, met name gericht op adviezen bij beleids- en efficiency vraagstukken waarbij gebruik wordt gemaakt van kwantitatieve methoden uit de Operations Research en Econometrie". De belangrijkste onderdelen moesten worden: consultancy en ontwikkelen van beleidsondersteunende systemen inclusief programmatuur.

In 1987 verhuisde ORTEC van de benedenverdieping in Rotterdam naar een kantorengebouw in dezelfde stad. Al in 1989 werd ook dit pand te klein en verhuisde ORTEC naar de Molenwiek in Gouda, waar een groot deel nog altijd is gevestigd. In 1990 werd de VOF omgezet in ORTEC Consultants BV.

Tot 1991 had ORTEC nog nauwelijks structuur. Het was een groep mensen die gezamenlijk projecten deed, maar van afdelingen was nog geen sprake. In 1991 veranderde dit door het ontstaan van de afdelingen Pensioenen, Productieplanning, Logistiek en Planning & Indeling.

Vanaf 1993 gaat ORTEC de grens over met eerst een samenwerking met Ditel in Belgie en later de oprichting van ORTEC Frankrijk. In 1996 volgt een uitbreiding in Indonesie en in 1997 Frankrijk. In 2000 wordt hetzelfde gedaan in Zuid-Afrika en Amerika.

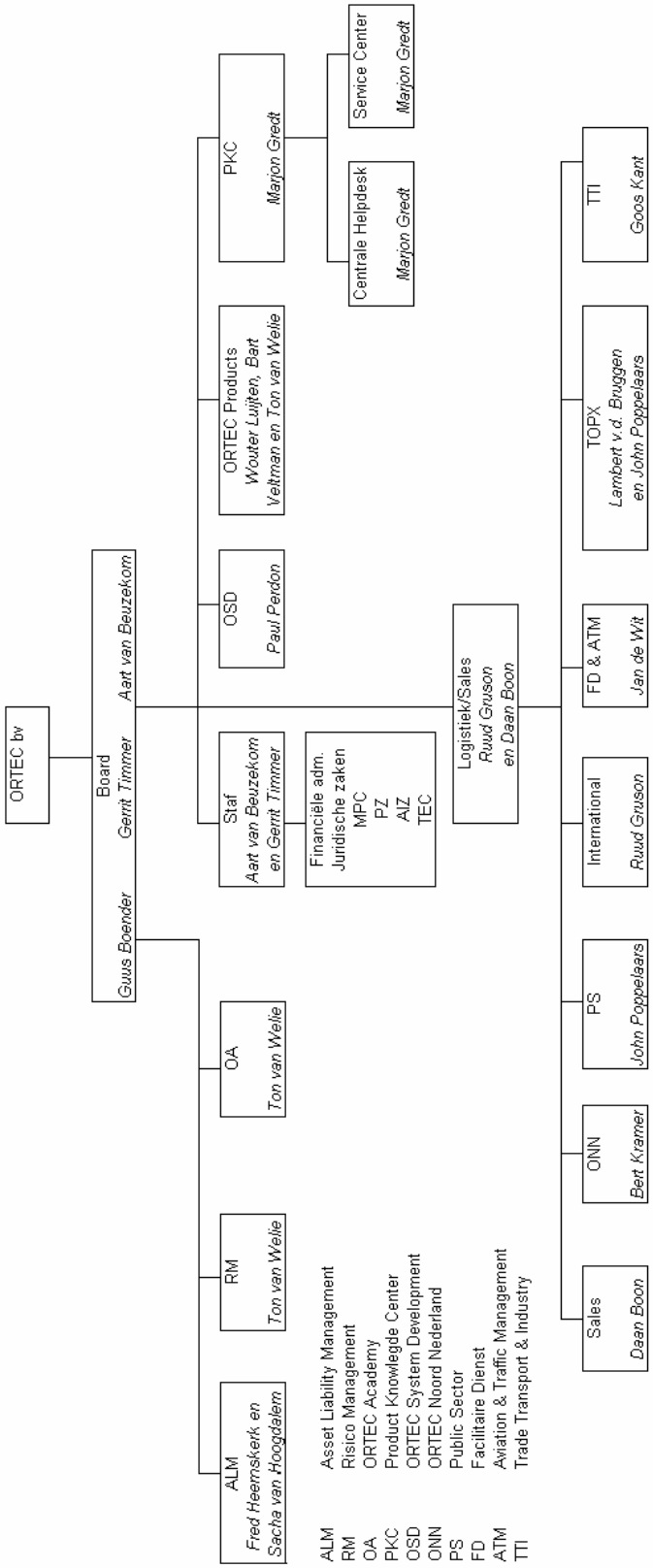
In 2002 is ORTEC, met een gemiddelde jaarlijkse groeit van 30%, uitgegroeid tot een onderneming met ongeveer 300 full time medewerkers en een jaaromzet van 50 miljoen gulden. Het houdt zich bezig met allerlei projecten die in de kern iets te maken hebben met planning, simuleren of optimaliseren. De kernbezigheden omvatten consultancy, software pakketverkoop, maatwerk en opleidingen. De doelgroep waar ORTEC voor werkt is zeer gevarieerd. Zo behoren ziekenhuizen, olieraffinaderijen, vliegvelden, verzekeringsmaatschappijen, pensioenfondsen, overheid, taxicentrales, banken en spoorwegen tot de clientèle.

In de loop der jaren heeft ORTEC een groot aantal softwarepakketten gebouwd die verkocht en onderhouden worden. De belangrijkste producten zijn:

- HARMONY, een planning softwarepakket voor workforce management.
- PLANWISE, ontwikkeld voor productie planning, materiaal management, task scheduling en voorraadbeheer.
- ORION, een planningstool voor order genereren en voorraadbeheer.
- SHORTREC, een routeplanningsysteem voor transporten.
- SHORTROUTE, een fleet track & trace systeem.
- PEARL, analyseren van beleggingsbeslissingen

2. Huidige organisatiestructuur

Het aantal afdelingen is in de tijd ook sterk toegenomen. In 2002 ziet de organisatie er als volgt uit:



De afdeling ALM, wat staat voor Asset liability management, houdt zich bezig met consultancy op het gelijknamige gebied. In opdracht van de Rabobank is het software pakket ALS, Asset Liability System, ontwikkeld. Naast de consultancy verkoopt de afdeling het product en maakt het aanpassingen aan het pakket.

De afdeling RM (Risicomanagement) is in 2000 ontstaan en verkoopt 2 softwarepakketten, WALS en Pearl. WALS is een aangepaste versie van ALS voor woningcorporaties. Pearl is een programma waar beleggingsbeslissingen mee kunnen worden geanalyseerd.

De stafafdelingen zijn PZ (personeelszaken), TEC (technische zaken) en AIZ (Algemene interne zaken).

Vóór 2005 had ORTEC een productgerichte structuur als het gaat om de grote softwarepakketten. Er zijn voor elk product verschillende afdelingen. Zo houdt de afdeling Arbeidstijdsmanagement zich bezig met HARMONY. De afdeling verkoopt het product, lost bugs in het programma op, ontwikkelt het programma en doet de klantservice.
In 2005 is ORTEC overgegaan van een product gerichte structuur naar een taak gerichte structuur. Er zijn nu drie afdelingen OSD, Products en PKC die ieder een deeltaak hebben overgenomen. OSD houdt zich bezig met het ontwikkelen van de softwarepakketten, PKC richt zich op het helpen van de klanten en Products stelt de toekomstige functionaliteit van de producten vast. De oude afdelingen zijn opgesplitst en ondergebracht in de nieuwe afdelingen.

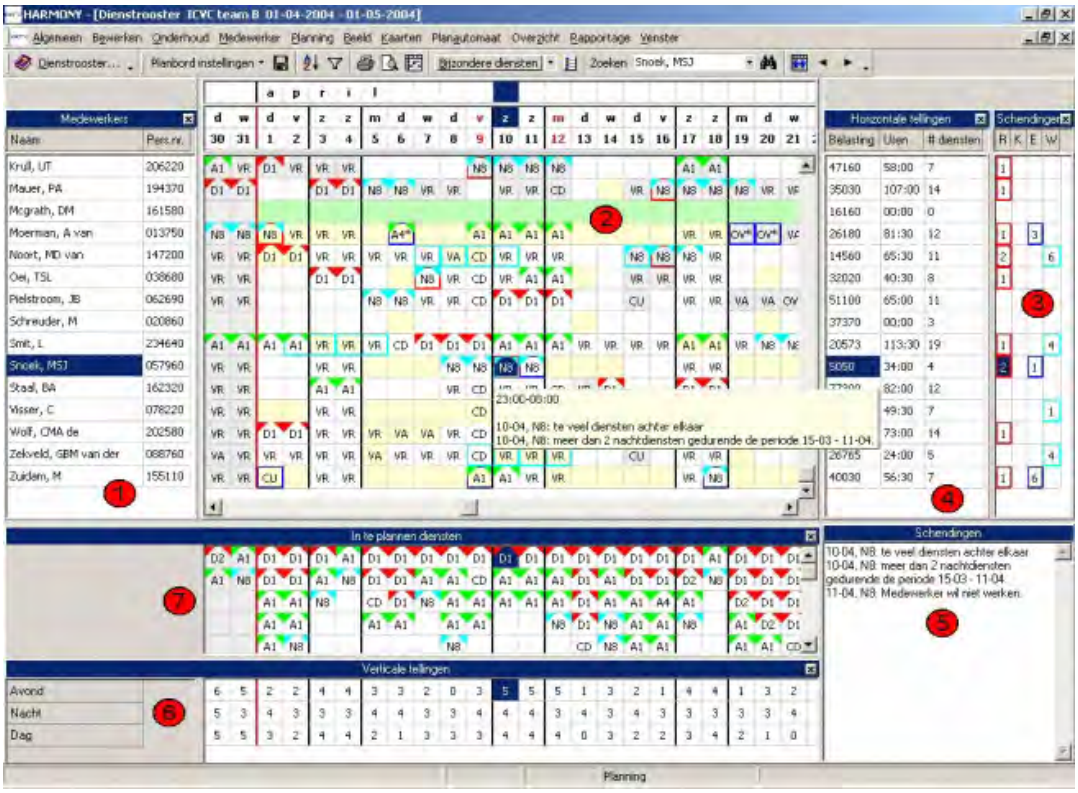
Mijn stage heb ik gedaan in de afdeling OSD. Daar heb ik me bezig gehouden met het product HARMONY waar hieronder meer uitleg over wordt gegeven.

3. Harmony

Onderstaande beschrijving van HARMONY is afkomstig uit 'Dienstreeksen plannen in personeelsroosters' geschreven door Maarten Kroon ([5], Kroon, 2004). Het is geschreven in 2004 als onderdeel van de stage die Maarten Kroon bij ORTEC heeft gelopen.

HARMONY is een geavanceerd planningssysteem voor dienstroostering. HARMONY onderscheidt zich door de mogelijkheid dienstroosters automatisch te genereren. Dit betekent dat een algoritme achter HARMONY de diensten, die ingeroosterd moeten worden, toekent aan de beschikbare medewerkers. Hierbij houdt het systeem rekening met de (complexe) regels uit de arbeidstijdenwet, met individuele wensen van medewerkers en met de kwalificaties die nodig zijn om specifieke diensten uit te voeren. Middels zogenoemde roostercriteria kan de gebruiker de planautomaat van HARMONY sturen. HARMONY zorgt dat er snel ingespeeld kan worden op ad hoc verstoringen in de planning, bijvoorbeeld als gevolg van ziekte of onvoorziene werkzaamheden. Een nauwkeurige registratie en verwerking van gewerkte uren, ziekte en verlof zijn andere speerpunten.

In HARMONY wordt de planning weergegeven in het *planbord.* Dit is een grafische representatie van ingeplande diensten bij medewerkers. Verder is er nog veel meer informatie zichtbaar, welke aan de hand van Figuur 44 kort besproken zal worden.



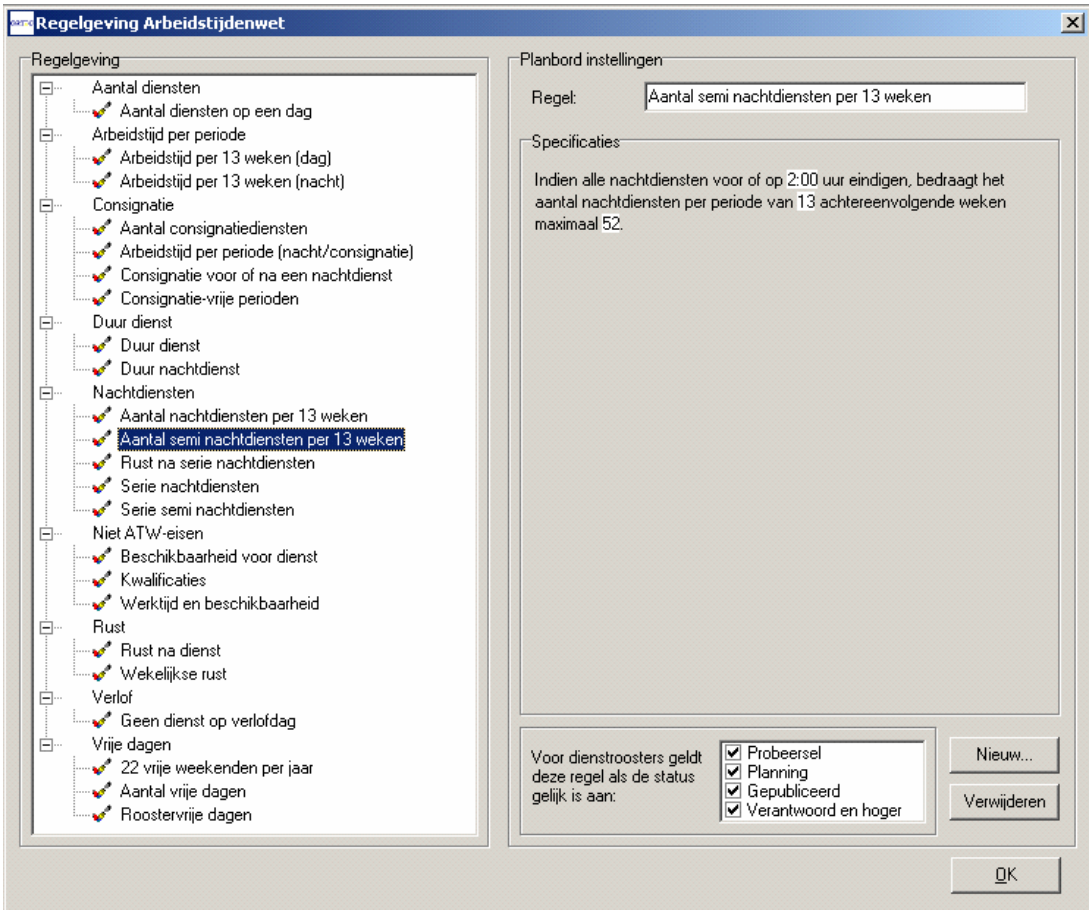
Figuur 44: Planbord HARMONY

In het figuur zijn de verschillende onderdelen van het planbord met een nummer gemarkeerd. Aan de hand van deze nummers worden deze onderdelen besproken.

1. Hier staan de medewerkergegevens. In het planbord heeft elke medewerker een eigen regel, en alle informatie die op die regel staat geldt dan ook voor die medewerker. Voor de medewerkergegevens kan de gebruiker van HARMONY zelf instellen wat daar getoond moet worden. In dit geval staan daar de naam en het personeelsnummer gegeven. Andere medewerkerkenmerken die gebruikt zouden kunnen worden zijn bijvoorbeeld het telefoonnummer, e-mail adres of contract uren.
2. Dit is waar de daadwerkelijke planning staat weergegeven. Elke regel is dus de planning van de bijbehorende medewerker. De diensten die gewerkt moeten worden staan met de dienstnaam aangegeven en er zijn ook nog een aantal kleureffecten om de planning duidelijker te maken. Deze kunnen overigens ook weer door de gebruiker naar believen ingesteld worden. Zo betekent een groene ondergrond dat een medewerker ziek is op die dagen, een gele ondergrond betekent dat de medewerker bepaalde wensen heeft op die dagen en een grijze ondergrond betekent dat de desbetreffende diensten *vastgezet* zijn en dus niet meer veranderd kunnen worden. Ook zijn de kaders van diensten af en toe gekleurd, de betekenis hiervan wordt bij punt 3 besproken.
3. Hier staan de schendingen die bij de planning horen. Er zijn vier type schendingen, namelijk schendingen op basis van *regelgeving*, *kwalificaties*, *eisen* en *wensen*. Hiermee corresponderen de vier kolommen. Ieder type schending heeft zijn eigen kleur en deze zie je dan ook weer terugkomen in de omkadering van de diensten waarop de schending betrekking heeft.
4. Bij de horizontale tellingen kunnen tellingen ingesteld worden die de gebruiker in staat stellen snel te kunnen zien of de planning een goede planning is of niet. Zo kunnen bijvoorbeeld de belasting, het aantal uren en het aantal diensten per medewerker snel gezien worden.
5. Als er sprake is van een schending dan kan in het planbord (2) de dienst die de schending veroorzaakt geselecteerd worden. Hier komt dan enige informatie te staan over de oorzaak van de schending.

6. Naast horizontale tellingen bevat HARMONY ook verticale tellingen. Hier worden bepaalde type diensten verticaal geteld, er kan dan bijvoorbeeld snel gezien worden hoeveel nachtdiensten er op een bepaalde dag zijn ingepland.
7. Dit zijn de nog *in te plannen diensten*. De diensten die hier staan zijn nog niet aan een medewerker toegewezen en zouden in principe nog zoveel mogelijk ingepland moeten worden.

Elke medewerker is gekoppeld aan een verzameling *arbeidsvoorwaarden*. Hierin staat informatie over de bezoldiging en de regelgeving voor deze medewerker. In Figuur 45 staat een voorbeeld van wat er zoal in de regelgeving kan staan. Verderop zal daar dieper op in gegaan worden.



Figuur 45: Regelgeving HARMONY

Planautomaten

HARMONY stelt de gebruiker een aantal planautomaten voor het genereren van dienstroosters ter beschikking:

- Inplannen dienstreeksen
- Inplannen open diensten
- Rooster genereren

De drie genoemde planautomaten kunnen apart aangeroepen worden, maar zowel 'Inplannen dienstreeksen' als 'Inplannen open diensten' zijn niet bedoeld om een volledig rooster mee te genereren. Hiervoor is (zoals de naam al aangeeft) 'Rooster genereren' bedoeld. Rooster genereren werkt in twee fases.

In de eerste fase wordt een genetisch algoritme uitgevoerd. De functie hiervan is vooral het vinden van globaal optimale roosters. Dit algoritme maakt gebruik van zowel 'inplannen dienstreeksen' als 'inplannen open diensten'. Het genetisch algoritme werkt met een aantal roosters parallel. Voor meer informatie over de werking van een genetisch algoritme verwijs ik naar Bijlage C: 'Basisprincipes genetisch algoritme'.

De gebruiker kan instellen hoeveel parallelle roosters er gebruikt worden; dit heet de *omvang* van een generatie. Over het algemeen is het zo dat hoe groter de omvang, hoe beter het eindresultaat. Echter de rekentijd neemt wel toe: een generatie van omvang 10 zal per generatie 5 keer zoveel tijd nemen als wanneer de generatie een omvang van 2 heeft. Naast de omvang is ook de maximale rekentijd instelbaar. Als deze tijd ingesteld wordt, kan de omvang van de generaties gedurende het proces afnemen. Dit gebeurt op grond van een schatting van het aantal benodigde generaties: als de automaat denkt dit aantal niet te halen, wordt de omvang verlaagd. Als het genetisch algoritme ontdekt dat er geen verbetering te vinden is, schakelt de automaat over op de tweede fase. Veelal zijn er dan 40 of (veel) meer generaties geweest.

De tweede fase is een verbeter fase en is vooral gericht op het lokaal optimaliseren van de oplossing. Het beste in de eerste fase gevonden rooster wordt grondig onderzocht op mogelijke verbeteringen. Als in een slag een verbetering wordt gevonden, wordt hetzelfde nog eens uitgevoerd. Deze fase kan daardoor lang duren, ook in verhouding tot de eerste fase. De verbeteringen die zo gevonden worden, kunnen echter aanzienlijk zijn.

Regels & criteria

Het inrichten van de gegevens voor de planautomaten is geen eenvoudige zaak. De instellingen beïnvloeden het resultaat van de planning, en het aanpassen van de instellingen kan de resultaten aanzienlijk verbeteren. De filosofie van HARMONY is dat alles mag en alles goed is, zolang niet expliciet aangegeven is dat dit niet de bedoeling is. Dit betekent dat er veel ingesteld moet worden om tot een goed rooster te komen: alle intelligentie de een planner/gebruiker heeft, moet ingebracht worden in HARMONY. Deze instellingen betreffen de *regels* en de *criteria*.

Definitie Alles wat zegt dat iets wel of niet mag (door de planautomaat) noemen we een *regel*. De planautomaat mag geen rooster genereren waarin de regels overtreden worden. Regels zijn dus zogenaamde harde voorwaarden, voorwaarden waar altijd aan voldaan moet worden.

Definitie De *criteria* bepalen wat wel of niet mag, maar nu is de overtreding echter toegestaan, op straffe van een belasting. Criteria zijn daarom zachte voorwaarden, omdat hier niet altijd aan voldaan hoeft te worden.

Voor zowel de regels als de criteria geldt dat ze enerzijds op groepsniveau kunnen worden gebruikt en anderzijds op persoonlijk niveau. Een regel of criterium kan dus voor een hele roostergroep gelden, maar ook voor één bepaalde medewerker. Binnen HARMONY kunnen aan medewerkers ook wensen toegekend worden. Deze worden opgevat als criteria. Ze mogen dus wel overtreden worden, maar als dit voor komt dan wordt er een belasting opgelegd. Er is ook de mogelijkheid om een wens als eis in te voeren en in dat geval wordt het opgevat als een persoonlijke regel, welke dus niet overtreden mag worden.

De doelstelling van alle planautomaten is om zoveel mogelijk diensten in plannen. Bij dit inplannen moet rekening gehouden worden met alle ingestelde regels; deze regels mogen niet overtreden worden. Daarnaast dient er zoveel mogelijk aan de criteria voldaan te worden. De criteria mogen wel overtreden worden, maar als dit gebeurd, wordt hierop een belasting gelegd. De automaat probeert dus zoveel mogelijk diensten in te plannen, zonder de regels te overtreden, en daarbij de belasting zo laag mogelijk te houden.

Belasting

Per criterium wordt een belasting opgelegd op elke overtreding hiervan. Per overtreding worden eerst de basiskosten van de overtreding bepaald, en om de belasting te verkrijgen wordt dit getal vermenigvuldigd met het gewicht van het criterium. De basiskosten zullen groter worden naarmate

de afwijking van het criterium groter is. Voor wel/niet criteria[4] is dit niet van toepassing, maar bijvoorbeeld wel als het gaat om criteria die te maken hebben met de lengte van een serie diensten. In dat geval worden twee afwijkingen met lengte 1 lager belast dan één afwijking van lengte 2.

Het gewicht van een criterium kan ingesteld worden met waardes van 0 tot 10000. Het criterium wordt niet in beschouwing genomen indien het gewicht op 0 is ingesteld. Aangezien de planautomaat de belasting zo laag mogelijk probeert te houden zullen bijvoorbeeld 16 overtredingen van criterium A met gewicht 10 door de planautomaat ingeruild worden tegen één overtreding van criterium B met een gewicht van 150. Hieruit blijkt dat het gewicht een grote invloed kan hebben op het resultaat van de planautomaat.

Voorbeelden regels

Op dit moment bestaan er binnen HARMONY 59 regels. Om enig inzicht te krijgen in wat regels zijn, volgen er enkele voorbeelden van regels die binnen HARMONY gebruikt kunnen worden. Hierbij zijn de cursief en onderstreept gedrukte waardes instelbaar door de gebruiker.

Aantal diensten in een periode

Het maximaal aantal diensten per periode van *13* weken is *25*. Hierbij loopt een week van maandag tot en met zondag.

Definitie nachtdienst

Een dienst telt als nachtdienst als een deel van de dienst tussen *0:00* en *6:00* valt.

Aantal nachtdiensten

Indien sommige nachtdiensten na *2:00* uur eindigen, bedraagt het aantal nachtdiensten per periode van *13* achtereenvolgende weken maximaal *25*. Een week loopt hierbij van maandag tot en met zondag.

Aantal vrije weekenden

Het aantal vrije weekenden van *60* uur is per *4* weken minimaal *2*. Een weekend omvat hierbij de periode van *zaterdag 0:00* tot *maandag 4:00* uur. Het weekend mag dus niet op of voor zondag eindigen. Een week loopt hierbij van woensdag tot en met dinsdag.

Dagelijkse rust

De minimale onafgebroken rusttijd bedraagt *11* aaneengesloten uren in een periode van 24 achtereenvolgende uren. De minimale rusttijd mag éénmaal in een periode van *7* maal 24 achtereenvolgende uren worden ingekort tot tenminste *8* uur.

Maximaal aantal uren per dag

De maximum arbeidstijd per dag bedraagt *12* uur.

Rust na serie nachtdiensten

Na het beëindigen van een reeks van minimaal *3* achtereenvolgende nachtdiensten moet een onafgebroken rust van tenminste *48* uur worden genoten.

Dit is maar een kleine greep uit de verzameling van regels die HARMONY rijk is. Sommige regels zijn erg eenvoudig, zoals het maximale aantal uren per dag, maar anderen zitten een stuk gecompliceerder in elkaar. Zie onder meer het aantal vrije weekenden.

Voorbeelden criteria

Veel criteria hebben te maken met series, hierbij kan gedacht worden aan:

[4] Dit zijn criteria waar wel of niet aan voldaan kan worden. Er is dus geen mate van overtreding, er is slechts sprake van wel of geen overtreding.

- Serie lengte
  Bijvoorbeeld: Hoeveel diensten minimaal/maximaal na elkaar?

- Serie spreiding
  Bijvoorbeeld: Hoeveel diensten per periode van een aantal weken?
  Hoeveel dagen zonder dienst na een serie diensten?

- Serie opbouw
  Bijvoorbeeld: Welke diensten mogen na elkaar plaatsvinden?
  Welke diensten zijn geoorloofd?

Om de instelling van criteria te vereenvoudigen, eindigt elk criterium met de volgende instelling:

> Dit criterium is geldig voor medewerkers met een beschikbaarheid
> van minimaal *0* uur en maximaal *48* uur per week.

Hierbij zijn de cursief en onderstreept gedrukte waardes wederom parameters die door gebruiker ingesteld kunnen worden. Op deze manier kunnen criteria voor parttimers anders ingesteld worden. Zo is het voor de gebruiker mogelijk om algemene criteria toch niet voor alle medewerkers te laten gelden. In het geval van een wens die als eis ingesteld wordt, komt deze instelling uiteraard niet tevoorschijn, omdat deze toch al persoonsgebonden zijn.

Enkele criteria zoals deze in HARMONY gebruikt kunnen worden zijn:

Aantal dagen rust na dienst

De rust na een serie diensten die eindigt met een dienst uit de dienstenverzameling *Diensten nachtploeg* met de naam *A, L, N*, bedraagt minimaal *3* dagen.

Het kostentype van dit criterium is kwadratisch. Dit wil zeggen dat de basiskosten kwadratisch oplopen naarmate het aantal dagen rust verder van het minimum afzit.

Maximaal verschil in begintijden

Een opvolgende dienst begint maximaal *0:30* uur eerder en maximaal *0:30* uur later.

Minimaal/Maximaal aantal diensten

Instellingen voor het totaal aantal diensten van de dienstenverzameling *Diensten nachtploeg* met de namen *A, L, N* in een periode van *1* week:

Minimaal: *2*
Maximaal: *5*
Gewenst: *4*

Dit criterium geldt voor diensten die beginnen tussen *maandag* *0:00* uur en *zaterdag* *0:00* uur.

Hierbij kan ook aangevinkt worden of het criterium geldig is voor alle dagen van de week. Het kostentype is wederom kwadratisch en een week loopt van maandag tot en met zondag.

Voorbeelden wensen

De volgende type wensen zijn beschikbaar binnen HARMONY. De cursief en onderstreept gedrukte velden zijn instelbaar door de gebruiker.

Afwezigheidsvorm

Medewerker *Jansen* met arbeidsvoorwaardengroep *AVWG* en personeelsnummer 14051980 is afwezig van *11-11-2004* *0:00* tot *18-11-2004* *0:00*.

Hierbij kan ook nog een toelichting meegegeven worden.

Dienstwens

Medewerker *Jansen* wenst op *20-11-2004* *wel* de volgende dienst: *A*

Deze wens wordt gebruikt om aan te geven of een medewerker een bepaalde dienst wel of niet wil op een bepaalde dag.

Terugkerende wens

Medewerker *Jansen* heeft gedurende *15:00* tot *20:00* uur (Duur in uren: 5) als wens: *Niet werken*.

Het repeterende patroon van deze wens (wel of niet werken) kan als volgt ingesteld worden:

- Er kan gekozen worden voor een (meer) maandelijks patroon, of anders voor een patroon dat een geheeltallig aantal weken duurt.
- In geval van een wekelijks patroon kunnen de dagen van de week waarop de wens betrekking heeft worden opgegeven.
- In geval van een maandelijks patroon kan gekozen worden op welke dagen van de maand de wens betrekking heeft.
- Er moet ook een startdatum voor de wens gegeven worden en een einddatum is optioneel.

Er kan ook nog een toelichting meegegeven worden. Als gekozen is voor *Wel werken* dan kan deze wens niet als eis ingesteld worden.

Verlofwens

Medewerker *Jansen* wenst van *25-11-2004* tot *05-12-2004* verlof van het type *Vakantie*.

Wel/niet werken in een periode

Medewerker *Jansen* heeft in de periode van *07-12-2004* *7:00* uur tot *09-12-2004* *22:00* uur als wens: *Niet werken*

Wens met omschrijving

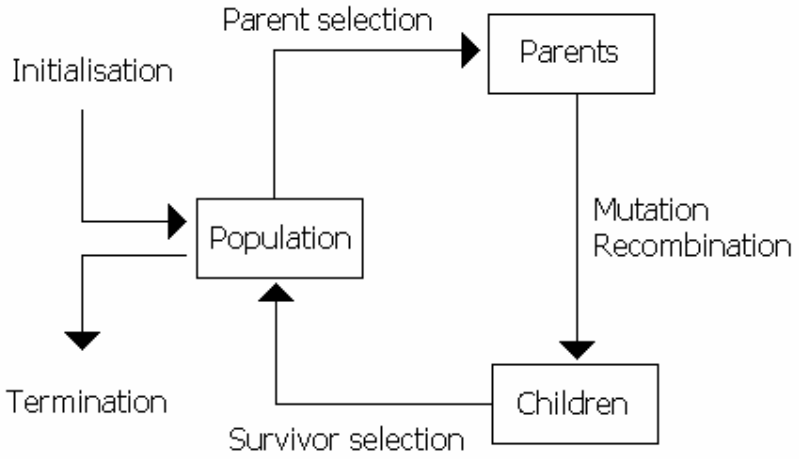
Medewerker *Jansen* heeft in de periode van *15-12-2004* *7:00* uur tot *15-12-2004* *14:00* uur de volgende wens: *Samenwerken met De Vries*

Bijlage C. Basisprincipes genetisch algoritme

In deze bijlage worden de basisprincipes van een genetisch algoritme uitgelegd. Voor meer details over het genetisch algoritme verwijs ik naar ([3], Eiben, 2003).

Het genetisch algoritme vindt zijn oorsprong in de evolutieleer van Darwin. Hij beschreef de evolutie op de volgende manier. In een omgeving bevinden zich een aantal individuen. De omgeving bevat een beperkt aantal grondstoffen waarvan de individuen moeten leven. De individuen sterven na een tijd. Om als een soort te blijven bestaan planten de individuen zich voortdurend voort. Hierdoor zullen de individuen met elkaar moeten strijden om de grondstoffen om te kunnen voortbestaan. Die individuen die dit het beste doen zullen overleven, terwijl de andere individuen zullen sterven. Dit fenomeen wordt ook wel 'survival of the fittest' genoemd. Tijdens het voortplanten ontstaan telkens nieuwe individuen die verschillen van hun ouders. Door recombinatie krijgen de kinderen eigenschappen mee van de ouders, maar ook veranderen de kinderen los van hun ouders door mutaties. Door deze veranderingen en de voortdurende selectie van beter geschikte individuen zullen generatie op generatie individuen overleven die beter kunnen strijden voor de grondstoffen.

Deze ideeën hebben toepassing gevonden bij het genetisch algoritme. Het principe van 'survival of the fittest' is de drijfveer voor deze methode. De omgeving is het probleemdomein. De randvoorwaarden van het probleem vormen de grondstoffen waar de individuen om moeten strijden. De individuen zijn oplossingen binnen het probleem. Elke generatie vindt er een selectie plaats tussen de individuen om te kijken wie kinderen mogen produceren, waarna alle individuen zullen moeten strijden om te mogen overleven. Dit gebeurt door aan elke oplossing een waarde mee te geven, ook wel fitness genoemd. Deze waarde geeft aan in hoeverre een individu een oplossing geeft voor het probleem. Door 'survival of the fittest' moeten uiteindelijk de beste oplossingen voor het probleem overblijven. Dit proces herhaalt zich net zo lang tot er een bevredigende oplossing is gevonden, of als er een maximaal aantal generaties is doorlopen. Schematisch ziet een algemeen genetisch algoritme er als volgt uit:



Figuur 46: Schematisch genetisch algoritme

Een genetisch algoritme bestaat uit een vijftal componenten. Dit zijn de initialisation, parent selection, recombination, survivor selection en de stopcondition.

Initialisatie

Om het algoritme te starten moeten er een aantal initiële oplossingen worden gegenereerd. Deze oplossingen vormen de eerste generatie.

Parent selection

Uit de population worden individuen geselecteerd die gezamenlijk children maken. De keuze welke individuen dit mogen doen wordt gebaseerd op de fitness. Hierdoor hebben de beste individuen de meeste kans om children te maken, die daardoor de goede eigenschappen kunnen overnemen.

Mutatie

Elk individu kan zelf veranderd worden. Door deze mutatie ontstaat een nieuwe oplossing.

Recombination

Meerdere individuen kunnen met elkaar gecombineerd worden om nieuwe oplossingen te creëren.

Survivor selection

Nadat de children zijn gemaakt wordt bepaald welke individuen de generatie mogen overleven. De volgende generatie kan gevormd worden door alleen de children, maar ook de individuen uit de huidige generatie kunnen vaak meedingen voor een plek in de volgende generatie. Dit selectieproces gebeurt opnieuw op basis van fitness. De beste individuen hebben de meeste kans te overleven. Ook wordt vaak het verschil tussen de individuen meegenomen in de afweging om ervoor te zorgen dat de individuen niet teveel op elkaar gaan lijken.

Stopcondition

Uiteindelijk stopt het algoritme als een oplossing wordt gevonden die goed genoeg wordt geacht of als een maximaal aantal generaties is doorlopen.

Door het toepassen van evolutieleer is het genetisch algoritme in staat snel goede oplossingen te vinden voor grote problemen waarvan het probleemdomein nauwelijks bekend is. Met zulke problemen hebben heuristieken vaak moeite om te komen tot goede oplossingen. Het genetisch algoritme kan echter niet garanderen dat een gevonden oplossing ook de optimale oplossing is.

Bijlage D. Testcase

Deze bijlage bespreekt de testset waarmee de verschillende methoden met elkaar worden vergeleken. Het bevat de noodzakelijke gegevens over medewerkers, ingestelde regels en criteria en de relevante historie voorafgaande aan de roosterperiode.

De gebruikte testcase heeft veel weg de testcase zoals die ook gebruikt is in ([2] Van der Put, 2005). Op twee punten is die testcase anders. In de eerste plaats is de historie van die testcase niet overgenomen. Dit heeft als reden dat de historie die in ([2] Van der Put, 2005) is gebruikt niet in de scriptie staat beschreven. Daarnaast is voor medewerker A en E de eis toegevoegd om niet te werken op de eerste dag van de maand. Dit is gedaan om de testcase wat moeilijker te maken.

De testcase bevat een afdeling bestaande uit 16 medewerkers genaamd A tot P. Gedurende de maand mei 2004 moeten 4 verschillende soorten diensten worden gedraaid. Alle medewerkers hebben de benodigde kwalificaties om al deze diensten uit te voeren. De details van de diensten en de bijbehorende bezettingseisen voor deze periode zijn te vinden in onderstaande tabel.

Dienstnaam	D (Dag dienst)	E (Vroege dienst)	L (Late dienst)	N (Nacht dienst)
Werktijden	08:00-17:00	07:00-16:00	14:00-23:00	23:00-07:00
Werk	08:00-12:00	07:00-11:00	14:00-18:00	23:00-07:00
Pauze	12:00-13:00	11:00-12:00	18:00-19:00	
Werk	13:00-17:00	12:00-16:00	19:00-23:00	
Bezettingseis op weekdag	3	3	3	1
Bezettingseis op weekenddag	2	2	2	1

Medewerker	Contracturen per week
A	36
B	36
C	20
D	36
E	36
F	36
G	36
H	20
I	20
J	36
K	36
L	36
M	36
N	36
O	32
P	36

1. Regels

De regels die zijn ingesteld voor de testcase zijn hieronder per categorie vermeldt.

Niet Arbeidstijdenwet-eisen

Beschikbaarheid voor een dienst
Een medewerker moet beschikbaar zijn voor een dienst.

Kwalificaties
Om een dienst te mogen uitvoeren dient een medewerker te beschikken over de vereiste kwalificaties.

Werkuren en beschikbaarheid
Een medewerker mag in een roosterperiode niet meer dan 4 uren werken boven het aantal uren dat hij of zij beschikbaar is voor de betreffende roostergroep.

Vrije dagen

Aantal vrije weekenden
Minimaal 2 vrije weekenden van 60 uren per 5 weken, omvattende zaterdag 0:00 tot maandag 4:00 uur. Ook geldig voor bereikbaarheidsdiensten. Ook geldig voor diensten van het soort Consignatie, Aanwezigheid, Piket.

Arbeidstijd per periode

Arbeidstijd per periode (dag)
De maximale arbeidstijd per week bedraagt gemiddeld 36 uur in een periode van 13 achtereenvolgende weken.

Nachtdiensten

Nachtdienst series
Indien sommige nachtdiensten na 2:00 uur eindigen, bedraagt het aantal achtereenvolgende nachtdiensten maximaal 3.

Aantal nachtdiensten
Indien sommige nachtdiensten na 2:00 uur eindigen, bedraagt het aantal nachtdiensten per periode van 5 achtereenvolgende weken maximaal 3.

Rust na serie nachtdiensten
Na het beëindigen van een reeks van minimaal 1 achtereenvolgende nachtdienst moet een onafgebroken rust van tenminste 42 uur worden genoten.

Aantal diensten

Aantal diensten op een dag
Het maximaal aantal diensten met arbeid dat op een dag begint is 1.

Rust

Rust na dienst, twee uitzonderingen
De minimale onafgebroken rusttijd bedraagt 11 uur in een periode van 24 achtereenvolgende uren.
De minimale onafgebroken rusttijd na afloop van een nachtdienst die eindigt na 2:00 uur bedraagt 14 uur.
De minimale rusttijd mag eenmaal in een periode van 21 maal 24 achtereenvolgende uren worden ingekort tot tenminste 8 uur en mag in die periode nog een maal worden verkort tot 10 uur.

Series

Dienst Series
Het aantal achtereenvolgende diensten is maximaal 6.

2. Criteria

De criteria die zijn ingesteld voor de testcase zijn hieronder per categorie vermeldt.

Minimaal en maximaal aantal diensten

Min/Max aantal diensten (crit)

Instellingen voor het totaal aantal diensten in een periode van 5 weken:

- Minimaal: 0
- Gewenst: 2
- Maximaal: 3

❖ Dit criterium is geldig voor alle dagen van de week.
❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen N.
❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
❖ Gewicht = 1000

Totaal aantal diensten (crit)

Instellingen voor het totaal aantal diensten in een periode van 1 weken:

- Minimaal: 4
- Gewenst: 5
- Maximaal: 5

❖ Dit criterium is geldig voor alle dagen van de week.
❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen D, E, L, N.
❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 30 en niet meer dan 48 uur per week.
❖ Gewicht = 10.

Totaal aantal diensten (crit)

Instellingen voor het totaal aantal diensten in een periode van 1 weken:

- Minimaal: 2
- Gewenst: 2
- Maximaal: 3

❖ Dit criterium is geldig voor alle dagen van de week.
❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen D, E, L, N.
❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 30 uur per week.
❖ Gewicht = 10.

Geen dienst of anders minimum aantal diensten

Compleet Weekend (crit)

Voor diensten zoals hieronder geselecteerd, beginnend tussen vrijdag 22 uur en maandag 0 uur dient te gelden: of geen diensten, of tenminste 2 diensten.

❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen D, E, L, N.
❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
❖ Gewicht = 1000.

Rust

Aantal dagen rust na een dienst (crit)

De rust na een serie diensten die eindigt met een dienst zoals hieronder geselecteerd, bedraagt ten minste 2 dagen.

- ❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de diensten verzameling DNSTVRZ_Test met de namen D, E, L.
- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
- ❖ Gewicht = 100.

Aantal dagen rust na een nachtdienst (crit)

De rust na een serie diensten die eindigt met een dienst zoals hieronder geselecteerd, bedraagt tenminste 1 dag.

- ❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen N.
- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
- ❖ Gewicht = 100.

Serielengte

Geen losse diensten

Vermijd het inplannen van losse diensten met arbeid.

- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
- ❖ Gewicht = 1000.

Lengte van dienstseries

Instellingen voor de lengten van series diensten:

- Minimaal: 2
- Gewenst: 2
- Maximaal: 5

- ❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen N.
- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
- ❖ Gewicht = 1000.

Lengte van dienstseries (crit)

Instellingen voor de lengte van series diensten

- Minimaal: 2
- Gewenst: 3
- Maximaal: 3

- ❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen E.
- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
- ❖ Gewicht = 10.

Lengte van dienstseries (crit)

Instellingen voor de lengte van series diensten:

- Minimaal: 2
- Gewenst: 2
- Maximaal: 3

- ❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen D, E, L, N.
- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 30 uur per week.

- ❖ Gewicht = 10.

Lengte van dienstseries (crit)

Instellingen voor de lengte van series diensten:

- Minimaal: 4
- Gewenst: 5
- Maximaal: 6

- ❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen D, E, L, N.
- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 30 en niet meer dan 48 per week.
- ❖ Gewicht = 10.

Lengte van dienstseries (crit)

Instellingen voor de lengte van series diensten:

- Minimaal: 2
- Gewenst: 2
- Maximaal: 3

- ❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen L.
- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
- ❖ Gewicht = 10.

Lengte van nachtdienstseries (crit)

Instellingen voor de lengte van series diensten:

- Minimaal: 2
- Gewenst: 2
- Maximaal: 3

- ❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen N.
- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 30 uur per week.
- ❖ Gewicht = 1000

Serieopbouw

Geen vroege dienst na een nachtdienst: Na een dienst met naam E dienen geen van de hieronder geselecteerde diensten ingepland te worden.

- ❖ Selecteer diensten op basis van naam: dit criterium geldt voor diensten uit de dienstenverzameling DNSTVRZ_Test met de namen N.
- ❖ Dit criterium is geldig voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
- ❖ Gewicht = 1.

Maximaal verschil in begintijden (crit.)

Een opvolgende dienst begint maximaal 0:30 eerder en maximaal 0:00 later.

- ❖ Dit criterium geldt voor medewerkers met een beschikbaarheid van ten minste 0 en niet meer dan 48 uur per week.
- ❖ Gewicht = 5.

Wel/niet werken in een periode

Medewerker *Aap* hebben in de periode van *01-04-2004 0:00* uur tot *01-05-2004 23:59* uur als wens: *Niet werken*

- ❖ Gewicht = eis.

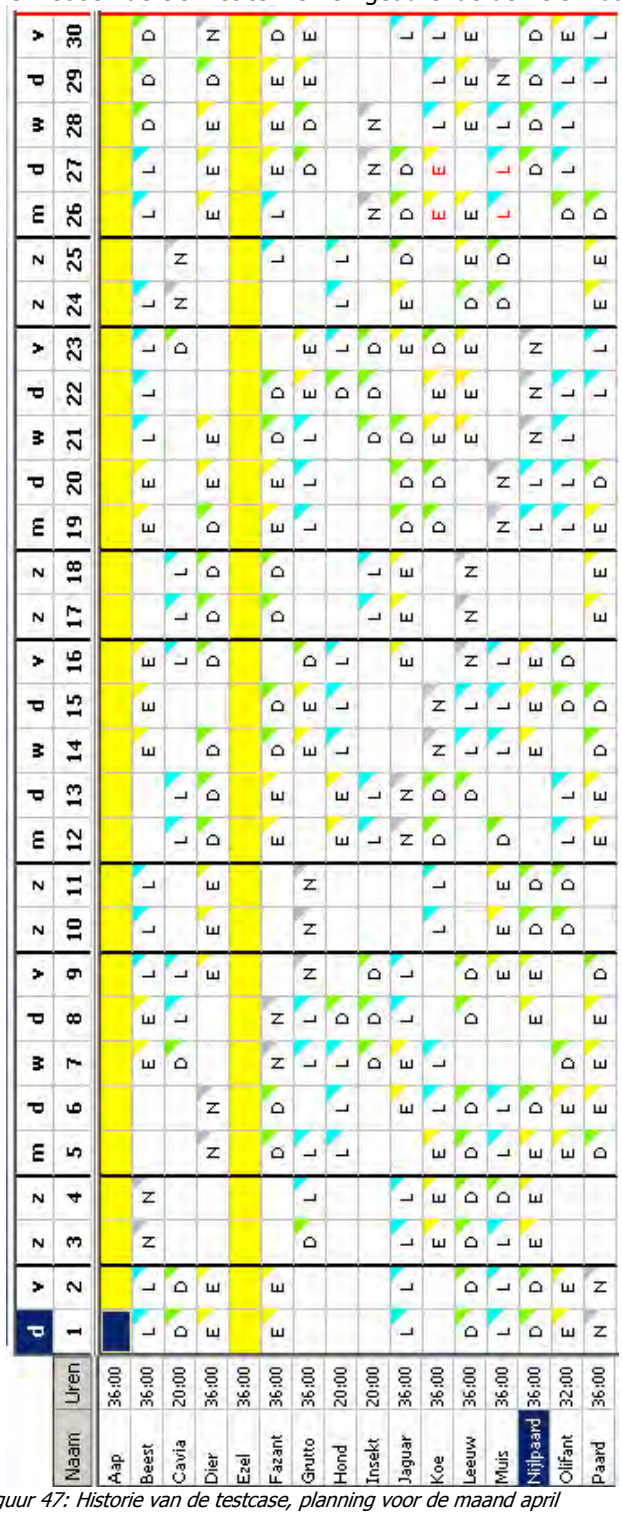
Wel/niet werken in een periode

Medewerker *Ezel* hebben in de periode van *01-04-2004 7:00* uur tot *01-05-2004 20:00* uur als wens: *Niet werken*

- Gewicht = eis.

3. Historie

Medewerker Aap en Ezel hebben de eis niet te werken gedurende de hele maand.



Figuur 47: Historie van de testcase, planning voor de maand april