Crew Rostering: Een Rooster Probleem voor de Rondvaart

Wouter Radder

BWI Werkstuk

Crew Rostering: Een Rooster Probleem voor de Rondvaart

Wouter Radder

BWI Werkstuk

Vrije Universiteit
Faculteit der Exacte Wetenschappen
Studie Bedrijfswiskunde en Informatica
De Boelelaan 1081a
1081 HV Amsterdam

Februari 2008

Inhoudsopgave

Inleiding

Het maken van dienstroosters is een probleem waar veel organisaties mee te maken hebben. Denk bijvoorbeeld aan een rooster voor buschauffeurs, vliegtuigpiloten of verpleegkundigen in een ziekenhuis. Het maken van dergelijke roosters, en alles wat daarbij te kijken komt, wordt in de wetenschap aangeduid als het *Crew Management Problem* (CMP), en is een complex probleem. Daarom wordt het doorgaans opgesplitst in twee subproblemen: het *Crew Scheduling Problem* (CSP) en het *Crew Rostering Problem* (CRP). Bij het CSP draait het om het vinden van 'optimale' diensten voor een bepaalde periode. Bij het CRP worden de uitvoerende medewerkers op een zo goed mogelijke wijze aan deze diensten toegewezen.
In dit werkstuk concentreren we ons op het CRP. We gaan op zoek naar een oplossingsmethode voor een specifiek CRP uit de praktijk. Het betreft een rondvaart organisatie die haar schippers moet inroosteren op de verschillende vaarten. We zullen dan ook naar dit probleem refereren als het *rondvaart probleem*.

Het Crew Rostering Problem

Bij een CRP draait het om het toekennen van medewerkers aan uit te voeren diensten. Hierbij moet rekening gehouden worden met een aantal 'harde condities', waar de oplossing per se aan moet voldoen. Zo kan een medewerker bijvoorbeeld, op basis van zijn vaardigheden, niet altijd voor alle diensten ingezet worden. Hiernaast zijn er doorgaans ook 'zachte condities', aan welke bij voorkeur voldaan dient te worden. Deze hebben bijvoorbeeld betrekking op een eerlijke verdeling van het werk. Bij het maken van een zo goed mogelijk rooster is het dus zaak om al deze condities in beschouwing te nemen.
Welke condities er spelen, en hoe belangrijk ze zijn, varieert per situatie en wordt onder andere beïnvloed door de aard van het werk en de omstandigheden binnen de specifieke organisatie.
De hoeveelheid en vorm van deze condities bepaalt in grote mate de complexiteit van het onderliggende probleem. Dit maakt het lastig om een generiek bruikbare oplossingsmethode voor het CRP te vinden. In de loop der jaren is er vanuit verschillende vakgebieden onderzoek gedaan naar het CRP, met name vanuit de *Operations Research* (OR) en de *Artificial Intelligence* (AI). In dit werkstuk wordt een oplossingsmethode uit beide categorieën besproken. Er wordt geanalyseerd wat de sterke en zwakke punten van deze technieken zijn, in het bijzonder, in relatie tot het rondvaart probleem.

Het Rondvaart Probleem

De organisatie die we beschouwen is een vrijwilligersorganisatie, die toeristische rondvaarten organiseert over een aantal verschillende (vaste) routes. Op iedere boot zijn twee medewerkers aanwezig: één schipper en één spreker. Er wordt gevaren volgens een vast vaarschema, waarin alle vaarten voor een vaarseizoen zijn vastgelegd. Sommige vaarten uit het vaarschema hebben een specifiek thema; niet alle medewerkers zijn geschikt als spreker op deze vaarten. Naast de schippers op de vaarten is er ook een walschipper en een stand-by dienst. Net als het vaarschema worden deze diensten vooraf vastgelegd. De organisatie is op zoek naar een geautomatiseerde oplossing voor het inroosteren van de schippers over de verschillende vaarten en diensten.

Doel van dit Werkstuk

Het doel is het vinden van een oplossingsmethode voor het CRP, dat geschikt is voor het rondvaart probleem. Hiervoor zullen een aantal technieken besproken en vergeleken worden.

Structuur van het werkstuk

Dit werkstuk bestaat uit 5 hoofdstukken. In hoofdstuk 1 wordt het probleem beschreven, hierbij wordt zowel aandacht besteed aan de algemene kenmerken van het CRP, als aan de

specifieke eigenschappen van het rondvaart probleem. In hoofdstuk 2 en 3 kijken we naar een aantal verschillende oplossingsmethoden. Aan de hand van voorbeelden wordt bestudeerd hoe deze methoden toegepast kunnen worden op het CRP. Aan het einde van hoofdstuk 3 wordt, op basis van de sterke en zwakke eigenschappen van de besproken oplossingsmethoden, een oplossingsmethode gekozen voor het rondvaart probleem.
In hoofdstuk 4 werken we de gekozen oplossingsmethode uit, en kijken we vooruit naar de ontwikkelingen die op korte termijn plaats zullen vinden.

Hoofdstuk 1: Probleem Beschrijving

In dit hoofdstuk gaan we nader in op het Crew Rostering Problem. In paragraaf 1.1 wordt het probleem op een hoog niveau gemodelleerd. In paragraaf 1.2 wordt er gekeken naar verschillende kwaliteitseigenschappen van roosters. Vervolgens analyseren we in paragraaf 1.3 welke harde condities doorgaans een rol spelen. In paragraaf 1.4 beschrijven we de specifieke eigenschappen van het CRP bij onze rondvaart organisatie, en in paragraaf 1.5 wordt kort aandacht besteed aan de verschillende praktische toepassingen van het CRP. Tenslotte beschrijven we in paragraaf 1.6 hoe we te werk gaan bij het zoeken naar een oplossingsmethode voor het rondvaart probleem.

1.1 Een Formulering voor het Crew Rostering Problem

In deze paragraaf stellen we op hoog niveau, een algemeen bruikbaar model op voor het CRP. We hanteren de volgende definitie voor het CRP:

Definitie 1: Crew Rostering Problem
Het Crew Rostering Problem betreft de problematiek van het optimaal toewijzen van medewerkers aan vastgelegde, uit te voeren diensten over een gegeven periode, op een dusdanige wijze dat er wordt voldaan aan een verzameling van harde condities.

We beschouwen de situatie waarbij we op zoek zijn naar een rooster voor een periode van *D* dagen. In deze periode moet een totaal van *#S* diensten toegewezen worden aan een totaal van *#E* medewerkers. Om te beginnen definiëren we een verzameling voor de uit te voeren diensten:

diensten	$S = \{ S[i]: i \varepsilon \{1, \ldots, \#S\}\}$.

De dag, begintijd en eindtijd van een willekeurige dienst S[j] beschouwen we als gegeven, en noteren we als volgt:

dag	$S[j][dag] \varepsilon \{1, 2, \ldots, D\}$,
begintijd	$S[j][start] \varepsilon (0, 1440)$ (aantal minuten vanaf middernacht),
eindtijd	$S[j][eind] \varepsilon (0, 1440)$ (aantal minuten vanaf middernacht).

De diensten moeten worden uitgevoerd door medewerkers, hiervoor definiëren we de verzameling van medewerkers:

medewerkers	$E = \{ E[i]: i \varepsilon \{1, \ldots, \#E\}\}$.

De oplossing van het probleem, een rooster, kan worden gemodelleerd als een binaire verzameling met alle mogelijke toewijzingen van medewerkers aan diensten:

toewijzingen	$A = \{A[i][j]: A[i][j] \varepsilon \{0,1\}, i \varepsilon \{1, \ldots, \#E), j \varepsilon \{1, \ldots, \#S\}\}$

Hierbij geldt dat A[i][j] = 1 als medewerker E[i] is toegewezen aan dienst S[j] en 0 als dit niet het geval is.

Om een optimale oplossing te vinden moeten we kunnen bepalen hoe goed een gegeven oplossing is. Hiervoor gebruiken we de zogenaamde *fitness functie*:

fitness functie	$F: A \rightarrow R.$

De vorm van deze fitness functie is cruciaal en bepaalt in grote mate de geschiktheid van de verschillende oplossingsmethoden. Het opstellen van een goede fitness functie is een complex probleem, er kunnen immers vele factoren zijn die een rol spelen bij de kwaliteit van een rooster. Over het algemeen is de fitness functie ofwel gericht op kosten minimalisatie, ofwel op een eerlijke verdeling van het werk. Combinaties hiervan zijn denkbaar, maar in de praktijk moeilijk inzetbaar omdat het vereist de kosten van een rooster te vergelijken met de eerlijkheid. In de volgende paragraaf bespreken we de verschillende elementen die terugkomen in de fitness functie.
Bij het optimaliseren van de fitness functie moet rekening gehouden worden met een aantal harde condities. In paragraaf 1.3 gaan we nader in op deze harde condities. De exacte vorm van deze condities kan uiteenlopen. We noteren de verzameling van #C condities als volgt:

harde condities	$C = \{ C[i]: i \varepsilon \{1, \ldots, \#C\}\}.$

Verder voegen we de volgende functie toe aan het model:

conditie functie	$CF: A \rightarrow \{0,1\}.$

In het bovenstaand geldt dat CF(A) = 1 betekent dat oplossing A aan alle harde condities voldoet, en CF(A) = 0 dat er minimaal één conditie overschreden is. In het geval dat CF(A) gelijk aan 1 is, spreken we van een *valide* oplossing.

Met de geïntroduceerde notatie kunnen we het CRP als volgt formuleren:

Probleem 1: Crew Rostering Problem	
maximaliseer	F(A),
onder	CF(A) = 1.

1.2 Het Evalueren van Roosters

In ons model wordt ons rooster geëvalueerd door middel van een fitness functie. Deze functie beoordeelt een rooster in de vorm van een numerieke waarde. Welke criteria precies een rol spelen bij deze beoordeling verschilt per geval, maar over het algemeen hebben ze betrekking op de volgende twee factoren:

Kosten
Vooral in traditionele modellen, afkomstig uit de luchtvaartindustrie, vormen de kosten van het rooster de grootste drijfveer voor optimalisatie. Omdat de kosten van een rooster lineair zijn in relatie tot de beslissingsvariabelen, zijn bekende OR technieken zoals Lineair Programmeren uitermate geschikt voor deze problemen.

Eerlijke verdeling
In de loop der jaren is er ook steeds meer interesse ontstaan in methoden voor het maken van ‘eerlijke’ roosters. Met een ‘eerlijk rooster’ doelen we op een rooster waarbij de totale hoeveelheid werk zo gelijkmatig mogelijk over de medewerkers verdeeld is, en waarbij zo goed mogelijk rekening gehouden is met persoonlijke voorkeuren van de medewerkers.

Afhankelijk van de situatie, kunnen er vele factoren zijn die een rol spelen bij de eerlijkheid van een oplossing. Idealiter worden al deze factoren meegenomen in de fitness functie, in de praktijk kan dit echter lastig zijn. Hiervoor zijn een aantal oorzaken.

1. Ten eerste vereist dit dat de verschillende mogelijke onwenselijke situaties in een rooster door het model geïdentificeerd en gekwantificeerd kunnen worden.

2. Ten tweede moet het voorkomen van onwenselijke situaties op de één of andere manier tegen elkaar afgewogen worden om zo de kwaliteit van het rooster als geheel te bepalen.

3. Tenslotte moet de resulterende fitness functie niet de complexiteit van het probleem dusdanig verhogen dat de beoogde oplossingsmethode niet meer toepasbaar is.

Fitness functies die op eerlijke roosters georiënteerd zijn, hebben de neiging om de complexiteit van het probleem te vergroten. Voornamelijk OR methoden lijden onder deze toename in complexiteit. Desalniettemin zijn er verscheidene technieken beschikbaar om dergelijke factoren mee te nemen in de oplossingsmethode. In hoofdstuk 4 gaan we nader in op het evalueren van roosters.

1.3 Harde Condities

Een rooster is alleen bruikbaar indien het voldoet aan een aantal gestelde, harde condities. Deze condities zijn onder te verdelen in een aantal categorieën. In deze paragraaf worden de categorieën kort besproken en worden er enkele voorbeelden van condities uit de betreffende categorieën gegeven.

1.3.1 Volledigheid

Over het algemeen wordt er van een rooster geëist dat het volledig is, dat wil zeggen, aan alle diensten is een uitvoerende medewerker gekoppeld. Of, in termen van ons model:

volledigheid	$\text{som}(i \in \{1, \ldots, \#E\}: A[i][j]) = 1.$

In sommige gevallen wordt er voor gekozen om de volledigheid van het rooster niet als harde conditie op te nemen in het model, maar in plaats daarvan te verwerken in de fitness functie. In deze gevallen wordt de volledigheidsconditie versoepeld tot de volgende conditie, die ervoor zorgt dat er niet meer dan één medewerker aan iedere dienst toegekend wordt:

versoepelde volledigheid	$\text{som}(i \in \{1, \ldots, \#E\}: A[i][j]) \leq 1.$

Vereiste Vaardigheden

In veel gevallen vereisen verschillende soorten diensten ook verschillende vaardigheden van de medewerkers. Dit nemen we op in het model door middel van de verzameling:

vaardigheden	$Sk = \{Sk[i][j]: Sk[i][j] \in \{0,1\}\}.$

Hierbij geldt dat Sk[i][j] gelijk is aan 1 als medewerker E[i] dienst S[j] kan uitvoeren, en 0 als dit niet het geval is. Dit levert ons de volgende conditie op:

vereiste vaardigheden	$A[i][j] \leq Sk[i][j]$ voor alle $i \in \{1, \ldots, \#E\}$, $j \in \{1, \ldots, \#S\}$.

1.3.2 Wetgeving en Bedrijfsspecifieke Regelgeving
De wetgeving stelt bepaalde eisen aan de werktijden van de medewerkers. Hiernaast hebben de meeste organisaties ook nog hun eigen regelgeving betreffende de werktijden van hun personeel. Over het algemeen hebben deze betrekking op het totaal aantal werk/vrije dagen, het aantal achtereenvolgende werk/vrije dagen of speciale regelgeving voor zwaardere diensten (bijvoorbeeld nachtdiensten of fysiek zwaar werk).

1.3.3 Eerlijke Verdeling
Het is mogelijk, en in sommige gevallen wenselijk, om harde condities op te nemen die de verdeling van het werk over de tijd beïnvloeden, om zo een 'eerlijk' rooster te garanderen. Het is echter gebruikelijker om deze condities als zachte condities te beschouwen en in beschouwing te nemen in de fitness functie. Vooral wanneer het een 'krap' rooster betreft (relatief weinig medewerkers voor relatief veel diensten) is het niet aan te raden deze als harde condities op te nemen.

1.3.4 Kosten
Indien de fitness functie primair georiënteerd is op het maken van een eerlijk rooster, kan het soms wenselijk zijn om een bovengrens voor de bijbehorende kosten als harde conditie op te nemen. Door deze grens te variëren kan er vervolgens 'met de hand' een afweging gemaakt worden tussen de kosten en de eerlijkheid van het rooster. Dit is echter minder geschikt voor problemen van grote omvang.

1.4 Het Rondvaart Probleem
In deze paragraaf beschrijven we de specifieke karakteristieken van het rondvaart probleem. We beginnen met een algemene beschrijving van wat de organisatie doet, en kijken vervolgens welke gevolgen dit heeft op het CRP.

We beschouwen een organisatie die rondvaarten organiseert over een aantal verschillende (vaste) routes. Gemiddeld varen er elk jaar rond de 50.000 passagiers met de organisatie. De organisatie is ontstaan uit een vrijwilligersproject voor 50-plussers, en bestaat nog steeds volledig uit vrijwilligers. Dit maakt de tevredenheid van de medewerkers een zeer belangrijke factor binnen de organisatie: de medewerkers doen het werk immers voor hun plezier.

1.4.1 Het Vaarschema
De organisatie beschikt over een zevental boten, en vaart over een kleine 10 verschillende routes. In het vaarschema is exact vastgelegd welke boot wanneer vaart, en over welke route. Iedere vaart duurt ongeveer 45 minuten, onafhankelijk van de route. Over het algemeen vaart een specifieke boot elke dag dezelfde route, maar hier zijn enkele uitzonderingen op. Enkele vaarten hebben een specifiek thema, waardoor niet alle medewerkers zijn geschikt als spreker op deze vaarten. Ook deze themavaarten liggen vast in het vaarschema.

1.4.2 Diensten
De medewerkers werken in diensten, die gebaseerd zijn op dagdelen (ochtend, middag of avond) in combinatie met een specifieke boot. Een typische dienst is dus "woensdag middag, schipper 1 op boot 5". De diensten liggen dus impliciet vast in het vaarschema. Op iedere vaart dienen twee medewerkers aanwezig te zijn: een schipper en een spreker. Over het algemeen wisselen deze medewerkers hun rol (schipper en spreker) om de vaart af. Ook hier zijn echter weer enkele uitzonderingen te vinden (zowel op basis van het thema van de vaart als op basis van de persoonlijke situatie van de medewerker).
Naast de medewerkers op de boten zijn er ook nog een aantal andere rollen die vervuld dienen te worden. Om te beginnen is er op iedere vaardag een zogenaamde walschipper dienst. Op de drukkere dagen zijn er ook regelmatig één of twee "stand-by" diensten, welke

ingezet kunnen worden in geval van afwezigheid van de ingeroosterde medewerkers. Ook deze walschipper en stand-by diensten liggen vooraf vast in het vaarschema.
Ter illustratie toont de onderstaande figuur de zogenaamde dagstaat van 2 juli 2008, waarop het vaarschema en de overige diensten van die dag zichtbaar zijn.

Dagstaat 02-07-2008

Walschipper: onbekend vanaf 13:00

		W1 (44)	W7 (24)	W3 (16)	W4 (16)	W5 (22)	W6 (24)	W2 (12)	Stanby 1	Stanby 2
11	00						11:00 west			
	15						vrij geboek			
	30		11:30 west				24 0			
	45		vrij geboek							
12	00		26 0				12:00 west			
	15						vrij geboek			
	30		12:30 west				24 0			
	45		vrij geboek							
13	00		26 0				13:00 west			
	15						vrij geboek			
	30		13:30 west				24 0			
	45		vrij geboek	13:45 oost H	13:45 oost W	13:45beek op				
14	00		26 0	vrij geboek	vrij geboek	vrij geboek	14:00 west			
	15	14:15 west		16 0	16 0	22 0	vrij geboek			
	30	vrij geboek	14:30 west			14:30 beek af	24 0			
	45	44 0	vrij geboek	14:45 oost W	14:45 oost H	vrij geboek				
15	00		26 0	vrij geboek	vrij geboek	22 0	15:00 west	15:00beek op		
	15	15:15 west		16 0	16 0		vrij geboek	vrij geboek		
	30	vrij geboek	15:30 west			15:30beek op	24 0	12 0		
	45	44 0	vrij geboek	15:45 oost H	15:45 oost W	vrij geboek		15:45 beek af		
16	00		26 0	vrij geboek	vrij geboek	22 0	16:00 west	vrij geboek		
	15	16:15 west		16 0	16 0	16:15 beek af	vrij geboek	12 0		
	30	vrij geboek	16:30 west			vrij geboek	24 0			
	45	44 0	vrij geboek			22 0		16:45 west		
17	00		26 0					vrij geboek		
	15							12 0		
	30									
	45									

Figuur 1: dagstaat van 2 juli 2008

Op de horizontale as zien we de verschillende boten (W1 t/m W7) en de eventuele stand-by diensten (die in dit geval niet aanwezig zijn).

1.4.3 Persoonlijke Voorkeuren
De schippers zijn niet fulltime beschikbaar voor de organisatie, maar geven dit aan door middel van een beschikbaarheidsformulier. Hierop dienen alle medewerkers minimaal 6 dagdeel-weekdag combinaties aan te geven waarop ze beschikbaar zijn. Ook dienen de medewerkers aan te geven hoeveel vaardiensten ze maximaal per week willen doen (het minimum hiervoor is twee). Tevens kunnen er een aantal andere persoonlijke voorkeuren opgegeven worden. Deze hebben onder andere betrekking op de weekenden, vrije en vakantiedagen, dubbele diensten en walschipper diensten.

1.4.4 Huidige Werkwijze
Momenteel is de werkwijze als volgt. Om te beginnen wordt er automatisch een “startrooster” gegenereerd. Merk op dat dit geen volledig rooster is. Vervolgens worden handmatig de gaten in het startrooster opgevuld, en de uitzonderingsgevallen verwerkt. Ook wordt er met de hand getracht de eerlijkheid van het rooster te verbeteren. Hiervoor hebben de roostermakers zelf een aantal intuïtieve methoden bedacht om de kwaliteit van een rooster te evalueren en te verbeteren.

Er zijn twee factoren die een grote rol spelen bij de beslissing om dit proces te innoveren:

1. *Toename hoeveelheid handwerk*
   Door het toenemende aantal uitzonderingen en andere veranderingen binnen de organisatie is de benodigde hoeveelheid handwerk in de loop van de tijd aanzienlijk toegenomen. Dit brengt twee nadelen met zich mee. Ten eerste de hoeveelheid werk zelf. De roostermakers beseffen dat, door het hoge aantal en de dynamische aard van de uitzonderingen, het nodige handwerk waarschijnlijk onvermijdelijk is. Toch hebben ze het gevoel dat de hoeveelheid van dit werk aanzienlijk gereduceerd kan worden, en er betere ondersteuning gegeven kan worden bij het verrichten van dit handwerk.
   Het tweede nadeel is dat deze werkwijze soms het gevolg kan hebben dat medewerkers zich persoonlijk benadeeld voelen, en (al dan niet terecht) het gevoel kunnen krijgen dat er sprake is van “vriendjespolitiek” bij het genereren van de roosters.

2. *Inzichtelijkheid en beïnvloedbaarheid van de startoplossing*
   Het programma waarmee de beginroosters momenteel genereerd worden, is voor de roostermakers een ‘black box’. Deze stap in het roosterproces wordt uitgevoerd door een derde partij. De roostermakers trekken echter de continuïteit van deze partij in twijfel, en zouden bovendien graag meer zicht en invloed hebben op het genereren van de (begin)roosters.

Bij het zoeken naar een geschikte methode voor het oplossen van het probleem is het dus van belang om deze twee factoren in beschouwing te nemen.

1.4.5 Observaties
Op basis van het bovenstaande doen we een aantal belangrijke observaties met betrekking tot het rondvaart CRP:

1. *Eerlijk Rooster*
   Om te beginnen merken we op dat het een vrijwilligersorganisatie betreft; dus de kosten van iedere mogelijk oplossing moet nagenoeg gelijk zijn, en dus geen rol spelen bij de beslissingen omtrent het rooster. Hierdoor ligt de aandacht volledig op het maken van een ‘zo eerlijk mogelijk’ rooster.

2. *Persoonlijke Voorkeuren*
   De persoonlijke voorkeuren van de verschillende medewerkers spelen een grote rol in hun beleving van de eerlijkheid van een rooster.

3. *Uitzonderingen*
   Er zijn relatief veel gevallen die door de organisatie als “uitzondering” beschouwd worden. Deze hebben voornamelijk betrekking op eigenschappen van specifieke medewerkers en thema’s.

4. *Beperkte Omvang*
   De omvang van het probleem is relatief klein, dat wil zeggen, een gegeven oplossing van het probleem is nog dusdanig goed te overzien dat het maken van handmatige wijzigingen een realistische optie is.

5. *Technische Kennis Roostermakers*
   De roostermakers zijn de nodige hoeveelheid handwerk gewend, en hebben een relatief hoge (technische) kennis van het probleem.

1.4.6 Kwaliteit Rooster

We identificeren de volgende factoren die een rol spelen bij onze rondvaart organisatie en de kwaliteit van een rooster bepalen:

1. *Verdeling hoeveelheid werk*
   De medewerkers geven niet alleen aan op welke dagdelen zij beschikbaar zijn, maar ook hoeveel dagdelen in de week. Bij het beoordelen van het rooster op een eerlijke verdeling van de hoeveelheid werk moet hier dus rekening mee gehouden worden. Verder merken we op dat het aantal diensten weliswaar een indicatie geeft van de hoeveelheid werk, maar niet een heel nauwkeurige. Dit komt doordat de diensten verschillende aantallen vaarten bevatten, op basis van de boot en de datum van de vaart.

2. *Spreiding over de periode*
   De diensten van alle medewerkers dienen zo evenwichtig mogelijk verspreid te zijn over de periode van het rooster. Vooral voor de lange termijn is deze factor van belang. De boten zijn namelijk niet overdekt, dus varen in de kou en regen is aanzienlijk minder aantrekkelijk dan varen op een zonnige zomermiddag.

3. *Variatie in routes en boten*
   Over het algemeen wordt variatie over verschillende route's door de schippers op prijs gesteld. Het varen van dezelfde route bij iedere dienst kan als saai ervaren worden. Aangezien er in het vaarschema een zekere relatie is tussen de boten en de route's, impliceert variatie in de boten ook variatie in de routes. Daarnaast zijn de boten van een aantal verschillende typen, en het bestuur van de organisatie is van mening dat het een goede zaak is de schippers regelmatig op de verschillende typen boten varen.

4. *Variatie in collega's*
   Ook variatie in collega schippers wordt als belangrijk beschouwd voor de kwaliteit van het rooster.

5. *Weekenden*
   De weekenden spelen een zeer belangrijke rol bij het maken van de roosters, en zijn altijd een gevoelig punt geweest. Het merendeel van de medewerkers ervaart de weekenddiensten (in meer of mindere mate) als 'extra vervelend'. Ondanks het feit dat sommige medewerkers (bepaalde) weekend diensten eventueel minder vervelend vinden dan andere medewerkers, geeft het bestuur van de organisatie aan dat ze willen dat het aantal weekend diensten zo evenwichtig mogelijk verdeeld wordt over de medewerkers. Wel stelt de organisatie haar medewerkers in de gelegenheid om aan te geven of zij er de voorkeur aan geven om hun weekend diensten zoveel mogelijk in dezelfde weekenden in te plannen.

6. *Dubbele diensten*
   Uit de praktijk is gebleken dat sommige medewerkers de voorkeur geven aan een volledige dag varen, dat wil zeggen, zowel een ochtend als een middag dienst op dezelfde dag. De mate waarin aan deze wens wordt voldaan beïnvloedt dus de kwaliteit van het rooster.

7. *Verdeling walschipper diensten*
   Voor de walschipper diensten geldt dat de medewerkers de diensten verschillend beoordelen. De organisatie geeft aan de walschipper diensten voor iedere medewerker op één van de volgende vier manieren te willen verwerken in het rooster:

a. walschipper diensten zijn extra en worden niet meegerekend als vaarbeurt,
b. walschipper diensten worden wel meegerekend als vaarbeurt,
c. alleen varen, geen walschipper diensten,
d. alleen walschipper, niet varen.

8. *Verdeling stand-by diensten*
   Voor de stand-by diensten geldt simpelweg dat ze zo eerlijk mogelijk verdeeld moeten worden. Dat wil zeggen, een zo laag mogelijke spreiding in het totale aantal stand-by diensten per medewerker.

Voor het rondvaartprobleem zoeken we een oplossingsmethode waarbij zo veel mogelijk van deze factoren in beschouwing genomen worden.

1.5 Toepassingen

Het CRP kent vele praktische toepassingen, ieder met zijn eigen specifieke kenmerken. De meest bestudeerde, en tevens de eerste branche waarin geautomatiseerde oplossingen toegepast zijn, is de transport sector. Voorbeelden zijn de luchtvaartindustrie en openbaar vervoer diensten. Een typisch kenmerk van CRP's uit deze sector is dat, naast de begin- en eindtijden van de diensten, er ook rekening gehouden moet worden met de geografische locatie. Een andere bekende, en veel bestudeerde toepassing van het CRP komt uit de gezondheidszorg, en betreft het inroosteren van verpleegkundigen. Overige voorbeelden van toepassingen waar het CRP speelt, zijn nooddiensten, productie processen, toerisme en financiële diensten.

1.6 Oplossingsmethoden

Het CRP valt onder de categorie van zogenaamde combinatorische problemen. Dit zijn problemen waarbij gezocht wordt naar een optimale oplossing in een (eindige) oplossingsruimte. In de loop der jaren is er veel onderzoek naar dit probleem gedaan. Er is interesse voor dit probleem vanuit verscheidene vakgebieden, voornamelijk in de *Operations Research* en *Artificial Intelligence* wordt dit probleem veel bestudeerd. In hoofdstuk 2 en 3 worden verschillende technieken uit deze twee vakgebieden besproken, en wordt geanalyseerd in welke mate deze oplossingen geschikt zijn voor het rondvaart probleem.

Hoofdstuk 2: Mathematisch Programmeren

In dit hoofdstuk beschouwen we een veel gebruikte techniek voor het oplossen van het CRP: Mathematisch Programmeren (MP). Deze techniek is afkomstig uit de Operations Research, en kan worden ingezet voor een groot aantal verschillende problemen.
Het CRP kan geformuleerd worden als *Linear Integer Program* (LIP). Dit Linear Integer Program is een instantie van het beruchte *Set Covering Problem* (SCP), dat als volgt geformuleerd kan worden:

Probleem 2: Set Covering Problem (SCP)

minimaliseer	som(j ε {1, 2, …, n}: c[j] * x[j]),	
onder	som (j ε {1, 2, …, n}: a[i][j] * x[j]) ≥ 1,	voor i ε {1, 2, …, m},
	x[j] ε {0,1},	voor j ε {1, 2, …, n},

waarbij A = (a[i][j]) een m · n matrix is bestaande uit 0'en en 1'en.

Dit is een intensief bestudeerd, maar zeer complex probleem. Een exacte oplossings-methode is in theorie beschikbaar (de zogenaamde Branch and Bound (BB) methode), maar is in de praktijk niet altijd toepasbaar. Hierdoor wordt er in de praktijk in veel gevallen gebruikt gemaakt van een benadering.
De complexiteit van het probleem wordt veroorzaakt door de eis dat de beslissingvariabelen x[j] geheeltallig zijn. Een voor de hand liggende benaderingsmethode is dan ook het loslaten van deze eis. Hierdoor ontstaat een zogenaamde LP Relaxatie van het oorspronkelijke LIP. Dit nieuwe probleem kan relatief eenvoudig worden opgelost. Vervolgens wordt er een geheeltallige oplossing gezocht in de buurt van de gevonden oplossing. De resultaten van de methode zijn echter wisselvallig, afhankelijk van het specifieke probleem waarvoor de methode ingezet wordt. Een betere methode voor het benaderen van dit probleem is de zogenaamde Lagrange Relaxatie. Deze wordt besproken in de volgende paragraaf.

2.1 Lagrange Relaxatie

Lagrange Relaxatie is een methode voor het benaderen van *Linear Integer Programs* die in de praktijk veel toegepast wordt. In [9] wordt bewezen dat de methode altijd gelijke of betere benaderingen geeft dan LP Relaxatie.
Het basis idee van de methode is dat het op te lossen *Linear Integer Program* gezien kan worden als een relatief eenvoudig basisprobleem, met een aantal complexe bijcondities. Deze complexe condities worden vervolgens niet als harde condities opgenomen, maar als "strafpunten" in de te optimaliseren functie.
De methode gaat als volgt te werk. Beschouw het volgende Linear Integer Program.

Probleem 3: Linear Integer Program

minimaliseer	$c^T x$,
onder	$Ax \leq b$,
	$x \geq 0$, x geheeltallig.

De condities ($Ax \leq b$) worden opgesplitst in een groep met complexe condities ($Dx \leq e$), en een groep eenvoudige condities ($Gx \leq h$). De Lagrange Relaxatie van probleem 3 wordt geformuleerd door de eerste groep weg te laten uit de condities, en de term $\mu^T(e - Dx)$ van de criterium functie af te trekken. Hierbij is μ een vector van niet-negatieve getallen, die geïnterpreteerd kan worden als een weging voor de verschillende condities. Deze vector wordt ook wel de *Lagrange Multiplier* genoemd. Dit resulteert in het volgende probleem:

Probleem 4: Lagrange Relaxatie van Probleem 3

minimaliseer	$c^T x - \mu^T (e - Dx)$,
onder	$Gx \leq h$,
	$x \geq 0$, x geheeltallig.

Het idee is nu dat, door het weglaten van de juiste condities, dit probleem eenvoudig op te lossen is.

Lagrange Relaxatie levert over het algemeen goede benaderingen voor het Set Covering Problem. Het concept achter de methode (het verplaatsen van condities naar de criterium functie) is iets wat we ook terug zien in de *Artificial Intelligence*.

2.2 Branch and Bound

Zoals gezegd in de inleiding van dit hoofdstuk, is dit de enige bekende exacte oplossingsmethode voor het Set Covering Problem. We noteren de te minimaliseren functie als F(x), en de (eindige) oplossingsruimte *U.* Branch en Bound gaat als volgt in werking

1. *Branching*
   Op basis van de zogenaamde *branching rule*, wordt de oplossingsruimte *U* opgesplitst in *n* disjuncte deelverzamelingen U_1, U_2, …, U_n, waarvoor geldt dat de vereniging van U_1, U_2, …, U_n gelijk is aan *U*.

2. *Bounding*
   Voor iedere nieuwe deelverzameling U_i wordt vervolgens een onder- en bovengrens van F(x) bepaald.

3. *Pruning*
   Wanneer voor een deelverzameling U_i geldt dat de bijbehorende ondergrens hoger is dan de tot dan toe gevonden laagste bovengrens, wordt de deelverzameling geschrapt.

Dit proces wordt recursief herhaald (voor deelverzamelingen U_i) totdat er of nog maar één oplossing over is, of wanneer er een deelverzameling gevonden wordt waarvoor de ondergrens gelijk is aan de bovengrens. Het bepalen van de ondergrens wordt doorgaans gedaan door middel van Lagrange Relaxatie.
Het probleem van deze methode zit in de rekentijd. Deze neemt exponentieel toe ten opzichte van de grootte van de oplossingsruimte. Dit maakt de methode in de praktijk niet altijd toepasbaar. Het kiezen van een slimme branching *method* kan de rekentijd enigszins beperken, maar de relatie blijft exponentieel.
In verband met deze explosieve toename in rekentijd van de Branch and Bound methode, zijn een aantal heuristische varianten op de Branch en Bound methode ontwikkeld, die gericht zijn op het verkleinen van de zoekruimte van het algoritme. Dit kan bijvoorbeeld bereikt worden door het toepassen van een alternatieve (agressievere) pruning methode. Een voorbeeld van een heuristische toevoeging aan de Branch and Bound methode is het toepassen een zogenaamd *Greedy Algorithm* in de *pruning* fase. *Greedy Algorithms* zijn afkomstig uit de AI, en werken volgens het principe telkens de lokaal optimale keuze te maken. Dit komt er op neer dat bij de *pruning* alle deelverzamelingen U_i geschrapt worden, met uitzondering van de deelverzameling met de laagste bovengrens. Het is echter niet aan te raden een dergelijk *pruning* methode toe te passen in de eerste stappen van het Branch and Bound algoritme, omdat er dan te snel potentieel goede oplossingen weggegooid worden. Het kiezen van een geschikt moment om een dergelijke methode toe te passen is een complex probleem, waarvoor in de praktijk doorgaans “trial and error” toegepast wordt.

2.3 Branch and Bound voor het CRP

In [2] wordt een CRP geformuleerd als een LIP, en opgelost met behulp van de Branch and Bound methode. Er worden drie formuleringen gepresenteerd. In deze paragraaf bespreken we één van deze formuleringen, de zogenaamde *Generalized Assignment Formulation*.
Het betreft een CRP bij een afval verwerkingsbedrijf, waarbij de dagelijkse diensten op een eerlijke manier verdeeld moeten worden over de medewerkers. Hierbij dient rekening te worden gehouden met de verschillende vaardigheden van de medewerkers.

Het probleem wordt gemodelleerd als zogenaamd *Multilevel Bottleneck Assignment Problem*. Het model gaat uit van een periode van *L* dagen, en veronderstelt dat het aantal dagelijks uit te voeren diensten gelijk is aan het aantal medewerkers. Indien dit in de praktijk niet het geval is kunnen er *dummy* diensten opgenomen worden.

De volgende notatie wordt gehanteerd:

- De verzameling van alle uit te voeren diensten wordt genoteerd als *N*, en dienst nummer *i* dienst van dag *l* als u[i][l].
- N[l] is de verzameling van alle diensten op dag *l*.
- Aan iedere dienst wordt een zogenaamde 'workload' toegekend. De workload van dienst u[i][l] wordt genoteerd als w[i][l], en geeft een indicatie van de hoeveelheid werk van de dienst.
- De verzameling medewerkers wordt aangeduid met W. W[l][i] is de verzameling van medewerkers die, op basis van vaardigheden, geschikt zijn voor het uitvoeren van dienst u[i][l].
- Met s[i][l] noteren we de totale 'workload' van de medewerker die aan dienst u[i][l] toegewezen is, over de periode dag 1 tot en met l. Het maximum van alle s[i][l] wordt genoteerd als *z*. Deze waarde wordt als criterium functie gebruikt in de optimalisatie.
- De beslissingsvariabelen worden genoteerd als u[i][l][d] ε {0,1}, waarbij u[i][l][d] gelijk is aan 1 indien dienst u[i][l] aan medewerker *d* toegekend wordt, en 0 anders.

Met deze notatie wordt het probleem geformuleerd als het volgende Linear Integer Program:

Probleem 5: Linear Integer Program voor CRP

minimaliseer z

onder

(5a) $\text{som}(u[i][l] \varepsilon N: w[i][l] * u[i][l][d]) \leq z, \quad d \varepsilon W,$

(5b) $\text{som}(d \varepsilon W[i][l]: u[i][l][d]) = 1, \quad u[i][l] \varepsilon N,$

(5c) $\text{som}(u[i][l] \varepsilon N[l]: u[i][l][d]) = 1, \quad d \varepsilon W, \; l \varepsilon \{1, 2, \ldots, L\},$

(5d) $u[i][l][d] \varepsilon \{0,1\}.$

Conditie 5b zorgt ervoor dat iedere dienst wordt toegekend aan exact één medewerker uit de verzameling met geschikte medewerkers. Conditie 5c garandeert dat iedere medewerker op iedere dag exact aan één dienst toegewezen wordt.

Er wordt Langrange Relaxatie toegepast op conditie 5b, dit resulteert in het volgende probleem:

Probleem 6: Lagrange Relaxatie van Probleem 5

minimaliseer $z - \text{som}(u[i][l] \varepsilon N: \text{som}(d \varepsilon W[i][l]: \mu[i][l] * u[i][l][d]))$,

onder

(6a) $\text{som}(u[i][l] \varepsilon N: w[i][l] * u[i][l][d]) \leq z$, $d \varepsilon W$,

(6b) $\text{som}(u[i][l] \varepsilon N[l]: u[i][l][d]) = 1$, $d \varepsilon W$, $l \varepsilon \{1, 2, \ldots, L\}$,

(6c) $u[i][l][d] \varepsilon \{0,1\}$.

Dit probleem kan als parametrisch ten opzichte van z beschouwd worden. De optimale waarde van de criterium functie van dit probleem wordt genoteerd als L[z](μ), met bijbehorende toewijzingen u*[i][l][d](μ,z). Voor gegeven waarden van z en μ kan dit probleem opgedeeld worden in |W| onafhankelijke *Multiple Choice Knapsack* (MCKP) problemen, één voor iedere medewerker. Dit MCKP is een bekend probleem waar veel onderzoek naar gedaan is, en exacte oplossingsmethoden zijn beschikbaar.
Om de oplossing van het parametrische probleem uit te buiten, wordt de volgende functie beschouwd:

$F(z) = z - L[z](\mu)$
$= \text{som}(i,l,d: \mu[i][l] * u^*[i][l][d](\mu,z))$.

Deze functie is monotoon stijgend in z. Voor lage waarden van z geldt één van de volgende twee uitspraken:

1. Probleem 6 is niet oplosbaar. Omdat probleem 6 een relaxatie van probleem 5 is, impliceert dit dat ook probleem 5 niet oplosbaar is.

2. F(z) is negatief. Dit zou betekenen dat het optimum van het oorspronkelijke probleem kleiner is dan het optimum van de relaxatie. Ook in dit geval is probleem 5 niet oplosbaar. Aangezien F(z) monotoon stijgend is in z, is de gevonden waarde voor z dus een ondergrens voor probleem 5.

Voor hogere waarden van z geldt dat F(z) positief is, en dat er geen uitspraak over het oorspronkelijke probleem gedaan kan worden.
De scherpst mogelijke ondergrens voor probleem 5 wordt gevonden door de hoogste waarde van z te zoeken, waarvoor geldt dat $F(z) < 0$. Dit wordt gedaan door middel van binary search.

De gevonden ondergrens wordt vervolgens gebruikt in een Branch and Bound procedure. Bij het “branchen” wordt de verzameling medewerkers W[i][l] die een bepaalde dienst u[i][l] kunnen uitvoeren, gesplitst in twee deelverzamelingen van gelijke grootte.

2.4 Sterke en Zwakke Eigenschappen

MP technieken hebben over het algemeen de eigenschap dat ze kwalitatief zeer goede oplossingen produceren voor het gedefinieerde model. Vooral voor kostengevoelige problemen, waarbij een kleine verbetering in de kwaliteit van de oplossing een relatief grote kostenbesparing teweeg brengt, is dit een zeer aantrekkelijke eigenschap. Een goed

voorbeeld hiervan is de luchtvaartindustrie, waar het toepassen van MP technieken tot grote kostenbesparingen geleid heeft.
Een grote beperking is echter de eis dat zowel de fitness functie als de harde condities, gemodelleerd moeten kunnen worden als lineaire combinatie van de beslissingsvariabelen.
Dit heeft het gevolg dat niet altijd alle factoren die in de praktijk een rol spelen in het model opgenomen kunnen worden. Voornamelijk condities die betrekking hebben op de eerlijkheid van het rooster, kunnen moeilijk in de modellen opgenomen worden. Dit is een tweede reden die deze methoden voornamelijk geschikt maakt voor CRP's waarbij kosten minimalisatie centraal staat.
Een ander nadeel van MP technieken is dat de modellen relatief inflexibel zijn: het toevoegen van condities of het aanpassen van de fitness functie kan er in veel gevallen toe leiden dat de gebruikte oplossingsmethode niet meer toepasbaar is.
Tenslotte merken we op dat door de wiskundige complexiteit, het implementeren van deze methode relatief moeilijk, en dus duur is. In de onderstaande tabel worden de sterke en zwakke eigenschappen van MP technieken als oplossingsmethode voor het CRP op een rij gezet.

Sterke Eigenschappen	Zwakke Eigenschappen
• kwalitatief goede oplossingen • veel bestudeerde aanpak • "kant en klare" modellen beschikbaar voor een aantal specifieke problemen	• lineaire beperking • relatief inflexibel • hoge complexiteit • dure ontwikkeling

Tabel 1: sterke en zwakke eigenschappen van MP als oplossingsmethode voor het CRP

2.5 Het Rondvaart Probleem

In deze paragraaf analyseren we in welke mate MP technieken geschikt zijn voor het in hoofdstuk 1 geïntroduceerde rondvaart probleem. Zoals aangegeven in hoofdstuk 1 ligt de nadruk op het maken van een eerlijk rooster. In paragraaf 1.4.6 hebben we aangegeven welke factoren de kwaliteit van het rooster bepalen. In de volgende paragrafen analyseren we voor elk van deze factoren in welke mate ze meegenomen kunnen worden bij het gebruik van MP als oplossingsmethode.

Verdeling hoeveelheid werk

De verdeling van de hoeveelheid werk is cruciaal voor de eerlijkheid van een rooster. Een goede maat voor de verdeling van het werk, is de variantie in de totale hoeveelheid werk van de verschillende medewerkers. Het probleem is echter dat deze variantie niet lineair is ten opzichte van de beslissingsvariabelen, en dus niet gebruikt kan worden in de fitness functie van een LIP.
Er moet dus een andere indicatie voor de verdeling van het werk gevonden worden. In de in paragraaf 2.3 beschreven methode wordt hiervoor de maximale totale hoeveelheid werk van de medewerkers gebruikt. De vraag is echter in hoeverre dit maximum een indicatie geeft over de spreiding van de totale hoeveelheid werk per medewerker. We merken op dat dit maximum een betere indicatie geeft voor de spreiding bij "krappe" roosters. Dat wil zeggen, roosters waarbij er niet veel meer medewerkers beschikbaar zijn dan dat er nodig zijn om de geplande diensten mee uit te voeren. Bij commerciële organisaties is dit doorgaans het geval, aangezien dit vanuit financieel oogpunt efficiënter is; in het rondvaart probleem is dit echter niet zo.
Tevens merken we op dat bij het rondvaart probleem, de beoordeling van deze verdeling rekening moet houden met de opgegeven persoonlijke voorkeuren van de medewerkers. Alleen de variantie van de totale hoeveelheid werk geeft op zich dus geen goede indicatie voor hoe eerlijk het werk verdeeld is. We moeten de totale hoeveelheid werk dus relateren aan de opgegeven gewenste hoeveelheid werk. In de in paragraaf 2.3 beschreven methode

kan dit gedaan worden door niet te kijken naar de totale hoeveelheid werk van de medewerkers, maar deze te verminderen met de “gewenste” hoeveelheid werk.

Spreiding over de periode
Een andere belangrijke factor is de spreiding van de diensten over de periode. Ook hier geldt echter weer dat deze spreiding doorgaans gemeten wordt door middel van een variantie, en dus geen lineair verband heeft met de beslissingsvariabelen. In de in paragraaf 2.3 beschreven methode wordt dan ook geen rekening gehouden met deze factor. We merken opnieuw op dat dit probleem minder speelt bij “krappe” roosters; hier is immers weinig ruimte voor spreiding van de medewerkers over de periode.

Weekenden
Zoals aangegeven vormen de weekenden altijd een gevoelig punt bij de rondvaart organisatie, en wordt er gestreefd naar een evenwichtige verdeling van het aantal weekend diensten.
Wanneer we kijken naar de verdeling van de weekenden, geldt ook hier dat deze lastig als lineaire combinatie van de beslissingsvariabelen uit te drukken is, en dus niet op een directe manier aan het LIP toegevoegd kan worden.
Een mogelijke manier om hier mee om te gaan, is het opsplitsen van het rondvaart probleem in twee subproblemen, één voor het roosteren van de weekenden en één voor de overige dagen. Uiteraard zijn deze twee problemen theoretisch niet onafhankelijk, maar omdat een evenwichtige verdeling van de weekenddiensten zeer hoge prioriteit heeft, zal een methode die deze twee subproblemen achtereenvolgens oplost naar verwachting acceptabele oplossingen geven voor het oorspronkelijke probleem.

Voor de overige besproken factoren in paragraaf 1.4.6 (“variatie in routes, boten en collega’s”, “dubbele diensten” en de “verdeling van walschipper/stand-by diensten”) geldt dat ze in het model op te nemen zijn door extra variabelen te introduceren. In de praktijk heeft dit echter twee nadelen:

1. het introduceren van extra variabelen en condities heeft in veel gevallen het gevolg dat de gebruikte oplossingsmethode voor het bepalen van een onder/bovengrens, niet meer toepasbaar is.

2. de bijkomende groei van de oplossingsruimte kan tot gevolg hebben dat de Branch en Bound methode door een toename in de rekentijd niet effectief meer toepasbaar is.

In de praktijk betekent dit dat er bij problemen met complexe condities vaak een heuristische oplossingsmethode toegepast moet worden.

2.5.1 Conclusie
Al met al moeten we concluderen dat de Mathematisch Programmering geen geschikte oplossingsmethode is voor het rondvaart probleem. Dit komt voornamelijk door het hoge aantal en de diversiteit van de factoren die een rol spelen bij de kwaliteit van een oplossing. Doordat een aantal van deze factoren niet, of slechts bij benadering, in het model opgenomen kunnen worden, gaat het voordeel van een exacte (of in ieder geval kwalitatief goede) oplossing in zekere mate verloren. Daarnaast spelen ook de beperkte flexibiliteit en relatief hoge kosten een rol bij de beslissing om niet voor een MP oplossing te kiezen.

Hoofdstuk 3: Heuristische Zoekmethoden

In de kunstmatige intelligentie (AI) zijn een aantal heuristische zoekmethoden voor combinatorische problemen ontwikkeld. Deze kunnen toegepast worden op het CRP. In dit hoofdstuk bespreken we een aantal van deze methoden. Men spreekt van een hybride oplossingsmethode, indien de methode twee of meer (al dan niet heuristische) technieken combineert. Deze technieken kunnen afkomstig zijn uit verschillende vakgebieden. Voordat we de specifieke zoekmethoden bespreken, kijken we eerst naar een algemeen kenmerk van deze methoden.

3.1 Reproductie en Survival

Veel van de heuristische zoekmethoden werken, op hoog niveau, volgens hetzelfde concept: reproductie en survival. Methoden die dit concept volgen gaan als volgt te werk:

1. *Initialisatie*
   Genereer één of meerdere startoplossingen (die voldoen aan alle harde condities van het probleem). Initialiseer de 'huidige verzameling van oplossingen' met deze startoplossingen.

2. *Reproductie*
   Genereer, op basis van de huidige verzameling oplossingen, één of meerdere nieuwe kandidaat oplossingen (die voldoen aan alle harde condities van het probleem).

3. *Survival*
   Vernieuw de huidige verzameling van oplossingen door de relatief slechtere huidige oplossingen te verwijderen, en de relatief betere kandidaat oplossingen toe te voegen.

Stap 2 (reproductie) en stap 3 (survival) worden vervolgens herhaald totdat er aan een bepaalde stopconditie voldaan is. In de praktijk heeft deze conditie doorgaans betrekking op de kwaliteit van de (best) gevonden oplossing, de rekentijd van het algoritme of een combinatie van beide. In de volgende paragrafen worden een aantal oplossingsmethoden besproken die dit concept volgen, en worden voorbeelden gegeven van hoe deze methode op het CRP toegepast kunnen worden.

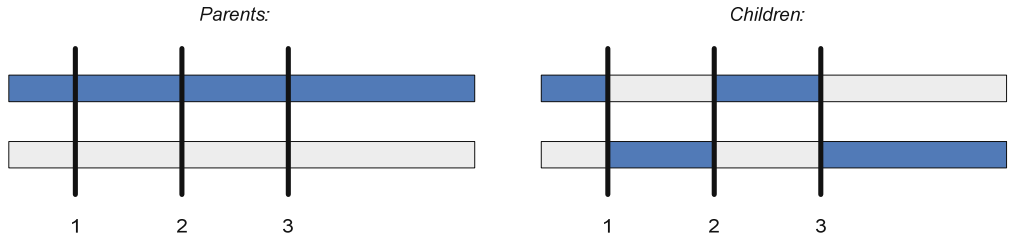
3.2 Genetische Algoritmen

Het eerste voorbeeld van een klasse algoritmen die volgens het beschreven concept werken, zijn de zogenaamde Genetische Algoritmen. Het idee achter deze klasse stamt af van Darwin's evolutietheorie; "Survival of the fittest". Gedurende het algoritme "evolueert" een populatie van oplossingen. Hierbij probeert het algoritme de populatie in de richting van de optimale oplossing te laten evolueren.
Oplossingen worden genetisch gerepresenteerd. Een individuele oplossing wordt een chromosoom genoemd, en bestaat uit een rij van genen. Traditioneel wordt een gen als binaire variabele gemodelleerd, maar andere representaties zijn ook mogelijk.
Bij de initialisatie wordt, doorgaans op volledig willekeurige basis, een populatie van chromosomen gegenereerd. Tijdens de reproductie wordt een selectie van paren uit deze populatie gekruist en/of gemuteerd, om zo een nieuwe generatie van oplossingen te genereren. De chromosomen uit deze nieuwe generatie worden vervolgens geëvalueerd, de betere chromosomen uit de nieuwe generatie vervangen de minder goede uit de huidige populatie. Dit proces wordt herhaald totdat er een oplossing van voldoende kwaliteit gevonden is, of een bepaalde tijd verstreken is.

3.2.1 Selectie
Het idee is dat de beste chromosomen uit de populatie gebruikt worden voor het creëren van een nieuwe generatie. Binnen Genetische Algoritmen wordt deze stap aangeduid met selectie. Voor deze selectie zijn meerdere methoden beschikbaar. De meeste van deze methoden zijn stochastisch, en dusdanig opgezet dat ook een klein deel van de minder goede chromosomen meegenomen worden bij het genereren van de nieuwe generatie. Twee bekende, en veel gebruikte methoden, zijn *Tournament Selection* en *Fitness Proportion Selection*.
Bij *Tournament Selection* wordt een willekeurige groep van chromosomen geselecteerd. Vervolgens wordt er een "toernooi" tussen deze chromosomen gehouden, waarbij het chromosoom met de hoogste fitness functie wint. Er worden een aantal van deze toernooien gehouden, en de winnaars worden vervolgens gebruikt voor reproductie.
Bij *Fitness Proportion Selection* (ook wel bekend als *Roulette-Wheel* Selection) worden chromosomen geselecteerd door middel van "trekkingen" uit de populatie (*A*). Ieder chromosoom *u* heeft een kans p[u] om gekozen te worden bij een trekking. Deze kans is gebaseerd op de waarde van de fitness functie voor dit chromosoom (F(u)), en wordt als volgt berekend:

$$p[u] = F(u) / som(a \varepsilon A: F(a)).$$

3.2.2 Reproductie
Ook voor de reproductie zelf zijn er meerdere methoden beschikbaar, waaronder de bekende methoden: *K-Point Crossover* en *Mutation*. Bij K-point crossover worden 2 chromosomen gemixed, op basis van k punten. In figuur 2 wordt illustratief weergegeven hoe 3-point crossover in zijn werk gaat.



Figuur 2: 3-point Crossover

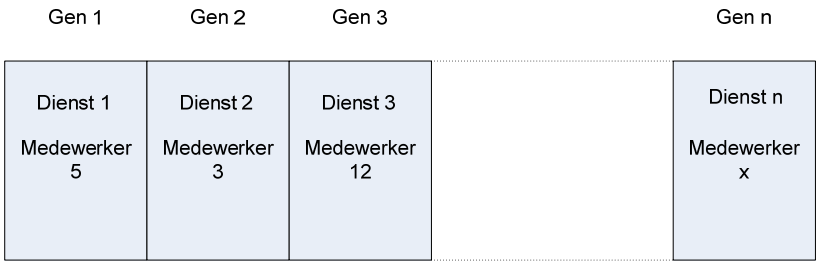
Bij mutatie worden één of meerdere bits binnen een enkel chromosoom aangepast. Dit gebeurt doorgaans op willekeurige basis. De traditionele methode is het toekennen van een stochastische variabele aan iedere bit, waarvan de waarde vervolgens bepaalt of de betreffende bit al dan niet aangepast wordt. Diverse variaties op deze methode zijn mogelijk.
In de praktijk komt het regelmatig voor dat een combinatie van *crossover* en *mutation* toegepast wordt.

3.2.3 Survival
Wanneer een nieuwe generatie van chromosomen gecreëerd is, moet worden bepaald welke van deze nieuwe chromosomen aan de populatie toegevoegd worden, en ten koste van welke chromosomen uit de populatie. Ook hiervoor zijn weer verschillende methoden toepasbaar. De methoden verschillen in het aantal chromosomen dat wordt vervangen, en de manier waarop de chromosomen gekozen worden. Een voorbeeld is de zogenaamde *Elitist Strategy*, waarbij alleen het "slechtste" chromosoom uit de populatie wordt vervangen met het beste chromosoom uit de nieuwe populatie (indien deze beter is).

3.2.4 Conclusie
Een grote uitdaging bij het in de praktijk inzetten van een Genetisch Algoritme, is het vinden van een geschikte genetische representatie van de oplossing, die het mogelijk maakt om op adequate wijze te kunnen reproduceren en evalueren. Hiernaast kan het opstellen van een reproductie methode, die chromosomen genereert die valide oplossingen voor het probleem vormen, een complex probleem zijn.
Zoals in deze paragraaf duidelijk geworden is, kan er bij het toepassen van een genetisch algoritme bij een aantal stappen uit verschillende methoden gekozen worden. Dit resulteert in een groot aantal mogelijke combinaties, waardoor in de praktijk het “fine tunen” van een algoritme een tijdrovende en complexe bezigheid kan worden.
In de volgende paragraaf geven we een voorbeeld van hoe een genetisch algoritme kan worden toegepast op het CRP.

3.3 Genetisch Algoritme voor het CRP
In [5] wordt een manier gepresenteerd om Genetische Algoritmen te gebruiken als oplossingsmethode voor het CRP. In deze paragraaf bespreken we de gebruikte aanpak.

3.3.1 Genetische Representatie
In de genetische representatie van de oplossing heeft ieder gen betrekking op een uit te voeren dienst. De waarde van het gen refereert naar de medewerker die aan de dienst wordt toegewezen. In de onderstaande figuur wordt dit schematisch weergegeven.



Figuur 3: genetische representatie van een rooster

Het grote voordeel van deze representatie is dat het expliciet vastgelegd wordt, en er geen tijdrovende algoritmen vereist zijn voor het “vertalen” van de oplossing. Een nadeel is echter dat alle harde condities afgedwongen worden, en dus representeert niet ieder chromosoom een valide oplossing van het probleem (zo kan een medewerker bijvoorbeeld aan twee gelijktijdige diensten toegekend worden). Om ervoor te zorgen dat het algoritme valide oplossingen produceert, worden er in de fitness functie “strafpunten uitgedeeld” voor het overschrijden van deze condities. Hiernaast worden er een aantal zogenaamde “repair operators” toegepast, hier komen we in paragraaf 3.3.4 op terug.

3.3.2 Selectie en Reproductie
Er wordt gebruik gemaakt van *Tournament Selection*: twee chromosomen uit de populatie worden willekeurig gekozen, en de beste van de twee wordt gebruikt voor reproductie. Bij deze reproductie wordt in eerste instantie 2-point crossover toegepast. Vervolgens worden er een aantal willekeurige mutaties op het resulterende chromosoom uitgevoerd. Het aantal mutaties dat uitgevoerd wordt, wordt gedurende het algoritme lineair verlaagd tot een minimum van 2. Vervolgens wordt het chromosoom uit de populatie met de laagste fitness functie vergeleken met het nieuwe chromosoom, en vervangen indien het nieuwe chromosoom beter is.

3.3.3 Fitness Functie
Aan iedere conditie van het probleem worden “overschrijdingskosten” toegekend. De fitness functie telt vervolgens voor iedere conditie het aantal overschrijdingen, en schaalt deze met de bijbehorende overschrijdingskosten. Zowel harde als zachte condities worden meegenomen in de fitness functie.

3.3.4 Repair Operators
Bij het toepassen van het tot dusver beschreven algoritme worden oplossingen gevonden met een relatief goede waarde voor de fitness functie. Er worden echter nog relatief veel harde condities overschreden in deze oplossingen. Om dit te voorkomen worden er repair operators toegepast. Deze repair operators bevatten domeinkennis voor het specifieke probleem, en gebruiken deze kennis om invalide oplossingen te repareren. De operators voeren de volgende handelingen uit:

1. Toewijzingen annuleren
   Wanneer een medewerker aan meerdere diensten op het zelfde tijdstip is toegewezen, worden op één na alle toewijzingen geannuleerd

2. Nieuwe toewijzen
   De ontstane gaten worden opgevuld met, willekeurig gekozen, beschikbare medewerkers.

3. Omwisselen van toewijzingen
   Op basis van specifieke domeinkennis worden de toegewezen diensten van twee medewerkers omgewisseld. Dit bijvoorbeeld om zo een meer homogeen rooster voor de medewerkers te creëren.

Het toepassen van deze repair operators brengt echter het nadeel met zich mee dat het zoekgedrag meer lokaal wordt. Het te vroeg toepassen van deze operators verslechtert dan ook de kwaliteit van de oplossing. In de praktijk is het vinden van het juiste moment van het toepassen van de repair operators een kwestie van fine-tunen door middel van trial and error.

3.3.5 Resultaten
Het beschreven algoritme is getest op een aantal datasets uit de praktijk, en was in het grootste deel van de gevallen in staat redelijke oplossingen te produceren, dat wil zeggen, oplossingen waarbij geen harde condities overschreden worden, en die met de hand niet eenvoudig (te veel) te verbeteren zijn. Het is echter in al deze gevallen nodig gebleken om domeinkennis te gebruiken om acceptabele roosters te produceren.

3.4 Local Search
Een *Local Search* algoritme is een iteratief algoritme, waarbij vanuit een willekeurige startoplossing, stap voor stap naar “nabij gelegen” oplossingen verplaatst wordt. *Local Search* staat ook wel bekend als *Neighbourhood Search*. Bij het maken van deze selectie, mag slechts gekozen worden uit oplossingen die in de buurt (*neighbourhood*) van de huidige oplossing liggen. Wederom zijn er voor het selecteren van een nieuwe oplossing verschillende methoden mogelijk.
Een voorbeeld van Local Search is *Steepest Descent*. Hierbij wordt telkens die oplossing uit de neighbourhood van de huidige oplossing gekozen, welke de grootste verbetering in de fitness functie met zich mee brengt. Indien geen enkele oplossing in de neighbourhood tot verbetering leidt, stopt het algoritme.
Een andere variant van *Local Search* is *Tabu Search*. Bij Tabu Search wordt een zogenaamde *tabu list* bijgehouden, waar reeds bezochte oplossingen op geplaatst worden.

Bij het kiezen van een nieuwe oplossing mogen geen oplossingen gekozen worden die op de *tabu list* voorkomen.

3.5 Hybride Tabu Search voor het CRP

In [6] wordt een methode gedemonstreerd waarbij *Tabu Search* wordt ingezet voor het maken van roosters voor verpleegkundigen. In deze paragraaf beschrijven we de gebruikte aanpak.

3.5.1 Probleembeschrijving

Het CRP dat in [6] behandeld wordt betreft het roosteren van verpleegkundigen in Belgische ziekenhuizen. De roosters worden doorgaans op maandelijkse basis gemaakt. Op een afdeling werken gemiddeld rond de 20 verpleegkundigen met verschillende kwalificaties..
De verschillende uit te voeren diensten, en bijbehorende vereiste kwalificaties, liggen vooraf vast. Net als bij het rondvaart probleem wordt de nadruk gelegd op de eerlijkheid van het rooster. Ook hier spelen diverse factoren een rol bij de kwaliteit van een rooster, denk bijvoorbeeld aan een eerlijke verdeling van nachtdiensten. Ook hebben de verschillende verpleegkundigen uiteenlopende persoonlijke voorkeuren, die bij het maken van het rooster in beschouwing genomen dienen te worden.

3.5.2 Model Aannamen

Op basis van hun kwalificaties worden de verpleegkundigen opgedeeld in categorieën. De diensten worden gemodelleerd als combinatie van dagdeel (ochtend, middag of avond) en datum. Een typische dienst kan dus zijn: “ochtenddienst, maandag 23 januari”. Voor de diensten ligt vast hoeveel medewerkers uit elke categorie er vereist zijn. In tegenstelling tot het in hoofdstuk 1 geïntroduceerde model, zijn de diensten hier dusdanig gemodelleerd dat er meerdere verpleegkundigen aan één dienst toegewezen dienen te worden. Dit heeft het voordeel dat er automatisch afgedwongen wordt dat verpleegkundigen niet twee keer op hetzelfde tijdstip ingeroosterd worden.
De eis dat aan iedere dienst het juiste aantal medewerkers met de juiste kwalificaties toegewezen wordt, is de enige harde conditie die gebruikt wordt, alle overige (zachte) condities worden opgenomen in de fitness functie.
Deze fitness functie is, net als bij het in paragraaf 3.3 besproken genetische algoritme, gebaseerd op het aantal overschrijdingen van de verschillende zachte condities. Voor het samenstellen van de fitness functie wordt een generiek raamwerk gebruikt [3]. Hierin worden de zachte condities generiek gemodelleerd op basis van een aantal kernvariabelen, waardoor het mogelijk is om eenvoudig condities toe te voegen voor een specifiek probleem. In het volgende hoofdstuk gaan we nader op dit raamwerk in.

3.5.2 Initialisatie

Bij de initialisatie van het algoritme wordt een valide startoplossing gegenereerd. Dit gebeurt op één van de volgende 3 manieren

1. *Huidige Rooster*
   Deze optie wordt gebruikt indien er een wijziging is ontstaan in de situatie (bijvoorbeeld door het wegvallen van een verpleegkundige).

2. *Rooster Vorige Periode*
   In sommige gevallen kan het handig zijn om bij het maken van een rooster voor een nieuwe periode te starten met het rooster van de vorige periode. Dit is met name het geval indien het rooster van de vorige periode van goede kwaliteit is, en er weinig veranderd is ten opzichte van deze periode.

3. *Willekeurig Rooster*

Indien methode 1 en 2 niet geschikt zijn, wordt er een startoplossing op basis van volledige willekeur gegenereerd.

Methode 1 en 2 lijken in eerste instantie aantrekkelijker, maar brengen het gevaar met zich mee dat het algoritme snel vast blijft zitten in een lokaal optimum. Hiernaast is in de praktijk gebleken dat het algoritme ook prima in staat is om een goede oplossing te genereren vanuit een willekeurig gegenereerde startoplossing.

3.5.3 Tabu Search

De *neighbourhood* van een zekere oplossing bestaat uit alle valide oplossingen waarbij exact één toewijzing van een dienst anders is. In iedere iteratie wordt als nieuwe huidige oplossing simpelweg de oplossing met de hoogste fitness functie uit de *neighbourhood* gekozen. Uiteraard mogen er geen oplossingen uit de *tabu list* gekozen worden. Deze *tabu list* wordt samengesteld door na het kiezen van een nieuwe oplossing te kijken naar de veranderde dienst: alle oplossingen die, ten opzichte van de nieuwe huidige oplossing, een andere waarde hebben voor deze (en nabije) dienst(en), worden (tijdelijk) op de *tabu list* geplaatst. Op deze manier wordt het algoritme gedwongen steeds verschillende toewijzingen aan te passen, waardoor het zoekgedrag van het algoritme minder lokaal wordt.
Merk op dat het algoritme een nieuwe oplossing ook accepteert wanneer dit in een verslechtering van de huidige oplossing resulteert. Er wordt pas gestopt indien er na een bepaald aantal iteraties geen verbeteringen meer gevonden zijn ten opzichte van het tot dan toe best gevonden rooster. Dit verkleint de kans dat het algoritme vast blijft zitten in een lokaal optimum. Bij het testen van het Tabu Search algoritme is, ter vergelijking, ook een test gedaan met een Steepest Descent algoritme (zie paragraaf 3.4), dat stopt indien er geen betere oplossing in de *neighbourhood* aanwezig is. De resultaten van één van de de tests zijn te vinden in paragraaf 3.5.6.

3.5.4 Heuristische Toevoegingen

Het beschreven algoritme produceert redelijke oplossingen, maar er zijn een aantal gebieden waarop de oplossingen nog voor verbetering vatbaar zijn. Er worden 3 heuristische methoden beschouwd die hierbij uitkomst zouden kunnen brengen.

1. *Diversificatie 1: Weekenden Eerst*
   Evenals bij het rondvaart probleem spelen ook hier de weekenden een belangrijke rol. Ondanks het feit dat de invloed van de bijbehorende conditie vergroot kan worden door middel van een zogenaamde kostparameter, produceert het *tabu search* algoritme oplossingen waarbij de weekenden niet voldoende benadrukt worden. Bovendien is gebleken dat “oneerlijkheden” in de verdelingen van de weekenden meer in het oog springen dan andere overschrijdingen van zachte condities.
   Dit wordt veroorzaakt door het hoge aantal condities; vele condities met lage kostparameters wegen samen op tegen één conditie met een hoge kostparameter.
   In de praktijk is gebleken dat het onmogelijk is om parameter instellingen te vinden die goede oplossingen voor de weekends garanderen. Om deze reden kan er voor worden gekozen de weekenden eerst in te roosteren, en de gevolgen hiervan op de overige condities te negeren.

2. *Diversificatie 2: Slechtste Persoonlijke Rooster*
   Bij de gebruikte evaluatie methode [3] wordt het rooster van iedere medewerker afzonderlijk beoordeeld. Het idee achter deze heuristiek is dat verbeteringen in het rooster van deze “slechtste medewerker” ook leiden tot verbeteringen in het totale rooster.
   Voor iedere medewerker (uit dezelfde categorie) wordt gekeken naar het effect van het omwisselen van alle toewijzingen uit een gedeelte van het rooster (altijd minder dan de halve planningsperiode). Alle mogelijkheden worden bekeken (iets wat redelijk

tijdrovend is), en de omwisseling die leidt tot de grootste verbetering voor de fitness functie wordt toegepast.

3. *Greedy Shuffling*
   Bij het toepassen van het oorspronkelijke *tabu search* algoritme op een aantal test datasets, is gebleken dat (menselijke) roostermakers in veel gevallen, op basis van een visuele weergave van het rooster, in staat waren om door middel van een aantal kleine wijzigingen verbeteringen aan te brengen in het rooster. Een methode om dit te voorkomen is *Greedy Shuffling*. Dit is een zogenaamde *Brute Force* methode, die alle "kleine" wijzigingen "probeert", en de beste toepast.
   Analoog aan de beschreven methode bij diversificatie 2, worden voor paren medewerkers alle mogelijke omwisselingen van gedeelten van het rooster en de bijbehorende waarde van de fitness functie berekend. Dit wordt gedaan voor alle combinaties van medewerkers uit de overeenkomstige categorie. Vervolgens worden alle mogelijke wijzigingen, die een verbetering vormen, in een lijst geplaatst en aflopend gesorteerd op de waarde van de fitness functie van het resulterende rooster. De eerste wijziging uit de lijst wordt toegepast en uit de lijst verwijderd. Ook alle andere wijzigingen, waar één van dezelfde medewerkers bij betrokken is, worden uit de lijst verwijderd. De wijzigingen op de lijst worden op deze manier toegepast totdat de lijst leeg is.
   Dit proces (bepalen/evalueren mogelijke omwisselingen en "afwerken" van de lijst met omwisselingen) wordt net zo lang herhaald totdat er geen enkele verbetering meer gevonden wordt.
   Het mag duidelijk zijn dat dit een tijdrovend proces is, maar tests hebben uitgewezen dat het toepassen van deze methode aanzienlijke verbeteringen kan aanbrengen in de gevonden oplossingen van het *Tabu Search* algoritme. Een groot deel van deze verbeteringen uiten zich niet direct in de fitness functie, maar in het feit dat het maken van verbeteringen met de hand praktisch onmogelijk geworden is.

3.5.5 Twee Hybride Algoritmen

Indien een heuristische methode wordt gecombineerd met een ander algoritme spreekt men van een *Hybride Algoritme*. Na uitvoerig testen van (combinaties van) de drie beschreven heuristische methoden in combinatie met het beschreven *Tabu Search* algoritme, worden in [6] uiteindelijk 2 hybride algoritmen voor het verpleegkundigen CRP gepresenteerd.

1. *TS1: Tabu Search met Diversificatie 1 en 2*
   Het eerste algoritme combineert *Tabu Search* met de twee beschreven diversificatie methoden. Indien het Tabu Search algoritme al een aantal iteraties lang geen verbetering gevonden heeft, wordt diversificatie 1 toegepast. Indien diversificatie 1 niet tot een gewijzigde oplossing leidt, wordt diversificatie 2 toegepast. Het algoritme stopt indien na een (groter) aantal iteraties geen verbetering gevonden is.

2. *TS2: Tabu Search met Greedy Shuffling*
   Bij dit algoritme wordt eerst het standaard Tabu Search algoritme uitgevoerd. Op de gevonden oplossing wordt vervolgens de beschreven Greedy Shuffling methode toegepast.

Het tweede algoritme produceert betere oplossingen, maar is veel tijdrovender. In de praktijk is het eerste algoritme zeer geschikt om te analyseren in welke mate aan de condities voldaan kan worden, en te experimenteren met de kostparameters. Vervolgens kan TS2 gebruikt worden voor het genereren van de echte oplossing, die in de praktijk gebruikt zal worden.

3.5.6 Resultaten

De beschreven algoritmen zijn in [6] uitvoerig getest. Als testdata zijn twee problemen uit de praktijk gebruikt. In de onderstaande tabel worden resultaten van één van de tests weergegeven.

Algoritme	Fitness Functie	Rekentijd (minuten)
Steepest Descent	2594	1:26
Tabu Search (kort)	2435	2:05
Tabu Search (lang)	1915	40:58
TS1	1341	6:00
TS2	1264	20:15

Tabel 2: Testresultaten van verschillende Local Search Algoritmen

De absolute waarden van de fitness functie hebben voor ons (door gebrek aan context) uiteraard weinig betekenis, maar we kunnen wel de uitkomsten van de verschillende algoritmen vergelijken. In het experiment is het oorspronkelijke Tabu Search algoritme uitgevoerd met twee verschillende instellingen voor het stopcriterium. In de korte variant is het stopcriterium op een dusdanige manier gekozen dat de resulterende rekentijd in dezelfde orde van grootte ligt als bij het *Steepest Descent* algoritme. Bij de lange variant is een extreem ruime stopconditie gekozen, om te zien in welke mate de oplossing verbetert bij het opvoeren van de rekentijd.

In tabel 2 zien we dat beide varianten van *Tabu Search* in deze test beter presteerden dan *Steepest Descent* (een lagere waarde voor de fitness functie correspondeert met minder overschrijdingen van condities). Ook zien we dat de hybride varianten TS1 en TS2 een significante verbetering brengen ten opzichte van oorspronkelijke *Tabu Search*. Dezelfde constateringen werden bij alle uitgevoerde tests gemaakt.

De gevonden oplossingen van TS2 waren allen beter dan die van TS1, maar de verschillen in de fitness functie zijn niet heel groot. Toch is in de praktijk gebleken dat de roosters gegenereerd door TS2 als veel beter ervaren worden.

De algoritmen TS1 en TS2 zijn geïmplementeerd in commercieel pakket (genaamd Plane), dat succesvol ingezet wordt in een aantal Belgische ziekenhuizen.

3.6 Sterke en Zwakke Eigenschappen

Over het algemeen zijn de heuristische zoekmethoden in staat om binnen relatief beperkte (reken)tijd, oplossingen van voldoende kwaliteit te genereren. Er is echter geen garantie voor een optimale oplossing. Een bekend probleem van deze methoden is echter dat ze soms de neiging vertonen “vast te blijven zitten” op een lokaal optimum. De mate waarin dit een probleem vormt, is sterk afhankelijk van de situatie.

Het merendeel van de zoekmethoden biedt de mogelijkheid om zeer complexe factoren in beschouwing te nemen bij de condities en fitness functie van het model. Ook het toevoegen van condities en het uitbreiden van de fitness functie kan doorgaans zonder problemen plaatsvinden.

Dit is een sterke eigenschap van de methoden, maar brengt ook een aantal moeilijkheden met zich mee. De kwaliteit van de fitness functie is cruciaal voor de kwaliteit van de oplossingen die de methoden produceren. Naarmate de omvang en complexiteit van de condities/fitness funtie toeneemt, wordt ook het vinden van (de parameters voor) een goede fitness functie moeilijker. In het volgende hoofdstuk gaan we hier nader op in.

Bij veel van de beschikbare algoritmen moet, vaak op een aantal plaatsen, een keuze gemaakt worden uit verschillende beschikbare submethoden (denk bijvoorbeeld aan de keuze van een selectie methode in een Genetisch Algoritme). Hiernaast kan het aantal in te stellen parameters aanzienlijk meegroeien met de omvang van het probleem. Welke submethoden en parameter instellingen de beste oplossing genereren kan in de praktijk

sterk situatie afhankelijk zijn. Dit betekent dat er bij het maken van een rooster de nodige tijd besteedt moet worden aan het experimenteren met deze parameters en submethoden.
De zoekmethoden hebben doorgaans de eigenschap dat ze relatief eenvoudig, en dus goedkoop, te implementeren zijn. Hier komt bij dat, doordat de zoekmethoden veelal hetzelfde concept volgen, het niet moeilijk is verschillende zoekmethoden in één systeem te implementeren.
In de onderstaande tabel worden de sterke en zwakke eigenschappen nogmaals op een rij gezet.

Sterke Eigenschappen	Zwakke Eigenschappen
• redelijke oplossingen in beperkte tijd • complexe fitness functie / condities • flexibel model • eenvoudig te implementeren en uit te breiden • eenvoudig meerdere methoden in één systeem te verwerken • intuïtief karakter	• geen optimale oplossing • veel parameters om te “tweaken” • gebruik vereist zekere mate van inzicht in de zoekmethode • gevaar lokaal optimum

Tabel 3: Sterke en zwakke eigenschappen van heuristische zoekmethoden als oplossingsmethode voor het CRP

3.7 Het Rondvaart Probleem

In deze paragraaf analyseren we de mate waarin de Heuristische Zoekmethoden toegepast kunnen worden op het rondvaart probleem. Om te beginnen merken we op dat, in theorie, alle in paragraaf 1.4.6 genoemde factoren (die een rol spelen bij de kwaliteit van het rooster) in het model opgenomen kunnen worden. In paragraaf 4.1 laten we zien hoe deze factoren meegenomen kunnen worden bij de evaluatie van een rooster. In dit opzicht zijn de heuristische zoekmethoden zeer geschikt voor het rondvaart probleem, waarbij een relatief groot aantal, complexe condities spelen. Ook de overige genoemde sterke eigenschappen van deze methoden spreken erg aan, zowel bij de rondvaart organisatie als de leverancier, die de uiteindelijke roosterapplicatie moeten implementeren.
Omdat er geen kosten in het spel zijn, is het minder van belang dat er een exacte oplossing gevonden wordt, en vormt deze beperking van de zoekmethoden dus geen groot bezwaar. De roosters worden voor een relatief lange periode gemaakt, en zoals aangegeven zijn de roostermakers de nodige hoeveelheid handwerk gewend. Uiteraard willen we met de uiteindelijke oplossing de hoeveelheid handwerk verminderen, maar we hebben hier dus de nodige speling, en het experimenteren met verschillende instellingen wordt niet als een probleem beschouwd.
Hiermee rest de vraag in welke mate de zoekmethoden in staat zijn kwalitatief goede oplossingen voor het rondvaart probleem te produceren. Hierover zijn we optimistisch; de structuur van het rondvaart probleem lijkt in veel opzichten op die van het in [6] besproken verpleegkundigen probleem.

3.7.1 Conclusie

De sterke en zwakke punten van de heuristische zoekmethoden sluiten erg goed aan bij de specifieke eigenschappen van het rondvaart probleem en de betrokken partijen. Er wordt in de praktijk dan ook gekozen voor een oplossingsmethode uit deze categorie. De mogelijkheid om alle factoren die in de praktijk een rol spelen op te nemen in het model, en de flexibiliteit van de methoden geven de doorslag bij deze beslissing. Vanwege de cruciale rol van de fitness functie bij deze methoden, wordt er in eerste instantie een project opgestart waarbij de aandacht grotendeels ligt op het implementeren van een fitness functie. Tevens zal er een relatief eenvoudige, intuïtieve heuristiek geïmplementeerd worden, die gebruik

maakt van deze fitness functie. In het volgende hoofdstuk wordt de gekozen oplossingsmethode verder uitgewerkt.

Hoofdstuk 4: Oplossingsmethode Rondvaart Probleem

We hebben gekozen om in eerste instantie de nadruk te leggen op het opstellen van een fitness functie voor het rooster probleem. In paragraaf 4.1 beschrijven we een generiek bruikbaar raamwerk voor rooster evaluatie. In paragraaf 4.2 wordt vervolgens omschreven hoe dit raamwerk ingezet kan worden voor het rooster probleem. In paragraaf 4.3 beschrijven we een simpel algoritme om roosters mee te maken. In paragraaf 4.4 motiveren we de keuze voor dit algoritme, en bekijken we de korte termijn plannen omtrent de automatisering van het rondvaart probleem.

4.1 Een Raamwerk voor Rooster Evaluatie

Het bepalen van de kwaliteit van een gegeven oplossing, is een cruciale stap in de in hoofdstuk 3 beschreven algoritmen. In [3] wordt een generiek raamwerk opgesteld om roosters mee te evalueren. In deze paragraaf bespreken we de gevolgde aanpak.
De fitness van een oplossing wordt bepaald aan de hand van het aantal overschrijdingen van zachte condities (net als bij het in paragraaf 3.3 besproken genetische algoritme). Het raamwerk biedt een generieke methode waarmee een breed scala aan zachte condities gemodelleerd kan worden, en een iteratief algoritme waarmee overschrijdingen van deze condities, per medewerker, geteld kunnen worden. Er wordt dus voor iedere afzonderlijke medewerker een beoordeling van het rooster gemaakt. Deze afzonderlijke beoordelingen worden vervolgens gebruikt om tot een eindoordeel over het gehele rooster te komen.

4.1.1 Time Units

De methode werkt met zogenaamde *time units*. Deze *time units* verdelen iedere dag uit de planningsperiode in blokken van uit te voeren soorten diensten. Het totaal aantal *time units* is dus gelijk aan het aantal dagen in de planningsperiode, vermenigvuldigd met het aantal soorten diensten. De verzameling time units wordt genoteerd als T.
Het concept van de time units laat zich het beste uitleggen aan de hand van een voorbeeld. Hiervoor gebruiken we het rondvaart probleem. We categoriseren de diensten op basis van de eigenschappen die we willen meenemen in de evaluatie.
Voor de “vaardiensten” zijn dit het dagdeel van de betreffende dienst (ochtend, middag of avond), de rol van de medewerker (schipper of spreker) en de betreffende boot (W1 t/m W7). Dit levert ons dus 42 (3 dagdelen * 2 rollen * 7 boten) soorten diensten. Hierbij komen de walschipper en (twee) stand-by diensten er nog bij, wat ons al met al 45 time units per dag oplevert. Om het voorbeeld overzichtelijk te houden beschouwen we de situatie waarbij er slechts één boot aanwezig is, en laten we de walschipper en stand-by diensten buiten beschouwing; dit levert ons dus 6 time units per dag op. In de onderstaande figuur worden de time units van twee achtereenvolgende dagen weergegeven.

Dag j						Dag j + 1					
Ochtend Spreker W1	Ochtend Schipper W1	Middag Spreker W1	Middag Schipper W1	Avond Spreker W1	Avond Schipper W1	Ochtend Spreker W1	Ochtend Schipper W1	Middag Spreker W1	Middag Schipper W1	Avond Spreker W1	Avond Schipper W1

Figuur 4: Indeling in time units van het rondvaart probleem met 1 boot

4.1.2 Persoonlijk Rooster

Een rooster kan worden gerepresenteerd op basis van deze time units, door voor iedere medewerker, voor iedere time unit, aan te geven of de medewerker in deze time unit “used” of “idle” is. Het persoonlijke rooster van een willekeurige medewerker *p* kan gezien worden als een functie van een time unit naar een binaire variabele. Een 1 betekent dat de medewerker in de betreffende time unit “used” is, en een 0 dat dit niet het geval is.

Persoonlijk rooster
S[p]: T → {0,1}.

In figuur 5 wordt een voorbeeld van een rooster voor twee dagen gegeven, waarbij de verschillende diensten verdeeld zijn over 4 medewerkers.

	Dag j						Dag j + 1					
	Ochtend Spreker W1	Ochtend Schipper W1	Middag Spreker W1	Middag Schipper W1	Avond Spreker W1	Avond Schipper W1	Ochtend Spreker W1	Ochtend Schipper W1	Middag Spreker W1	Middag Schipper W1	Avond Spreker W1	Avond Schipper W1
Medewerker 1	1	0	0	1	0	0	0	1	1	0	0	0
Medewerker 2	0	1	1	0	0	0	1	0	0	1	0	0
Medewerker 3	0	0	0	0	1	0	0	0	0	0	0	1
Medewerker 4	0	0	0	0	0	1	0	0	0	0	1	0

Figuur 5: Time unit representatie van een rooster

4.1.3 Numberings
Om de verschillende condities generiek te evalueren, maakt de methode gebruik van zogenaamde numberings. Formeel is een numbering een functie van een time unit naar een nummer:

Numbering
N[i]: T → {-M, -M + 1, …, M -1, M, *U*}.

Hierbij is i ε {1, 2, …, I), en I het totaal aantal numberings. M is een positief, geheel getal, en *U* representeert de time units waarvoor de numbering ongedefinieerd is. Een numbering kan gezien worden als een manier om de time units te tellen, gebaseerd op een specifieke eigenschap van de betreffende time unit. In figuur 6 worden drie voorbeelden van numberings gegeven, over een periode van 4 dagen. De eerste numbering “telt” de verschillende dagen, de tweede telt de weekenden, en de derde telling de ochtend diensten.

	Dag j (vrijdag)						Dag j + 1 (zaterdag)					
	Ochtend Spreker W1	Ochtend Schipper W1	Middag Spreker W1	Middag Schipper W1	Avond Spreker W1	Avond Schipper W1	Ochtend Spreker W1	Ochtend Schipper W1	Middag Spreker W1	Middag Schipper W1	Avond Spreker W1	Avond Schipper W1
Numbering N[1]	0	0	0	0	0	0	1	1	1	1	1	1
Numbering N[2]	U	U	U	U	U	U	0	0	0	0	0	0
Numbering N[3]	0	0	U	U	U	U	1	1	U	U	U	U

	Dag j + 2 (zondag)						Dag j + 3 (maandag)					
	Ochtend Spreker W1	Ochtend Schipper W1	Middag Spreker W1	Middag Schipper W1	Avond Spreker W1	Avond Schipper W1	Ochtend Spreker W1	Ochtend Schipper W1	Middag Spreker W1	Middag Schipper W1	Avond Spreker W1	Avond Schipper W1
Numbering N[1]	2	2	2	2	2	2	3	3	3	3	3	3
Numbering N[2]	1	1	1	1	1	1	U	U	U	U	U	U
Numbering N[3]	2	2	U	U	U	U	3	3	U	U	U	U

Figuur 6: Drie numberings over 4 dagen

Voor een gegeven medewerker *p*, noemen we een time unit *e* een event indien de medewerker in deze time unit "used" is. Met ES[p] noteren we de totale verzameling van events behorende bij persoonlijk rooster van persoon S[p], en met E[N[i]][S[p]] de verzameling van events van S[p], die gedefinieerd zijn in telling N[i]:

$ES[p] = \{e: S[p](e) = 1\},$

$E[N[i]][S[p]] = \{e: S[p](e) = 1, N[i](e) \neq U\}.$

4.1.4 Numbering Constraints

De condities van het probleem worden gemodelleerd door middel van acht verschillende typen *numbering constraints*. Een *numbering constraint* bestaat uit een type, een waarde en een *numbering* waar de conditie op gedefinieerd is.

1. max_total
   Dit eerste *numbering constraint* type stelt een bovengrens aan het totaal aantal *events* in *numbering* N[i] binnen het rooster van medewerker *p*. De waarde van max_total dient een geheel getal te zijn.

   # E[N[i]][S[p]] ≤ max_total.

   Dit type numbering constraints kan gebruikt worden om allerlei verschillende condities mee te modelleren. Een typische toepassing is het gebruik van constraints van dit type om een bovengrens te stellen aan de totale hoeveelheid werk die aan een medewerker toegekend wordt.

2. min_total
   Dit type stelt een ondergrens aan het totaal aantal *events* in *numbering* N[i] binnen het rooster van medewerker *p*. Net als bij max_total, dient de waarde een geheel getal te zijn.

   # E[N[i]][S[p]] ≥ min_total.

3. max_pert
   De waarde van max_pert bevat een rij getallen, genoteerd als max_pert[a], voor alle a = 0, 1, …, *M*. De waarde max_pert[a] is een bovengrens voor het aantal events dat plaatsvindt in de time units die in de betreffende numbering waarde a hebben:

   #{e ε E[N[i]][S[p]]: N[i](e) = a} ≤ max_pert[a], voor alle a = 1, 2, …, *M.*

   In combinatie met numbering 2 uit figuur 6 kan dit type bijvoorbeeld gebruikt worden om te zorgen dat een medewerker slechts een beperkt aantal weekenddiensten per week kan uitvoeren.

4. min_pert
   Dit type is geheel analoog aan max_pert, met het verschil dat het hier een ondergrens betreft in plaats van een bovengrens. Dit type *numbering constraint* kan gebruikt worden om een zekere mate van spreiding van werk over de periode af te dwingen, bijvoorbeeld door een *numbering constraint* te maken die zorgt dat iedere medewerker ten minste een bepaald aantal diensten per week of maand moeten uitvoeren.

Voordat we de volgende typen bespreken, bespreken we eerst de volgende definitie:

Definitie 2:
We noemen twee getallen, a en b (met a ≤ b) *consecutive* in relatie tot numbering N[i] en medewerker *p* dan en slechts dan als voor ieder nummer *m* uit {a, …, b} numbering N[i] een event uit E[N[i]][S[p]] naar m projecteert.

5. max_consecutive
   Dit type beperkt het maximaal aantal achtereenvolgende events van medewerker *p* binnen een specifieke *numbering* N[i]. Een typisch voorbeeld is het beperken van het aantal achtereenvolgende nachtdiensten bij verpleegkundigen.

6. min_consecutive
   Dit type is geheel analoog aan max_consecutive, maar stelt een ondergrens (in tegenstelling tot een bovengrens.

7. max_between
   Dit type stelt een bovengrens aan de periode tussen twee *non-consecutive* events, en kan bijvoorbeeld gebruikt worden om te het werk van de medewerkers te verspreiden over de planningsperiode.

8. min_between
   Wederom geheel analoog aan max_between, maar dan een ondergrens. Dit type is bijvoorbeeld bruikbaar zwaar fysiek werk, of werk waarbij de medewerkers blootgesteld worden aan schadelijke stoffen, en kan ervoor zorgen dat medewerkers niet twee van deze diensten achter elkaar doen.

Voor een gegeven rooster kunnen overschrijdingen van de verschillende *numbering constraints* met een eenvoudig, iteratief algoritme geteld worden. De uitwerking van dit algoritme laten we hier achterwegen, hiervoor verwijzen we naar [3].

4.2 Evaluatie voor het Rooster Probleem

In deze paragraaf laten we zien hoe het in paragraaf 4.1 beschreven raamwerk gebruikt kan worden voor het rondvaart probleem. We gebruiken de in paragraaf 4.2.1 geïntroduceerde opdeling in time units (waarbij iedere dag ingedeeld wordt in 45 time units (42 voor de vaarten, 1 voor de walschipper dienst en 1 voor beide stand-by diensten).
In paragraaf 1.4.6 van hoofdstuk 1 hebben we de factoren die de kwaliteit van het rooster bepalen gedefinieerd. We beschrijven voor ieder van deze factoren hoe deze vertaald kunnen worden in *numbering constraints*, gebruik makend van de beschreven *time units*.

4.2.1 Verdeling hoeveelheid werk

Het doel is het werk zo eerlijk mogelijk te verdelen, op een dusdanige wijze dat er rekening wordt gehouden met de opgegeven voorkeuren van de medewerkers (hoeveelheid gewenste diensten). Dit kan worden gerealiseerd door beperkingen (minimum en maximum) te stellen aan het totale aantal diensten van een medewerker. Dit doen we door twee numbering constraints op te stellen van het type min_total en max_total.
Om rekening te houden met de persoonlijke voorkeuren van de medewerkers, willen we dat de waarde van deze twee numbering constraints verschillend is per medewerkers. Het in paragraaf 4.1 gepresenteerde raamwerk is eenvoudig aan te passen om hier mee om te gaan door de waarde van een numbering constraint niet als constante op te nemen, maar als functie van een medewerker.
We bepalen van de waarde van de max_total en min_total numbering contraints voor een zekere medewerker op basis van de “ideale hoeveelheid werk” van deze medewerker. Deze ideale hoeveelheid wordt bepaald door de totale hoeveelheid werk proportioneel (in relatie tot de opgegeven gewenste hoeveelheden) te verdelen over de medewerkers. Vervolgens bepalen we de waarde voor de constraints door middel van een marge op de ideale

hoeveelheid werk. Door deze marge te variëren kunnen we dus de invloed van deze constraints op het rooster veranderen.
Voor de constraints hebben we een numbering nodig waarmee we het totaal aantal “beurten” kunnen tellen. Merk op dat dit aantal beurten niet gelijk is aan het totaal aantal ingeroosterde diensten, omdat we de medewerkers verschillende voorkeuren kunnen opgeven met betrekking tot hoe de walschipper diensten worden meegeteld. Dit lossen we op door, analoog aan de waarde van de constraints, ook de numberings medewerker specifiek te maken. Afhankelijk van de voorkeuren van de medewerker, wordt er voor de numbering constraints een numbering gebruikt die de walschipper diensten al dan niet mee telt

4.1.2 Spreiding over de periode
Voor het meten van de spreiding over de periode gebruiken we een vergelijkbare aanpak als bij paragraaf 4.2.1: we bepalen eerst de ideale situatie, en kijken vervolgens in hoeverre het rooster daarvan afwijkt.
We meten de spreiding over de periode door te kijken naar het aantal diensten per week van de medewerkers. Het ideale aantal wordt bepaald door het aantal ingeroosterde diensten te delen door het aantal weken in de planningsperiode. Door middel van een marge worden de waarden voor het minimale en maximale aantal diensten per week bepaald.
Deze waarden worden vervolgens gebruikt in twee numbering constraints, van het type min_pert en max_pert. De constraints zijn gedefinieerd op een numbering die de weken telt.
Door de gebruikte numbering medewerker specifiek te maken, kan gezorgd worden dat de opgegeven vakantie perioden niet meegeteld worden. Door de gekozen periode (1 week) en de marge te variëren kan de invloed van deze conditie op de oplossing veranderd worden.

4.2.3 Variatie in Routes en Boten
Omdat er in de gekozen opdeling in time units geen informatie over de routes opgenomen is, kunnen we geen numbering constraints formuleren die zorgen dat de oplossing rekening houdt met variatie in de route’s. In theorie kunnen we de time unit oplossing uitbreiden met informatie over de routes door de time units verder op te splitsen op basis van de verschillende mogelijke routes. Echter, zoals aangegeven in paragraaf 1.4.6 wordt variatie in de routes ook in zekere mate gegarandeerd door variatie in de boten. We kiezen er in eerste instantie voor om dit uit te buiten, en ons te concentreren op variatie in de boten.
Wederom kiezen we voor een aanpak waarbij we uitgaan van een ideale situatie, en analyseren in hoeverre het rooster daarvan afwijkt. Hiervoor beschouwen we voor iedere medewerker het aantal toegewezen diensten per boot. Voor een gegeven boot bepalen we het ideale aantal door het totaal aantal ingeroosterde vaardiensten van de medewerker proportioneel te verdelen over de boten. De proporties bij dit verdelen worden bepaald op basis van de totale aantallen vaardiensten per boot.
Voor iedere boot wordt er vervolgens numbering constraints van het type min_total en max_total toegevoegd. De waarden worden opnieuw bepaald door middel van een marge op het ideale aantal diensten voor de betreffende boot. Iedere constraint is gebaseerd op een numbering die de vaardiensten op de betreffende boot telt.

4.2.4 Variatie in Collega’s
Net als voor de routes, geldt ook voor de collega’s dat deze niet in het time unit schema zijn opgenomen. In tegenstelling tot de routes, is dit voor de collega’s echter ook niet mogelijk, omdat de ingeroosterde collega’s geen statische eigenschap van de diensten zijn. Om deze factor toch mee te nemen in de evaluatie is dus ‘maatwerk’ nodig. Het is echter relatief eenvoudig om een algoritme te ontwikkelen en te implementeren dat voor iedere medewerker het totaal aantal verschillende collega’s medewerkers telt. Op basis van deze telling in relatie tot het aantal ingeroosterde vaarten wordt de penalty cost voor de variatie in collega’s bepaald.

4.2.5 Weekenden

Het beoordelen van de weekenden gebeurt op twee aspecten. Ten eerste wordt er gekeken naar de kwantitatieve verdeling van de weekend diensten. Dit gebeurt op een vergelijkbare manier als bij het totaal aantal diensten, alleen wordt er in dit geval geen rekening gehouden met het opgegeven gewenste aantal wekelijkse diensten; de weekend diensten dienen immers zo evenwichtig mogelijk verdeeld te worden. Opnieuw worden er twee numbering constraints van het type min_total en max_total gebruikt, gedefinieerd op een numbering die de weekend diensten telt.
Het tweede aspect waar de verdeling van de weekend diensten op beoordeeld wordt, is de mate waarin rekening gehouden is met de persoonlijke voorkeur van de medewerkers om al dan niet volledige weekenden te werken. Dit aspect nemen we mee door te kijken naar het aantal diensten per weekend. We stellen een numbering constraint van het type max_pert op, en definiëren deze op een numbering die de weekenden telt, en geven de constraint de waarde 1. Het idee is dat we het overschreiden van deze contraint verschillend waarderen, afhankelijk van de opgegeven voorkeur van de betreffende medewerker. Indien deze medewerker de voorkeur geeft aan volledige weekenden, kennen we 'negatieve kosten' toe aan het overschrijden van de constraint. Overschrijdingen hebben in dit geval dus een positieve invloed op de fitness functie.
Hiervoor moeten we het raamwerk opnieuw enigszins aanpassen: ook de penalty cost voor het overschrijden van de verschillende numbering constraints beschouwen we als een functie van de medewerker.

4.2.6 Dubbele Diensten

De dubbele diensten kunnen op vergelijkbare wijze als de volledige weekenden gemodelleerd worden. We stellen een numbering contraint op van het type max_pert met waarde 1 en definiëren deze op de numbering die de weekdagen telt. Net als bij de weekenden, wordt de waardering van het overschrijden van deze constraint afhankelijk van de medewerker gemaakt.

4.2.7 Verdeling Walschipper en Stand-by Diensten

In paragraaf 1.4.6 worden vier mogelijke behandelingswijzen met betrekking tot de walschipper diensten gegeven. Mogelijkheid a en b worden al in beschouwing genomen bij de evaluatie van de verdeling van de totale hoeveelheid werk, zoals omschreven in paragraaf 4.1.1. Mogelijkheid c en d kunnen in het model meegenomen worden door de walschipper en vaardiensten als (vereiste) vaardigheden te modelleren, en worden dus meegenomen bij de harde condities van het model.
We zullen de verdeling van de walschipper diensten daarom beoordelen op de evenwichtigheid van de verdeling. Dit geldt ook voor de stand-by diensten. We doen dit op vergelijkbare wijze als bij de verdeling van de totale hoeveelheid werk. Het ideale aantal walschipper / stand-by diensten wordt bepaald door middel van het totale aantal walschipper /stand-by diensten te delen door het totaal aantal medewerkers. Vervolgens wordt er door middel van een marge een onder en bovengrens opgesteld, welke gebruikt worden als waarden in numbering constraints van het type min_total / max_total gedefinieerd op numberings die de walschipper / stand-by diensten telt.

4.3 Een Iteratieve Heuristiek

In deze paragraaf geven we een beknopte beschrijving van het algoritme dat in eerste instantie geïmplementeerd zal worden. Het algoritme is gebaseerd op de zogenaamde Bottleneck Value heuristiek. Deze heuristiek gaat als volgt te werk:

1. Bepaal voor iedere dienst hoeveel medewerkers de dienst kunnen uitvoeren. Bepaal, op basis van dit aantal de *Bottleneck Value* van de dienst op een dusdanige wijze dat de diensten met minder beschikbare medewerkers een hogere bottleneck value hebben.

2. Ken één voor één, op volgorde van aflopende bottleneck value, willekeurige medewerkers toe aan de dienst, hierbij rekening houdend met de harde condities van het probleem. Indien er geen medewerkers beschikbaar zijn, wordt er geen medewerker aan de dienst toegewezen.

Het algoritme dat we zullen gebruiken is geïnspireerd door deze bottleneck heuristiek, en kan gezien worden als een verfijnde versie hiervan. Net als bij het beschreven bottleneck value algoritme, worden de diensten geordend op aflopende volgorde van een bottleneck value. Bij het bepalen van deze waarde zal niet alleen rekening gehouden worden met het aantal beschikbare medewerkers, maar ook met de aard van de dienst. Dit heeft in eerste instantie betrekking op de weekend diensten, welke een hoge bottleneck value zullen krijgen.
In tegenstelling tot bij het hierboven beschreven algoritme, wordt niet per definitie de dienst met de hoogste bottleneck value als eerste toegewezen, maar wordt er een zekere mate van willekeur geïntroduceerd door het in paragraaf 3.2.1 beschreven Fitness Proportion Selection toe te passen.
Bij het toewijzen van de dienst wordt de medewerker niet gekozen op willekeurige basis, maar wordt rekening gehouden met de impact van de toewijzing op de kwaliteit van het rooster. Dit doen we door het evaluatie raamwerk toe te passen op deelroosters. We bepalen voor iedere mogelijke toewijzing aan de betreffende dienst, de waarde van de fitness voor het resulterende deelrooster. Vervolgens kiezen we de toewijzing die tot de beste waarde van de fitness functie leidt.
Na het doen van een toewijzing wordt, voor de resterende in te roosteren diensten, de bottleneck value opnieuw bepaald. Op deze manier worden de gevolgen van de gemaakte toewijzingen meegenomen bij het kiezen van de volgende in te roosteren dienst.
Dit proces wordt vervolgens herhaald totdat alle diensten toegewezen zijn, of er geen toewijzingen meer mogelijk zijn zonder harde condities te overschrijden.

4.4 Conclusie en Vooruitblik

Het doel van dit werkstuk was het vinden van een oplossingsmethode die geschikt was voor het rondvaart probleem. We hebben in dit werkstuk oplossingsmethode uit de Operations Research (mathematisch programmeren) en uit de Artificial Intelligence (heuristische zoekmethoden) geanalyseerd, en hebben geconcludeerd dat de heuristische zoekmethoden de meest adequate oplossingsmethode voor het rondvaartprobleem is.
Dit wordt voornamelijk veroorzaakt door het grote aantal “menselijke” factoren die een rol spelen bij het rondvaart probleem. Ook de hoge flexibiliteit van deze oplossingsmethode heeft een rol gespeeld bij deze beslissing.
Bij het implementeren van een geautomatiseerde oplossing zal in eerste instantie de nadruk gelegd worden op het evalueren van roosters. Hiervoor wordt gebruik gemaakt van een generiek raamwerk om condities mee te modelleren. We hebben een eenvoudig heuristisch algoritme om roosters mee te genereren gepresenteerd dat gebruik maakt van het evaluatie raamwerk. Indien de praktijk uitwijst dat oplossingen die op deze methode geproduceerd worden nog van onvoldoende kwaliteit zijn, kunnen we het algoritme uitbreiden met de in paragraaf 3.5.4 beschreven Greedy Shuffling heuristiek, of het in paragraaf 3.5 beschreven hybride tabu search algoritme of een andere vorm van local search proberen.

Referenties

1. A.T. Ernst, H. Jiang, M. Krishnamoorthy, D. Sier, “Staff Scheduling and rostering: A review of applications, methods and models” European Journal Of Operations Research Vol. 153 Issue 1, 2004

2. R. Aringhieri, A. Ceselli, R. Cordone, “Models and Algorithms for Balanced Rostering With Limited Skills”, 2005

3. E.K. Burke, P. De Causmaecker, S. Petrovic, G. Vanden Berghe, “Fitness Evaluation for Nurse Scheduling Problems”, Proceedings of CEC2001, 2001

4. E.K. Burke, P. De Causmaecker, S. Petrovic, G. Vanden Berghe, “A Multi Criteria Meta-heuristic Approach to Nurse Rostering”, Proceedings of CEC2002, 2002

5. M. Gröbner, P. Wilke, “Rostering with a Hybrid Genetic Algorithm”, Proceedings of Fifth International Conference on Artificial Neural Networks and Genetic Algorithms, 2001

6. E.K. Burke, P. De Causmaecker, G. Vanden Berghe, “A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem”, SEAL1998 ,1998

7. H. Chuin LAU, S. Chong LAU, “Efficient Multi-Skill Crew Rostering via Constrained Sets”

8. D. E. Goldberg, “Genetic Algorithms in Search, Optimization and Machine Learning”, 1989

9. H.C. Tijms, A.A.N. Ridder, “Mathematische Programmering”, 2001