Dienstroosters bij Connexxion

EEN TOEPASSING VAN ZELFROOSTERING

Arthur Bosma

1 september 2010

Inhoudsopgave

Inhoudsopgave 3

Voorwoord 7

Samenvatting 9

1 Inleiding 11

1.1 Planningsproces in een busmaatschappij . . . 11

1.2 De matching . . . 12

1.2.1 Zelfroostering . . . 13

1.2.2 Blokindeling . . . 13

1.3 Harde en zachte eisen . . . 13

1.4 Opbouw van het verslag . . . 15

2 Probleemomschrijving 17

2.1 Kenmerken van het bloksysteem . . . 17

2.1.1 De indeling van diensten bij blokken . . . 17

2.1.2 Roostervrije dagen en verlofdagen . . . 17

2.2 Het maken van een matching . . . 18

2.3 Kenmerken van de chauffeurs . . . 19

2.3.1 Reserve . . . 19

2.3.2 Fulltime en parttime . . . 19

2.4 Wensen vanuit Connexxion . . . 20

2.5 Kenmerken van Flexplan . . . 21

3 Mathematisch model 23

3.1 Het semi-assignment probleem . . . 23

3.2 Aanvullende restricties . . . 25

3.2.1 Dagrust . . . 25
3.2.2 Weekrust . . . 25
3.2.3 Nachtdiensten . . . 27
3.2.4 Weekenddiensten en gebroken diensten . . . 29
3.2.5 Werkdagen en vrije dagen . . . 29
3.3 Realiteitswaarde van het model . . . 30

4 Literatuuronderzoek **31**
4.1 Terminologie . . . 31
4.2 Roosterplanning in de literatuur . . . 32
4.3 Wensen van de werknemers . . . 34
4.4 Persoonlijke dienstroosters . . . 34
4.4.1 Days-off en shift scheduling . . . 34
4.4.2 Dienstroosters per chauffeur . . . 35
4.4.3 Dienstroosters per dag . . . 35
4.5 Keuze voor dienstroosters per chauffeur . . . 36

5 Deelprobleem **37**
5.1 De graaf . . . 38
5.2 Randvoorwaarden van de chauffeurs . . . 39
5.2.1 Vakantieschema . . . 39
5.3 De takkosten . . . 40
5.3.1 Huidige kostencomponenten . . . 41
5.3.2 Mogelijke kostencomponenten . . . 42
5.4 Ondergrens van de kosten . . . 43
5.5 Het SPPRC algoritme . . . 44
5.6 Resultaten . . . 48

6 Masterprobleem **51**
6.1 Lagrange Relaxatie . . . 52
6.1.1 Lagrange Multipliers . . . 52
6.1.2 Resultaten van de Lagrange Relaxatie . . . 53
6.2 Swap-Algoritme . . . 53
6.3 Max-Flow algoritme . . . 54
6.4 Vrije dagen verschuiven . . . 56

6.4.1 Enkele dag en weekend . . . 56
6.4.2 Tabu Search . . . 56
6.5 Overige functies . . . 57
6.5.1 Strafkosten . . . 57
6.5.2 Toewijzingen vastzetten . . . 57
6.5.3 Matchingspercentage verbeteren . . . 57
6.6 Algoritmes combineren . . . 58
6.6.1 Overzicht van de gebruikte methodieken . . . 58

7 Resultaten **61**
7.1 Matching voor 91 dagen . . . 62
7.1.1 Matching per kwartaal . . . 62
7.2 Matching voor 364 dagen . . . 63
7.2.1 Weekrust . . . 64
7.2.2 Aantal vrije dagen per half jaar . . . 65
7.2.3 Resultaten voor 364 dagen . . . 66
7.3 Vergelijking met FlexPlan . . . 67
7.3.1 Rekentijd . . . 67
7.3.2 Matchingspercentage . . . 67
7.3.3 Matching van de werkdagen . . . 67
7.3.4 Matching van de roostervrije dagen . . . 68
7.3.5 Matching van alle dagen . . . 69
7.3.6 Onzuiverheden . . . 69

8 Conclusies en aanbevelingen **71**
8.1 Oplossingsmethode . . . 71
8.1.1 Deelprobleem . . . 71
8.1.2 Masterprobleem . . . 71
8.2 Toepassing van het model . . . 72
8.3 Prestaties van het model . . . 73
8.4 Toekomstig onderzoek . . . 73
8.4.1 Toelaatbaarheid van de oplossing . . . 73
8.4.2 Oplossing verbeteren . . . 73
8.4.3 Rekentijd . . . 74
8.4.4 Planningsstructuur . . . 74

A Rules van de matching **75**

Bibliografie **77**

Voorwoord

De afdeling Future Technology van Connexxion houdt zich bezig met het ontwikkelen en verbeteren van de logistieke softwaresystemen waar Connexxion mee te maken heeft. Voor mijn afstudeerproject van de studie Operational Research and Management aan de Universiteit van Amsterdam heb ik zes maanden bij deze afdeling stage gelopen.

Een belangrijk onderdeel van de productie van Connexxion bestaat uit het verzorgen van het lokale en regionale busvervoer in grote delen van Nederland. Om het busvervoer efficiënt en betrouwbaar te kunnen aanbieden, besteedt Connexxion veel tijd aan het maken van goede dienstroosters. Door recente ontwikkelingen rond de flexibilisering van de arbeidsmarkt is het wenselijk om de buschauffeurs meer inspraak te geven in hun dienstroosters.

Tijdens mijn stageperiode hebben we een model ontwikkeld om dienstroosters voor de buschauffeurs te ontwikkelen, waarbij de chauffeurs hun eigen wensen aangeven.

Ik zou bij deze graag Michiel Houthuizen willen bedanken voor de moeite die hij heeft gedaan om binnen Connexxion op zoek te gaan naar een geschikte stageplek en stageopdracht. Daarnaast wil ik mijn collega's bij Future Technology bedanken voor het feit dat ze me vanaf de eerste dag in de groep hebben opgenomen en voor hun steun bij het uitwerken van de opdracht. In het bijzonder wil ik mijn collega Henk Post bedanken voor de begeleiding en voor de verbeteringen die ik dankzij zijn suggesties kon doorvoeren. Tenslotte wil ik Ton Volgenant bedanken voor de begeleiding vanuit de universiteit.

Arthur Bosma
Juli 2010

Samenvatting

Connexxion is een groot vervoersbedrijf dat onder meer regionaal en lokaal openbaar busvervoer aanbiedt. Eén van de uitdagingen voor Connexxion is het maken van de dienstroosters van de buschauffeurs. Het maken van dienstroosters is een bekend logistiek probleem waar veel onderzoek naar gedaan is.

Een relatief nieuwe ontwikkeling op het gebied van roosterplanning is zelfroostering, een proces waarbij buschauffeurs meer invloed hebben op hun dienstrooster door hun wensen aan te geven over werktijden en vrije dagen. We hebben een heuristische methode ontwikkeld om de dienstroosters te maken met als doel om zo veel mogelijk aan de wensen van de buschauffeurs te voldoen.

Er zijn twee soorten eisen waar de dienstroosters aan moeten voldoen.

- Voorwaarden ten behoeve van de werknemers, zoals CAO-afspraken. Dit zijn voornamelijk eisen die over meerdere dagen lopen.
- Voorwaarden die bedoeld zijn om te voldoen aan de vraag naar de busdiensten. Deze voorwaarden zijn per dag apart gespecificeerd.

We stellen een oplossingsmethode voor die het probleem oplost in twee stappen, gebaseerd op de hierboven beschreven tweedeling. Eerst maken we voor elke chauffeur een toegelaten dienstrooster. Vervolgens proberen we de dienstroosters te combineren in een complete matching. We passen daarbij heuristieken toe om zo aan de vraag te voldoen.

We hebben de resultaten van onze oplossingsmethode vergeleken met de resultaten van FlexPlan, het softwareprogramma dat Connexxion op dit moment gebruikt om een matching te maken. Ons model heeft een oplossing gegenereerd waarvan het matchingpercentage van vergelijkbare waarde is als het matchingpercentage van FlexPlan, zowel bij de matching van de werkdagen als bij de matching van de roostervrije dagen. Daarnaast is de rekentijd van ons model minder dan een uur, terwijl deze bij FlexPlan meerdere uren bedraagt.

Uit de resultaten blijkt dat ons model in staat is om de prestaties van FlexPlan te evenaren. We waren echter niet in staat om de omstandigheden waaronder FlexPlan een matching heeft gemaakt exact na te bootsen. Verder onderzoek is daarom nodig om te bepalen of de prestaties van ons model gewaarborgd blijven als we extra voorwaarden toevoegen.

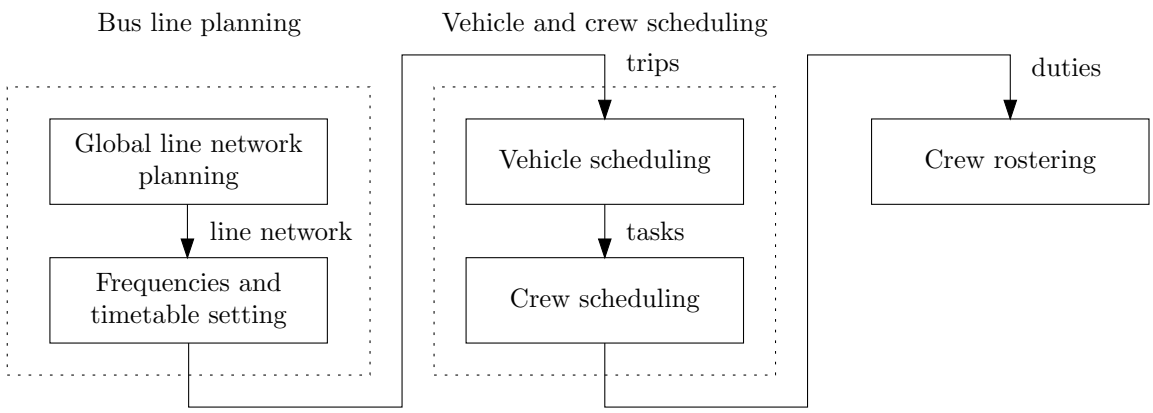
Hoofdstuk 1

Inleiding

1.1 Planningsproces in een busmaatschappij

Sinds 2001 is de markt voor het regionaal openbaar vervoer geliberaliseerd. Meerdere busmaatschappijen wedijveren om het busvervoer in de verschillende delen van Nederland aan te mogen bieden. Connexxion is op dit moment marktleider op het gebied van regionaal personenvervoer in Nederland. Zij vervoeren dagelijks meer dan één miljoen mensen naar hun bestemming [Connexxion (2008)].

De planning van het busvervoer is een belangrijk onderdeel van het werk van Connexxion. Connexxion heeft meer dan 50 vestigingen in Nederland en voor elke vestiging apart worden planningen gemaakt voor de bussen en voor de buschauffeurs (in het vervolg kortweg chauffeurs genoemd). Bij Connexxion onderscheiden we in het planningsproces vijf fasen [Zijp (2005)]. In figuur 1.1 staat een overzicht van deze fasen die we nu kort verklaren.



Figuur 1.1: *Traditioneel planningsproces van een busmaatschappij*

Global line network planning Allereerst wordt vastgelegd waar de bushaltes komen te liggen, welke buslijnen er bestaan en op welke haltes elke buslijn stopt.

Frequencies and timetable setting In deze fase worden de frequenties van de bussen bepaald (bijvoorbeeld vier keer per uur per buslijn). Daarna wordt een tijdschema voor de buslijnen gemaakt afhankelijk van de gewenste frequenties.

Vehicle scheduling Als bekend is welke buslijnen verreden moeten worden, dan kunnen de bussen toegekend worden aan de lijnen met als doel om het aantal benodigde bussen en het aantal MAT-ritten te minimaliseren. MAT-ritten (materiaalritten of in het Engels ook wel 'deadheads' genoemd) zijn ritten waarbij een buschauffeur de bus zonder passagiers naar een bepaalde plek brengt. Deze ritten leveren geen inkomsten op en zijn daarom onaantrekkelijk voor Connexxion.

Na de vehicle scheduling fase zou een deel van de busplanning er bijvoorbeeld als volgt uit kunnen zien: een bus rijdt van 6:05 uur tot 7:30 lijn 12 heen en terug, waarna de chauffeur een MAT-rit uitvoert naar het startpunt van lijn 15 om daar lijn 15 te starten om 8:00 uur.

Crew scheduling In deze fase wordt de busplanning gekoppeld aan de chauffeurs opdat er diensten ontstaan die voldoen aan bepaalde voorwaarden. Zo moet elke dienst bijvoorbeeld een pauze van minimaal een half uur bevatten. De diensten hebben een vaste start- en eindtijd en een bepaalde toewijzing van bussen en lijnen, maar vooralsnog is niet bekend welke chauffeur deze dienst zal rijden.

Crew Rostering In de laatste fase worden de dienstroosters van de chauffeurs opgesteld, waarbij de verschillende diensten aan specifieke chauffeurs worden toegekend voor elke dag van het jaar. Dit wordt bij Connexxion de matching genoemd.

Connexxion staat voor de uitdaging om al deze fasen zo goed mogelijk te doorlopen. Zij hebben daarom al verscheidene malen onderzoek laten doen naar het verbeteren en/of integreren van enkele van deze stappen. Quak (2003) heeft een model gemaakt om het ontwerpen van het netwerk van de buslijnen en het bepalen van de frequenties (de eerste en tweede fase) te automatiseren. Zijp (2005) heeft verder onderzocht of het mogelijk is om de derde en de vierde fase te integreren terwijl van den Heuvel (2007) een integratie van de tweede en derde fase voorstelt. We concentreren ons hier op de laatste fase: het maken van de matching.

1.2 De matching

Bij Connexxion is op dit moment sprake van een splitsing in twee soorten matchings. In de meeste delen van Nederland worden de diensten verdeeld in weekroosters, waarbij elk weekrooster gemiddeld evenveel werkuren bevat. Deze roosters worden dan gerouleerd onder de chauffeurs, zodat de chauffeurs elk weekrooster precies eenmaal rijden. Dit proces gebeurt nu al bijna geheel automatisch met behulp van het softwareprogramma HASTUS. Dit pakket bestaat al enkele tientallen jaren en is populair bij busvervoersbedrijven, zie bijvoorbeeld Blais e.a. (1990). De matching wordt beoordeeld op toelaatbaarheid en op de mate van verdeling van werkuren over de weekroosters.

Sinds kort heeft Connexxion het busvervoer in delen van Friesland overgenomen van Arriva, één van de concurrende busvervoersbedrijven. In deze regio's werd bij Arriva op een andere manier gepland dan bij Connexxion gebruikelijk was. Bij de overname is besloten om de methode van Arriva te handhaven. We zullen hier onderzoeken of het mogelijk is om verbeteringen door te voeren in deze planningsmethode, zelfroostering genaamd.

1.2.1 Zelfroostering

Bij zelfroostering wordt de chauffeurs de mogelijkheid geboden om wensen in te dienen over werktijden en vrije dagen. Elke chauffeur geeft van tevoren een wens aan voor elke dag van het jaar. Vervolgens wordt de matching gemaakt met als criterium om zo veel mogelijk aan deze wensen tegemoet te komen. Op deze manier hebben de chauffeurs meer invloed op hun dienstrooster.

Bij de overname van deze regio's in Friesland is besloten om een aparte CAO te maken zodat zelfroostering (binnen Connexxion ook wel 'Werken naar Wens' genoemd) kan blijven bestaan. Inmiddels bestaat bij Connexxion ook de wil om dit systeem uit te breiden naar andere delen van Nederland.

1.2.2 Blokindeling

Als een chauffeur een wens indient voor een bepaalde werktijd, dan maakt hij gebruik van bepaalde raamtijden, die door Connexxion 'blokken' worden genoemd. Elk type blok representeert een bepaald tijdsinterval op de dag waarin de chauffeur werkt. Een blok A beslaat bijvoorbeeld een interval van 4:30 uur tot 14:00 uur.

Elke dienst (de diensten worden opgesteld in fase 4 van figuur 1.1) wordt van tevoren ingedeeld bij een bepaald type blok. De blokken zijn ruim gekozen, zodat het mogelijk is om alle diensten in te delen bij een bepaald type blok. Een overzicht van de bestaande blokken bij Connexxion staat in tabel 1.1. In de hoofdstukken 2 en 3 zullen we een uitgebreide verklaring geven voor de specifieke eigenschappen van elk blok.

De indeling van de diensten naar blokken wordt per vestiging gedaan en valt buiten de scope van dit onderzoek. Bij het maken van de matching weten we dus al hoeveel blokken van elk type ingepland moeten worden en we zullen de chauffeurs vervolgens toekennen aan de blokken die op die dag nodig zijn. De blokken zijn dusdanig gedefinieerd dat elke chauffeur elke dag precies één blok krijgt toegewezen (het VRIJ-blok wordt gebruikt om roostervrije dagen weer te geven). De chauffeurs geven voor elke dag aan welk blok ze toegewezen willen krijgen.

1.3 Harde en zachte eisen

Connexxion stelt een aantal eisen aan de matching, waarbij onderscheid gemaakt wordt tussen 'harde' en 'zachte' eisen.

Harde eisen zijn eisen waar Connexxion aan moet voldoen. We onderscheiden twee soorten harde eisen.

- Voorwaarden ten behoeve van de werknemers, zoals CAO-afspraken en contractuele verplichtingen. Voorbeelden hiervan zijn het aantal vrije dagen in het jaar of het maximum aantal opeenvolgende werkdagen.
- Connexxion moet alle busritten verrijden, zoals deze zijn afgesproken met de opdrachtgevers (meestal de gemeentes waar de bussen rijden). Connexxion moet dus voldoen aan de vraag naar de busdiensten. Per dag moeten alle benodigde blokken worden toegekend.

Blok	Code	Starttijd	Eindtijd	Dagen	Opmerkingen
VRIJ	0	-	-	Ma-Zo	Roostervrij
RES	46	-	-	Ma-Zo	Reservedienst
A	30	4:30	14:00	Ma-Vr	GOLFPAK
B	31	6:00	16:00	Ma-Vr	
C	32	8:00	18:30	Ma-Vr	
D	33	6:00	19:00	Ma-Vr	Gebroken
E	34	13:00	24:00	Ma-Vr	
F	35	15:00	26:00	Ma-Vr	GOLFPAK
G	36	23:00	31:00	Ma-Vr	GLP/GOLFPAK
S	47	4:30	14:30	Za	GOLFPAK
H	37	6:00	16:00	Za	
I	38	8:00	18:30	Za	
J	39	13:00	24:00	Za	
K	40	15:00	26:00	Za	GOLFPAK
L	41	23:00	31:00	Za	GLP/GOLFPAK
U	48	4:30	14:00	Zo	GOLFPAK
M	42	6:00	16:00	Zo	
T	49	8:00	18:30	Zo	
N	43	13:00	24:00	Zo	
O	44	15:00	26:00	Zo	GOLFPAK
P	45	23:00	31:00	Zo	GLP/GOLFPAK

Tabel 1.1: *Een overzicht van de verschillende blokken die gebruikt worden voor de matching bij Connexxion. De blokken met als eindtijd 26:00 uur en 31:00 uur duren tot respectievelijk 2:00 uur en 7:00 uur de volgende dag. Ze vallen echter onder de dag van de starttijd.*

Zachte eisen zijn wensen waar Connexxion graag aan wil voldoen, maar die niet verplicht zijn. Hieronder vallen de voorkeuren die de chauffeurs van tevoren opgeven.

Een matching is toegelaten (in de literatuur vaak 'feasible' genoemd) als voldaan wordt aan alle harde eisen. Daarnaast kunnen we de matching beoordelen aan de hand van de zachte eisen, die we hieronder verder uitwerken.

1. **Matchingspercentage** De belangrijkste maatstaf is het matchingspercentage. Dit is het aantal ingewilligde wensen van de chauffeurs uitgedrukt als percentage van het totaal aantal wensen van de chauffeurs. Vanuit Connexxion wordt gestreefd naar een zo hoog mogelijk matchingspercentage, waarbij een minimum van 80% wordt aangehouden.

2. **Flexibiliteit** Een andere zachte eis is de flexibiliteit van de oplossing, de mogelijkheid om een bepaalde oplossing later nog aan te passen. Op dit moment gebeuren deze aanpassingen vaak nog met de hand, maar bij Connexxion wenst men dat dit ook geautomatiseerd wordt.

3. **Rekentijd** Uit de praktijk is gebleken dat het vaak een lange tijd kost om een matching te maken, ook al worden de berekeningen gedaan met behulp van planningssoftware. We besteden daarom ook aandacht aan de rekentijd.

1.4 Opbouw van het verslag

In hoofdstuk 2 beschrijven we de huidige situatie bij Connexxion. Daarna zullen we in hoofdstuk 3 het mathematische model opstellen, waarbij we de verschillende harde eisen omzetten naar restricties in het model. In hoofdstuk 4 geven we een overzicht van de bekende literatuur over personeelsplanning en besteden we ook aandacht aan de bestaande oplossingsmethodes. In hoofdstuk 5 en hoofdstuk 6 beschrijven we de methode die we hebben gebruikt om een matching op te stellen. In hoofdstuk 7 zullen we de resultaten van deze methode beschrijven, toegepast op werkelijke data van de vestiging Dokkum. Tenslotte stellen we in hoofdstuk 8 onze conclusies op.

Hoofdstuk 2

Probleemomschrijving

In delen van Friesland wordt gewerkt met zelfroostering waarbij voor alle chauffeurs een apart dienstrooster wordt gemaakt, afhankelijk van hun wensen. Het doel is om zo veel mogelijk aan de wensen te voldoen. Connexxion wil dit systeem uitbreiden naar andere delen van Nederland en daarom wordt er bij Connexxion veel aandacht besteed aan het verbeteren van de zelfroostering.

2.1 Kenmerken van het bloksysteem

De blokindeling bij Connexxion leidt tot een aantal specifieke kenmerken, die we in deze paragraaf zullen benoemen.

2.1.1 De indeling van diensten bij blokken

In het vorige hoofdstuk hebben we uitgelegd dat Connexxion eerst diensten maakt die de chauffeurs gaan rijden en dat deze diensten worden ingedeeld bij de blokken. Deze blokindeling van Connexxion heeft als nadeel dat het ten koste gaat van de flexibiliteit.

Een dienst die bijvoorbeeld van 5:59 uur tot 13:30 uur duurt, zal ingedeeld worden bij blok A terwijl de dienst eigenlijk pas anderhalf uur later begint dan 4:30 uur. Toch zal het voor de voorwaarden lijken alsof de chauffeur al om 4:30 uur begint met werken. Dit heeft een beperkende invloed op de combinatie van opeenvolgende blokken die we kunnen kiezen.

2.1.2 Roostervrije dagen en verlofdagen

Er zijn verschillende voorwaarden aan de matching met betrekking tot roostervrije dagen en verlofdagen, daarom is het belangrijk om hier onderscheid tussen te maken.

Roostervrije dagen (of kortweg vrije dagen) zijn dagen waarop geen werk wordt ingepland, omdat chauffeurs elke week recht hebben op een bepaalde vorm van 'weekend'. Deze dagen kunnen (onder bepaalde criteria) vrij worden ingedeeld door de roostermaker. Het VRIJ-blok representeert een roostervrije dag.

Verlofdagen of vakantiedagen zijn dagen waarop een chauffeur wel werk krijgt toegewezen, want hij krijgt immers doorbetaald tijdens verlof, maar op deze dagen heeft de chauffeur vrij genomen en zal hij dus geen diensten rijden (voor de matching wordt er geen onderscheid gemaakt tussen verlof en vakantie). De verlofdagen worden grotendeels van tevoren vastgelegd in overleg met de chauffeur en de roostermaker kan deze dagen dus niet beïnvloeden. Voor de verlofdagen gebruiken we het RES-blok, een dummy blok waar geen diensten aan gekoppeld zijn.

2.2 Het maken van een matching

Het maken van de matching (bij zelfroostering) is een uitgebreid proces. Hieronder geven we een schets van de huidige situatie bij Connexxion. We bespreken stapsgewijs hoe een matching wordt gemaakt vanaf de Crew Scheduling fase (stap 0) tot het moment dat de dienstroosters gemaakt zijn.

0. Voor elke vestiging wordt een busschema gemaakt en worden de chauffeursdiensten ontwikkeld (fase 3 en 4 uit figuur 1.1). Voor elke bus is nu bekend wanneer en waar deze wordt ingezet. Ook voldoen de diensten aan de algemene CAO-voorwaarden, maar ze zijn nog niet toegekend aan de chauffeurs zelf.

1. De chauffeursdiensten worden toegekend aan een blok. Dit gebeurt nog handmatig en per vestiging wordt dit door de planner ter plaatse gedaan. Alle diensten die bijvoorbeeld tussen 4:30 en 6:00 uur starten zullen worden ingedeeld als een blok A, omdat ze te vroeg starten om bij een ander blok ingedeeld te kunnen worden.

2. Bij elke vestiging is nu per dag bekend hoeveel chauffeurs per bloktype nodig zijn om alle diensten te vullen. Voor ons model definiëren we dit als de 'vraag' naar de blokken.

3. De vestigingsmanager kan de vraag naar blokken kunstmatig beïnvloeden. Vanwege ziekte en cursussen zal niet elke chauffeur altijd beschikbaar zijn op elke dag dat hij werk krijgt toegewezen en daarom wordt het aantal benodigde blokken meestal met 10% tot 20% vergroot. Op deze manier wordt rekening gehouden met onvoorziene omstandigheden waar we tijdens het maken van de matching nog geen weet van kunnen hebben.

4. Elke chauffeur geeft zijn wensen op voor het volgende jaar. Een overzicht van deze wensen en van de benodigde blokken op elke dag wordt in de computer ingevoerd met behulp van een softwareprogramma genaamd FlexPlan. Dit proces loopt grotendeels gelijk met de voorgaande stappen.

5. Vervolgens geeft de roostermaker aan op welke dagen de chauffeurs verlofdagen hebben opgenomen (zie paragraaf 2.1.2 voor een uitleg over het verschil tussen roostervrije dagen en verlofdagen).

6. FlexPlan bepaalt eerst voor elke chauffeur een schema van werkdagen en roostervrije dagen (in de literatuur vaak een pattern genoemd, zie ook hoofdstuk 4). Op de werkdagen wordt nog geen blok toegekend, maar wel staat vast dat hij op deze dagen geen VRIJ-blok zal krijgen. De wensen van de chauffeurs met betrekking tot roostervrije dagen worden zo veel mogelijk ingewilligd, de overige wensen worden aanvankelijk nog genegeerd.

7. FlexPlan gaat nu voor elke dag blokken toekennen aan de chauffeurs die een werkdag hebben. Opnieuw wordt zo veel mogelijk aan de wensen voldaan.

8. Het rooster is klaar en het matchingspercentage is bekend. Per vestiging weet elke chauffeur nu in welk blok hij werkt op elke dag. Vervolgens wordt per dag bepaald welke dienst iedereen moet rijden. Dit gebeurt pas op het laatste moment, zodat er rekening gehouden kan worden met allerlei actuele omstandigheden (zoals ziekte).

We gaan een model ontwikkelen dat stap 4 tot en met 7 implementeert en uitvoert, zodat de data die bij stap 4 en 5 wordt ingelezen kan worden gebruikt bij het zoeken van een oplossing in stap 6 en 7. Dit zijn de stappen die op dit moment worden uitgevoerd door een softwareprogramma genaamd FlexPlan.

2.3 Kenmerken van de chauffeurs

Connexxion maakt van tevoren een schatting van het aantal chauffeurs dat nodig is om aan de vraag naar de blokken te kunnen voldoen. Het aanbod van chauffeurs is bij het maken van de matching constant en dit onderdeel houden we hier dan ook buiten beschouwing.

2.3.1 Reserve

In paragraaf 2.2 hebben we aangegeven dat het aantal benodigde blokken van tevoren wordt verhoogd met een bepaald reservepercentage. In de praktijk is dit reservepercentage meestal een redelijke inschatting van wat er daadwerkelijk nodig is aan reserve. Soms zal het echter voorkomen dat er reserve tekort is, of dat er reserve over is. Dit soort onregelmatigheden worden op de vestiging opgelost door de planners.

Als er reserve over is, dan heeft de planner een aantal opties om ervoor te zorgen dat de chauffeur niet voor niets naar zijn werk komt. Zo laten ze de chauffeur bijvoorbeeld tijdens een ander blok rijden dan waar ze waren ingedeeld. Het komt bij Connexxion weinig voor dat een chauffeur niets te doen heeft, ook al krijgt hij een blok toegekend waar aanvankelijk nog geen dienst voor beschikbaar is.

2.3.2 Fulltime en parttime

Bij het matchen wordt onderscheid gemaakt tussen fulltime en parttime chauffeurs. Parttime chauffeurs werken vaak op specifieke dagen, bijvoorbeeld alleen op maandag en dinsdag. In dat geval moeten deze chauffeurs werk krijgen op maandag en dinsdag; ze mogen geen roostervrije dag toegewezen krijgen op deze dagen.

In andere gevallen werkt de chauffeur wel alle dagen, maar in totaal bijvoorbeeld slechts 20 uur per week. In dat geval krijgt hij wel een normaal blok toegewezen, maar dan wordt er bij de vestiging rekening mee gehouden dat hij binnen dat blok een korte dienst krijgt van bijvoorbeeld 4 uur.

Het is ook mogelijk dat de chauffeur alsnog vrij krijgt op de betreffende dag, omdat er geen kortlopende diensten meer beschikbaar zijn. Dit hoeft geen problemen te geven als het aantal extra ingeplande blokken (zie stap 5) groot genoeg is.

Bij het maken van de matching hoeven we dus geen rekening te houden met het aantal uur dat een chauffeur daadwerkelijk aan het werk is. De vestigingsmanager kan verschillende middelen inzetten om specifieke omstandigheden op te lossen.

2.4 Wensen vanuit Connexxion

Vanuit Connexxion zijn een aantal wensen ontstaan met betrekking tot de flexibiliteit bij het maken van de matching. Er bestaan meerdere theorieën en meningen over hoe een matching kan worden beoordeeld en daarom zou men de mogelijkheid willen hebben om verschillende methoden met elkaar te vergelijken. Hieronder geven we een overzicht van de manieren waarop de matching beoordeeld kan worden.

Minimale matchingspercentage We optimaliseren door de matching van de chauffeur met het laagste matchingspercentage zo veel mogelijk te verbeteren.

Gemiddelde matchingspercentage Het optimaliseren van het gemiddelde matchingspercentage zorgt ervoor dat het totale matchingspercentage beter is, maar dat sommige chauffeurs waarschijnlijk veel slechtere resultaten behalen dan hun collega's.

Verdeling van de wensvervulling Het beoordelen van het matchingspercentage houdt geen rekening met de subjectieve beleving van de chauffeurs. Een chauffeur die bijvoorbeeld afwisselend ochtend- en nachtdiensten aanvraagt, zal over het algemeen niet zo veel moeite hebben met de toewijzing van een ochtenddienst terwijl hij een nachtdienst wenste. Iemand die altijd nachtdiensten wenst, heeft over het algemeen een grotere aversie tegen ochtenddiensten dan de chauffeur met een gemengd wensenpakket. Mogelijk kunnen we het model daarom sturen om de ochtenddiensten aan chauffeurs toe te wijzen die hier minder moeite mee hebben. Aan de andere kant vindt men het ook niet eerlijk als het aanvragen van heel eenzijdige wensen betekent dat je nooit ander type diensten krijgt. Met dit gegeven zou men dan ook willen kunnen experimenteren.

Uiteindelijk wil men de matching kunnen beoordelen op grond van een combinatie van de hierboven beschreven maatstaven, waarbij sommige maatstaven eventueel zwaarder meetellen dan andere.

Daarnaast bestaan er ook enkele varianten voor de manier waarop de wensen worden ingediend. Hieronder geven we enkele voorbeelden.

Eén wens per dag De chauffeurs geven aan welk blok ze toegewezen willen krijgen op elke dag van de planningsperiode. Op dit moment wordt bij Connexxion gewerkt met dit systeem.

Puntensysteem Bij dit systeem mogen de chauffeurs punten toekennen aan dagen waarop zij het belangrijk vinden dat aan hun wens voldaan wordt. Deze dagen tellen dan ook extra zwaar mee bij de beoordeling van de matching.

Meerdere keuzes De chauffeurs geven nu alleen een eerste keuze aan voor een bepaalde dag, maar het is ook mogelijk dat ze daarnaast een tweede of derde keuze aangeven voor hun wens op een bepaalde dag. Op die manier is de kans kleiner dat de chauffeur iets krijgt dat hij absoluut niet wil hebben (en daarmee kan de verdeling van de wensvervulling bijvoorbeeld verbeterd worden).

Niet-gewenste blokken De chauffeurs geven nu aan wat hun wens is voor een bepaalde dag. Het is ook mogelijk om de chauffeurs aan te laten geven welke blokken ze juist niet willen krijgen.

Deze methodes kunnen ook gecombineerd worden om een steeds completer beeld te krijgen van de wensen van de chauffeurs. Daarbij moet er wel rekening mee gehouden worden dat een uitgebreider systeem vaak ook meer tijd en moeite van de chauffeurs vereist om alles naar behoren in te vullen.

Binnen Connexxion wil men graag de mogelijkheid hebben om te experimenteren met verschillende maatstaven en methodes. We zullen onderzoeken in hoeverre het mogelijk is deze flexibiliteit toe te voegen aan het model. Aanvankelijk zullen we uitgaan van het maximaliseren van het gemiddelde matchingspercentage om onze modellen te testen, maar we zullen later nog terugkomen op de mogelijkheid tot het flexibiliseren van onze doelfunctie.

2.5 Kenmerken van Flexplan

FlexPlan wordt op dit moment gebruikt om een matching te maken voor de chauffeurs die werken met zelfroostering. FlexPlan begint met het inplannen van de vrije dagen. De reden hiervoor is tweeledig: aan de ene kant is het indelen van de vrije dagen vaak het moeilijkste onderdeel van de planning, waardoor de kans op het vinden van een toegelaten oplossing groter is als hiermee begonnen wordt.

Daarnaast blijkt uit de praktijk dat chauffeurs bij het indienen van hun wensen meestal meer waarde hechten aan de wensen met betrekking tot de roostervrije dagen en minder waarde aan een specifiek blok op de werkdagen. Eventuele klachten over de matching komen ook meestal van chauffeurs die ontevreden zijn met de toekenning van de roostervrije dagen. Door te beginnen met de vrije dagen in te plannen, zullen de wensen voor de vrije dagen relatief vaker ingewilligd worden.

Een nadeel van FlexPlan is dat het programma vaak een lange tijd (in de orde van enkele uren) nodig heeft voordat het een oplossing vindt. FlexPlan heeft op dat moment vaak geen optimale oplossing gevonden, maar een (goede) toegelaten oplossing. Een snellere methode zou als voordeel kunnen hebben dat er meer tijd beschikbaar is voor het verbeteren van de oplossing waardoor betere matchingspercentages gerealiseerd kunnen worden.

Daarnaast is FlexPlan (nog) niet flexibel genoeg om alle wensen van Connexxion te implementeren (zie paragraaf 2.4).

Hoofdstuk 3

Mathematisch model

3.1 Het semi-assignment probleem

We willen een matching maken waarbij we de 'opbrengst' maximaliseren, die we reeds in de inleiding hebben gedefinieerd als het matchingspercentage. Elke wens die wordt ingewilligd brengt een virtuele opbrengst met zich mee, doordat de toewijzing het matchingspercentage vergroot. Het maakt daarom niet uit of we het percentage of het aantal ingewilligde wensen maximaliseren.

Voor onze doelfunctie is het makkelijker als we gewichten toekennen aan bepaalde toewijzingen. We kennen daarom positieve gewichten toe aan de toewijzingen die overeen komen met de wensen van de chauffeurs en een gewicht van 0 aan alle andere mogelijke toewijzingen. Op deze manier hebben we de opbrengsten van ons model gedefinieerd en kunnen we de doelfunctie opstellen.

We kunnen het matchen zien als een 'assignment probleem' met aanvullende restricties (de harde eisen die door Connexxion worden gesteld). We zullen deze restricties in paragraaf 3.2 beschrijven. Hier bespreken we hoe we het assignment probleem hebben gebruikt om het mathematische model van dit probleem te beschrijven.

Het assignment probleem bestaat uit een bipartiete graaf. In Nicoletti (1975) wordt een model ontwikkeld om complete dienstroosters toe te kennen aan de werknemers. De knopen aan de ene zijde van de bipartiete graaf representeren de werknemers, terwijl de dienstroosters door de knopen aan de andere zijde worden weergegeven. We gebruiken een soortgelijke representatie, in dit geval met aan de ene zijde de groep chauffeurs en aan de andere zijde de groep van alle mogelijke blokken (de blokken in tabel 1.1).

De vraag naar de verschillende blokken is vaak groter dan 1 en daarom kunnen we hier beter spreken van een semi-assignment probleem. Het semi-assignment probleem is een generalisatie van het assignment probleem waarbij de elementen aan één zijde een capaciteit hebben die groter kan zijn dan 1 (de blokken), terwijl de elementen aan de andere zijde een capaciteit hebben die exact gelijk is aan 1 (de chauffeurs). Het assignment probleem en het semi-assignment probleem kunnen in polynomiale tijd opgelost worden (zie bijvoorbeeld Ahuja e.a. (1993) of Ford en Fulkerson (1962)) en zo kunnen we een matching maken als we voor elke dag d een semi-assignment probleem oplossen. Deze matching voldoet aan de vraag naar de blokken, maar niet noodzakelijkerwijs aan de aanvullende restricties.

In het model definiëren we voor elke dag d in de planningsperiode (normaal gesproken een

jaar) een semi-assignment probleem met kantenverzameling $\mathcal{E}^d$. Een kant tussen een chauffeur c en een blok b is gedefinieerd als het mogelijk is om de chauffeur op die dag aan het blok toe te kennen. In uitzonderlijke gevallen (zoals medische omstandigheden van de chauffeur) kan het voorkomen dat bepaalde toewijzingen niet zijn toegestaan, maar in de meeste gevallen zal een kant (c, b) bestaan, voor elke chauffeur en elk blok dat op dag d gevraagd wordt.

We schrijven eerst het semi-assignment probleem uit en daarna leggen we het model uit. We gaan uit van de situatie waarbij alle chauffeurs fulltime werken en dat de week loopt van zondag tot en met zaterdag (dat betekent dus dat dag 1, 8, 15, 22 etc. op zondag vallen). De vergelijkingen in dit hoofdstuk worden steeds voorafgegaan door een overzicht van nieuw geïntroduceerde variabelen en notatie.

$$
\begin{aligned}
c \in \mathcal{C} &= \text{de verzameling chauffeurs.} \\
b \in \mathcal{B} &= \text{de verzameling toe te wijzen blokken.} \\
d \in \mathcal{D} &= \text{de dagen in de planningsperiode.} \\
(c, b) \in \mathcal{E}^d &= \text{de verzameling kanten (edges) op dag } d. \\
X_{cb}^d &= \begin{cases} 1 & \text{als chauffeur } c \text{ op dag } d \text{ blok } b \text{ toegewezen krijgt.} \\ 0 & \text{anders.} \end{cases} \\
r_{cb}^d &= \text{de kosten behorende bij de toewijzing } X_{cb}^d. \\
A_b^d &= \text{de vraag naar de blokken } b \text{ op dag } d.
\end{aligned}
$$

$$
\min \sum_{d \in \mathcal{D}} \sum_{(c,b) \in \mathcal{E}^d} r_{cb}^d X_{cb}^d \tag{3.1}
$$

onder voorwaarde van:

$$
\sum_{c:(c,b) \in \mathcal{E}^d} X_{cb}^d = A_b^d \qquad \forall b \in \mathcal{B}; \forall d \in \mathcal{D} \tag{3.2}
$$

$$
\sum_{b:(c,b) \in \mathcal{E}^d} X_{cb}^d = 1 \qquad \forall c \in \mathcal{C}; \forall d \in \mathcal{D} \tag{3.3}
$$

$$
X_{cb}^d \in \{0, 1\} \qquad \forall (c, b) \in \mathcal{E}^d; \forall d \in \mathcal{D} \tag{3.4}
$$

Dit is het semi-assignment probleem voor deze situatie (voor meer informatie over het semi-assignment probleem, zie bijvoorbeeld Kennington en Wang (1992)). Vaak wordt voorwaarde (3.4) geschreven als $X_{cb}^d \geq 0$, omdat de keuze voor 0 of 1 automatisch volgt door de totale unimodulariteit van (3.2) en (3.3). Vanwege de aanvullende restricties kunnen we deze keuzes echter niet garanderen. Daarom maken we de variabele X_{cb}^d hier expliciet binair.

De opbrengsten r_{cb}^d kiezen we zo dat de wensen van de chauffeurs het meest aantrekkelijk worden, terwijl alle andere matchings lagere waarden hebben. De chauffeurs $c \in \mathcal{C}$ hebben een aanbod van één eenheid, terwijl elk blok $b \in \mathcal{B}$ een vraag heeft van A_b^d op dag $d \in \mathcal{D}$. Deze vraag is afhankelijk van de diensten die de betreffende dag nodig zijn, en zal in een aantal gevallen gelijk zijn aan 0. De vraag naar een willekeurig weekendblok zal bijvoorbeeld 0 zijn op een doordeweekse dag.

3.2 Aanvullende restricties

Het semi-assignment probleem kunnen we optimaal oplossen. De matching moet echter wel voldoen aan de harde eisen met betrekking tot onder meer de CAO afspraken. Een overzicht van de harde eisen (of 'rules') staat in bijlage A. Hieronder hebben we deze rules in het model verwerkt als extra voorwaarden.

3.2.1 Dagrust

Tussen twee blokken moet de chauffeur een nachtrust van 11 uur krijgen. Dit is echter een verwarrende term in deze context, omdat deze rust op elk moment van de dag mag vallen. Daarom spreken we hier liever van 'dagrust': de rust die elke dag gegeven moet worden om de chauffeur de kans te geven om uit te rusten.

Voor elk blok b hebben we een verzameling $\mathcal{N}_b$ die alle blokken bevat die minder dan 11 uur na blok b beginnen. Als twee blokken te kort na elkaar beginnen, dan mag een chauffeur maximaal één van beide toewijzingen krijgen. Deze voorwaarde is uitgewerkt in (3.5).

$$\begin{aligned} \mathcal{N}_b &= \text{de verzameling van blokken die minder dan 11 uur} \\ &\quad\ \text{na het einde van blok } b \text{ starten} \\ b_N &= \text{een blok in de verzameling } \mathcal{N}_b \end{aligned}$$

$$X^d_{cb} + X^{d+1}_{cb_N} \leq 1 \qquad \forall c \in \mathcal{C}; \forall d = 1, ..., D-1; \forall b \in \mathcal{B}; \forall b_N \in \mathcal{N}_b \tag{3.5}$$

3.2.2 Weekrust

Een chauffeur mag nooit meer dan vijf dagen achtereen blokken toegewezen krijgen, tenzij deze reeks blokken een zaterdag én een zondag bevat. In dat geval is het maximum zes blokken achtereenvolgend. We hebben deze voorwaarde gevat in drie verschillende restricties.

Om specifieke blokken weer te geven, gebruiken we de codes zoals die ook te vinden zijn in tabel 1.1. In dit geval gebruiken we het symbool b_0 om het blok met code 0 weer te geven, het roostervrije blok.

De eerste restrictie, (3.6), bepaalt dat het nooit mogelijk is om meer dan zes opeenvolgende dagen blokken toegewezen te krijgen. Vervolgens geldt dat we een extra restrictie opleggen aan die reeksen van dagen waar niet zowel een zaterdag als een zondag in voorkomen. Dit zijn slechts twee mogelijkheden, namelijk de reeks maandag tot en met vrijdag en de reeks zondag tot en met donderdag. Vanaf elke andere startdag is het mogelijk om een reeks van zes dagen te maken met zowel een zaterdag als een zondag erin. We hebben daarom voor deze twee startdagen aparte restricties gemaakt, zodat we in die gevallen een maximum van vijf opeenvolgende dagen hanteren. We maken daarbij gebruik van restricties die zich voor elke week in het jaar herhalen. We gebruiken daarbij de eigenschap dat dag $7d+1$ altijd op zondag valt, dag $7d+2$ altijd op maandag, etcetera.

In (3.7) hebben we deze voorwaarde uitgewerkt voor de periode van zondag tot en met donderdag ($t = 1, \ldots, 5$) en in (3.8) voor de periode van maandag tot en met vrijdag. Als de

chauffeur op alle vijf de dagen moet werken, dan is de som gelijk aan 5 en dan leggen we de restrictie op dat de toewijzing van die chauffeur aan blok b_0 de volgende dag positief moet zijn, zodat alleen de toewijzing $X_{cb_0}^{7d+6} = 1$ resteert.

$$\begin{aligned} b_0 &= \text{het blok dat een roostervrije dag representeert} \\ \mathcal{B}^+ = \mathcal{B} \setminus \{b_0\} &= \text{de verzameling van alle blokken exclusief roostervrij} \\ t &= \text{een hulpvariabele voor de tijd in dagen} \\ D &= \text{het aantal dagen in de planningsperiode } = |\mathcal{D}| \end{aligned}$$

$$\sum_{t=0}^{6} \sum_{b \in \mathcal{B}^+} X_{cb}^{d+t} \leq 6 \qquad \forall c \in \mathcal{C}; \forall d = 1, ..., D-6 \tag{3.6}$$

$$\sum_{t=1}^{5} \sum_{b \in \mathcal{B}^+} X_{cb}^{7d+t} - 4 \leq X_{cb_0}^{7d+6} \qquad \forall c \in \mathcal{C}, \forall d = 0, ..., 51 \tag{3.7}$$

$$\sum_{t=2}^{6} \sum_{b \in \mathcal{B}^+} X_{cb}^{7d+t} - 4 \leq X_{cb_0}^{7d+7} \qquad \forall c \in \mathcal{C}, \forall d = 0, ..., 51 \tag{3.8}$$

Elke week moet een chauffeur ergens 45 aaneengesloten uren rust krijgen. Er is een uitzondering op deze regel, want elke twee weken mag deze rust eenmaal verkort worden tot minimaal 36 uur. Deze verkorte uren moeten dan wel binnen 22 dagen gecompenseerd worden en dan moeten ook in één keer alle verkorte uren gecompenseerd worden. De compensatie mag zowel op een weekrust gegeven worden (de weekrust zal dan dus 45 + compensatie uren bedragen) of op een nachtrust (die dan 11 + compensatie uren duurt).

Bij het matchen wordt de verkorte weekrust zelden tot nooit toegekend. Vaak wordt uitgegaan van een versimpelde voorwaarde waarbij de chauffeurs elke week twee dagen aaneengesloten vrij krijgen, waardoor altijd voldaan wordt aan de rustperiode van 45 uur. De verkorte weekrust komt in de praktijk vooral voor als een chauffeur op het laatste moment invalt vanwege bijvoorbeeld ziekte. Daarom zullen we hier ook uitgaan van de versimpelde versie met een weekrust van 45 uur.

We hebben deze restrictie geïmplementeerd met behulp van de extra binaire variabele W_c^d en een extra set van blokken $\mathcal{M}_b$. Deze set bevat voor elk blok b alle blokken die minimaal 21 uur na blok b starten (op de volgende dag).

Uit (3.9) volgt dat $W_c^d = 1$ als chauffeur c de rustperiode van 45 uur heeft gehad op dag d. Deze rustperiode start altijd met een blok op dag $d-2$ (dit kan ook een roostervrije dag zijn), gevolgd door een roostervrije dag en daarna gevolgd door een dag waarop een blok gepland is dat minimaal 21 uur later begint. In dit geval stellen we dat elk blok in verzameling $\mathcal{M}_{b_0}$ zit en dat blok b_0 onderdeel is van $\mathcal{M}_b$ voor elke b. Met andere woorden: elk blok start minimaal 21 uur na een roostervrije dag en een roostervrije dag 'start' altijd minimaal 21 uur na een willekeurig blok. Op deze manier kunnen we alle mogelijke combinaties van 45 uur afvangen in (3.9).

In (3.10) stellen we vervolgens dat de weekrust minimaal één keer in elke week gegeven moet worden.

$\mathcal{M}_b =$ de verzameling van blokken die minimaal 21 uur
na het einde van blok b starten

$b_M =$ een blok in de verzameling $\mathcal{M}_b$

$$W_c^d = \begin{cases} 1 & \text{als de rustperiode van 45 uur van chauffeur } c \text{ op dag } d \text{ eindigt} \\ 0 & \text{anders} \end{cases}$$

$$X_{cb}^{d-2} + X_{cb_0}^{d-1} + \sum_{j \in \mathcal{M}_b} X_{cj}^d - 2 \leq W_c^d \qquad \forall c \in \mathcal{C}; \forall b \in \mathcal{B}; \forall d = 3, ..., D \tag{3.9}$$

$$\sum_{t=1}^{7} W_c^{7d+t} \geq 1 \qquad \forall c \in \mathcal{C}; \forall d = 0, ..., 51 \tag{3.10}$$

3.2.3 Nachtdiensten

Bij Connexxion gelden er aparte regels voor de nachtdiensten vanwege de onregelmatige werktijden die daarmee gepaard gaan. Dit type regels is opgesplitst in twee delen: de regels voor de 'echte' nachtdiensten (de GLP blokken; blok G, L en P) en de regels voor diensten die voor een gedeelte 's nachts vallen (de zogenaamde GOLFPAK diensten, de diensten waarvan de letters in het woord GOLFPAK voorkomen aangevuld met de blokken U en S). De GOLFPAK diensten vallen allemaal geheel of gedeeltelijk tussen 24:00 uur 's avonds en 6:00 uur 's ochtends de volgende dag. De GLP blokken lopen van 23:00 uur tot 7:00 uur de volgende ochtend. Voor deze blokken geldt de regel dat als een chauffeur een GLP blok krijgt, dan moet hij ten minste drie opeenvolgende GLP blokken toegewezen krijgen. Op deze manier hoeven de chauffeurs niet te vaak om te schakelen tussen dag- en nachtwerk.

Om deze restrictie te implementeren, introduceren we een extra variabele Y_c^d. We weten dat een chauffeur start met een serie van GLP-blokken als de chauffeur vandaag een GLP-blok krijgt, terwijl hij gisteren een ander blok toegewezen had gekregen, zie (3.11). De twee daarop volgende dagen moet de chauffeur ook een GLP-blok toegewezen krijgen, zie (3.12) en (3.13).

$$Y_c^d = \begin{cases} 1 & \text{als chauffeur } c \text{ op dag } d \text{ start met een serie van minimaal drie GLP blokken} \\ 0 & \text{anders} \end{cases}$$

$\mathcal{B}^G = \{b_{36}, b_{41}, b_{45}\} \subset \mathcal{B}$
$=$ de verzameling van G, L en P blokken

$$\sum_{b \in \mathcal{B}^G} X_{cb}^d - \sum_{b \in \mathcal{B}^G} X_{cb}^{d-1} \leq Y_c^d \qquad \forall c \in \mathcal{C}; \forall d = 2, ..., D \tag{3.11}$$

$$\sum_{b \in \mathcal{B}^G} X_{cb}^{d+1} \geq Y_c^d \qquad \forall c \in \mathcal{C}; \forall d = 2, ..., D-1 \tag{3.12}$$

$$\sum_{b \in \mathcal{B}^G} X_{cb}^{d+2} \geq Y_c^d \qquad \forall c \in \mathcal{C}; \forall d = 2, ..., D-2 \tag{3.13}$$

$$Y_c^d \in \{0, 1\} \qquad \forall c \in \mathcal{C}; \forall d \in \mathcal{D} \tag{3.14}$$

Voor de GOLFPAK blokken (de GLP blokken vallen hier dus ook onder) geldt de regel dat de chauffeurs na minimaal drie van dit soort blokken altijd 48 uur nachtrust moeten krijgen.[1] Om deze voorwaarde te implementeren, hebben we een nieuwe binaire variabele geïntroduceerd genaamd Z_c^d, die gelijk is aan 1 als een chauffeur een serie van minimaal drie GOLFPAK diensten achter elkaar heeft gekregen en 0 anders. In vergelijking (3.15) zorgen we ervoor dat Z_c^d altijd gelijk aan 1 wordt gekozen als de chauffeur c de afgelopen drie dagen een GOLFPAK dienst toegewezen heeft gekregen.

Het is niet van tevoren bekend hoeveel opeenvolgende blokken de chauffeur precies krijgt toegewezen voordat de pauze van 48 uur volgt. We hebben daarom in vergelijking (3.16) een uitdrukking opgesteld om te waarborgen dat de chauffeur ergens in de dagen na de serie van nachtdiensten een pauze van 48 uur krijgt. De restricties uit paragraaf 3.2.2 zorgen er voor dat de chauffeur maximaal drie dagen later aan zijn pauze begint (want hij heeft al drie nachtdiensten gemaakt en in totaal mag hij maximaal zes blokken achtereenvolgend toegewezen krijgen). We laten daarom de teller t lopen van 0 tot en met 3. Op het moment dat hij aan zijn pauze begint, zorgen we er nu voor dat hij direct twee opeenvolgende dagen rust krijgt (dag $d+t+1$ en dag $d+t+2$).

In de meeste gevallen zijn twee opeenvolgende vrije dagen genoeg om 48 uur rust te krijgen. In het zeldzame geval echter dat de chauffeur eindigt met een GLP blok (dat eindigt om 7:00 uur 's ochtends) en daarna na twee dagen een vroeg blok krijgt toegewezen, begint de chauffeur al om 4:30 of 6:00 uur. Dan heeft hij minder dan 48 uur pauze gehad, want hij mag pas na 7:00 uur die dag beginnen. We hebben daarom de extra restrictie (3.17) ingevoerd die deze combinatie van toewijzingen verbiedt.

$$Z_c^d = \begin{cases} 1 & \text{als chauffeur } c \text{ op dag } d \text{ een serie van drie GOLFPAK diensten heeft gehad} \\ 0 & \text{anders} \end{cases}$$

$$\begin{aligned} \mathcal{B}^V =& \{b_{30}, b_{31}, b_{37}, b_{42}, b_{47}, b_{48}\} \subset \mathcal{B} \\ =& \text{ de verzameling van 'vroege' blokken: A, B, H, M, S en U} \\ \mathcal{B}^N =& \{b_{30}, b_{35}, b_{36}, b_{40}, b_{41}, b_{44}, b_{45}, b_{47}, b_{48}\} \subset \mathcal{B} \\ =& \text{ de verzameling van de nachtdiensten: blok S, U en de GOLFPAK blokken} \end{aligned}$$

$$\sum_{t=0}^{2} \sum_{b \in \mathcal{B}^N} X_{cb}^{d-t} - 2 \leq Z_c^d \qquad \forall c \in \mathcal{C}; \forall d = 3, ..., D \tag{3.15}$$

$$\sum_{b \in \mathcal{B}^+} X_{cb}^{d+t} + X_{cb_0}^{d+t+1} + Z_c^d - 2 \leq X_{cb_0}^{d+t+2} \qquad \forall c \in \mathcal{C}; \forall t = 0, ..., 3; \forall d = 1, ..., D-5 \tag{3.16}$$

$$\sum_{b \in \mathcal{B} \setminus \mathcal{B}^G} X_{cb}^{d} \geq \sum_{b \in \mathcal{B}^V} X_{cb}^{d+3} \qquad \forall c \in \mathcal{C}; \forall d = 1, ..., D-3 \tag{3.17}$$

Tenslotte mag een chauffeur elke opeenvolgende 91 dagen maximaal 29 nachtdiensten toegewezen krijgen (van dag 1 t/m 91, dag 2 t/m 92 etc.). De nachtdienstblokken worden weergegeven met behulp van $\mathcal{B}^N$. Deze voorwaarde is uitgewerkt in vergelijking (3.18).

[1] Het zal regelmatig voorkomen dat een chauffeur drie GOLFPAK blokken achtereenvolgens krijgt toegewezen, bij de GLP blokken is dit zelfs verplicht (zie paragraaf 3.2.3).

$$\sum_{d=1}^{91} \sum_{b \in \mathcal{B}^N} X_{cb}^{d+91t} \leq 29 \qquad \forall c \in \mathcal{C}; \forall t = 0, ..., 3 \tag{3.18}$$

3.2.4 Weekenddiensten en gebroken diensten

Elke vier weken mag een chauffeur niet meer dan drie zaterdagen werken, zie (3.19). Daarnaast mag een chauffeur maximaal 9 zondagen werken in 13 weken, zie (3.20). Deze voorwaarden zijn doorlopend, ze gelden dus voor elke mogelijke combinatie van 4 opeenvolgende weken. Elke week moeten we daarom opnieuw controleren of nog aan de voorwaarde voldaan wordt.

$$\sum_{d=1}^{4} \sum_{b \in \mathcal{B}^+} X_{cb}^{7d+7t} \leq 3 \qquad \forall c \in \mathcal{C}; \forall t = 0, ..., 48 \tag{3.19}$$

$$\sum_{d=1}^{13} \sum_{b \in \mathcal{B}^+} X_{cb}^{7d-6+7t} \leq 9 \qquad \forall c \in \mathcal{C}; \forall t = 0, ..., 39 \tag{3.20}$$

Een chauffeur die op zondag moet werken, moet ook altijd ingepland worden op zaterdag, zie voorwaarde (3.21). Andersom is niet verplicht en daarom kunnen we dit in wiskundige notatie ook als volgt noteren: zondag werken $\Rightarrow$ zaterdag werken.

$$\sum_{b \in \mathcal{B}^+} X_{cb}^{7d-6} \leq \sum_{b \in \mathcal{B}^+} X_{cb}^{7d} \qquad \forall c \in \mathcal{C}; \forall d = 1, ..., 51 \tag{3.21}$$

Elke week mag een chauffeur maximaal twee gebroken diensten (diensten met minimaal anderhalf uur pauze) toegewezen krijgen. De som over alle gebroken diensten (blok D met code 33) mag daarom niet meer dan 2 per week zijn. Deze voorwaarde is uitgewerkt in (3.22).

$$b_{33} = \text{ het blok dat de gebroken dienst representeert}$$

$$\sum_{t=0}^{6} X_{cb_{33}}^{7d+t} \leq 2 \qquad \forall c \in \mathcal{C}; \forall d = 0, ..., 51 \tag{3.22}$$

3.2.5 Werkdagen en vrije dagen

Een chauffeur moet over een bepaalde periode altijd een bepaald aantal dagen werken en een aantal dagen roostervrij zijn. Eén van de eisen aan de chauffeurs is dat zij minimaal 56 en maximaal 61 werkdagen krijgen toegewezen per 91 dagen (13 weken). Deze voorwaarden zijn verwerkt in respectievelijk (3.23) en (3.24).

$$\sum_{d=1}^{91} \sum_{b \in \mathcal{B}^+} X_{cb}^{d+91t} \geq 56 \qquad \forall c \in \mathcal{C}; \forall t = 0, ..., 3 \tag{3.23}$$

$$\sum_{d=1}^{91} \sum_{b \in \mathcal{B}^+} X_{cb}^{d+91t} \leq 61 \qquad \forall c \in \mathcal{C}; \forall t = 0, ..., 3 \tag{3.24}$$

De laatste restricties betreffen de vrije dagen van de chauffeurs. Elke chauffeur moet elke 91 dagen minimaal 30 vrije dagen toegewezen krijgen, zie (3.25), en elke 182 dagen (of 26 weken) precies 65 vrije dagen, zie (3.26).

$$\sum_{d=1}^{91} X_{cb_0}^{d+91t} \geq 30 \qquad \forall c \in \mathcal{C}; \forall t = 0, ..., 3 \tag{3.25}$$

$$\sum_{d=1}^{182} X_{cb_0}^{d+182t} = 65 \qquad \forall c \in \mathcal{C}; \forall t = 0, 1 \tag{3.26}$$

3.3 Realiteitswaarde van het model

We hebben hierboven één voorwaarde genegeerd, die stelt dat de chauffeurs altijd gemiddeld 9 van elke 14 dagen moeten werken. Deze voorwaarde wordt echter al ondervangen door de andere voorwaarden op de werkdruk en de weekrust. Deze voorwaarde hebben we hier daarom niet verwerkt in een aparte restrictie.

We hebben bij het formuleren van de voorwaarden daarnaast geen rekening gehouden met eventuele uitzonderingen op de regels. De meeste regels zijn bedoeld om de chauffeurs te beschermen, maar vaak is het toegestaan om de regels te negeren als de chauffeurs dit zelf wensen. De implementatie van dit soort uitzonderingen vereist vaak meerdere extra voorwaarden, waardoor het model al snel onoverzichtelijk wordt. We hebben de uitzonderingen hier weggelaten, maar we zullen ze wel zoveel mogelijk toepassen bij onze oplossingsmethode.

In hoofdstuk 4 zullen we bespreken welke literatuur er over personeelsplanning geschreven is. Daar zullen ook bestaande oplossingsmethoden aan bod komen en zullen we nagaan welke methoden we eventueel kunnen gebruiken om een matching voor Connexxion te ontwikkelen.

Hoofdstuk 4

Literatuuronderzoek

Het onderzoek naar personeelsplanning heeft inmiddels een rijke geschiedenis. De eerste stappen naar een op de Operations Research gebaseerde aanpak werden gemaakt toen Dantzig (1954) een model opstelde waarbij de personeelsplanning werd geformuleerd als een set covering probleem. Er zijn vele verschillende varianten van dit probleem. In Lau (1996) is reeds bewezen dat deze problemen NP-hard zijn tenzij we uitgaan van een sterke versimpeling van het probleem.

4.1 Terminologie

In dit hoofdstuk spreken we consequent van 'werknemers', omdat de artikelen niet allemaal specifiek over buschauffeurs gaan. Veelal maakt dit echter geen verschil voor de toepasbaarheid van de beschreven methoden. Ook blijkt uit de literatuur dat veel van de voorwaarden die bij Connexxion gelden (met betrekking tot werktijden en vrije dagen), universeel zijn voor personeelsplanning.

In de loop der tijd zijn er al verscheidene overzichtsartikelen geschreven over de literatuur die tot nu toe verschenen is op het gebied van personeelsplanning (zoals Alfares (2004), Burke e.a. (2004) en Ernst e.a. (2004a)). Veel van de uitspraken in dit hoofdstuk zijn gebaseerd op conclusies uit die artikelen.

In de literatuur is vooral veel aandacht besteed aan het crew scheduling probleem (fase 4 uit figuur 1.1) en minder aan het crew rostering probleem, waar we hier mee te maken hebben (Bianco e.a. (1992)). De verschillende termen van de personeelsplanning worden ook vaak door elkaar gebruikt. Grötschel e.a. (2003) besteedt bijvoorbeeld aandacht aan 'duty scheduling', het achter elkaar plannen van bepaalde taken om dagdiensten te maken. Dit proces valt bij Connexxion en in veel andere artikelen onder de crew scheduling fase.

In de laatste jaren ontstaat steeds meer een algemene consensus over de terminologie op het gebied van personeelsplanning. Ernst e.a. (2004b) geven een overzicht van de meest gangbare termen en die hebben we hier overgenomen.

Diensten (of duty's) Een dienst is een bepaalde taak waarvan de begin- en eindtijd bekend is inclusief eventuele pauzes; dit soort diensten worden in de crew scheduling fase ontwikkeld.

Pattern Een pattern is het schema van werkdagen en roostervrije dagen van een werknemer,

zie bijvoorbeeld Caprara e.a. (2003). De precieze taken of blokken die de werknemer op de werkdagen toegewezen krijgt, zijn hier nog niet in meegenomen.

Line of work (LoW) Een reeks van verschillende dagen waarvan de diensten en de vrije dagen al zijn ingepland, maar waar nog geen werknemer aan verbonden is.

Days-off scheduling Bij days-off scheduling worden de vrije dagen van een werknemer ingepland, waardoor een pattern ontstaat.

Shift scheduling Shift scheduling richt zich op het toekennen van bepaalde shifts aan werknemers, waarbij een shift overeenkomt met de blokken waar Connexxion mee werkt. Vaak worden er nog pauzes tijdens de shifts ingepland, wat bij Connexxion al van tevoren is gedaan.

Tour scheduling Bij tour scheduling worden de bovenstaande twee problemen gecombineerd en worden ze tegelijkertijd opgelost (zoals beschreven in Baker (1976)).

Bidline Bij veel luchtvaartmaatschappijen kunnen de werknemers 'bieden' op de geplande LoW's, waarna gepoogd wordt om zo veel mogelijk aan deze wensen te voldoen. Dit is anders dan bij Connexxion, want bij de bidlines wordt geboden op een gehele reeks van diensten en vrije dagen, terwijl Connexxion een systeem aanbiedt waarbij per dag wensen ingevuld kunnen worden.

Preferential bidding Net zoals bij de bidlines wordt dit systeem veel toegepast bij luchtvaartmaatschappijen. Bij preferential bidding kunnen de werknemers van tevoren voorkeuren uitspreken over specifieke diensten en vrije dagen, vaak met een bepaald gewicht. Het zijn hier meestal niet wensen per dag, maar algemene voorkeuren over werktijden en routes.

Personalized rostering Als het dienstrooster voor elke werknemer specifiek wordt gemaakt, dan spreken we van personalized rostering (in tegenstelling to bidline generation, waar eerst een rooster wordt gemaakt waar nog geen werknemer aan gekoppeld is).

4.2 Roosterplanning in de literatuur

In de literatuur wordt vooral aandacht besteed aan het opstellen van dienstroosters waarbij de voorkeuren van de werknemers niet worden meegenomen. Binnen deze groep bestaat er veel onderscheid in zowel de probleemomschrijving als de gebruikte oplossingsmethoden. Hieronder geven we een korte samenvatting van de beschikbare literatuur, waarbij we ook bespreken of het mogelijk is om de transitie naar onze situatie te maken.

Sommige artikelen richten zich op de days-off planning. Als we een algoritme kunnen maken voor de days-off planning, dan kunnen we de blokken toekennen door een semi-assignment probleem op te lossen (zie hoofdstuk 3). In Berrada e.a. (1996) en Alfares (1998) worden bijvoorbeeld modellen gegeven om het days-off scheduling probleem op te lossen, maar het betreft hier altijd algemene roosters die niet gemaakt zijn om aan persoonlijke wensen te voldoen. Daarnaast maken deze roosters meestal gebruik van een bepaalde vorm van patroonherhaling. De wensen van de chauffeurs bij Connexxion zijn echter vaak acyclisch van aard en daarom is het niet wenselijk om patroonherhaling toe te passen. We hebben geen literatuur kunnen vinden die de days-off planning toepast om persoonlijke days-off planningen te maken. Wel weten we dat deze aanpak door FlexPlan wordt toegepast.

Veel artikelen zijn geschreven over tour scheduling (Morris en Showalter (1983), Baxter en Mosby (1988), Gärtner e.a. (2001), Day en Ryan (1997), Bard e.a. (2003), Sodhi en Norris (2004) en Caprara e.a. (1998)), maar deze leggen de focus op cyclische planning. Ook is het doel zelden om aan de wensen van specifieke werknemers te voldoen, maar wil men een algemeen doel nastreven zoals het minimaliseren van het aantal benodigde werknemers, of het gelijkmatig verdelen van de werkdruk. Voorbeelden hiervan zijn te vinden in Yunes e.a. (2005), Lezaun e.a. (2007), Teodorović en Lucić (1998), Caprara e.a. (1997), Caprara e.a. (1999), Ryan (1992) en Caprara e.a. (2001). Sommige van de latere artikelen zetten daarbij ook de eerste stappen naar een integratie van crew scheduling en crew rostering. De methodes die in deze artikelen worden gebruikt zijn er meestal niet op gericht om persoonlijke roosters te maken, maar om een groot aantal verschillende LoW's te maken waarna deze pas in de laatste stap aan een naam worden gekoppeld. In ons geval willen we de wensen van elk individu al direct meenemen bij het maken van de LoW's en daarom zijn de methodes die hier worden beschreven niet goed bruikbaar voor de situatie bij Connexxion.

In de literatuur is regelmatig aandacht voor het maken van rouleringen. Townsend (1988), Sodhi en Norris (2004) en Lezaun e.a. (2006) besteden hier bijvoorbeeld aandacht aan. Rouleringen roteren wekelijks tussen de werknemers, zodat de werknemers iedere week een ander dienstrooster hebben. In een bepaalde periode krijgt elke werknemer dan elk weekrooster een keer, zodat het werk eerlijk verdeeld wordt over alle werknemers. Dit systeem gebruikt Connexxion nu voor het grootste deel van de vestigingen, maar de rouleringen zijn niet eenvoudig aan te passen om aan de wensen van zelfroostering te voldoen.

In Caprara e.a. (2003) wordt een model gemaakt dat uit twee stappen bestaat. Eerst wordt een pattern gemaakt en daarna worden specifieke taken toegewezen door herhaaldelijk een transportprobleem op te lossen; dit is een speciale vorm van het assignment of het semi-assignment probleem (zie bijvoorbeeld Hillier en Lieberman (2005)) en daarom lijkt dit veel op het door ons gebruikte mathematische model. Bij dit model is echter geen sprake van een wensensysteem en daarom zoeken ze alleen naar een toegelaten oplossing, waardoor dit model een stuk eenvoudiger is dan de situatie bij Connexxion.

Er zijn verschillende algoritmes die zich specifiek op busplanning richten. Zo hebben Bianco e.a. (1992) een heuristisch algoritme ontwikkeld om een Bottleneck Assignment Problem (BAP) op te lossen, waarbij ze de maximale werkdruk minimaliseren. Dit model is ook toepasbaar op onze situatie als we er voor kiezen om het minimale matchingspercentage per chauffeur te maximaliseren. Een nadeel van deze methode is echter dat het voor langere planningsperioden veel tijd in beslag neemt.

Gamache e.a. (1999) maken een dienstrooster voor de luchtvaart, waarbij het doel is om zo veel mogelijk taken toe te kennen aan de normale werknemers zodat de reserveploeg zo weinig mogelijk taken hoeft uit te voeren. Junker e.a. (1999) en Fahle e.a. (2002) gebruiken een combinatie van column generation en constraint programming om het crew rostering probleem op te lossen. Kohl en Karisch (2004) beschrijven een uitgebreid rekenprogramma, genaamd Carmen Crew Rostering, dat de voorgaande artikelen combineert in een programma dat oplossingen kan vinden voor allerlei aspecten van de crew rostering.

In alle bovengenoemde artikelen blijkt echter dat de oplossingsmethode gericht is op een model met als doel om gelijkwaardige dienstroosters te maken en daarom is het bijna onmogelijk om deze methoden te integreren met persoonsgebonden wensen.

4.3 Wensen van de werknemers

Er is relatief weinig onderzoek gedaan naar het tegemoet komen aan de wensen van de werknemers bij het maken van roosters, omdat het gebruik van wensen een nieuwe ontwikkeling is op het gebied van roosterplanning. Beaulieu e.a. (2000) ontwikkelen een model om het inplannen van artsen te automatiseren. Zij maken wel melding van het implementeren van wensen die de artsen van tevoren opgeven, maar het betreft hier meer algemene voorkeuren die pas terugkomen bij de evaluatie van het dienstrooster. Bij het bepalen van een oplossing worden de wensen nog niet meegenomen.

Het bidline systeem is het bekendste systeem in de literatuur waarbij de werknemers betrokken worden bij het planningsproces. Eerst worden de 'bidlines' gemaakt, complete LoW's voor een langere periode. Vervolgens geven de werknemers aan welke LoW zij zouden willen krijgen. Uiteindelijk worden zo veel mogelijk wensen ingewilligd, eventueel met een vorm van preferentie zoals 'seniority'. Bij seniority wordt er onderscheid gemaakt tussen werknemers op grond van het aantal jaren dat ze in dienst zijn. Voorbeelden van artikelen over dit type dienstroosters (vooral toegepast bij luchtvaartmaatschappijen) zijn Jarrah (1997), Weir en Johnson (2004), Christou e.a. (1999), Jones (1989) en meer recentelijk Boubaker e.a. (2010). Hoewel de werknemers hier veel inspraak hebben, worden de LoW's in de praktijk nog altijd apart gemaakt zonder dat de werknemers hier invloed op hebben. Bij zelfroostering is het juist de bedoeling dat de werknemers al direct invloed hebben op het maken van de LoW's.

Over preferential bidding is relatief weinig geschreven en bijna altijd in combinatie met seniority. Onder meer Butchers e.a. (2001), Gamache e.a. (2007), Desrosiers e.a. (2000) en Gamache e.a. (1998) hebben hier onderzoek naar gedaan. De methodes zijn allemaal gericht op het maken van een optimaal rooster vanuit één werknemer, terwijl het dienstrooster voor alle andere werknemers op zijn minst feasible is. Alle oplossingsmethodes gaan ervan uit dat er onderscheid gemaakt kan worden tussen de werknemers en deze situatie zal dan ook nooit gelijkwaardige dienstroosters opleveren. Bij Connexxion hebben alle werknemers evenveel rechten en daarom is het belangrijk dat iedere werknemer ongeveer hetzelfde matchingspercentage heeft.

4.4 Persoonlijke dienstroosters

De laatste ontwikkeling in de literatuur is het systeem van personalized rostering. Dit is een nieuw onderzoeksgebied dat zich richt op het maken van unieke persoonlijke dienstroosters. Daarbij worden de wensen van alle werknemers apart meegenomen en dit komt dan ook het dichtst in de buurt van zelfroostering. We stellen eerst twee mogelijke oplossingsmethoden die gebaseerd zijn op de gevonden literatuur. Daarna stellen we een nog een eigen oplossingsmethode voor.

4.4.1 Days-off en shift scheduling

Op dit moment wordt het probleem door FlexPlan opgesplitst in days-off en shift scheduling (zie hoofdstuk 2). In de literatuur komt deze aanpak van roosterplanning ook regelmatig voor. Day en Ryan (1997) onderscheiden zich met een methode die breed toepasbaar is. Zij genereren voor een periode van twee weken alle mogelijke patterns en kiezen dan per chauffeur de beste op basis van een arbitraire kostenfunctie. Vervolgens worden de shifts op de werk-

dagen ingepland. Als we de kostenfunctie aanpassen, dan kunnen we deze methode relatief makkelijk omzetten naar een model voor zelfroostering. In het artikel is de roosterperiode echter slechts twee weken terwijl deze bij ons een jaar is.

De toepassing van deze methode voor langere roosterperioden is bij de Nederlandse Spoorwegen (NS) onderzocht (de Pont (2006)). Zij passen de methode van Hartog e.a. (2005) aan, die weer gebaseerd is op Day en Ryan (1997). Eerst worden alle diensten opgedeeld in een soort blokkenstructuur zoals bij Connexxion ook bestaat (hoewel zij minder blokken gebruiken). Vervolgens gaan zij voor elke week alle mogelijke combinaties van blokken generen en kiezen ze de beste voor elke werknemer. Daarna pogen ze voor de volgende week om een nieuw weekrooster aan de eerste toe te voegen. Uit de resultaten blijkt echter dat deze methode veel rekentijd kost doordat er veel 'backtracking' plaats vindt als het onmogelijk blijkt te zijn om een nieuw weekrooster toe te voegen.

4.4.2 Dienstroosters per chauffeur

Een andere methode, beschreven in Maenhout en Vanhoucke (2007), splitst het probleem op in een 'deelprobleem' voor elke werknemer en een 'masterprobleem' waarbij de dienstroosters voor elke werknemer worden samengevoegd. Het deelprobleem bestaat uit het oplossen van een 'Shortest Path Problem met Resource Constraints' (SPPRC). Voor elke werknemer maken ze een graaf waarbij elke knoop een bepaald blok op een specifieke dag representeert (de roostervrije dagen tellen hier ook als een blok). Het masterprobleem is vervolgens om voor elke dag uit te zoeken welke blokken te weinig of te veel zijn toegewezen. Als we bijvoorbeeld in totaal vijf chauffeurs hebben toegewezen aan blok A op dag 1 terwijl we eigenlijk tien keer een blok A moeten toewijzen, dan is de oplossing voor die dag niet toegelaten.

Het nadeel van deze methode is dat het veel tijd kost om de optimale oplossing van een SPPRC te vinden. Deze methode heeft echter als voordeel dat we elke iteratie een toegelaten dienstrooster hebben voor een chauffeur, zodat het niet nodig is om bij het oplossen van het masterprobleem rekening te houden met de rules.

4.4.3 Dienstroosters per dag

We stellen zelf een derde methode voor, die een afgeleide is van het mathematisch model. Elke dag kunnen we oplossen als een semi-assignment probleem. Op die manier kennen we elke dag precies genoeg blokken toe om te voldoen aan de vraag naar de blokken (het aantal blokken dat toegekend moet worden). We combineren de oplossingen zodat we voor elke chauffeur een dienstrooster hebben voor elke dag van de planningsperiode. Daarbij zorgen we ervoor dat de oplossingen toegelaten zijn met betrekking tot de rules.

Deze aanpak is een soort inverse van de tweede methode. Bij die methode maken we eerst een oplossing waarbij we geen rekening houden met de vraag naar de blokken maar wel met de rules. Bij deze methode houden we aanvankelijk wel rekening met de vraag naar de blokken maar niet met de rules. Uiteindelijk bestaat het probleem bij beide methodes uit twee fasen, waarbij we in de tweede fase proberen om ook aan de overige voorwaarden te voldoen.

Het voordeel van deze methode is dat het oplossen van het semi-assignment probleem erg makkelijk en snel kan. Aan de andere kant wordt het mogelijk erg moeilijk om aan de rules te voldoen.

4.5 Keuze voor dienstroosters per chauffeur

De oplossingsmethodes die we in de vorige paragrafen hebben beschreven, hebben allen als kenmerk dat de methode een kostenfunctie gebruikt die geen invloed heeft op de bruikbaarheid van de methode. Dat betekent dat we naar vrij inzicht bepaalde kostencomponenten kunnen toevoegen en verwijderen, of van een bepaald gewicht kunnen voorzien. Op die manier kunnen we later nog besluiten om bepaalde maatstaven wel of niet mee te nemen, afhankelijk van de wensen binnen Connexxion.

De eerste methode splitst het probleem op in days-off en shift scheduling, zoals FlexPlan ook doet. Het risico van deze methode is dat het weinig toevoegt aan dit reeds aanwezige softwarepakket. Daarnaast blijkt dat er weinig beschikbare literatuur is over succesvolle toepassingen van deze methode. We hebben er daarom voor gekozen om een andere aanpak te gebruiken.

Bij Connexxion is gebleken dat de rules vaak het grootste struikelblok vormen bij het vinden van een toegelaten dienstrooster. Vooral het maken van een toegelaten pattern van vrije dagen en werkdagen voor een chauffeur geeft vaak moeilijkheden.

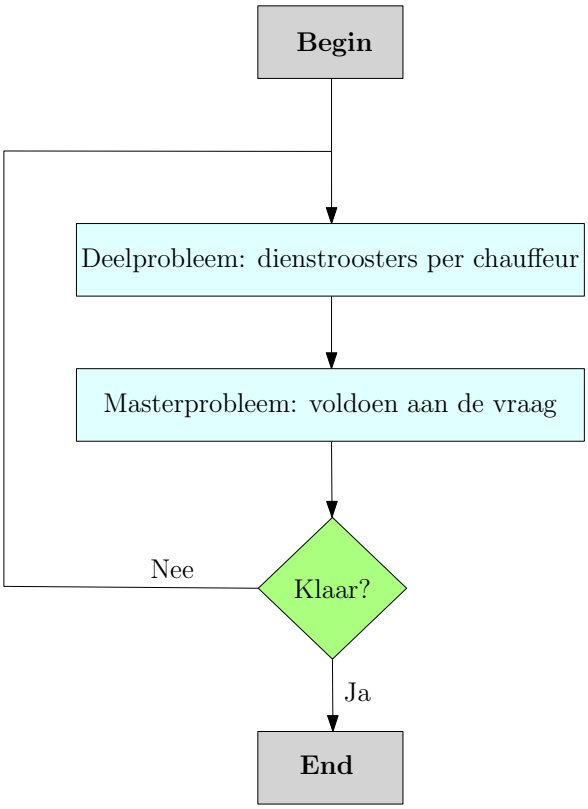
We hebben daarom besloten om dienstroosters per chauffeur op te stellen (de methode uit paragraaf 4.4.2), omdat we inschatten dat deze methode een grotere kans heeft om uiteindelijk tot een toegelaten oplossing te komen.

In hoofdstuk 5 beschrijven we in detail hoe we een dienstrooster opstellen voor één chauffeur en in hoofdstuk 6 beschrijven we hoe we vervolgens iteratief proberen om aan de vraag naar de blokken te voldoen. In hoofdstuk 7 beschrijven we de resultaten van deze methode.

Hoofdstuk 5

Deelprobleem

In het vorige hoofdstuk hebben we beschreven hoe we gaan proberen om de matching te maken. We gaan eerst dienstroosters per chauffeur opstellen, wat we hier definiëren als het deelprobleem. Daarna gaan we proberen om deze dienstroosters te combineren om zo aan de vraag naar de blokken te voldoen. Deze tweede stap noemen we het masterprobleem. In figuur 5.1 geven we een schematisch overzicht van de wisselwerking tussen het deelprobleem en het masterprobleem.



Figuur 5.1: *Een schematisch overzicht van het gebruikte algoritme om de matching te maken. Het algoritme bestaat uit het oplossen van een deelprobleem (het opstellen van dienstroosters per chauffeur) en het oplossen van een masterprobleem (het voldoen aan de vraag naar de blokken).*

Onze oplossingsmethode voor het deelprobleem genereert dienstroosters die als input dienen voor het masterprobleem. Als het niet lukt om het masterprobleem direct op te lossen, dan zullen we opnieuw het deelprobleem oplossen waarbij we bepaalde parameters en restricties aanpassen. In hoofdstuk 6 leggen we uit hoe we het masterprobleem oplossen en welke maatregelen we nemen voordat we een nieuwe iteratie uitvoeren. In dit hoofdstuk beperken we ons tot het oplossen van het deelprobleem.

Het deelprobleem bestaat uit het vinden van een dienstrooster voor één chauffeur. We lossen een Shortest Path Probleem met Resource Constraints (SPPRC) op met behulp van dynamische programmering. Afhankelijk van de voorwaarden kunnen we een aantal resources definiëren, op grond waarvan we verschillende paden met elkaar kunnen vergelijken. Gedurende de uitvoering van het algoritme onthouden we alleen de Pareto-optimale paden, dat zijn de paden die niet gedomineerd worden door andere paden. Het optimale pad bevindt zich altijd in de verzameling van dominante paden (Bentley (1980)), zodat we alle andere paden kunnen negeren. Het algoritme voor het oplossen van de SPPRC is gebaseerd op het algoritme van Irnich en Desaulniers (2005).

5.1 De graaf

We maken een graaf G waarbij elke knoop staat voor een bepaald blok op een dag (we gebruiken de blokken uit tabel 1.1). De planningsperiode is in totaal 52 weken (364 dagen). We gebruiken een dummy startknoop s met uitgaande takken naar alle knopen van dag 1 en een dummy eindknoop t met ingaande takken van dag 364 naar t toe. Elke dag heeft maximaal negen knopen en het aantal knopen is daarom maximaal $364 \times 9 = 3276$. Elke knoop heeft maximaal negen uitgaande takken (naar alle knopen van de volgende dag) en daarom is het aantal takken begrensd door $3276 \times 9 = 29484$.

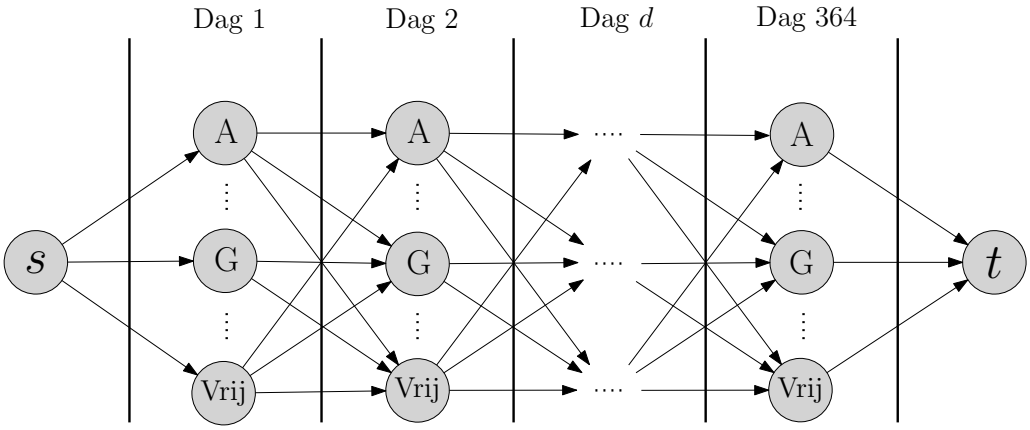
We verwerken enkele rules al direct in de graaf. We geven nu een kort overzicht van deze rules.

- Eén van de rules stelt dat een chauffeur die zaterdag vrij krijgt, ook zondag vrij moet krijgen. Dat betekent dat er van het roostervrije blok op zaterdag alleen een tak bestaat naar het roostervrije blok op zondag.

- Een chauffeur mag nooit eerder dan 11 uur beginnen na de eindtijd van een blok. We kunnen van tevoren al bepalen welke combinaties van blokken op deze manier verboden zijn. Er bestaat daarom geen tak tussen twee blokken op opeenvolgende dagen als de tussentijd minder dan 11 uur is.

In figuur 5.2 staat een schematisch overzicht van de SPPRC graaf. In dit geval hebben we slechts drie blokken getoond, A (van 4:30 tot 14:00 uur, G (van 23:00 tot 7:00 uur de volgende ochtend) en 'VRIJ', het blok dat een roostervrije dag weergeeft. De planningsperiode is 364 dagen, met een dummy startknoop en een dummy eindknoop. We gaan er voor het gemak van uit dat alle getoonde dagen weekdagen zijn (in het weekend worden andere blokken gebruikt).

De graaf is acyclisch, waarbij elke tak altijd naar de volgende dag gaat. Meestal bestaan er takken van een knoop naar alle knopen op de volgende dag, maar de tak van blok G naar A bestaat bijvoorbeeld niet, omdat de tussentijd minder dan 11 uur is.



Figuur 5.2: *Een schematische weergave van de SPPRC graaf. We tonen enkele weekdagen waarbij we de blokken A, G en VRIJ hebben uitgelicht. Van blok G naar A bestaat geen tak omdat de tussentijd minder dan 11 uur is.*

5.2 Randvoorwaarden van de chauffeurs

We zijn tot nu toe uitgegaan van een model met alleen fulltime chauffeurs. In de praktijk werkt een groot deel van de chauffeurs parttime en voor hen gelden andere voorwaarden dan voor fulltime chauffeurs. Een parttime chauffeur die voor 50% werkt, heeft slechts 50% van het aantal werkdagen in een bepaalde periode. De rules voor het aantal werkdagen en het aantal roostervrije dagen moeten daarom aangepast worden voor deze chauffeurs. Elke chauffeur krijgt daarom een persoonlijk aantal vrije dagen.

Voor sommige chauffeurs gelden verder speciale regelingen die vaak te maken hebben met de leeftijd of medische achtergrond van de chauffeur. Eén van deze randvoorwaarden is bijvoorbeeld dat chauffeurs geen nachtdiensten krijgen toegewezen als ze ouder dan 50 jaar zijn.

5.2.1 Vakantieschema

Van tevoren wordt een vakantieschema opgesteld, zodat bekend is wanneer de chauffeurs vakantie krijgen en op welke feestdagen ze vrij zijn. Ook wordt van de vakantie al besloten welke dagen als werkdagen tellen en welke als roostervrije dag. We houden tijdens het bepalen van de SPPRC bij hoe veel roostervrije dagen en hoe veel werkdagen een chauffeur heeft gehad en daarom gebruiken we dit vakantieschema ook voor ons model. In tabel 5.1 staat een voorbeeld van een vakantieschema.

De kolom 'Telt als' geeft aan of de dagen in de periode tellen als werkdag (waarvoor de chauffeur verlof op moet nemen) of als roostervrije dag. Deze chauffeur heeft bijvoorbeeld vrij op nieuwjaarsdag, tweede paasdag, hemelvaartsdag en met kerst. De kerstdagen tellen als roostervrije dagen en de overige feestdagen als werkdagen. Onder de dubbele lijn staat de vakantieperiode. De vakantie van deze chauffeur duurt van 19 augustus tot en met 7 september, maar in die periode neemt hij slechts tien vakantiedagen op omdat de overige dagen als roostervrije dagen tellen.

We beschouwen het vakantieschema van een chauffeur als gegeven. Dat heeft invloed op het

Van	Tot en met	Beschrijving	Telt als
01-01-2010	01-01-2010	Nieuwjaarsdag	Werk
05-04-2010	05-04-2010	Tweede Paasdag	Werk
13-05-2010	13-05-2010	Hemelvaartsdag	Werk
25-12-2010	26-12-2010	Kerst	Vrij
19-08-2010	22-08-2010	Vakantie	Vrij
23-08-2010	27-08-2010	Vakantie	Werk
28-08-2010	29-08-2010	Vakantie	Vrij
30-08-2010	03-09-2010	Vakantie	Werk
04-09-2010	07-09-2010	Vakantie	Vrij

Tabel 5.1: *Een voorbeeld van een vakantieschema van een chauffeur. Boven de dubbele lijn staan de feestdagen waarop de chauffeur vrij is en onder de dubbele lijn worden de vakantiedagen van de chauffeur vermeldt. De derde kolom geeft aan of we de dagen rekenen als werkdagen of als roostervrije dagen.*

dienstrooster, omdat we vooral rond de vakantieperiode worden beperkt in de keuzemogelijkheden.

De dagen die als verlofdag tellen, worden gemarkeerd als 'reservedienst' (door een RES-blok toe te wijzen). Op deze manier wordt kunstmatig afgedwongen dat we geen blok toekennen aan de chauffeur op deze dagen.

5.3 De takkosten

Elke (gerichte) tak heeft een 'VanKnoop', de knoop waar de tak vandaan komt, en een 'NaarKnoop', de knoop waar de tak naar toe gaat. Daarnaast heeft een tak bepaalde kosten, die we van tevoren definiëren.

De takkosten van de graaf zijn van nature niet gespecificeerd, omdat er niet direct kosten zijn verbonden aan het toewijzen van bepaalde blokken. Het loon van de chauffeurs is immers onafhankelijk van hun dienstrooster, ze krijgen een vast uurloon gebaseerd op hun contract. Er zijn dan ook alleen virtuele kosten verbonden aan de toewijzing van bepaalde blokken.

We gebruiken de takkosten in het algoritme daarom als stuurmiddel om bepaalde toewijzingen aantrekkelijker of juist minder aantrekkelijk te maken. Een belangrijke reden om het algoritme te sturen is het verbeteren van het matchingspercentage. We baseren de takkosten daarom op de wensen van de chauffeurs. We hebben ervoor gekozen om steeds uit te gaan van de wensen voor de NaarKnoop van de tak. De takken vanuit s hebben dus takkosten die gebaseerd zijn op de wens van de chauffeur voor dag 1. De takken vanaf dag 365 naar dag t hebben allemaal takkosten van 0.

We hebben aanvankelijk gekozen voor een systeem waarbij de takkosten 0 zijn als de NaarKnoop overeenkomt met de wens van de chauffeur, en 1 anders. Op deze manier is het goedkoopste pad altijd het pad dat de wensen van de chauffeur volgt.

Het gebruik van takkosten van 0 of 1 heeft als voordeel dat het eenvoudig te implementeren is, maar deze methode heeft ook een aantal nadelen.

- Er is weinig variatie in de kosten. Elke tak die niet aan de wensen voldoet, zal daarom dezelfde kosten hebben en het algoritme kan dan minder goed onderscheid maken tussen de verschillende takken en paden.

- Een aantal bekende en veelgebruikte oplossingsmethodes die we kunnen gebruiken voor het masterprobleem zijn erg gevoelig voor de takkosten. Een voorbeeld hiervan is Lagrange Relaxatie, dat over het algemeen veel moeite heeft om tot een oplossing te komen bij dit soort binaire takkosten.

Voor het oplossen van het deelprobleem zijn takkosten van 0 of 1 eenvoudig te implementeren en makkelijk in het gebruik. Als we echter later vaker willen gaan itereren en meer controle willen uitoefenen op de toewijzingen, dan is deze methode niet flexibel genoeg. We hebben er daarom voor gekozen om meer variatie in de kosten aan te brengen.

5.3.1 Huidige kostencomponenten

We geven een overzicht van de verschillende manieren waarop de takkosten worden beïnvloed.

- De wensen van de chauffeurs zijn de belangrijkste maatstaf voor Connexxion. Het zal echter niet altijd mogelijk zijn om aan alle wensen van de chauffeurs te voldoen. In dat geval vindt een chauffeur het over het algemeen minder erg als hij een blok krijgt dat 'in de buurt' ligt. Als iemand bijvoorbeeld een blok A (een ochtendblok) wenst, dan is het verschil met de andere ochtendblokken (B of C) veel minder groot dan het verschil met een nachtblok (F of G). Het is daarom aannemelijk dat de kosten voor een blok G op deze dag groter zijn dan de kosten voor een blok B en dat implementeren we met behulp van strafkosten op de takken die qua tijdinterval ver bij de wens vandaan liggen.

  We hebben de kosten daarom aangepast zodat deze nuance wordt meegenomen. Voor elk blok bepalen we het verschil in starttijd en het verschil in eindtijd met de wens. De som van deze verschillen bepaalt de kosten voor dit blok. De kosten van een blok A op een dag dat de wens een blok B is, wordt dan als volgt berekend: $Kosten = |4{,}5-6|+|14-16| = 3{,}5$ (zie tabel 1.1 voor de start- en eindtijd van elk blok). We zeggen daarom dat blok A een factor van 3,5 bij blok B vandaan ligt. We houden een maximum aan van 10 voor deze factor, zodat het verschil niet te groot wordt. Het verschil tussen blok A en G is bijvoorbeeld 10, terwijl het eigenlijk $|4{,}5-23| + |14 - 31| = 25{,}5$ zou moeten zijn. Een roostervrije dag krijgt ook de verschilfactor 10 mee, omdat dit blok geen starttijd of eindtijd heeft (ervan uitgaande dat de chauffeur geen roostervrij blok wenst op de betreffende dag).

- Op elke dag weten we wat de vraag is naar de verschillende blokken. Sommige blokken zijn bijvoorbeeld slechts 1 of 2 keer nodig, terwijl andere blokken 10 keer gevraagd worden op een bepaalde dag. Om het oplossen van het masterprobleem te versnellen, verwerken we dit gegeven in de kosten. De kosten voor elke knoop worden daarom afhankelijk van het blok en de dag van de knoop verhoogd met de volgende term: $AantalKeerGewenst(Blok, Dag) - Vraag(Blok, Dag)$. Als een blok veel nodig is en weinig gewenst wordt, dan zal deze term negatief zijn en dan worden bepaalde toewijzingen aantrekkelijker. In het andere geval worden de kosten positief en wordt de toewijzing minder aantrekkelijk.

- Als we de SPPRC bepalen voor een chauffeur, dan weten we al wat de voorgaande chauffeurs toegewezen hebben gekregen. Zo kunnen we nagaan of bepaalde blokken al genoeg of meer dan genoeg toegewezen zijn op een specifieke dag. Tegelijkertijd hebben we een groep chauffeurs die geen of weinig wensen hebben ingevuld. We willen de toewijzingen van deze chauffeurs eigenlijk sturen naar de knopen die we nog nodig hebben.

  Daarom maken we de takkosten oneindig voor takken naar knopen die niet meer nodig zijn, mits de chauffeur op de betreffende dag geen wens had ingevuld. Wensvervulling heeft in dit geval dus voorrang boven de sturende werking van deze kostencomponent.

  We sorteren de chauffeurs eerst op grond van het aantal ingevulde wensen van hoog naar laag. We hebben dan zo veel mogelijk informatie over de andere chauffeurs op het moment dat we de dienstroosters bepalen voor de chauffeurs die weinig tot geen wensen hebben ingevuld. Op die manier halen we het maximale effect uit deze strafkosten.

- Gedurende de verschillende iteraties van het masterprobleem kan het voorkomen dat bepaalde blokken te veel en bepaalde blokken te weinig zijn toegewezen. Met behulp van Lagrange Relaxatie of met een heuristische methode is het kortste pad algoritme zodanig te sturen dat de tekorten of overschotten worden weggewerkt totdat we een toegelaten oplossing voor het master probleem krijgen. We zullen hier verder op ingaan in hoofdstuk 6.

- We willen het matchingspercentage maximaliseren van elke chauffeur, maar tegelijkertijd willen we niet dat de verschillen onderling te groot zijn. Gemiddeld wordt een matchingspercentage van 80% nagestreefd en de chauffeurs die een lager percentage hebben, willen we zo veel mogelijk op het niveau van 80% krijgen. Daarom gebruiken we een schalingsparameter voor de strafkosten. Als het matchingspercentage van een chauffeur lager dan 80% is, dan is de schaling ook lager waardoor de wensen relatief meer invloed hebben op de kosten. Deze schaling kunnen we pas toepassen vanaf de tweede iteratie, omdat we eerst een oplossing voor de SPPRC moeten hebben gevonden voordat we het matchingspercentage kunnen berekenen.

5.3.2 Mogelijke kostencomponenten

Eventueel is het mogelijk om de takkosten nog verder aan te passen. Hieronder geven we een overzicht van een aantal mogelijke kostencomponenten die nu nog niet gebruikt worden, maar die eventueel kunnen worden toegevoegd als dit wenselijk is binnen Connexxion.

- Bij Connexxion geven chauffeurs op dit moment altijd een wens op voor een bepaald blok. Er bestaan echter ook zelfroosteringssystemen waarbij een chauffeur naast een eerste keuze, ook een tweede of zelfs derde keuze kan opgeven voor elke dag. Dit systeem kan eventueel ook bij Connexxion worden toegepast, waarbij de verschillende keuzes verschillende strafkosten met zich mee brengen.

- Bij Connexxion heeft men aangegeven dat het wenselijk is dat de planners de mogelijkheid krijgen om zelf bepaalde toewijzingen te sturen. We kunnen in dit model eenvoudig een extra kostencomponent toevoegen die de gebruiker kan activeren of deactiveren, zodat bepaalde toewijzingen aantrekkelijk of minder aantrekkelijk worden.

- Voor Connexxion is het relatief duur om bepaalde chauffeurs veel overwerk te geven. Chauffeurs die relatief veel overwerk krijgen, zouden daarom meer strafkosten kunnen

krijgen voor blokken die veel overwerk bevatten. Op deze manier worden de opbrengsten van overwerk beter verdeeld over de chauffeurs. We passen deze strafkosten toe op het moment dat we reeds een oplossing hebben gegenereerd, zodat deze kosten bij een eventuele volgende iteratie effect hebben op de toewijzingen.

Door verschillende kostencomponenten samen te voegen, krijgen we een veel gevarieerdere graaf. Daardoor kunnen we flexibeler omgaan met de wensen van de Connexxion en verwachten we dat het masterprobleem makkelijker op te lossen is.

In de komende paragrafen beschrijven we het algoritme dat we gebruikt hebben om de SPPRC op te lossen.

5.4 Ondergrens van de kosten

We beginnen het algoritme door vanuit elke knoop de lengte van het kortste pad naar t te berekenen, waarbij we geen rekening houden met de resources. De afstanden gebruiken we in het SPPRC algoritme op twee manieren, die gebaseerd zijn op het A* algoritme (zie Rich en Knight (1990)).

1. De lengte van het kortste pad van s naar t is een ondergrens of lower bound (LB) op de kosten van het SPPRC. Deze ondergrens gebruiken we tijdens het SPPRC algoritme om te bepalen of een pad naar t optimaal is. De LB kan niet verbeterd worden, dus als we een pad naar t vinden dat gelijk is aan de LB, dan kunnen we het algoritme al voortijdig afbreken.

2. Als we een pad naar t vinden dat niet gelijk is aan de LB, dan kunnen we dit pad alsnog gebruiken om andere paden eerder af te kappen. Van de andere (deel)paden weten we de kosten en we weten de kortst mogelijke afstand naar t vanaf het uiteinde van het pad. Als we een pad hebben waarvan deze twee waarden tezamen groter zijn dan of gelijk zijn aan de kosten van het huidige beste pad naar t, dan kan dat pad nooit beter worden dan ons beste pad.

We bepalen de kortste paden met behulp van een variant van een 'pull' algoritme dat speciaal bedoeld is voor acyclische grafen (Ahuja e.a. (1993)). De rekentijd van dit algoritme is in orde $O(m)$, met m het aantal takken. Het algoritme berekent de kortste paden van s naar alle andere knopen. We willen echter de kortste paden hebben van alle knopen naar t. Als we alle takken van richting te verwisselen, kunnen we t als startknoop gebruiken om zo de kortste paden van alle knopen naar t vinden.

Het algoritme vereist verder dat de knopen topologisch geordend zijn. Dat betekent dat de knopen in een volgorde moeten staan waarbij we voor elke tak kunnen zeggen dat het rangnummer van de startknoop kleiner is dan het rangnummer van de eindknoop. Deze rangorde is makkelijk te vinden als elke tak loopt van een dag naar de dag ervoor (we hebben de takken van richting veranderd): we starten met de knopen van de laatste dag en lopen vervolgens van achter naar voren langs de dagen. Een tak waarvan de startknoop op dag $k+1$ ligt, heeft een rangnummer dat altijd lager is dan dat van de eindknoop op dag k. De knopen op dezelfde dag kunnen we in willekeurige volgorde doorlopen, omdat er nooit takken tussen deze knopen lopen.

Algoritme 5.1 toont hoe we dit hebben geïmplementeerd. Voor elke knoop slaan we de lengte van het kortste pad naar t op onder het veld 'Afstand'. De kortste afstand van t naar zichzelf is 0 en de afstand van alle andere knopen initiëren we op oneindig. Vervolgens lopen we van achteren naar voren alle dagen langs en bepalen we de kortste paden van alle knopen op die dag. Tenslotte bepalen we de lengte van het kortste pad van s naar t en deze slaan we op als LB.

Algoritme 5.1. CalcLB(G)

```
1: for all KNOOP ∈ KNOOPVERZAMELING do
2:    KNOOP.AFSTAND = ∞
3: end for
4: DESTINATION.AFSTAND = 0
5:
6: for DAG = EINDDAG downto STARTDAG do
7:    for all KNOOP ∈ DAG.KNOOPVERZAMELING do
8:       for all TAK ∈ KNOOP.TAKVERZAMELING do
9:          NAARKNOOP = TAK.NAARKNOOP
10:         TAKKOSTEN = Kosten(KNOOP, NAARKNOOP)
11:         PADKOSTEN = NAARKNOOP.AFSTAND + TAKKOSTEN
12:         if PADKOSTEN < KNOOP.AFSTAND then
13:            KNOOP.AFSTAND = PADKOSTEN
14:         end if
15:      end for
16:   end for
17: end for
18:
19: for all TAK ∈ SOURCE.TAKVERZAMELING do
20:   NAARKNOOP = TAK.NAARKNOOP
21:   TAKKOSTEN = Kosten(SOURCE, NAARKNOOP)
22:   PADKOSTEN = NAARKNOOP.AFSTAND + TAKKOSTEN
23:   if PADKOSTEN < SOURCE.AFSTAND then
24:      SOURCE.AFSTAND = PADKOSTEN
25:   end if
26: end for
27: LB = SOURCE.AFSTAND
```

5.5 Het SPPRC algoritme

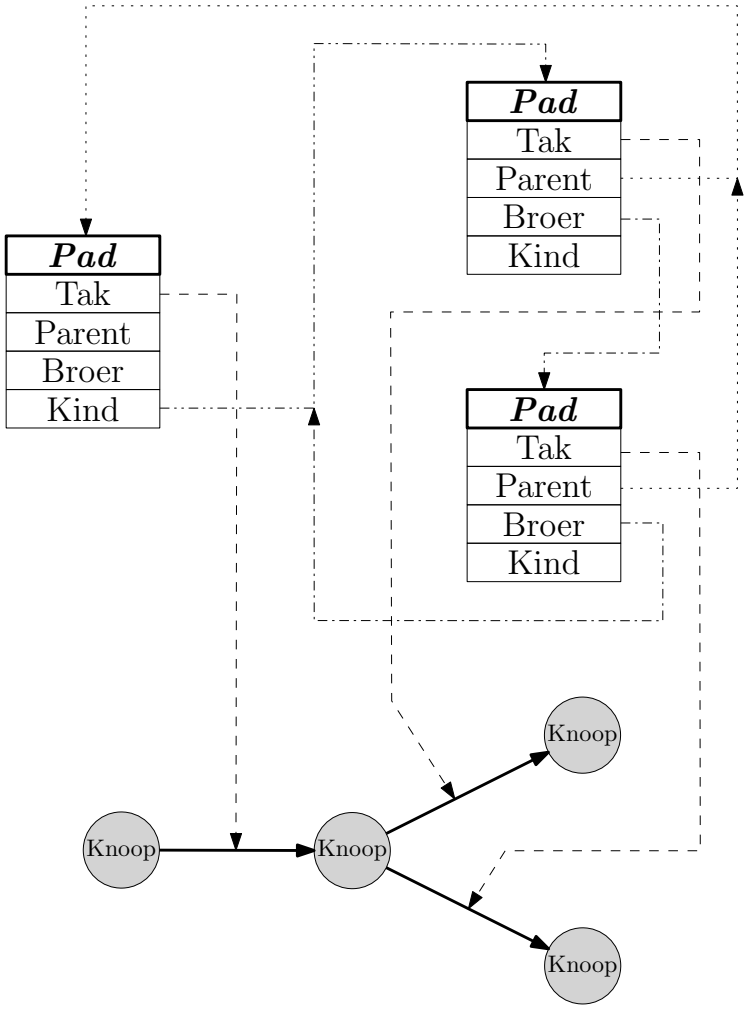
Gedurende het doorlopen van het SPPRC algoritme slaan we verschillende paden op. Elk pad heeft een aantal eigenschappen:

- De tak, die aangeeft waar het pad eindigt. Elk pad loopt vanaf s tot aan de NaarKnoop van de tak.
- Een ouder, de verwijzing naar de voorganger van het pad. Met behulp van de ouder kunnen we teruglopen langs alle takken van het pad tot aan s.

- De broers, paden met dezelfde ouder als het huidige pad. Elk pad heeft een verwijzing naar de voorgaande en naar de eerstvolgende broer. Op deze manier vormen de broers een zogenaamde 'Double Linked List'.

- De kinderen, paden die het huidige pad als ouder hebben. Elk pad heeft een verwijzing naar zijn eerst kind, waarna de overige kinderen bereikt kunnen worden door de lijst van broers van het kind te doorlopen.

In figuur 5.3 tonen we een schematische weergave van de paden, waaruit blijkt waar de verschillende eigenschappen naar toe verwijzen. We hebben een deel van de SPPRC graaf toegevoegd zodat de relatie tussen de takken en de paden duidelijk wordt. We hebben drie paden getoond, die ieder een unieke tak hebben. In de praktijk is het mogelijk dat we meerdere paden hebben voor dezelfde tak, als de paden via verschillende routes de huidige tak hebben bereikt.



Figuur 5.3: *Een schematische weergave van de datastructuur van een pad van het SPPRC algoritme. Elk pad heeft een verwijzing naar de betreffende tak (gestreepte lijn), naar zijn ouder (stippellijn), naar zijn eerstvolgende en vorige broer (streep-stippellijn) en naar zijn kinderen (streep-stip-stippellijn). Onderin staat een deel van de SPPRC graaf waaruit blijkt naar welke takken de paden verwijzen.*

Daarnaast gebruiken we twee verzamelingen om de paden op te slaan, een verzameling $\mathcal{U}$ voor de zogenaamde unprocessed paden en een verzameling $\mathcal{P}$ voor de processed paden. De unprocessed paden zijn paden waarvan we nog niet hebben onderzocht of we deze kunnen uitbreiden. Zij hebben nog geen kinderen, omdat we nog niet weten of het mogelijk is om een nieuwe toegelaten tak toe te voegen. De processed paden zijn paden waarvan we de uitgaande takken al beschouwd hebben en deze paden hebben minimaal één kind.[1] Elk pad is aanvankelijk unprocessed en gedurende het algoritme wordt elk pad processed.

Algoritme 5.2 beschrijft de oplossingsmethode voor het SPPRC (zie ook Irnich en Desaulniers (2005)). In de variabele BESTOPL slaan we de kosten van het beste pad naar t op dat tot nu toe gevonden is.

Algoritme 5.2. SPPRC(G, LB)

1: $\mathcal{P} \leftarrow \varnothing$
2: $\mathcal{U} \leftarrow \varnothing$
3: **for all** takken (s, b) **do**
4: $\quad \mathcal{U} \leftarrow \mathcal{U} \cup (s, b)$
5: **end for**
6: BESTOPL $\leftarrow \infty$
7: **while** $\mathcal{U} \neq \varnothing$ **do**
8: $\quad$ UnprocessedPadUitbreiden(G, LB)
9: **end while**
10: Sla beste pad op

We beginnen het algoritme met de unprocessed paden vanuit s (regel 3 en 4). Daarna gaan we de paden uit de verzameling $\mathcal{U}$ verwijderen. Tijdens het itereren voegen we ook weer paden toe aan $\mathcal{U}$ en daarom is het van tevoren niet bekend hoeveel iteraties nodig zijn. We hebben in algoritme 5.3 het deel van het algoritme beschreven dat in de while-lus staat.

De kosten van een pad of tak geven we aan met de letter C. Daarnaast gebruiken we het symbool $t(Q)$ om de eindknoop van het pad aan te geven (de NaarKnoop van de tak). Van deze knoop proberen we de uitgaande takken toe te voegen aan het pad om een nieuw pad te vormen. De letter v gebruiken we als hulpvariabele om nieuwe knopen aan te duiden die nog niet door het pad zijn bereikt.

[1] De processed paden zonder kinderen verwijderen we, omdat deze paden op geen enkele manier meer een toegelaten pad naar t kunnen worden.

Algoritme 5.3. UNPROCESSEDPADUITBREIDEN

1: KIES een pad $Q \in \mathcal{U}$
2: $\mathcal{U} \leftarrow \mathcal{U} \backslash \{Q\}$
3: $Kinderen_Q \leftarrow \varnothing$
4: **for** ALLE TAKKEN $(t(Q), v)$ **do**
5: **if** $v = t$ **then**
6: **if** $C_Q + C_{(t(Q),v)} <$ BESTOPL **then**
7: $Kinderen_Q \leftarrow Kinderen_Q \cup (Q, v)$
8: BESTOPL $\leftarrow C_Q + C_{(v(Q),v)}$
9: **if** BESTOPL = LB **then**
10: STOP
11: **end if**
12: **end if**
13: **else**
14: **if** $((Q, v)$ FEASIBLE$)$ **and** $(C_Q + C_{(t(Q),v)} + C^{MIN}_{(v,t)} <$ BESTOPL$)$ **then**
15: $\mathcal{U} \leftarrow \mathcal{U} \cup (Q, v)$
16: $Kinderen_Q \leftarrow Kinderen_Q \cup (Q, v)$
17: **end if**
18: **end if**
19: **end for**
20: **if** $Kinderen_Q \neq \varnothing$ **then**
21: $\mathcal{P} \leftarrow \mathcal{P} \cup Q$
22: Bepaal dominantie
23: **end if**

Elke iteratie bestaat uit het uitbreiden van één unprocessed pad. We controleren of we een toegelaten pad kunnen maken door het unprocessed pad uit te breiden met zijn uitgaande takken. Een pad is toegelaten als het voldoet aan de volgende eisen:

- Het pad dat we tot nu toe hebben gemaakt voldoet aan alle rules (zie bijlage A).
- Het pad kent geen blokken toe op dagen dat de chauffeurs verlof heeft opgenomen of een vaste vrije dag heeft gekregen (deze informatie komt uit het vakantieschema, tabel 5.1).
- Het pad voldoet aan eventuele randvoorwaarden die per chauffeur gelden (zie paragraaf 5.2).

Wanneer we nieuwe paden toevoegen aan $\mathcal{U}$, dan bepalen we de dominante paden (regel 22). Om de rekentijd binnen de perken te houden, bepalen we pas de dominantie als we alle uitgaande takken van een pad hebben beschouwd. We vergelijken de nieuwe paden met elkaar en met alle paden die al in $\mathcal{U}$ en $\mathcal{P}$ waren opgeslagen. Vervolgens verwijderen we alle gedomineerde paden. We bepalen de dominantie aan de hand van de kosten en de resources. Hieronder geven we een korte uitleg van de definities van kosten en resources in dit model.

Kosten De kosten zijn afhankelijk van het wensenpatroon van de chauffeur. Daarnaast worden de kosten beïnvloed door eventuele strafkosten die we op bepaalde toewijzingen

plaatsen. Paragraaf 5.3 gaat gedetailleerder in op de verschillende onderdelen waar de takkosten uit bestaan. We willen de kosten minimaliseren, daarom zijn lagere kosten dominant.

Nachtdiensten, zaterdagen, zondagen en gebroken diensten Er mogen niet teveel nachtdiensten, zaterdagdiensten, zondagdiensten en gebroken diensten toegekend worden in een bepaalde periode. We controleren altijd of een pad toegelaten is op grond van deze voorwaarden, maar daarnaast kunnen we al eerder met behulp van dominantie paden verwijderen die veel van dit soort diensten toekennen.

Vrije dagen Een chauffeur mag niet te veel en niet te weinig vrije dagen toegewezen krijgen. We mogen daarom niet twee paden vergelijken op grond van het aantal toegewezen vrije dagen. Wel weten we dat twee paden die evenveel vrije dagen toewijzen even goed zijn. We stellen daarom dat een pad alleen gedomineerd kan worden door paden die evenveel vrije dagen toekennen. Daarnaast behouden we alle paden die eindigen met een roostervrije dag, omdat deze paden van belang kunnen zijn voor de weekrust. Op deze manier verwijderen we geen paden die mogelijk tot de optimale oplossing kunnen leiden.

Een pad domineert een ander pad als de kosten kleiner of gelijk zijn en als het álle resources evenveel of minder gebruikt. Daarnaast moet minimaal één van de waarden (een resource of kosten) strict lager zijn. Paden die een even groot gebruik maken van de resources en dezelfde kosten hebben, worden dus behouden omdat deze paden mogelijk ook het beste pad naar t kunnen vormen.

Op het moment dat de verzameling $\mathcal{U}$ leeg is, stopt het algoritme. Als we geen pad hebben gevonden, dan weten we dat er geen toegelaten oplossing pad naar t bestaat. Als we wel een pad vinden, dan is het gevonden pad naar t optimaal. Dit pad representeert een toegelaten dienstrooster voor één chauffeur met minimale kosten.

5.6 Resultaten

Aanvankelijk hebben we het SPPRC algoritme toegepast om een dienstrooster voor de chauffeurs voor een heel jaar te bepalen. Al snel blijkt echter dat dit niet mogelijk is. Het algoritme heeft een lange tijd (in de orde van uren of misschien langer) nodig om alle mogelijke paden langs te gaan, ook al gebruiken we dominantie om het aantal paden te verminderen. In bijna alle gevallen is het algoritme daarom niet in staat om binnen afzienbare tijd een oplossing te vinden.

We hebben er daarom voor gekozen om een aantal aanpassingen door te voeren om het vinden van het optimale pad te versnellen.

1. We delen de planningsperiode op in kwartalen (13 weken). Bijna alle rules gelden voor een periode van maximaal 13 weken en daarom kost het relatief weinig moeite om deze overzetting te maken. We zullen proberen om het probleem voor meerdere perioden apart op te lossen om deze oplossingen daarna te combineren.

2. We passen het algoritme zo aan dat we gebruik maken van een combinatie van depth-first search met een methode waarbij de goedkoopste kant als eerste beschouwd wordt.

Op deze manier vinden we zo snel en goedkoop mogelijk een pad naar t. Dit is belangrijk, omdat we deze startoplossing kunnen gebruiken om een groot deel van de andere paden vooraf al af te wijzen.

Deze methode geeft in veel gevallen binnen een seconde een goed pad naar t. Het kost echter nog steeds veel tijd om alle andere paden af te gaan om te controleren of er niet betere paden gevonden kunnen worden. Ook voor 91 dagen blijkt het te lang te duren om alle paden af te gaan. Daarom laten we het algoritme maximaal 120.000 iteraties per chauffeur uitvoeren (een iteratie bestaat uit het beschouwen van de uitgaande takken van één unprocessed pad). Vaak blijkt dat het eerste gevonden pad niet meer verbeterd wordt in de overige 120.000 iteraties, en enkele tests wijzen uit dat dit pad vaak al dicht bij het optimum ligt.

In sommige extreme gevallen lukt het ondanks deze aanpassingen niet om binnen afzienbare tijd een pad naar t te kunnen vinden. Het betreft hier bijvoorbeeld chauffeurs die in de eerste weken veel vrije dagen aanvragen, waardoor ze aan het einde van de periode geen vrije dagen meer over hebben om weekend te krijgen. Het SPPRC algoritme zal in de beginperiode dan voldoen aan de wensen, maar zal pas aan het einde ontdekken dat de paden vervolgens niet meer toegelaten zijn. Uiteindelijk zal het algoritme wel een toegelaten pad vinden, maar daarvoor moet het algoritme zo veel dagen terug, dat het meer rekentijd kost dan we willen besteden aan het SPPRC algoritme.

Daarom hebben we nog een extra rule toegevoegd, namelijk de rule dat een (fulltime) chauffeur elke 14 dagen gemiddeld 9 werkdagen moet krijgen. Hier forceren we dat de chauffeurs altijd aan dit gemiddelde moeten voldoen, zodat de vrije dagen en werkdagen goed over de planningsperiode verdeeld worden. Dit zorgt ervoor dat het belangrijkste struikelblok voor het vinden van een toegelaten oplossing is verdwenen. Voor de overige rules kunnen we veel sneller ontdekken waar het fout gaat, zodat deze rules vaak geen probleem vormen.

De extra rule is vooral bedoeld als sturing en met dat doel voegen we deze dan ook toe als er na de vastgestelde 120.000 iteraties nog geen oplossing gevonden is voor de SPPRC. We hebben sinds deze aanpassing geen problemen meer gehad met het vinden van een pad naar t.

De aanpassingen leiden ertoe dat we niet meer kunnen spreken van een exacte oplossing, omdat de gevonden paden niet meer optimaal zijn. We hebben in dit geval echter gekozen voor snelheid boven optimaliteit, want de paden worden bij het uitrekenen van het masterprobleem vaak nog veranderd waardoor de optimaliteit niet behouden blijft.

In het volgende hoofdstuk bespreken we hoe we voldoen aan de vraag naar de blokken. We zullen daar ook uitgaan van een planningsperiode van 13 weken.

Hoofdstuk 6

Masterprobleem

In het vorige hoofdstuk hebben we laten zien hoe per chauffeur een toegelaten dienstrooster wordt opgesteld. Daarbij is echter geen rekening gehouden met voorwaarde (3.2), die eist dat we de vraag naar de blokken exact vervullen. Het is daarom mogelijk dat we sommige blokken te vaak hebben toegewezen en andere blokken te weinig. In dit hoofdstuk voegen we de verschillende deelroosters samen tot een eindoplossing. We onderzoeken of de vraag naar de verschillende blokken overeenkomt met de toewijzingen.

Binnen Connexxion wordt van tevoren bepaald hoeveel chauffeurs er nodig zijn bij een specifieke vestiging om te voldoen aan de vraag naar elk type blok. Deze data beschouwen we hier als gegeven. De totale vraag naar de blokken is echter niet gelijk aan het aantal chauffeurs. We beïnvloeden de vraag naar het reserve blok en naar het roostervrije blok kunstmatig om de vraag naar de blokken en het aanbod aan chauffeurs op elkaar aan te laten sluiten.

RES-blok Als een chauffeur verlof neemt, dan mogen we de chauffeur op die dagen geen blokken toekennen (zie ook paragraaf 5.2.1). De chauffeurs krijgen het RES-blok toegewezen op dagen dat ze verlof nemen. Dit is een dummy blok waar geen bestaande diensten aan zijn gekoppeld.

We weten van tevoren al hoeveel RES-blokken er op elke dag nodig zijn, omdat dit afhankelijk is van de data en niet van het algoritme. We zetten de vraag naar het RES-blok gelijk aan het aantal mensen dat op die dag verlof heeft genomen. Op deze manier zorgen we ervoor dat we altijd aan het juiste aantal chauffeurs het RES-blok toewijzen.

VRIJ-blok Elke dag krijgt een deel van de chauffeurs het VRIJ-blok toegewezen, het blok voor de roostervrije dag. De chauffeurs krijgen een VRIJ-blok als ze op die dag geen werk krijgen toegewezen. De vraag naar het VRIJ-blok is dus gelijk aan het totaal aantal chauffeurs minus de totale vraag naar de niet-vrije blokken.

Op deze manier balanceren we vraag en aanbod met elkaar. Het is daardoor mogelijk om exact aan de vraag van de blokken te voldoen zonder dat er blokken of chauffeurs overblijven.

Als de vraag naar het roostervrije blok voor een bepaalde dag negatief is, dan weten we dat er te weinig chauffeurs beschikbaar zijn om een toegelaten dienstrooster voor die dag op te stellen. Zo kunnen we al snel zien of er een tekort aan chauffeurs is op een bepaalde dag.

Voor elke dag en voor elk blok vergelijken we de vraag naar het blok met het aantal chauffeurs dat op die datum dat blok hebben gekregen. Het doel is om een toegelaten toewijzing te vinden waarbij de vraag naar elk blok precies overeenkomt met het aantal toewijzingen van dat blok.

We hebben verschillende methoden gebruikt om een toegelaten oplossing voor het masterprobleem te vinden. In de volgende paragrafen gaan we eerst dieper in op de verschillende methoden die we hebben toegepast. We hebben deze methoden later gecombineerd in een compleet algoritme, dat getoond wordt in figuur 6.2 en beschreven in paragraaf 6.6.

6.1 Lagrange Relaxatie

Het zal vaak voorkomen dat we een bepaalde knoop (een bepaald blok op een specifieke dag) te vaak toewijzen. We willen dat dit blok minder vaak wordt toegewezen dan nu het geval is. We kunnen het SPPRC algoritme sturen door extra strafkosten toe te kennen aan knopen die te vaak zijn toegewezen. Het algoritme zal dan mogelijk een andere knoop toewijzen zodat het overschot aan toewijzingen teniet wordt gedaan.

Lagrange Relaxatie is een bekende methode die dit gebruik van strafkosten als basis gebruikt voor het oplossen van allerlei modellen (zie bijvoorbeeld Ahuja e.a. (1993)). Lagrange Relaxatie heeft als voordeel dat we een ondergrens op onze kosten kunnen bepalen en daarmee kunnen we soms ook aantonen of een toegelaten oplossing optimaal is.

6.1.1 Lagrange Multipliers

We hebben daarom Lagrange Relaxatie toegepast, waarbij voorwaarde 3.2 gerelaxeerd is (de voorwaarde die stelt dat het aantal toewijzingen gelijk moet zijn aan de vraag). Een gerelaxeerde voorwaarde komt terug in de criteriumfunctie als extra kostenpost, waarbij de zogenaamde Lagrange Multipliers worden toegevoegd als schalingsparameter. In 6.1 tonen we de criteriumfunctie na toevoeging van de Lagrange Multipliers (gebaseerd op de notatie die we hebben gebruikt in hoofdstuk 3).

λ^i_{bd} = de Lagrange Multiplier behorende bij blok b op dag d tijdens iteratie i.

$$\min \sum_{(c,b)\in\mathcal{E}} \sum_{d\in\mathcal{D}} r^d_{cb} X^d_{cb} + \sum_{b\in\mathcal{B}} \sum_{d\in\mathcal{D}} \lambda^i_{bd} \left(\sum_{c:(c,b)\in\mathcal{E}^d} X^d_{cb} - A^d_b \right) \tag{6.1}$$

De Lagrange Multipliers zijn aanvankelijk gelijk aan 0, zodat we in de eerste iteratie alleen de overige kosten meenemen. Vervolgens gebruiken we de resultaten om de Lagrange Multipliers voor de volgende iteratie te berekenen. Er is veel onderzoek gedaan naar het vinden van goede methodes om de Lagrange Multipliers aan te passen. Bewezen is dat de Lagrange Multipliers aan twee eisen dienen te voldoen (Ahuja e.a. (1993)):

$$\lim_{i \to \infty} \lambda_{bd}^{i} = 0 \tag{6.2}$$

$$\lim_{i \to \infty} \sum_{j=1}^{i} \lambda_{bd}^{j} = \infty \tag{6.3}$$

In de literatuur zijn verscheidene methoden populair om de Lagrange Multipliers aan te passen (zie bijvoorbeeld Volgenant en Jonker (1982)), maar we beginnen hier met een eenvoudige methode gebaseerd op het voorbeeld dat gebruikt wordt in Ahuja e.a. (1993). Later kunnen we deze methode nog verfijnen als dit nodig is.

$$\lambda_{bd}^{i} = \min \left\{ \frac{4}{i+1}, \frac{4}{11} \right\} \tag{6.4}$$

6.1.2 Resultaten van de Lagrange Relaxatie

Lagrange Relaxatie garandeert niet dat we een toegelaten oplossing vinden. Een risico van deze methode (of van andere methoden waar we met strafkosten werken) is dat het algoritme te vaak een andere toewijzing kiest, terwijl we willen dat slechts een deel van de chauffeurs een andere toewijzing krijgen.

We hebben de Lagrange Relaxatie geïmplementeerd en bij het testen blijkt al snel dat er weinig verbetering wordt geboekt in het aantal knopen waarvoor vraag en toewijzingen nog niet overeen komen. Na het aanpassen van de Lagrange Multipliers verandert het aantal toewijzingen voor een groot deel van de knopenverzameling. In ongeveer driekwart van de gevallen heeft dit tot gevolg dat het aantal toewijzingen te ver doorschiet, van een overschot aan toewijzingen naar een tekort of andersom.

Dit wordt voornamelijk veroorzaakt doordat de kosten (nog) niet gevarieerd genoeg zijn voor de verschillende chauffeurs. Het algoritme wijst steeds hetzelfde blok toe aan de chauffeurs totdat de Lagrange Multipliers een bepaalde drempelwaarde (threshold) overschrijden waarna alle toewijzingen ineens veranderen.

Het algoritme convergeert daardoor niet naar een toegelaten oplossing. In de volgende paragrafen beschrijven we daarom alternatieve methodes om te proberen om een toegelaten oplossing te vinden.

6.2 Swap-Algoritme

Op de meeste dagen zal de vraag naar blokken nog niet overeenkomen met het aantal toegewezen blokken van elk type. Aanvankelijk hebben we daarom gekozen voor een Swap-Algoritme dat probeert om de overbodige blokken te verwisselen voor blokken waarvan een tekort is. Als we bijvoorbeeld te vaak een blok A hebben toegewezen, maar te weinig een blok B, dan zullen we proberen om de chauffeurs met een blok A te ruilen naar een blok B. Als dit mogelijk is (gegeven hun rooster op de overige dagen) dan passen we deze verwisseling toe. Op deze manier proberen we de vraag zoveel mogelijk overeen te laten komen met het aantal benodigde blokken op elke dag.

Het Swap-Algoritme wordt per dag apart opgelost, omdat we per dag aan de vraag naar de blokken willen voldoen. In de eerste iteratie doorlopen we de dagen simpelweg van begin tot eind, maar vanaf de tweede iteratie starten we met de dagen die we de vorige iteratie niet konden matchen. Deze dagen waren blijkbaar moeilijk te matchen, daarom geven we deze prioriteit.

De wensvervulling zal door deze methode minder kunnen worden, maar we passen dit algoritme alleen toe op de chauffeurs die een blok hebben gekregen waar een overschot van is. Het grootste gedeelte van de toewijzingen zal daarom behouden blijven en daarmee blijft ook het matchingspercentage redelijk intact. Bij een test bleek dat het matchingspercentage na een aanroep van het Swap-Algoritme met 15% gedaald was. Deze daling is ten opzichte van een dienstrooster waarbij iedereen een zo hoog mogelijk matchingspercentage heeft (na de eerste keer de SPPRC te hebben opgelost voor elke chauffeur).

Het aantal dagen waarop geen matching gevonden kan worden, wordt met deze methode al na één iteratie teruggebracht tot ongeveer 10%. Deze 10% blijft echter wel redelijk constant over de iteraties heen. Een oorzaak hiervan ligt in het feit dat het in veel gevallen wel mogelijk is om een chauffeur een roostervrije dag toe te kennen, maar het weghalen van iemands roostervrije dag is vaak niet toegestaan. We hebben daarom enkele andere functies gemaakt om het vinden van een toegelaten oplossing makkelijker te maken.

6.3 Max-Flow algoritme

We willen zo snel mogelijk een matching vinden voor een bepaalde dag die voldoet aan de vraag naar de blokken. In de hoofdstukken 3 en 5 hebben we al beschreven dat we het hoofdprobleem kunnen beschouwen als een reeks assignment problemen per dag. Vanuit datzelfde oogpunt gaan we nu proberen om een volledige matching per dag op te stellen, door voor elke dag een Max-Flow probleem op te lossen.

We gaan uit van een bipartiete graaf met aan de ene kant een knoop voor elke chauffeur en aan de andere kant een knoop voor elk blok dat we die dag moeten toekennen. Verder maken we een startknoop en een eindknoop en het doel is om de maximale flow te bepalen van de start- naar de eindknoop (in figuur 6.1 staat een voorbeeld van de Max-Flow graaf).

We hebben drie verschillende soorten takken, die we nu even kort bespreken.

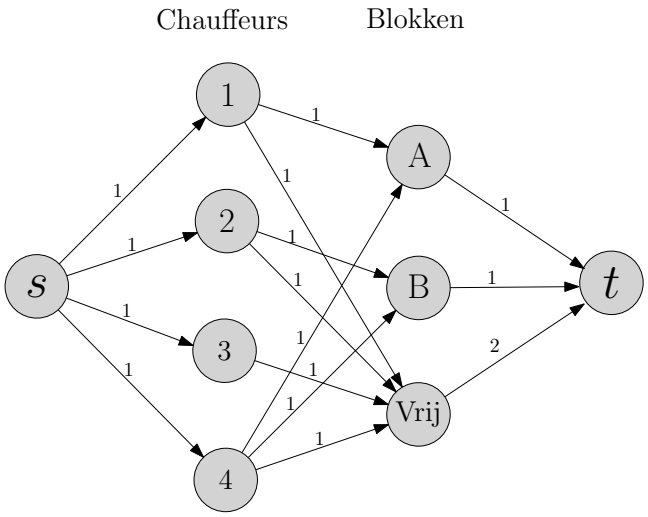
Start - chauffeurs Vanuit de startknoop gaat een tak naar elke chauffeur. Elk van deze takken heeft een capaciteit van 1, omdat de chauffeurs slechts één blok toegewezen kunnen krijgen.

Chauffeurs - blokken We maken een tak aan tussen een chauffeur en een blok als het mogelijk is om dit blok toe te wijzen aan de chauffeur op de dag waarvoor we het Max-Flow probleem oplossen. We beschouwen de toewijzingen op de overige dagen als gegeven, waardoor een deel van de toewijzingen onmogelijk zal blijken. De takken van de chauffeurs naar de blokken mogen elke willekeurige capaciteit hebben, zolang deze groter is dan 0. We hebben deze takken de capaciteit 1 gegeven.

Blokken - eind Van elke blokknoop gaat een tak naar de eindknoop. De capaciteit op deze takken is gelijk aan de vraag naar het betreffende blok op de dag. Op deze manier kunnen de blokken precies zo vaak toegewezen worden als nodig is.

We hebben een matching voor een bepaalde dag gevonden als we een Max-Flow vinden die gelijk is aan het aantal chauffeurs, want dit is alleen mogelijk als we alle chauffeurs toewijzen. De graaf is incompleet, het is dus van tevoren niet bekend of deze matching gevonden kan worden.

In figuur 6.1 staat een voorbeeld van hoe deze graaf eruit zou kunnen zien. We gebruiken drie blokken: blok A met een vraag van 1, blok B met een vraag van 1 en blok VRIJ met een vraag van 2. Er zijn vier chauffeurs, die allemaal andere mogelijk toewijzingen hebben. Chauffeur 1 kan alleen blok A en VRIJ toegewezen krijgen, chauffeur 2 alleen blok B en VRIJ, chauffeur 3 alleen blok VRIJ en chauffeur 4 kan alle blokken toegewezen krijgen. Het is makkelijk te zien dat het in deze graaf mogelijk is om een complete matching te maken (bijvoorbeeld de matching 1-A, 2-B, 3-VRIJ en 4-VRIJ). De maximale flow zal in dit geval dan ook gelijk zijn aan 4.



Figuur 6.1: *Een voorbeeld van de graaf die we opstellen voor het maken van een matching voor een specifieke dag, waarbij we een Max-Flow probleem oplossen. In dit geval zijn er vier chauffeurs en drie blokken (A, B en VRIJ). Bij elke tak vermelden we de capaciteit.*

Voor het Max-Flow probleem zijn verschillende algoritmes bekend die dit probleem in polynomiale tijd kunnen oplossen (zie Ahuja e.a. (1993)). We hebben gekozen voor een implementatie van de Shortest Augmenting Path variant van het Ford-Fulkerson algoritme. Deze methode vindt in $O(nm^2)$ tijd een optimale oplossing voor het Max-Flow probleem[1]. Een dag heeft ongeveer vijf tot acht mogelijke blokken en het aantal chauffeurs kan per vestiging variëren van ongeveer 50 tot 200. Het aantal takken is relatief laag, omdat er alleen takken van de chauffeurs naar (een deel van) de blokken gaan en niet tussen de chauffeurs onderling.

Deze methode blijkt in de praktijk erg goed te werken om snel een toegelaten matching te vinden per dag. Het nadeel van het Max-Flow algoritme is dat er geen mogelijkheid is om naar de wensen van de chauffeurs te kijken. Een aanroep van dit algoritme op een oplossing met maximaal matchingspercentage veroorzaakt een daling van 43% van het matchingspercentage. We gebruiken deze methode dan ook vooral om moeilijke dagen te matchen, meestal nadat we eerst het Swap-Algoritme hebben geprobeerd. Net zoals bij het Swap-Algoritme proberen we eerst de dagen te matchen die in de vorige iteratie niet gematcht konden worden.

[1] Met n het aantal knopen in de graaf en m het aantal kanten

6.4 Vrije dagen verschuiven

Ondanks de bovengenoemde heuristische methodes lukt het nog steeds niet altijd om een toegelaten oplossing te vinden. Dit komt vaak voor op het moment dat we te veel roostervrije dagen hebben toegewezen, omdat het moeilijk is om een roostervrije dag uit een dienstrooster weg te halen (vanwege de rules). Daarom hebben we een extra functie geïmplementeerd die het mogelijk maakt om de vrije dagen van een chauffeur te verschuiven. Als een chauffeur bijvoorbeeld eerst vrij had op zaterdag en zondag, dan kunnen we proberen om hem in plaats daarvan vrij te geven op donderdag en vrijdag. Op deze manier wordt wel voldaan aan de eis om de chauffeur weekend te geven en blijft het aantal toegewezen vrije dagen gelijk. Tegelijkertijd geeft het ons de mogelijkheid om op de zaterdag en de zondag andere blokken toe te kennen.

6.4.1 Enkele dag en weekend

Bij het verschuiven proberen we eerst om slechts één vrije dag te verschuiven. Als dit onmogelijk blijkt te zijn (bijvoorbeeld als de vrije dag onderdeel is van het weekend van de chauffeur), dan zullen we pogen om het gehele weekend te verschuiven.

Het is meestal niet mogelijk om de weekenden heel ver naar voren of naar achteren te verschuiven, omdat het weekend gebonden is aan de betreffende week. De weekenden verschuiven we daarom maximaal een week naar voren of naar achteren. Op deze manier houden we de rekentijd van deze functie beperkt. Uit tests is ook gebleken dat verder naar voren of naar achteren schuiven bijna nooit mogelijk is. De losse vrije dagen zijn vaak geen onderdeel van het weekend, daarom proberen we deze wel over de hele planningsperiode te verschuiven.

Deze functie helpt bij het snel vinden van een toegelaten oplossing, maar in het slechtste geval vermindert hij het matchingspercentage met vier vervulde wensen (de twee dagen waarop het weekend lag en de twee dagen waar we het naar toe verplaatsen). We gebruiken deze functie daarom alleen als zowel het Swap-Algoritme als het Max-Flow algoritme geen matching hebben kunnen maken op de dag.

6.4.2 Tabu Search

Bij het verschuiven van de vrije dagen bestaat het risico dat de vrije dagen 'heen en weer' worden geschoven. In dat geval alterneert het programma steeds tussen dezelfde oplossingen. Om dit tegen te gaan maken we gebruik van een Tabu Search methode (zie Glover (1989) en Glover (1990)).

Bij Tabu Search wordt gebruik gemaakt van een Tabu List (Taboe Lijst). De dagen die op de Tabu List staan mogen niet gebruikt worden om vrije dagen naar toe te schuiven (die dagen zijn 'taboe'). Elke keer dat we van een bepaalde dag de roostervrije dagen proberen weg te schuiven, zullen we deze dag toevoegen aan de Tabu List, zodat deze dag niet opnieuw te veel vrije dagen krijgt.

We hebben ervoor gekozen om de Tabu List een maximale lengte van tien te geven, zodat we nooit zo veel dagen uitsluiten dat het verschuiven onmogelijk is. Als we een element toevoegen aan een volle Tabu List, dan verwijderen we het oudste element van de lijst en voegen we het nieuwe element toe.

6.5 Overige functies

In deze paragraaf beschrijven we nog een aantal hulpfuncties die dienen om de hierboven besproken methoden te ondersteunen.

6.5.1 Strafkosten

In paragraaf 6.1.2 hebben we uitgelegd dat de resultaten van de Lagrange Relaxatie niet goed waren. We zijn daarom afgestapt van het gebruik van de Lagrange Multipliers om een oplossing te vinden. Wel gebruiken we nog strafkosten aan het einde van elke iteratie om bepaalde toewijzingen aantrekkelijk of minder aantrekkelijk te maken voor de volgende iteratie. Dit lijkt in de basis veel op de Lagrange Relaxatie, maar vanwege de overige heuristische methoden kunnen we nu niet meer van bepaalde oplossingen zeggen of deze optimaal zijn of niet. Daarnaast convergeren de strafkosten niet naar 0, maar blijven ze constant over de iteraties heen.

6.5.2 Toewijzingen vastzetten

De algoritmes in de vorige paragrafen worden allemaal gebruikt om op de een of andere manier de toewijzing van de chauffeurs te wijzigen. In de volgende iteratie van de SPPRC zal echter in veel gevallen opnieuw de wens van de chauffeur toegewezen worden. In sommige situaties zetten we daarom een bepaalde toewijzing vast, zodat we zeker weten dat de chauffeur die toewijzing ook krijgt tijdens de volgende iteratie van de SPPRC. Alleen als een vastgezette toewijzing leidt tot een niet toegelaten dienstrooster, dan zullen we deze toewijzing weer loslaten.

6.5.3 Matchingspercentage verbeteren

Zowel bij het Swap-Algoritme als bij de Max-Flow houden we geen rekening met het matchingspercentage en daarom is het mogelijk dat het matchingspercentage niet meer voldoet aan de eis van 80%.

We zoeken eerst een toegelaten oplossing voordat we streven naar een oplossing met een hoog matchingspercentage, omdat het vinden van een toegelaten oplossing al erg veel tijd kost. Als we een toegelaten oplossing hebben gevonden, dan zullen we proberen om het matchingspercentage te verbeteren door de toewijzingen van de chauffeurs onderling te wisselen. Onderlinge verwisselingen hebben tot gevolg dat de oplossing nog steeds voldoet aan de vraag, maar het is mogelijk dat de individuele dienstroosters niet meer toegelaten zijn met betrekking tot de rules. We verwisselen de toewijzingen van twee chauffeurs daarom alleen als wordt voldaan aan de volgende twee eisen:

1. Het totale matchingspercentage wordt niet verslechterd door de verwisseling. Dat betekent dat we de volgende mogelijke verwisselingen toestaan:
    - Het matchingspercentage van één of beide chauffeurs verbetert, zonder dat het matchingspercentage van iemand verslechtert. Dit is de meest ideale situatie.
    - Het matchingspercentage voor beide chauffeurs blijft ongewijzigd, omdat ze beiden niet hun wens toegewezen hadden gekregen op die dag. Deze situatie staan

we toe omdat het misschien een andere verwisseling mogelijk maakt, die wel tot verbeteringen leidt.

2. Voor beide chauffeurs is het mogelijk om het blok van de andere chauffeur toe te wijzen, zonder de toelaatbaarheid van de oplossing te schenden. De toewijzing moet daarom aan alle rules voldoen.

We proberen deze verwisselingen toe te passen voor alle mogelijke combinaties van twee chauffeurs en voor alle dagen. Het matchingspercentage kan op deze manier met een kleine 10% verbeterd worden in enkele seconden.

6.6 Algoritmes combineren

We hebben in de vorige paragrafen verschillende algoritmes besproken om tot een toegelaten oplossing te komen. In deze paragraaf leggen we uit hoe we deze algoritmes hebben gecombineerd om uiteindelijk (heuristisch) tot een oplossing te komen.

6.6.1 Overzicht van de gebruikte methodieken

We hebben de verschillende functies samengevoegd in een schema waarbij we proberen om een goede balans te vinden tussen het zo snel mogelijk vinden van een toegelaten oplossing en het vinden van oplossing met een zo hoog mogelijk matchingspercentage.

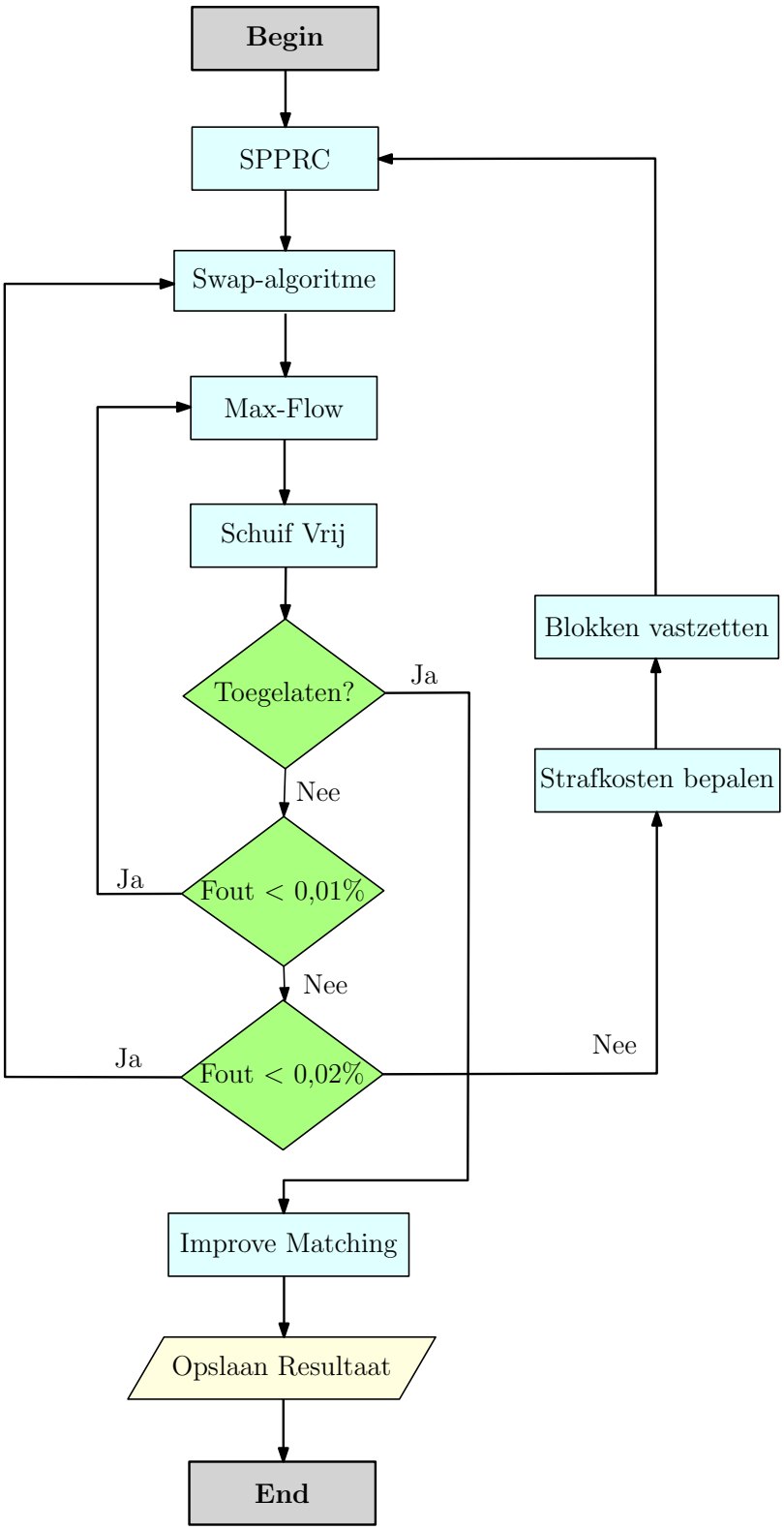
In figuur 6.2 staat een flow chart van de gebruikte heuristieken. We beginnen met het berekenen van de SPPRC en daarna zullen we achtereenvolgens het Swap-Algoritme en de Max-Flow toepassen om een toegelaten oplossing te vinden. Als dit nog niet voldoende is, dan zullen we proberen om de vrije dagen te verschuiven van de dagen waar nodig is.

Vervolgens hebben we een aantal loops die proberen om een toegelaten oplossing te forceren. Als we nog geen oplossing hebben gevonden, maar wel voor minder dan respectievelijk 1% of 2% van de knopen een verkeerd aantal toewijzingen hebben gevonden, dan zullen we de heuristische functies blijven aanroepen totdat we een toegelaten oplossing hebben gevonden.

Het algoritme zoals het in de flow chart beschreven wordt, kan leiden tot 'infinite loops'. Dit komt doordat we een deel van de informatie hebben weggelaten om het overzicht te behouden. Als de fout kleiner is dan 1% of 2% maar het lukt niet om een toegelaten oplossing te vinden, dan zal het algoritme vast blijven zitten in de loops. Daarom stoppen we ook als we na een iteratie van de loop geen verandering bemerken in het aantal foute knopen.

Als de oplossing na al deze heuristieken nog niet toegelaten is, dan zullen we een nieuwe oplossing opstellen van de SPPRC. De SPPRC zal echter opnieuw de startoplossing genereren. Daarom passen we de strafkosten aan en zetten we de toewijzingen vast voor de knopen die we na alle heuristische methoden nog steeds niet goed konden maken. Na de volgende iteratie van de SPPRC zullen dan vooral het aantal toewijzingen voor de moeilijke knopen gewijzigd zijn.

Uiteindelijk gebruiken we het schema in figuur 6.2 om een toegelaten oplossing te vinden voor het masterprobleem. In het volgende hoofdstuk zullen we de resultaten bespreken die we hebben behaald bij het toepassen van dit schema op verschillende situaties.



Figuur 6.2: *Een flow chart van de manier waarop de algoritmes gebruikt worden in combinatie met elkaar. Eventuele 'infinite loops' worden afgevangen, maar zijn niet in het schema verwerkt om het overzicht te behouden.*

Hoofdstuk 7

Resultaten

In de vorige hoofdstukken hebben we besproken hoe we een complete matching gaan maken voor de buschauffeurs van een willekeurige vestiging van Connexxion. In dit hoofdstuk onderzoeken we hoe goed ons model presteert als we het toepassen op een praktijksituatie. We gaan een matching maken voor de vestiging Dokkum voor het jaar 2010. Het planningsjaar 2010 loopt van zondag 27-12-2009 tot en met zaterdag 25-12-2010 (364 dagen).

De vestigingen zijn niet allemaal exact identiek. In Dokkum worden bijvoorbeeld nooit G, L of P diensten verreden. Dit vertaalt zich naar een vraag van 0 naar deze blokken. Deze onderlinge verschillen kunnen in zijn geheel worden afgevangen in de data, het is dus niet nodig om per vestiging aanpassingen in het model te maken.

De vestiging Dokkum heeft in totaal 68 chauffeurs. Elke chauffeur heeft een eigen contract waaruit blijkt hoeveel dagen in de week hij werkt en op welke dagen hij vrij heeft. Daarnaast heeft elke chauffeur een vakantieschema waaruit blijkt wanneer hij verlof heeft (zie tabel 5.1). We gebruiken deze informatie om een schema per chauffeur op te stellen waaruit blijkt wat hun vooraf bepaalde toewijzingen zijn.

We gebruiken de werkelijke wensen die de chauffeurs hebben opgegeven voor 2010. Dat betekent ook dat de wensen niet noodzakelijkerwijs volledig of correct zijn. We geven hieronder enkele voorbeelden van onregelmatigheden in de data.

- De chauffeurs vullen niet voor elke dag een wens in. In de praktijk blijkt dat een groot deel van de chauffeurs op één of meerdere dagen geen wens hebben ingevuld, maar uiteindelijk worden de meeste dagen wel ingevuld: ongeveer 90% van de chauffeurs vullen minimaal 90% van de dagen in. De overige chauffeurs vullen alleen voor specifieke dagen een wens in of ze hebben helemaal geen wensen ingevuld (bij de vestiging Dokkum hebben 2 chauffeurs geen wensen ingevuld).

- Chauffeurs vullen regelmatig wensen in op de dagen dat ze vakantie hebben genomen, waarschijnlijk omdat de vakantiedagen nog niet bekend waren op het moment van invullen of omdat de chauffeur er niet bij stil stond dat het hier een vakantiedag betreft. In het rooster worden deze dagen automatisch ingevuld als reservedienst (het RES-blok), terwijl de chauffeurs bijvoorbeeld een blok A wensen. Het is onmogelijk om op deze dag te voldoen aan de wens van de chauffeur, daarom hebben we de wensen op deze dagen genegeerd.

- In sommige gevallen voldoen de wensen niet aan de rules. Het is bijvoorbeeld mogelijk dat een chauffeur alleen maar nachtdiensten of gebroken diensten aanvraagt, terwijl er door de rules een duidelijk maximum is gesteld aan dit type diensten.

Het blijkt in een aanzienlijk deel van de gevallen onmogelijk om aan alle wensen van de chauffeurs te voldoen. We hebben daarom van tevoren voor elke chauffeur berekend wat het maximaal haalbare matchingspercentage is. Het gemiddelde van dit maximum over alle chauffeurs is gelijk aan 97%. Dat betekent dat we een bovengrens van 97% hebben voor het matchingspercentage van onze oplossing.

In de volgende paragraaf beschrijven we hoe we een matching voor 91 dagen hebben bepaald. Daarna zullen we deze oplossing uitbreiden voor een heel jaar.

7.1 Matching voor 91 dagen

We lossen eerst het deelprobleem (de SPPRC) op voor elke chauffeur, waarbij we steeds maximaal 120.000 iteraties uitvoeren per chauffeur. Voor een klein deel van de chauffeurs is het algoritme al eerder klaar met alle mogelijke paden langs gaan (bijvoorbeeld als de ondergrens op de kosten gehaald wordt), maar in het merendeel van de gevallen zal het programma na 120.000 iteraties de beste oplossing tot op dat moment teruggeven. De rekentijd van 120.000 iteraties is ongeveer 2 seconden.[1] Het oplossen van het deelprobleem voor alle 68 chauffeurs kost dan tussen de twee en drie minuten.

Vervolgens roepen we de verschillende functies van het masterprobleem aan. Het eenmaal doorlopen van al deze functies kost in de orde van een seconde aan rekentijd, maar uit figuur 6.2 blijkt al dat er verschillende mogelijkheden zijn waarop de functies meerdere malen worden aangeroepen. Toch kost het oplossen van het masterprobleem in ons voorbeeld nooit meer dan 20 seconden. Het oplossen van het deelprobleem neemt dus het grootste deel van de rekentijd in beslag.

7.1.1 Matching per kwartaal

We willen een oplossing vinden voor de gehele planningsperiode van een jaar. We moeten bij het bepalen van de oplossing rekening houden met de gevonden oplossingen voor de vorige periode. Aanvankelijk beschouwen we de kwartalen echter onafhankelijk van elkaar, zodat we kunnen onderzoeken wat de prestaties zijn van ons model bij de verschillende kwartalen. We zijn daarbij vooral geïnteresseerd in de prestaties gedurende de vakantieperiode (voornamelijk het derde kwartaal).

Bij het oplossen ontdekten we dat het voor het tweede kwartaal niet mogelijk is om een oplossing te vinden. De (fulltime) chauffeurs mogen maximaal 61 werkdagen per kwartaal krijgen, maar er blijken meer werkdagen per chauffeur nodig te zijn. Bij Connexxion bleek dit probleem bekend: de vestiging Dokkum had inderdaad een tekort van ongeveer drie chauffeurs op het moment van matchen. Dit werd opgelost door drie dummy chauffeurs (zonder wensen) toe te voegen voor het tweede kwartaal. Zij worden meegenomen in de matching zodat er extra dienstroosters worden gemaakt. Als een nieuwe chauffeur wordt aangenomen, kan deze

[1] Alle berekeningen zijn uitgevoerd op een computer met een Intel Pentium 4 processor met 3,2 Ghz en een intern geheugen van 3 GB

direct de plek van een dummy chauffeur innemen. We hebben deze methode overgenomen om tot een oplossing te kunnen komen voor het tweede kwartaal.

In tabel 7.1 geven we een vergelijking van de resultaten voor vier kwartalen. Het aantal benodigde iteraties varieert van 1 tot 4 en de rekentijd varieert evenredig van 152 seconden (ruim 2 minuten) tot 640 seconden (ruim 10 minuten). Meestal lukt het al snel om het aantal foute knopen kleiner dan 10 te maken, maar blijkt het meerdere iteraties te kosten om één of twee moeilijke dagen te matchen. Het gemiddelde matchingspercentage schommelt tussen 80% en 86%. Voor alle kwartalen halen we dus de gewenste waarde van 80%.

In totaal heeft het programma 8 iteraties gedraaid met een totale rekentijd van 1631 seconden (ruim 27 minuten). Het gemiddelde matchingspercentage over alle kwartalen is 83%.

Kwartaal	**Aant. Iter.**	**Rekentijd**	**Gem. Match. %**	**Opmerkingen**
1	1	152	83%	
2	1	239	86%	3 dummy chauffeurs
3	2	600	85%	
4	4	640	80%	
Totaal	8	1631	83%	

Tabel 7.1: *Een vergelijking van de oplossingen die het programma heeft gevonden voor alle vier kwartalen. We tonen het aantal iteraties van het hoofdprobleem tot aan het vinden van een oplossing, de rekentijd in seconden en het gemiddelde matchingspercentage over alle chauffeurs.*

We hebben daarnaast voor alle kwartalen een boxplot gemaakt van de gerealiseerde matchingspercentages. Elke boxplot bevat het minimum, het eerste kwartiel, de mediaan, het derde kwartiel en de maximumwaarde van het matchingspercentage. De boxplots zijn te vinden in figuur 7.1.

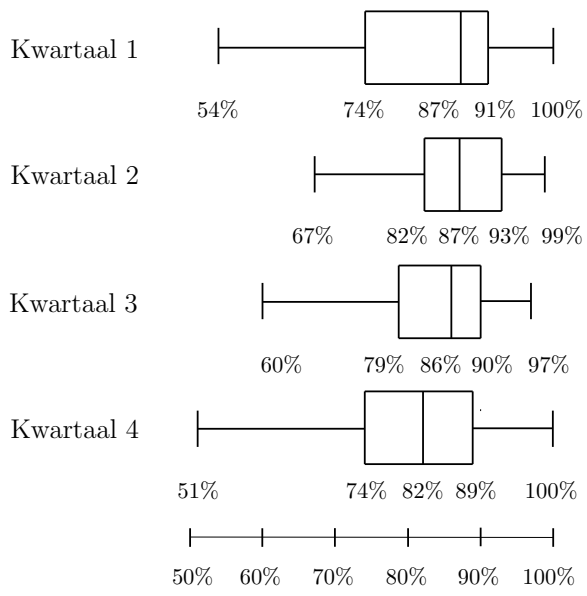
Uit de boxplots blijkt dat kwartaal 2 en 3 niet alleen het hoogste gemiddelde matchingspercentage hebben, maar dat de matchingspercentages ook veel dichter bij elkaar liggen. De minimumwaarde is ongeveer 10% groter dan bij de overige kwartalen en de verschillen tussen de mediaan en de kwartielwaarden zijn ook kleiner (de 'box' is smaller).

Het matchingspercentage kan nog minder worden als we de afhankelijkheid tussen de kwartalen meenemen bij het bepalen van de oplossing. In de volgende paragraaf zullen we deze afhankelijkheid terugbrengen in ons model.

7.2 Matching voor 364 dagen

Tot nu toe hebben we voor elk kwartaal apart een oplossing gegenereerd. We hebben er daarbij geen rekening mee gehouden dat sommige rules ervoor kunnen zorgen dat de oplossing voor een heel jaar niet meer toelaatbaar is. In deze paragraaf onderzoeken we hoe we een matching kunnen maken voor een langere periode waarbij we wel rekening houden met alle rules.

Als we een kwartaal hebben gematcht, dan zetten we deze oplossing vast en gaan we door naar het volgende kwartaal. We lossen nu eerst het SPPRC algoritme op voor het nieuwe kwartaal, waarbij we ook controleren of we voldoen aan alle rules met betrekking tot het vorige kwartaal. Dit heeft tot gevolg dat we meer takken zullen afwijzen, omdat deze leiden



Figuur 7.1: *Boxplots van de gerealiseerde matchingspercentages bij elk kwartaal, waarbij de kwartalen onafhankelijk van elkaar opgelost zijn. Van links naar rechts worden de minimumwaarde van het matchingspercentage getoond, het eerste kwartiel, de mediaan, het derde kwartiel en het maximum.*

tot onmogelijke paden.

In het eerste kwartaal zouden we eigenlijk de oplossing van 2009 moeten gebruiken als start-oplossing, omdat deze matching ook invloed heeft op de mogelijke dienstroosters in 2010. We hebben echter geen beschikking over deze data. Bij het oplossen van het eerste kwartaal hebben we daarom meer vrijheid dan bij de andere kwartalen.

Als we de oplossing van het vorige kwartaal als gegeven beschouwen, dan kan het voorkomen dat het onmogelijk is om een oplossing te vinden voor een nieuw kwartaal zonder de oplossing van het vorige kwartaal aan te passen. Dit wordt voornamelijk veroorzaakt door de rule voor de weekrust.

7.2.1 Weekrust

Elke week moet een chauffeur weekrust krijgen, waarbij de week loopt van maandag tot en met zondag. We controleren op elke zondag of we aan de weekrust hebben voldaan voor de afgelopen week, omdat we dan pas met zekerheid kunnen zeggen of de weekrust wel of niet gegeven is. Onze planningsperiode begint echter op een zondag en eindigt op een zaterdag.

Op dit moment komt het dan ook regelmatig voor dat de chauffeurs aan het einde van het kwartaal nog geen weekrust toegewezen hebben gekregen voor de laatste week. De zondag van die week valt immers in het volgende kwartaal. Bij het oplossen van het volgende kwartaal blijkt dan dat het vaak onmogelijk is om alsnog de weekrust toe te kennen en tegelijkertijd aan de vraag naar de blokken te voldoen.

Een oplossing voor dit probleem is om niet te beginnen op een zondag, maar op een maandag. Op die manier plannen we altijd volledige weken die al een weekrust toekennen. We hebben de planningsperiode daarom één dag opgeschoven, zodat we starten op maandag 28-12-2009.

De eerste dag (27-12-2009) nemen we niet mee in de matching voor een heel jaar. Uiteindelijk willen we de matching van 2009 meenemen bij het oplossen van de matching. We hebben besloten om de eerste dag weg te laten uit de matching zolang we geen beschikking hebben over deze data.

We hebben ons model nu een oplossing laten genereren voor de periode van 28-12-2009 tot en met 25-12-2010 (363 dagen). Het verschuiven van de planningsperiode blijkt echter niet voldoende te zijn. Vanaf het moment dat we een matching maken voor een aaneengesloten periode van 182 dagen, dan moeten we voldoen aan de rule dat een chauffeur een specifiek aantal vrije dagen moet krijgen in een half jaar. In de volgende paragraaf bespreken we de implementatie van deze rule.

7.2.2 Aantal vrije dagen per half jaar

De rule voor het aantal vrije dagen in een half jaar is zeer strict. Voor een kwartaal moeten we een chauffeur tussen 46% en 54% van het aantal vrije dagen voor een half jaar geven (voor een fulltime chauffeur tussen de 30 en 35 dagen), maar na een half jaar moet een chauffeur exact 100% van zijn vrije dagen hebben gehad. We missen dan de speling die nodig kan zijn om snel tot een oplossing te komen.

Als we deze regel toevoegen, blijkt dat de SPPRC heel veel moeite heeft om een toegelaten pad te vinden voor 182 dagen. Na een half jaar controleert het programma het aantal vrije dagen en vaak komt dit niet overeen met het vereiste aantal. Ook is het moeilijk om snel een oplossing te vinden die hier wel aan voldoet.

We hebben verder uitgezocht hoe veel vrije dagen we in totaal toewijzen in een jaar. We hebben dit op twee verschillende manieren onderzocht.

Vraag naar VRIJ-blok Het aantal toe te kennen roostervrije dagen kunnen we afleiden door de vraag naar het VRIJ-blok op te tellen voor alle dagen in de planningsperiode. Ons model is zo opgesteld dat we altijd voldoen aan de vraag naar het VRIJ-blok, omdat we de vraag naar de blokken en de toewijzingen matchen.

Eisen door de rules De rules stellen dat we elke fulltime chauffeur 130 vrije dagen moeten toekennen in een jaar (65 per half jaar). We kunnen nu voor elke chauffeur het aantal vrije dagen in een jaar uitrekenen en optellen. Zo bepalen we op een alternatieve wijze het aantal vrije dagen dat we in totaal moeten toewijzen.

Methode	Aantal vrije dagen
Vraag	10301
Rules	10358

Tabel 7.2: *Het aantal vrije dagen dat we in totaal moeten toewijzen in een jaar, uitgerekend met behulp van twee verschillende methodes.*

We hebben het aantal vrije dagen op beide manieren uitgerekend, waarbij we rekening hebben gehouden met de extra dummy chauffeurs in het tweede kwartaal. In tabel 7.2 hebben we deze twee waarden uiteengezet. Het is duidelijk dat beide methoden een ander aantal vrije dagen opleveren. Dat betekent dat het onmogelijk is om een complete matching te maken die voldoet aan de rule voor het aantal vrije dagen in een half jaar.

FlexPlan heeft echter wel een oplossing gegenereerd. Mogelijk zit er een fout in onze data of heeft men een aantal maatregelen genomen zodat FlexPlan wel tot een oplossing heeft kunnen komen. We weten al dat de matching is uitgevoerd met een tekort aan chauffeurs, dus misschien heeft men bij FlexPlan meer dummy chauffeurs toegevoegd, of zijn de dummy chauffeurs op een andere manier geïmplementeerd (bijvoorbeeld parttime, of juist het hele jaar door in plaats van slechts één kwartaal). Ons ontbrak echter de tijd om uit te zoeken wat de oorzaak van dit probleem is en hoe het kan worden opgelost.

We hebben de rule weggelaten die het aantal vrije dagen in een half jaar bepaalt. Voor het tweede kwartaal geldt nog wel dat een chauffeur niet minder dan 46% of meer dan 54% van zijn vrije dagen voor een half jaar toegewezen mag krijgen. Een (fulltime) chauffeur krijgt tussen de 60 en 70 vrije dagen toegewezen in een half jaar, terwijl hij 65 vrije dagen hoort te krijgen. In het algemeen geldt dat onze oplossing altijd tussen 92% en 108% van het toegestane aantal vrije dagen toekent in een half jaar. Onze oplossing verschilt dus relatief weinig van een toegelaten oplossing.

7.2.3 Resultaten voor 364 dagen

We hebben een oplossing opgesteld voor een heel jaar, waarbij we vanaf het tweede kwartaal rekening houden met de matching tot nu toe. We hebben opnieuw een tabel (zie tabel 7.3) gemaakt met daarin het aantal benodigde iteraties, de rekentijd in seconden en het gemiddelde matchingspercentage.

Kwartaal	Aant. Iter.	Rekentijd	Gem. Match. %	Opmerkingen
1	1	147	83%	
2	1	305	86%	3 dummy chauffeurs
3	1	370	85%	
4	6	1039	81%	
Totaal	9	1875	83%	

Tabel 7.3: *Een vergelijking van de oplossingen die het programma heeft gevonden voor alle vier kwartalen, waarbij we rekening houden met de oplossing tot dat moment. We tonen het aantal iteraties van het programma tot aan het vinden van een oplossing, de rekentijd in seconden en het gemiddelde matchingspercentage over alle chauffeurs.*

Kenmerkend is het feit dat zowel de rekentijd als het matchingspercentage van de oplossing weinig wordt beïnvloed door de toevoeging van de afhankelijkheid. Bij het derde kwartaal is de rekentijd nu zelfs korter dan toen we voor de kwartalen apart een matching maakten. Deze resultaten zijn een bevestiging van onze verwachting dat de planningsperiode redelijk eenvoudig kan worden opgedeeld in aparte kwartalen.

Het matchingspercentage is nog steeds boven 80%, waarmee we voldoen aan de eis van Connexxion. In de volgende paragraaf zullen we onze oplossing vergelijken met de oplossing van FlexPlan.

7.3 Vergelijking met FlexPlan

Bij Connexxion is reeds een matching gemaakt voor Dokkum voor het jaar 2010. In deze paragraaf vergelijken we de resultaten van de matching van FlexPlan met onze matching.

De oorspronkelijke berekeningen van FlexPlan zijn niet meer beschikbaar bij Connexxion, daarom hebben we eindoplossingen gebruikt als vergelijkingsmateriaal voor ons model. Deze zijn niet gelijk aan de oplossing van FlexPlan, omdat de roostermakers nog met de hand aanpassingen hebben doorgevoerd. Het matchingspercentage is hierdoor ongeveer 3% meer is geworden. Bij de vergelijking van de beide methodes zullen we er rekening mee houden dat de oplossing nog is aangepast.

FlexPlan heeft in tegenstelling tot ons wel rekening gehouden met de vorige planningsperiode (het jaar 2009). Daarnaast voldoen zij wel aan de rule voor het juiste aantal vrije dagen. FlexPlan is daarom meer beperkt geweest in het bepalen van een goede oplossing dan ons model en dat betekent dat de resultaten vertekend kunnen zijn.

7.3.1 Rekentijd

We hebben geen data van de hardware waarmee de berekeningen zijn uitgevoerd, noch van de exacte rekentijden. We weten echter wel dat de rekentijd van FlexPlan in de orde van enkele uren is. Ons programma vindt een oplossing in ongeveer een half uur, wat dus een verbetering is ten opzichte van FlexPlan.

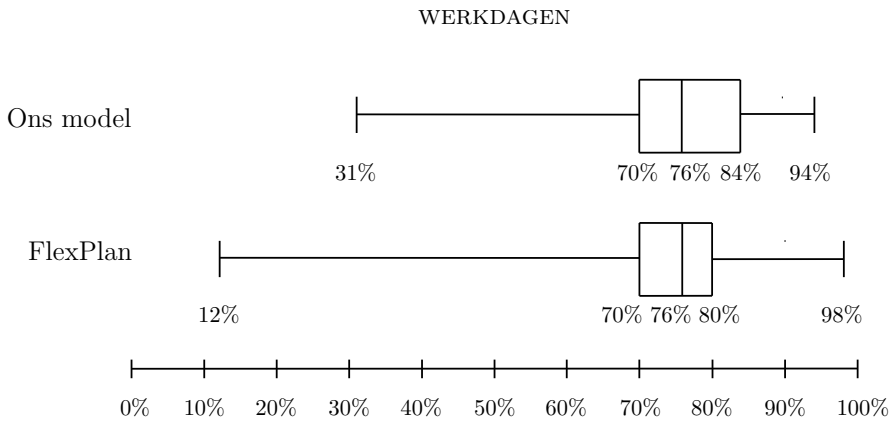
7.3.2 Matchingspercentage

Bij het matchen kent FlexPlan eerst roostervrije dagen toe en daarna pas de blokken op de werkdagen. Op deze manier is de matching van de roostervrije dagen naar verwachting beter dan de matching van de werkdagen. Bij het bepalen van het matchingspercentage wordt bij FlexPlan daarom ook onderscheid gemaakt tussen de roostervrije dagen en de werkdagen. We hebben de resultaten van onze matching vergeleken met de matching van FlexPlan op grond van de werkdagen, de roostervrije dagen en op grond van de matching als geheel, zodat de verschillen tussen de methodes duidelijker worden.

7.3.3 Matching van de werkdagen

Eerst onderzoeken we de verschillen in de matching van de werkdagen. Het gemiddelde matchingspercentage van de werkdagen verschilt erg weinig van elkaar. FlexPlan heeft een gemiddeld matchingspercentage van de werkdagen van 74,51% en ons programma behaalt een gemiddeld matchingspercentage van 74,89% voor de werkdagen. De matching van de werkdagen heeft zowel bij ons model als bij FlexPlan een matchingspercentage dat lager is dan 80%. Dit kan echter nog gecompenseerd worden door de matching van de roostervrije dagen.

We hebben opnieuw boxplots gemaakt van de belangrijkste data. In figuur 7.2 vergelijken we de matching van de werkdagen met elkaar. Hieruit blijkt dat de verschillen bij de boxplots ook niet erg groot zijn. De kwartielwaarden zijn bijna gelijk, maar ons model heeft wat minder lange staarten. Evengoed zijn de staarten naar beneden bij beide modellen erg lang, wat betekent dat beide oplossingen enkele extreme punten naar beneden hebben.

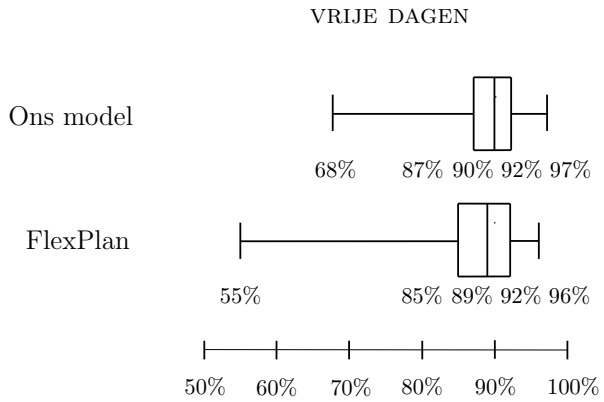


Figuur 7.2: *Boxplots van de gerealiseerde matchingspercentages van de werkdagen bij FlexPlan en bij ons model.*

7.3.4 Matching van de roostervrije dagen

Figuur 7.3 toont een vergelijking van de matching van de roostervrije dagen. Hier zijn enkele opmerkelijke resultaten te zien. We hadden verwacht dat FlexPlan een betere matching van de roostervrije dagen zou maken, maar dit blijkt niet het geval te zijn.

FlexPlan behaalt een gemiddeld matchingspercentage van 87,61%, terwijl ons model gemiddeld 89,00% haalt. Daarnaast blijkt uit de boxplots dat ons model ook minder lange staarten en een smallere box heeft. Ons model scoort op het gebied van de roostervrije dagen dus beter dan FlexPlan, hoewel de verschillen niet erg groot zijn.



Figuur 7.3: *Boxplots van de gerealiseerde matchingspercentages van de roostervrije dagen bij FlexPlan en bij ons model.*

We hadden vooraf verwacht dat ons model ongeveer hetzelfde matchingspercentage zou behalen voor de roostervrije dagen en voor de werkdagen. Het is nu duidelijk dat ons model, net zoals FlexPlan, een betere matching geeft voor de roostervrije dagen dan voor de werkdagen. Een verklaring hiervoor moet gezocht worden in onze heuristieken.

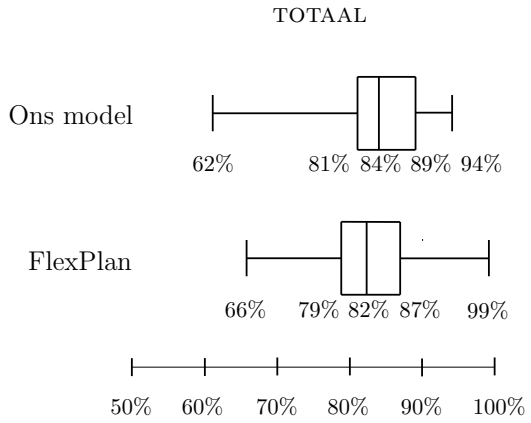
Als we het SPPRC algoritme hebben opgelost, dan hebben we voor bijna elke chauffeur een dienstrooster gemaakt met een erg hoog matchingspercentage voor zowel de werkdagen als de roostervrije dagen. Daarna gebruiken we het Swap-Algoritme om de vraag en het aantal toewijzingen gelijk te trekken. Het lukt bij dit algoritme vaak niet om veel vrije dagen te veranderen, omdat er veel eisen verbonden zijn aan de roostervrije dagen (zie ook paragraaf

6.2). Dat betekent dat het Swap-Algoritme vooral veel werkdagen verwisselt, terwijl de toewijzingen (en het matchingspercentage) van de vrije dagen grotendeels intact blijft. Het Swap-Algoritme zorgt er daarom voor dat het matchingspercentage van de roostervrije dagen relatief beter is dan het matchingspercentage van de werkdagen.

7.3.5 Matching van alle dagen

Uit de vorige paragrafen kunnen we concluderen dat ons model minstens even goed presteert bij zowel de matching van de werkdagen als van de vrije dagen. In deze paragraaf vergelijken we de resultaten van de oplossing als geheel. Voor elke chauffeur wordt het matchingspercentage bepaald als een gewogen gemiddelde van het matchingspercentage van de werkdagen en de roostervrije dagen.

In figuur 7.4 vergelijken we de matchingspercentages van de gehele periode met elkaar. De gemiddelde matchingspercentage waren respectievelijk 83,33% bij ons model en 82,74% bij FlexPlan. De boxplots verschillen niet veel van elkaar. Ons model scoort slechter op het gebied van de minimum- en de maximumwaarde, maar de mediaan en de kwartielwaarden liggen hoger.



Figuur 7.4: *Boxplots van de gerealiseerde matchingspercentages van het jaar 2010 bij FlexPlan en bij ons model.*

In de huidige situatie blijkt dus dat ons model resultaten behaalt die van een vergelijkbaar niveau zijn als de resultaten van FlexPlan. Ons model vindt deze oplossingen echter in een kortere rekentijd.

7.3.6 Onzuiverheden

Hieronder zetten we nogmaals de punten uiteen die ervoor kunnen zorgen dat de resultaten onzuiver zijn.

- FlexPlan houdt wel rekening met de rule voor het aantal roostervrije dagen in een half jaar, terwijl wij een bandbreedte toestaan van 8%.
- FlexPlan gebruikt waarschijnlijk meer chauffeurs of zet de chauffeurs op een andere manier in dan ons model zodat het mogelijk is om alle chauffeurs exact genoeg vrije dagen te geven.

- FlexPlan houdt rekening met de matching in 2009, waardoor zij vooral in het eerste kwartaal van 2010 meer beperkt zijn in het maken van de matching.
- De oplossing van FlexPlan is naderhand door de roostermakers geoptimaliseerd, waardoor het matchingspercentage van de oplossing ongeveer 3% is verbeterd ten opzichte van de oorspronkelijke oplossing.

We moeten er rekening mee houden dat we niet zo maar conclusies kunnen trekken. Verder onderzoek is dan ook nog nodig om te bepalen of ons model goed kan presteren in andere situaties en in het geval dat we alle rules volledig hebben geïmplementeerd. De resultaten zijn echter wel veelbelovend en we verwachten dat ons model uiteindelijk een geschikte vervanger kan zijn voor FlexPlan.

Hoofdstuk 8

Conclusies en aanbevelingen

We hebben een model ontwikkeld om een matching te maken voor de vestiging Dokkum, waar gebruik gemaakt wordt van zelfroostering. Het hoofddoel is om voor alle chauffeurs een dienstrooster te maken voor het jaar 2010, waarbij we een zo hoog mogelijk matchingspercentage nastreven. Een dienstrooster van een chauffeur bestaat uit de toewijzing van een specifiek blok aan elke dag van de planningsperiode.

8.1 Oplossingsmethode

Het model dat we voorstellen maakt een matching in twee stappen, waarbij eerst de dienstroosters per chauffeur worden gemaakt en vervolgens combineren we deze dienstroosters om te voldoen aan de vraag naar de blokken.

8.1.1 Deelprobleem

In de eerste stap lossen we een SPPRC (Shortest Path Problem with Resource Constraints) op waarbij elke knoop een unieke combinatie is van een blok en een dag. Door de takkosten aan te passen, kunnen we gewenste toewijzingen stimuleren of de kans op ongewenste toewijzingen verminderen. Op die manier kunnen we flexibeler omgaan met eventueel veranderende wensen vanuit Connexxion.

8.1.2 Masterprobleem

In de tweede stap proberen we aan de vraag naar de blokken te voldoen. Gebleken is dat het vinden van een toegelaten oplossing vaak al veel tijd en moeite kost. We hebben daarom meerdere heuristische methoden ontwikkeld om een oplossing voor het masterprobleem te vinden. De belangrijkste heuristische methodes zijn:

Swap-Algoritme Het Swap-Algoritme probeert toewijzingen van blokken waar een overschot van is te verwisselen met blokken waar een tekort van is (zie paragraaf 6.2).

Max-Flow algoritme Het Max-Flow algoritme maakt een matching voor een specifieke dag door een max-flow probleem op te lossen op een bipartiete graaf met aan de ene kant

de chauffeurs en aan de andere kant de blokken. Een tak tussen een chauffeur en een blok bestaat alleen als de toewijzing van de chauffeur aan het blok mogelijk is, gegeven het dienstrooster van de chauffeur op de overige dagen (zie paragraaf 6.3).

Vrije dagen verschuiven Voor een chauffeur proberen we de vrije dagen te verplaatsen van een bepaalde dag naar elders, om op die manier het aantal toegewezen vrije dagen op de dag te verminderen. Tegelijkertijd blijft het aantal vrije dagen van de chauffeur gelijk (zie paragraaf 6.4).

8.2 Toepassing van het model

Het oplossen van een SPPRC voor een heel jaar bleek erg moeilijk te zijn en te veel tijd te kosten. We hebben de planningsperiode daarom opgedeeld in kwartalen.

We hebben eerst voor alle vier kwartalen apart oplossingen gegenereerd, zonder daarbij onderlinge afhankelijkheid mee te nemen. De rekentijd voor een heel jaar bedraagt ongeveer een half uur, met een gemiddeld matchingspercentage van 83%.

Als we voor meerdere kwartalen samen een oplossing willen vinden, dan moeten we voldoen aan de rule voor het aantal vrije dagen per half jaar. Deze rule maakt het moeilijk om een oplossing voor de SPPRC te vinden. Daarnaast bleek het onmogelijk om deze rule te implementeren met de huidige data. De vraag naar de diensten was namelijk te hoog om het mogelijk te maken om de chauffeurs precies het benodigde aantal vrije dagen toe te kennen in een half jaar.

We hebben daarom een oplossing gemaakt waarbij deze rule niet wordt meegenomen, zodat we meer speling hadden met het aantal vrije dagen dat we mochten toewijzen. We verwachten niet dat deze rule ons matchingspercentage met meer dan enkele procentpunten zal verminderen, omdat we nooit meer dan 8% te veel of te weinig vrije dagen toewijzen.

We hebben onze resultaten vergeleken met FlexPlan. FlexPlan houdt wel rekening met het exacte aantal vrije dagen dat elke chauffeur in een half jaar moet krijgen. Aan de andere kant is de oplossing van FlexPlan reeds verbeterd door de roostermakers. We zijn daarom van mening dat we de oplossigen van ons model en van FlexPlan tot op bepaalde hoogte kunnen vergelijken.

We hebben onze resultaten vergeleken op grond van de matching van de werkdagen, van de vrije dagen en van de matching als geheel. Op alle vlakken behaalde ons model minstens zo goede resultaten als FlexPlan (zie tabel 8.1). De rekentijd was opnieuw ongeveer een half uur.

Matching van	**Ons model**	**FlexPlan**
Werkdagen	74,89%	74,51%
Roostervrije dagen	89,00%	87,61%
Totaal	83,33%	82,74%

Tabel 8.1: *Een vergelijking van FlexPlan met ons model. We vergelijken het gemiddelde matchingspercentage van de werkdagen, de roostervrije dagen en van de matching als geheel.*

8.3 Prestaties van het model

In paragraaf 1.3 hebben we drie maatstaven genoemd waarop we ons model kunnen beoordelen. We bespreken hier de prestaties van ons model met betrekking tot ieder van deze maatstaven.

1. **Matchingspercentage** In de huidige situatie is gebleken dat ons model minstens net zulke goede resultaten behaalt als FlexPlan op het gebied van het matchingspercentage. Ook zijn de roostervrije dagen beter gematcht dan de werkdagen, wat de chauffeurs over het algemeen belangrijk vinden.

2. **Flexibiliteit** Door het aanpassen van de takkosten, is het mogelijk om op verschillende manieren de oplossing te sturen zodat andere toewijzingen aantrekkelijker of minder aantrekkelijk worden.

3. **Rekentijd** De rekentijd van ons programma is minder dan een uur, terwijl FlexPlan meerdere uren nodig heeft voor het maken van een matching.

We concluderen dat ons model de potentie heeft om net zulke goede of zelfs betere oplossingen te genereren dan FlexPlan. Daarnaast bieden we meer flexibiliteit dankzij de mogelijkheid om te experimenteren met verschillende vormen van strafkosten.

8.4 Toekomstig onderzoek

In deze paragraaf bespreken we een aantal onderwerpen die interessant zijn voor volgend onderzoek.

8.4.1 Toelaatbaarheid van de oplossing

Verder onderzoek moet gaan uitwijzen of het mogelijk is om de rule voor het aantal vrije dagen in een half jaar toe te voegen zonder (veel) prestatieverlies. Daarbij moeten we ervoor zorgen dat de vraag naar het VRIJ-blok overeen komt met het aantal roostervrije dagen dat we volgens de rules moeten toewijzen. Daarnaast moet het SPPRC algoritme worden aangepast zodat het mogelijk is om snel een pad te vinden dat voldoet aan de rule.

Vervolgens is het van belang om te onderzoeken of er nog steeds een oplossing gevonden kan worden als de matching van 2009 meegenomen wordt bij het bepalen van de oplossing voor 2010.

8.4.2 Oplossing verbeteren

Bij het ontwikkelen van de heuristieken is gebleken dat het vinden van een toegelaten oplossing vaak al veel tijd en moeite kost. We hebben daarom vooral de focus gelegd op het vinden van een toegelaten oplossing en weinig tijd aan het verbeteren van deze oplossing. We hebben met een relatief simpele methode onze oplossing verbeterd met een kleine 10%. Mogelijk is hier een grotere verbetering haalbaar.

We hebben de chauffeurs op dit moment gesorteerd op het aantal ingevulde wensen en al onze algoritmes houden deze volgorde aan. Mogelijk zijn er verbeteringen haalbaar als we de sortering van de chauffeurs dynamisch maken. Voor de SPPRC is een sortering op het aantal ingevulde wensen van hoog naar laag bijvoorbeeld beter, maar bij het Swap-Algoritme is het misschien juist beter om te starten met de chauffeurs die weinig wensen hebben ingevuld. Het is ook mogelijk om de chauffeurs bijvoorbeeld te sorteren afhankelijk van het blok dat ze toegewezen hebben gekregen op een bepaalde dag.

8.4.3 Rekentijd

Hoewel de rekentijd op dit moment beter is dan van FlexPlan, verdient het verbeteren van de rekentijd toch enige aandacht. In veel gevallen is het mogelijk om binnen een paar iteraties een oplossing te vinden, maar soms komt het voor dat het relatief lang duurt om de laatste paar dagen te matchen (vaak vormen vooral de weekenddagen problemen). Als het lukt om dit soort dagen sneller te matchen, dan kan de rekentijd drastisch verlaagd worden en kan er ook voor gezorgd worden dat de rekentijd minder fluctueert bij verschillende invoerdata.

8.4.4 Planningsstructuur

Tot nu toe hebben we alleen gesproken over onderzoek naar het verbeteren van ons model. Het is echter ook mogelijk om verbeteringen te behalen door de planningsstructuur te veranderen.

Een mogelijkheid hiertoe ligt bijvoorbeeld in het verkorten van de planningsperiode. Op dit moment worden bij Connexxion dienstroosters voor een heel jaar gemaakt. Dat betekent dat de oplossingsruimte erg groot is en dat de chauffeurs al lang van tevoren hun wensen opgeven. Bij een planningsperiode van bijvoorbeeld een kwartaal hoeven de chauffeurs minder lang van tevoren hun rooster te plannen en tegelijkertijd is het maken van de matching waarschijnlijk minder moeilijk.

Tenslotte zou het mogelijk kunnen zijn om betere dienstroosters te creëren als de Crew Scheduling en Crew Rostering fases in zijn geheel gecombineerd worden. Hier is reeds onderzoek naar gedaan en daarbij is aangetoond dat dit vaak tot betere resultaten leidt, zie Freling e.a. (2004) en Caprara e.a. (1998). Op die manier zou het gebruik van blokken misschien ook niet meer nodig zijn. Nader onderzoek zou echter uit moeten wijzen of het bij Connexxion ook mogelijk is om deze fases succesvol te combineren.

Bijlage A

Rules van de matching

In figuur A.1 geven we een overzicht van de rules die Connexxion heeft bij het maken van de matching. We belichten hier kort enkele rules die niet direct voor zichzelf spreken.

Rule 4 De vierde rule in de figuur gaat over het aantal opeenvolgende diensten van de chauffeur. Een chauffeur mag niet te veel dagen achtereen werken zonder vrij te krijgen. In het algemeen mag een chauffeur maximaal 5 achtereenvolgende diensten rijden, waarna hij zijn weekrust krijgt. Soms zal het echter voorkomen dat de chauffeur in het weekend moet werken en in die gevallen geldt een uitzondering op deze regel. Een chauffeur mag namelijk een reeks van 6 opeenvolgende werkdagen toegewezen krijgen als deze reeks zowel een zaterdag als een zondag bevat.

In de figuur staan twee voorbeelden van hoe deze rule in de praktijk werkt. Het bovenste voorbeeld is niet toegestaan, omdat de chauffeur werkt van maandag tot en met zaterdag. Dit zijn 6 dagen, maar deze periode bevat alleen een zaterdag, terwijl 6 dagen alleen zijn toegestaan als de reeks zowel een zaterdag als een zondag bevat. Het tweede voorbeeld bevat ook een reeks van 6 werkdagen, maar deze keer is de reeks wel toegestaan omdat de reeks loopt van dinsdag tot en met zondag.

Rule 5 De vijfde rule betreft de weekrust van de chauffeurs. Elke week moet een chauffeur een onafgebroken rustperiode van 45 uur krijgen. Deze weekrust mag echter ingekort worden tot minimaal 36 uur. Als dit gebeurt, dan moet er binnen 22 dagen een aaneengesloten compensatie worden gegeven. Als de chauffeur bijvoorbeeld eens een weekrust van 38 uur heeft gekregen, dan moet hij binnen 22 dagen een keer een weekrust van minimaal $45+(45-38) = 52$ uur krijgen of een nachtrust van minimaal $11+(45-38) = 18$ uur.

Matching Rules Connexxion		
Periode	**Rule**	**Opmerkingen**
	Nachtrust is 11 uur tussen 2 blokken (dagen)	
	Na 3 nachtdiensten (GOLFPAK) of meer is de minimale nachtrust 48 uur	Geen verkorte rust na nachtdiensten
	Als de diensten van het type GLP zijn, dan minimaal 3 diensten aaneenvolgend	Dit zijn de "echte" nachtdiensten van 23:00 tot 7:00
	Niet meer dan 5 diensten achtereenvolgend. Uitzondering als er een zaterdag **en** een zondag in deze volgorde zit, dan is het maximum 6 diensten achtereenvolgend	Ma-Di-Wo-Do-Vr-Za-Zo-Ma-Di-Wo-Do-Vr-Za-Zo D D D D D D Off Off D D D D Off Off Fout, 6 dagen maar geen zaterdag EN zondag (D=Werkdag, Off=Roostervrije dag) Ma-Di-Wo-Do-Vr-Za-Zo-Ma-Di-Wo-Do-Vr-Za-Zo Off D D D D D D Off Off D D D Off Off Correct, 6 dagen aangesloten maar er zit een zaterdag en zondag in
7 dagen	Weekrust is 45 uur. Mag bekort worden tot minimaal 36 uur met compensatie binnen 22 dagen. Deze compensatie mag ook op een dagelijkse rust worden gegeven. Er mogen geen 2 bekorte rusten achter elkaar worden gegeven dus eenmaal per 14 dagen minimaal een keer 45 uur.	Ma-Di-Wo-Do-Vr-Za-Zo-Ma-Di-Wo-Do-Vr-Za-Zo D D D D D D Off D D D D Off D D Fout 2x 36 uur na elkaar Ma-Di-Wo-Do-Vr-Za-Zo-Ma-Di-Wo-Do-Vr-Za-Zo D D Off D D D D Off Off Off D D Off Off Correct, 36 uur op Wo maar 45 uur plus compensatie op Ma Di Wo
week	Maximaal 2 "D" blokken (gebroken diensten) per week	Gemiddeld 1 per week voor een jaar. Tenzij meer aangevraagd dan mag ook meer
28 dagen	Maximaal 3 zaterdagen werken	Meer mag op verzoek
13 weken	Minimaal 56 werkdagen	Dit is afhankelijk van de bezetting en afspraken
13 weken	Maximaal 61 werkdagen	Dit is afhankelijk van de bezetting en afspraken
13 weken	Een gemiddelde van 9 werkdagen per 14 dagen. Voor partimers afhankelijk van hun contract.	Uitgezonderd vakantie periode Week loopt van zondag t/m zaterdag
13 weken	Maximaal 9 zondagen werken	Minimaal 4 zondagen vrij
	Een zondag werken gaat altijd samen met de zaterdag werken dus zaterdag vrij is ook zondag vrij	Uitgezonderd wanneer de chauffeur hier zelf om vraagt
13 weken	Maximaal 29 nachtdiensten. Meer mag als aangevraagd met een maximum van 32	Nachtdiensten zijn alle diensten die een werktijd hebben die geheel of deels tussen 0h00 en 6h00 vallen (GOLFPAK diensten)
13 weken	Minimaal 46% van de vrije dagen (30 dagen voor full-time)	Dit is 46% van het aantal vrije dagen per periode van 26 weken
26 weken	Per periode van 26 weken moeten er voor de full-time chauffeurs 65 vrije dagen worden gepland	Dit is zowel voor de eerste als voor de tweede periode van 26 weken. Voor part-time afhankelijk van contract.

Figuur A.1: *De verschillende voorwaarden waaraan het dienstrooster moet voldoen*

Bibliografie

Ahuja, R., Magnanti, T., en Orlin, J. (1993). *Network flows. Theory, algorithms and applications*. Prentice-Hall, New-York.

Alfares, H. (1998). An efficient two-phase algorithm for cyclic days-off scheduling. *Computers Operations Research*, 25(11):913–923.

Alfares, H. (2004). Survey, categorization, and comparison of recent tour scheduling literature. *Annals of Operations Research*, 127:145–175.

Baker, K. (1976). Workforce allocation in cyclical scheduling problems: A survey. *Operational Research Quarterly*, 27(1):155–167.

Bard, J., Binici, C., en deSilva, A. (2003). Staff scheduling at the United States postal service. *Computers and Operations Research*, 30:745–771.

Baxter, J. en Mosby, M. (1988). Generating acceptable shift-working schedules. *The Journal of the Operational Research Society*, 39(6):537–542.

Beaulieu, H., Ferland, J., Gendron, B., en Michelon, P. (2000). A mathematical programming approach for scheduling physicians in the emergency room. *Health Care Management Science*, 3:193–200.

Bentley, J. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229.

Berrada, I., Ferland, J., en Michelon, P. (1996). A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences*, 30(3):183–193.

Bianco, L., Bielli, M., Mingozzi, A., Ricciardelli, S., en Spadoni, M. (1992). A heuristic procedure for the crew rostering problem. *European Journal of Operational Research*, 58:272–283.

Blais, J., Lamont, J., en Rousseau, J. (1990). The HASTUS vehicle and manpower scheduling system at the société de transport de la communauté urbaine de Montréal. *Interfaces*, 20(1):26–42.

Boubaker, K., Desaulniers, G., en Elhallaoui, I. (2010). Bidline scheduling with equity by heuristic dynamic constraint aggregation. *Transportation Research Part B*, 44:50–61.

Burke, E., de Caumaecker, P., vanden Berghe, G., en van Landeghem, H. (2004). The state of the art of nurse rostering. *Journal Of Scheduling*, 7:441–499.

Butchers, E., Day, P., Goldie, A., Miller, S., Meyer, J., Ryan, D., Scott, A., en Wallace, C. (2001). Optimized crew scheduling at air New Zealand. *Interfaces*, 31(1):30–56.

Caprara, A., Fischetti, M., Guida, P., Toth, P., en Vigo, D. (1999). Solution of large-scale railway crew planning problems: the Italian experience. *Computer-Aided Transit Scheduling*, 430:1–18.

Caprara, A., Fischetti, M., Toth, P., Vigo, D., en Guida, P. (1997). Algorithms for railway crew management. *Mathematical Programming*, 79:125–141.

Caprara, A., Focacci, F., Lamma, E., Mello, P., Milano, M., Toth, P., en Vigo, D. (1998). Integrating constraint logic programming and operations research techniques for the crew rostering problem. *Software Practice and Experience*, 28(1):49–76.

Caprara, A., Monaci, M., en Toth, P. (2001). A global method for crew planning in railway applications. *Computer-Aided Scheduling of Public Transport*, 505:17–36.

Caprara, A., Monaci, M., en Toth, P. (2003). Model and algorithms for a staff scheduling problem. *Mathematical Programming*, 98:445–476.

Christou, I., Zakarian, A., Liu, J., en Carter, H. (1999). A two-phase genetic algorithm for large-scale bidline-generation problems at Delta Air Lines. *Interfaces*, 29(5):51–65.

Connexxion (2008). *Connexxion Jaarverslag*.

Dantzig, G. (1954). A comment on Edie's "Traffic delays at toll booths". *Journal of the Operations Research Society of America*, 2(3):339–341.

Day, P. en Ryan, D. (1997). Flight attendant rostering for short-haul airline operations. *Operations Research*, 45(5):649–661.

de Pont, G. (2006). Personalized crew rostering at Netherlands railways. Master's thesis, Universiteit van Tilburg.

Desrosiers, J., Lasry, A., McInnis, D., Solomon, M., en Soumis, F. (2000). Air Transat uses ALTITUDE to manage its aircraft routing, crew pairing, and work assignment. *Interfaces*, 30(2):41–53.

Ernst, A., Jiang, H., Krishnamoorthy, M., Owens, B., en Sier, D. (2004a). An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, (127):21–144.

Ernst, A., Jiang, H., Krishnamoorthy, M., en Sier, D. (2004b). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, (153):3–27.

Fahle, T., Junker, U., Karisch, S., Kohl, N., Sellmann, M., en Vaaben, B. (2002). Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59–87.

Ford, L. en Fulkerson, D. (1962). *Flows in Networks*. The RAND Corporation, 1700 Main Street Santa Monica California United States.

Freling, R., Lentink, R., en Wagelmans, A. (2004). A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm. *Annals of Operations Research*, 127:203–222.

Gamache, M., Hertz, A., en Ouellet, J. (2007). A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers and Operations Research*, 34:2384–1295.

Gamache, M., Soumis, F., Marquis, G., en Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, 47(2):247–263.

Gamache, M., Soumis, F., Villeneuve, D., Desrosiers, J., en Gélinas, E. (1998). The preferential bidding system at Air Canada. *Transportation Science*, 32(3):246–255.

Gärtner, J., Musliu, N., en Slany, W. (2001). Rota: a research project on algorithms for workforce scheduling and shift design optimization. *AI Communications*, 14:83–92.

Glover, F. (1989). Tabu search - part I. *Journal On Computing*, 1(3):190–206.

Glover, F. (1990). Tabu search - part II. *Journal On Computing*, 2(1):4–32.

Grötschel, M., Borndörfer, R., en Löbel, A. (2003). Duty scheduling in public transit. *Mathematics - key technologies for the future.*

Hartog, A., Huisman, D., Abbink, E., en Kroon, L. (2005). Decision support for crew rostering at NS. *Econometric Institute Report.*

Hillier, F. en Lieberman, G. (2005). *Introductions To Operations Research.* McGraw-Hill, 8 edition.

Irnich, S. en Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers, G., Desrosiers, J., en Solomon, M., editors, *Column Generation*, GERAD 25th Anniversary Series, pages 33–65. Springer.

Jarrah, A. (1997). The problem of generating crew bidlines. *Interfaces*, 27(4):49–64.

Jones, R. (1989). Development of an automated airline crew bid generation system. *Interfaces*, 19(4):44–51.

Junker, U., Karisch, S., Kohl, N., Vaaben, B., Fahle, T., en Sellmann, M. (1999). A framework for constraint programming based column generation. *Proceedings of the 5th International Conference on the Principles and Practice of Constraint Programming*, 1713:261–274.

Kennington, J. en Wang, Z. (1992). A shortest augmenting path algorithm for the semi-assignment problem. *Operations Research*, 40(1):178–187.

Kohl, N. en Karisch, S. (2004). Airline crew rostering: Problem types, modeling, and optimization. *Annals of Operations Research*, 127:223–257.

Lau, H. (1996). On the complexity of manpower shift scheduling. *Computers & Operations Research*, 23(1):93–102.

Lezaun, M., Pérez, G., en de la Maza, E. S. (2006). Crew rostering problem in a public transport company. *Journal of the Operational Research Society*, 57:1173–1179.

Lezaun, M., Pérez, G., en de la Maza, E. S. (2007). Rostering in a rail passenger carrier. *Journal of Scheduling*, 10(4-5):245–254.

Maenhout, B. en Vanhoucke, M. (2007). A hybrid scatter search heuristic for personalized crew rostering in the airline industry. *To Appear In: European Journal Of Operational Research 2010*. Working Paper.

Morris, J. en Showalter, M. (1983). Simple approaches to shift, days-off and tour scheduling problems. *Management Science*, 29(8):942–950.

Nicoletti, B. (1975). Automatic crew rostering. *Transportation Science*, 9(1):33–42.

Quak, C. B. (2003). Bus line planning: A passenger-oriented approach of the construction of a global line network and an efficient timetable. Master's thesis, Technische Universiteit Delft, The Netherlands.

Rich, E. en Knight, K. (1990). *Artificial Intelligence*. McGraw-Hill Higher Education, 2nd edition.

Ryan, D. (1992). The solution of massive generalized set partitioning problems in aircrew rostering. *The Journal of the Operational Research Society*, 43(5):459–467.

Sodhi, M. en Norris, S. (2004). A flexible, fase, and optimal modeling approach applied to crew rostering at London underground. *Annals of Operations Research*, 127:259–281.

Teodorović, D. en Lucić, P. (1998). A fuzzy set theory approach to the aircrew rostering problem. *Fuzzy Sets and Systems*, 95:261–271.

Townsend, W. (1988). An approach to bus-crew roster design in London regional transport. *The Journal of the Operational Research Society*, 39(6):543–550.

van den Heuvel, A. (2007). Integrating timetabling and vehicle scheduling. Master's thesis, Universiteit Utrecht, The Netherlands.

Volgenant, T. en Jonker, R. (1982). A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *European Journal Of Operational Research*, (9):83–89.

Weir, J. en Johnson, E. (2004). A three-phase approach to solving the bidline problem. *Annals of Operations Research*, 127:283–308.

Yunes, T., Moura, A., en de Souza, C. (2005). Hybrid column generation approaches for urban transit crew management problems. *Transportation Science*, 39(2):273–288.

Zijp, E. (2005). Simultaneous scheduling of vehicles and crew. Master's thesis, Technische Universiteit Delft, The Netherlands.