Het toepassen van geheeltallige programmering bij het Student Scheduling Problem

Een case-studie

Hinke Visser
BWI-werkstuk
Begeleider: Prof. Dr. Rob van der Mei

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

Juni 2008

Voorwoord

Het BWI werkstuk is één van de laatste onderdelen van de studie Bedrijfswiskunde & Informatica aan de Vrije Universiteit in Amsterdam. In dit werkstuk wordt een onderzoek uitgevoerd waarin twee van de drie aspecten van de studie BWI aan bod komen: Bedrijfskunde, Wiskunde & Informatica.

In dit werkstuk heb ik een praktijkstudie gedaan naar de planning van trainingen bij het accountantsbedrijf Deloitte. Dit probleem is te omschrijven als een Student Scheduling Problem dat ik vervolgens met geheeltallige lineaire programmering (IP) probeer op te lossen. Met de resultaten is vervolgens gekeken welke strikte voorwaarden de mogelijke 'bottlenecks' zijn bij het optimaliseren van het probleem.

Graag wil ik Rob van der Mei bedanken bij het meedenken van de aanpak en zijn advies. Verder wil ik Michael van der Meulen en Daisy Helmig van Deloitte bedanken voor hun hulp bij het verkrijgen van de data.

Hinke Visser
Juni 2008

Management Samenvatting

Veel praktijkproblemen zijn te beschrijven als roosterproblemen, waarbij deze problemen weer onderverdeeld kunnen worden in verschillende soorten. Onder één van deze soorten behoort ook het zogenaamde Student Scheduling Problem. Dit is een specifiek roosterprobleem waarbij de wensen van de studenten centraal staan bij het maken van een rooster.

Er zijn verschillende oplossingstechnieken voor roosterproblemen, waaronder lineaire programmering. Omdat roosterproblemen van nature geheeltallig zijn (het gaat over aantallen mensen), zijn dit geheeltallige lineaire programmeringsproblemen (IP).

In deze case-studie wordt een roosterprobleem in de praktijk bij Deloitte opgelost. Het gaat om de planning van trainingen in twee trainingsweken. Omdat de trainingen door de medewerkers uitgekozen worden en daar een rooster bij gemaakt moet worden, kan dit probleem gezien worden als een Student Scheduling Problem.

Met behulp van geheeltallige lineaire programmering is een model gemaakt. Alle ‘hard’ en ‘soft’ constraints (voorwaarden) zijn opgesteld om tot een toelaatbare (feasible) oplossing te komen. Hiermee is een rooster kunnen maken en verschillende ‘bottlenecks’ zijn bekeken die de grootste invloed hebben op de oplossing.

Inhoudsopgave

1. Een roosterprobleem

1.1 Inleiding

In een grote accountantsorganisatie als Deloitte worden elk jaar trainingsweken aangeboden. Medewerkers schrijven zich in voor trainingen en deze moeten zo efficiënt mogelijk worden ingepland zodat zoveel mogelijk trainingen kunnen worden gevolgd en de kosten zo laag mogelijk zijn. Daarnaast zijn er ook andere beperkingen, onder andere wat betreft locatie, facilitators en het aantal deelnemers van een training.

Dit is een duidelijk voorbeeld van een roosterprobleem: het maken van een les- en/of werkrooster dat aan een hele hoop randvoorwaarden moet voldoen. Een kenmerk van het roosterprobleem is het vaak ontbreken van een perfecte oplossing. Wat zijn de voornaamste 'bottlenecks' die dit veroorzaken?

Het handmatig zoeken naar een oplossing van een roosterprobleem vereist veel dagen en/of personen aan werk. Gedurende de laatste dertig jaar is er veel gepubliceerd over het automatiseren van roosterproblemen. In deze case-studie wordt het plannen van de trainingsweken bij Deloitte geautomatiseerd en wordt op zoek gegaan naar de 'bottlenecks' die een 'perfecte' oplossing in de weg staan.

In de rest van dit hoofdstuk beschrijf ik verschillende soorten roosterproblemen, verschillende toepassingsgebieden en mogelijke oplossingstechnieken.

1.2 Verschillende soorten

De literatuur beschrijft een groot aantal soorten van het roosterprobleem. Deze soorten verschillen van elkaar op basis van het soort instituut (universiteit of middelbare school) en het type van de verschillende voorwaarden (constraints). Roosterproblemen worden vaak in drie hoofdgroepen verdeeld, [1]:

1. **School timetabling:** een wekelijks rooster voor alle klassen van een school, waarbij onderwijzers niet twee klassen op hetzelfde tijdstip zijn ingeroosterd en andersom.

2. **Examination timetabling:** een rooster voor de examens van een verzameling universiteitsvakken, waarbij de overlap van examens voor vakken met dezelfde studenten wordt geminimaliseerd en de examens voor de studenten zoveel mogelijk worden uitgespreid over de examenperiode.

3. **Course timetabling:** een wekelijks rooster van alle lessen van een verzameling universiteitsvakken, waarbij de overlap van de lessen van de vakken met dezelfde studenten wordt geminimaliseerd.

Dit is slechts een ruwe verdeling, want er zijn ook genoeg roosterproblemen die in meerdere groepen kunnen worden ingedeeld. Het 'school timetabling' probleem is ook wel bekend als het *class-teacher model.* In de simpelste vorm gaat het om het toekennen van vakken aan periodes op een dusdanige manier dat geen onderwijzer of klas is ingedeeld bij meer dan één les op een bepaald tijdstip. Een

bewijs is gegeven dat voor de simpelste vorm van dit probleem altijd een oplossing bestaat, zolang er minder lessen zijn dan mogelijke tijdstippen, [2].

Het grootste verschil tussen course timetabling en examination timetabling problemen is dat bij het examination probleem er telkens één examen bij een vak hoort, de voorwaarde voor tijdconflicten zeer strikt is en meerdere examens in één ruimte kunnen worden gegeven.

Het grootste verschil tussen course timetabling en school timetabling problemen is dat verschillende universiteitsvakken dezelfde studenten kunnen hebben, terwijl schoolklassen uit een vaste groep studenten bestaat.

Een variant van het course timetabling probleem is het Student Scheduling Problem (SSP). Een student wordt gevraagd om een aantal vakken te volgen, welke zelf gekozen moeten worden, waarna een rooster wordt gemaakt. Het doel is zoveel mogelijk te voldoen aan de wensen van de studenten door het verschaffen van een confict-vrij rooster. Er is in de literatuur laten zien dan dit probleem NP-hard is, [3]. NP-hard staat voor 'nondeterministic polynomial-time hard', dat wil zeggen dat voor het probleem geen polynomiaal-tijds algoritme bestaat maar dat de rekentijd exponentieel snel toeneemt in de probleemgrootte.
Wanneer alle trainingen op het wensenlijstje van de student moeten worden toegewezen aan de studenten ontstaat in de meeste gevallen een lege (blanco) geschikte oplossing.

De situatie bij Deloitte kan gezien worden als een Student Scheduling Problem; medewerkers mogen trainingen uitkiezen die ze willen volgen, waarna een rooster voor de gekozen trainingen moet worden gemaakt. Om niet tot een lege oplossing te komen formuleren we dit probleem als een *optimization problem:* wat geëist wordt is een rooster dat voldoet aan alle *hard constraints* (harde voorwaarden) en maximaliseert (of minimaliseert) een gegeven doelfunctie die de *soft constraints* bevat. Door het onderscheid te maken tussen 'hard constraints' (voorwaarden met prioriteit) en 'soft constraints' (niet onoverkomelijke randvoorwaarden) kan er mogelijk gekomen worden tot een acceptabele oplossing.

1.3 Toepassingsgebieden

Het SSP levert een *demand-driven timetabling*. Op basis van de vraag van studenten naar verschillende vakken wordt een rooster gemaakt. Deze specifieke vorm van het roosterprobleem wordt al bij enkele universiteiten toegepast, [4] en [5]. Maar deze vorm van het roosterprobleem kan bijvoorbeeld ook worden gevonden bij:

Het leger: Basis trainingen en gevorderde individuele trainingen voor soldaten moeten per week gedurende een heel jaar worden ingeroosterd. De vraag naar deze trainingen varieert tijdens het jaar en geldt als basis voor de planning van de trainingen. Het doel is het minimaliseren van gewenste trainingen die niet worden ingepland en het maximaliseren van een optimale bezetting van trainingen, [6]. Het grootste verschil met dit probleem en de universiteit is de wisselende vraag gedurende de tijdshorizon.

Vliegtuigmaatschappijen: Gedurende het jaar wisselen piloten van positie, bijvoorbeeld verandering van vliegtuigtype, promotie in functie of een nieuwe thuisbasis. Verschillende posities vereisen verschillende vaardigheden en daarom moeten piloten die wisselen van positie worden bijgeschoold. Bij voldoende aanvraag voor een training moet deze worden ingeroosterd in het werkrooster. Een

piloot minder in het werkrooster levert extra kosten voor de vliegtuigmaatschappij. Het doel is het minimaliseren van de lengte van trainingen en daarmee het beperken van de kosten, [7].
Het grootste verschil met dit probleem en de universiteit is de combinatie van werk en trainingen.

1.4 Oplossingstechnieken

De meeste roosterproblemen zijn NP compleet (zie sectie 1.2), ook al hebben ze de meest simpele vorm, het vinden van een optimale oplossing of het leveren van een optimale toegestane oplossing lijkt vaak onmogelijk wanneer de probleemgrootte erg groot wordt. Oplossingstechnieken zijn onder te verdelen in:

1. **Mathematisch programmeren**
   Het optimaliseren van een doelfunctie met een eindig aantal nevenvoorwaarden in een eindig aantal dimensies. Vaak bestaat de doelfunctie uit het maximaliseren van de opbrengst of het minimaliseren van de kosten. Bij roosterproblemen kunnen bijvoorbeeld de 'boete'-kosten worden geminimaliseerd wanneer iets niet geheel 'juist' wordt ingeroosterd.

2. **Contraint programming**
   Contraint programming is de studie van computationele systemen gebaseerd op beperkingen. Het idee achter contraint programming is om problemen op te lossen door beperkingen over het domein van het probleem te formuleren en daaruit voortvloeiend oplossingen te vinden die aan deze beperkingen voldoen.

3. **Meta-heuristieken**
   Heuristieken zoeken naar een aanvaardbare oplossing in een aanvaardbare tijd. Meta-heuristieken zijn één van de betere methoden om een aanvaardbare oplossing te vinden in gevallen waarin de zoekruimte heel groot is en in gevallen waarin het zelfs moeilijk is om een aanvaardbare (feasible) oplossing te vinden. Meta-heuristieken gebruiken meerdere heuristieken om oplossingen voor harde problemen te vinden. Voorbeelden van meta-heuristieken zijn simulated annealing, genetische algoritmen en tabu search.

In de literatuur zijn verschillende oplossingstechnieken voor het Student Scheduling Problem beschreven. Een wiskundige formulering is beschreven, waarbij de voorwaarden van het optimaliseringprobleem worden opgesplitst in harde en zachte. De enige harde voorwaarde is dat de wensen van de studenten moeten worden gerespecteerd, conflict in tijd is een zachte voorwaarde, [8]. Het SSP is tevens beschreven als een geheeltallig lineair programmeringprobleem wat Lagrange Relaxatie gebruikt om het op te lossen, [9], maar ook goal programming wordt als oplossingstechniek genoemd, [10].

Voor het roosterprobleem van trainingen bij Deloitte wordt in dit onderzoek (gemengd) geheeltallige lineaire programmering gebruikt, wat onder Mathematisch programmeren valt. Deze techniek wordt beschreven in het volgende hoofdstuk.

2. Geheeltallige lineaire programmering

2.1 Inleiding

In 1947 werd door George B. Dantzig de uitvinding gedaan van de simplex-methode voor lineaire programmering. Dit werd een gedenkwaardig jaar voor de toegepaste wiskunde want lineaire programmering (LP) is tot op de dag van vandaag één van de meest toegepaste wiskundige modellen in de praktijk. Bij een LP wordt de optimale waarde bepaald van een lineaire criteriumfunctie (doelfunctie) van meerdere beslissingsvariabelen. Deze variabelen moeten voldoen aan een aantal lineaire bijvoorwaarden, [11]. In de volgende paragrafen wordt de theorie van LP besproken en de link naar geheeltallige lineaire programmering (IP = integer programming) gemaakt. Tevens wordt beschreven hoe het programma kan worden opgelost en wat de nadelen zijn.

2.2 Theorie

Een lineair programma is een programma/model dat de volgende standaardvorm heeft:

$$\begin{array}{ll} \text{Maximaliseer} & c^T x \\ \text{Onderworpen aan} & Ax = b \\ & x \geq 0, \end{array}$$

waarbij x de vector van variabelen voorstelt waarvoor een oplossing gezocht moet worden, c en b vectoren met bekende coëfficiënten zijn en A een matrix van bekende coëfficiënten is. De uitdrukking $c^T x$ die gemaximaliseerd (of geminimaliseerd) moet worden, wordt de doelfunctie genoemd (objective function). De vergelijkingen $Ax = b$ zijn de beperkingen (constraints). Deze standaardvorm veronderstelt wel dat de rijen van de matrix A lineair onafhankelijk zijn.

De **simplex methode** lost LP-problemen op door op een 'slimme manier' door alle mogelijke toegelaten basisoplossingen van de standaardvorm van het LP-probleem heen te lopen. In het kort doorloopt de simplex-methode de volgende stappen, [11]:

1. Construeer een toegelaten basisoplossing als startoplossing.

2. Uitgaande van de huidige basisoplossing, bepaal een nieuwe basisoplossing die slechts één basisvariabele verschilt met de huidige basisoplossing. Daarvoor wordt één van de huidige niet-basis variabelen positief gemaakt en één van de huidige basisvariabelen gelijk aan nul gesteld. De keuze voor het wisselen van variabelen hangt af van verbetering van de criteriumwaarde en het toegelaten zijn van de nieuwe basisoplossing.

3. Ga door met construeren van verbeterde basisoplossingen totdat de criteriumfunctie niet verder verbeterd kan worden. Dit is dan de optimale oplossing gevonden met de simplex-methode.

Als vereist wordt dat de onbekende variabelen allemaal geheeltallig zijn, wordt het probleem een ***geheeltallig lineair programmeringsprobleem*** (**IP** = integer programming) genoemd. Wanneer, in het speciale geval, de variabelen alleen de waarden 0 en 1 mogen aannemen (in plaats van willekeurige gehele getallen) wordt het een ***0 -1 probleem*** genoemd. Als vereist wordt dat slechts enkele variabelen

geheeltallig moeten zijn, spreken we over een ***gemengd geheeltallig programmeringsprobleem*** (MIP = mixed integer programming).

In het algemeen is de rekenkundige complexiteit van IP problemen veel groter dan van LP problemen. IP problemen zijn meestal NP-compleet, dat wil zeggen dat de rekentijd exponentieel snel toeneemt in de probleemgrootte. Deze rekengrootte hangt bij IP problemen sterk af van:

1. **Het aantal geheeltallige variabelen en de gekozen formulering van het probleem**
   Om het aantal geheeltallige variabelen zo minimaal mogelijk te houden, is het altijd goed om na te gaan of sommige variabelen niet toch als continue variabelen kunnen worden gedeclareerd.

2. **De gekozen formulering van het probleem**
   Een vindingrijke formuleringsopzet is vaak vereist door de trucs die nodig zijn om bepaalde 'als-dan' voorwaarden aan het model toe te voegen met behulp van binaire variabelen (variabelen die alleen de waarden 0 of 1 kunnen aannemen). Over het algemeen geldt dat hoe scherper de variabelen begrensd kunnen worden, des te minder de rekentijd.

2.3 LP-relaxatie

De LP-relaxatie van een geheeltallig programmeringsprobleem is gedefinieerd als het LP-probleem dat ontstaat door in het IP-probleem (een deel van) de geheeltalligheidseisen te laten vallen (bijvoorbeeld de eis $x_j \in \{0,1\}$ wordt verzwakt tot $0 \leq x_j \leq 1$). De LP-relaxatie is zeer bruikbaar in het oplossingsproces van het oorspronkelijke IP-probleem. Er gelden hierbij twee principes:

1. De maximale critieriumwaarde van het oorspronkelijke geheeltallige LP-probleem is altijd kleiner dan of gelijk aan de optimale criteriumwaarde van de LP-relaxatie.
2. De criteriumwaarde van een toegelaten oplossing voor het geheeltallige LP-probleem geeft een ondergrens voor de maximale criteriumwaarde van het geheeltallige LP-probleem.

Bij de *Rounding-Off* procedure wordt de optimale oplossing van de LP-relaxatie naar boven of naar beneden afgerond (afhankelijk van de criteriumfunctie). Deze methode geeft echter niet noodzakelijk het juiste resultaat voor het oorspronkelijke IP probleem. Dit laat het volgende voorbeeld zien.

Probleemformulering:

$$\max(2x_1 + 3x_2)$$
$$x_1 + 5x_2 \leq 10$$
$$2x_1 + 4x_2 \leq 12$$
$$3x_1 + 2x_2 \leq 15$$
$$x_1, x_2 \geq 0 \quad \textit{en geheel}$$

Figuur 1 laat zien dat de afgeronde optimale oplossing (b) van de optimale oplossing van de LP-relaxatie (a) niet gelijk is aan de optimale oplossing van het IP-probleem (c).

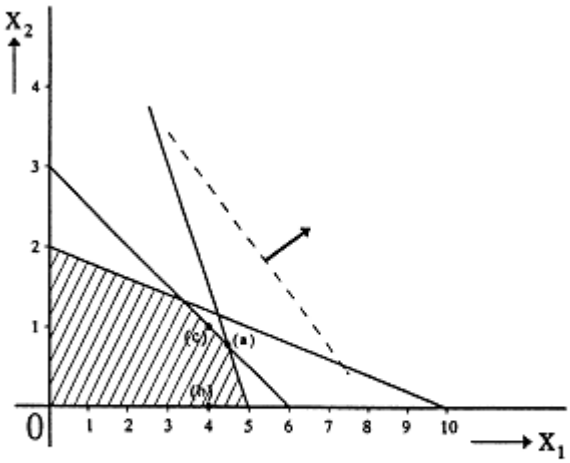

Figure 1. (a) is optimal LP-relaxation solution,
(b) is rounded-down solution, and
(c) is optimal ILP-solution.

Wanneer afronding tot een toegelaten oplossing leidt en deze oplossing voldoende dicht bij de bovengrens van de maximale criteriumwaarde van het geheeltallige LP-probleem ligt, is deze toegelaten oplossing vaak een goede sub-optimale oplossing. In de praktijk is men vaak met zo'n oplossing al heel erg tevreden.

2.4 Branch-en-Bound algoritme

Een klassieke en meest gebruikte benadering voor het oplossen van geheeltallige programmeringsproblemen is het branch-en-bound algoritme. Het basisidee van het branch-en-bound algoritme is om de totale verzameling van toegelaten oplossingen te splitsen (branching) in steeds kleinere deelverzamelingen, de grenzen van de criteriumwaarde voor elk corresponderend submodel te berekenen (bounding) en deze dan te gebruiken om sommige submodellen te verwerpen voor verdere beschouwing. De grenzen voor het huidige submodel kunnen verkregen worden door het model te vervangen met een eenvoudiger model (relaxatie). Deze procedure eindigt wanneer elk submodel of toegelaten oplossing heeft aangetoond geen betere oplossing te zijn dan tot nog toe gevonden.

De standaardvorm van een IP-model is:

$$\max\{c^T x \mid Ax \le b,\ x \ge 0,\ x\ geheel\}$$

met $x,\ c \in \Re^n,\ b \in \Re^m\ \ en\ A \in \Re^{mxn}$.
De LP-relaxatie van het IP-model geeft een grotere verzameling toegelaten oplossingen en voldoet aan:

$$\max\{c^T x \mid Ax \le b,\ x \ge 0\} \ge \max\{c^T x \mid Ax \le b,\ x \ge 0,\ x\ geheel\}$$

In de eerste stap van het branch-en-bound algoritme wordt de LP-relaxatie opgelost. De gevonden criteriumwaarde is een bovengrens voor de optimale criteriumwaarde van het originele IP-model. Als de oplossing van de LP-relaxatie integer (geheeltallig) is, zijn we klaar en is de oplossing ook de optimale oplossing van het originele IP-probleem. Wanneer de gevonden oplossing geen integer is, wordt de verzameling toegelaten oplossingen van de LP-relaxatie verdeeld in twee nieuwe LP-

modellen. Voor deze twee nieuwe modellen wordt de criteriumwaarde bepaald en gekeken of ze voor verdere beschouwing nuttig zijn. Dit model wordt weer verdeeld in twee nieuwe LP-modellen, etc.

2.5 Nadelen

Het is bekend dat lineaire programmering zeer populair is omdat veel praktische problemen gemodelleerd kunnen worden als een lineair programma of op z'n minst een benadering ervan. Er bestaat verder effectieve software waarmee oplossingen gevonden kunnen worden. Desondanks zijn er ook een aantal nadelen:

1. Voor alle coëfficiënten moeten numerieke waarden worden meegegeven. Maar in sommige situaties kan er niet volledige zekerheid worden gegeven over de werkelijke waarde van deze coëfficiënten.
2. Voor geheeltallige problemen zijn extra trucs nodig voor bepaalde voorwaarden om een optimale oplossing te verkrijgen. Het formuleren van deze trucs is tijdrovend.
3. Het probleem moet te schrijven zijn als LP of IP. Daardoor kunnen vaak niet de daadwerkelijke problemen opgelost worden, maar slechts vereenvoudigingen ervan.

3. Het model

In dit hoofdstuk ga ik het roosterprobleem bij Deloitte als praktijkvoorbeeld oplossen met behulp van geheeltallige lineaire programmering. Eerst zal ik het probleem beschrijven en de huidige situatie bij Deloitte. Vervolgens beschrijf ik het model en hoe het geïmplementeerd is. De resultaten worden besproken en de mogelijke bottlenecks worden aangekaard die de grootste invloed hebben op de resultaten.

3.1 Het probleem

Deloitte heeft als doel haar medewerkers zoveel mogelijk bij te leren en de mogelijkheid te geven zich te ontwikkelen. Daarvoor worden er ieder jaar drie trainingsweken ingepland, de zogenoemde 'TechWeeks', waar medewerkers trainingen kunnen volgen. Iedere medewerker heeft een bepaald functieniveau (consultant, senior consultant, manager, etc) en is ingedeeld in een bepaalde competentiegroep. Het curriculum bevat een trainingsschema per competentie en per functieniveau. Daarin wordt per functieniveau laten zien of een training verplicht ('Mandatory') is, wordt aangeraden ('Recommended') of een toegevoegde waarde heeft ('Value Added'). Elke medewerker maakt naar aanleiding van dit curriculum zijn eigen 'wensenlijstje' van te volgen trainingen, waarbij gefocust moet worden op de trainingen die verplicht zijn. Omdat de wensen van trainingen van de medewerkers centraal staat is dit probleem te zien als een Student Scheduling Problem (zie sectie 1.4).

Dit jaar is besloten dat TechWeek 1 voor alle eerstejaars consultants zal zijn. Er moet dus een planning worden gemaakt voor TechWeek 2 en TechWeek 3 (vanaf hierna genoemd week 1 en week 2). Alle wensen worden verzameld en moeten zo worden ingepland dat er een planning ontstaat voor deze twee weken dat aan de volgende voorwaarden voldoet:

1. de lengte van de trainingen moet gelijk zijn aan het aantal dagen wat ervoor staat (een training waar twee dagen voor staat, kan niet ineens op maar één dag worden ingepland)

2. er moet rekening worden gehouden met een maximum van vijf zalen per dag (een training waar geen zaal voor beschikbaar is kan niet gegeven worden)
3. de training moet op de juiste locatie worden gegeven (week 1 is in Amstelveen (op kantoor) en week 2 is in Noordwijkerhout (in een hotel zonder intra-net verbinding met Deloitte))
4. er moet rekening worden gehouden met de beschikbaarheid van de facilitator(s) (een training kan niet doorgaan als de facilitator(s) niet beschikbaar zijn)
5. trainingen die langer duren dan één dag zijn op achtereenvolgende dagen ingeroosterd (en binnen dezelfde week)
6. er mogen niet meer dan twee trainingen zijn die elkaar overlappen per dag per medewerker
7. een facilitator van een training die zelf ook trainingen volgt mag geen overlap hebben van zelf gekozen trainingen tijdens deze training
8. een training wordt pas ingepland bij minimaal acht aanmeldingen
9. een training wordt eventueel tweemaal aangeboden bij meer dan twintig aanmeldingen

Het uiteindelijke doel is om een planning te verkrijgen waarbij een medewerker zoveel mogelijk trainingen van zijn/haar wensenlijstje kan volgen en de kosten zo laag mogelijk zijn, wat gelijk staat aan het gebruik van zo min mogelijk zalen per dag.

3.2 De huidige situatie

De planning van de TechWeeks is afgelopen jaren handmatig uitgevoerd. Nadat de inschrijvingen op de trainingen waren ontvangen werd bepaald aan de hand van de hoeveelheid aanmeldingen (minder dan acht of meer dan twintig) of de desbetreffende training al dan niet gegeven zou worden of eventueel meerdere malen. Alle trainingen die gegeven zouden worden werden op ‘post-its’ geschreven en op een groot stuk wit muur werd net zo lang geschoven met de bekende gele papiertjes totdat er een redelijke planning ontstond. Deze planning werd vervolgens voorgelegd aan de facilitators en daar kwam dan weer feedback op. Omdat niet van te voren was nagegaan wat de beschikbaarheid was van de facilitators, was er telkens wel iets in de planning wat niet mogelijk was (bijvoorbeeld een facilitator op vakantie), waardoor er weer opnieuw geschoven moest worden voor een nieuwe oplossing. Deze werd dan weer teruggekoppeld, etc. Een erg tijdrovend proces, wat natuurlijk veel efficiënter moet kunnen.

3.3 Modelbeschrijving

Een (gemengd) geheeltallig programmerings model is opgesteld om een rooster te maken voor de trainingen. Het doel is het minimaliseren van het verschil van het verkregen rooster met een ‘perfect’ rooster, wat een rooster is wat voldoet aan alle wensen van de medewerkers en daarbij alle voorwaarden. Het doel van het model is dus ook het minimaliseren van de overlap van de gekozen trainingen door een medewerker op eenzelfde dag en het minimaliseren van het aantal bezette zalen per dag (kosten).
Het model is opgesteld met het gebruik van de volgende definities:
De verzameling van medewerkers, weergegeven door het personeelsnummer:
$i \in I, \quad I = \{31000381, 31003285, etc\}$
De verzameling van trainingen, weergegeven door het trainingsnummer:
$j \in J, \quad J = \{ERS002, ERS005, etc\}$

De verzameling van dagen, weergegeven door de afkorting (Engels) + weeknummer:
$k \in K,\ K = \{Mo1, Tu1, etc\}$

a_j = de lengte van training *j* in dagen

b_k = de beschikbare zalen op dag *k*

w_{ij} = de wens van medewerker *i* om training *j* te volgen $\begin{cases} 1 & training\ gekozen \\ 0 & training\ niet\ gekozen \end{cases}$

v_{jk} = de mogelijkheid om training *j* op dag *k* te geven $\begin{cases} 1 & trainingsdag\ mogelijk \\ 0 & trainingsdag\ niet\ mogelijk \end{cases}$

Een trainingsdag is bijvoorbeeld mogelijk als de facilitator op deze dag beschikbaar is en de locatie (intern of extern) geschikt is om deze training te geven.

De beslissingsvariabelen:

$x_{jk} = \begin{cases} 1 & training\ j\ wordt\ ingepland\ op\ dag\ k \\ 0 & anders \end{cases}$

z_{ik} = aantal trainingen die ingepland zijn op dag *k* voor medewerking *i*

De verschillende voorwaarden, de 'harde' voorwaarden waar aan voldaan moet zijn, kunnen dan opgesteld worden als:

1. check dat de lengte van de trainingen kloppen:
   $\sum_k x_{jk} = a_j \qquad \forall\ j \in J, k \in K$

2. check dat per dag niet meer dan het maximale aantal beschikbare zalen bezet zijn:
   $\sum_j x_{jk} \le b_k \qquad \forall\ j \in J, k \in K$

3. check dat de trainingen op de juiste locatie worden ingepland:
   $\sum_k v_{jk} x_{jk} = a_j \qquad \forall\ j \in J, k \in K$

4. check de beschikbaarheid van de facilitator(s)
   Dit kan deels met dezelfde vergelijking als de vorige voorwaarde.
   In de tabel v_{jk} wordt, naast de voorwaarde voor juiste locatie, ook aangegeven per dag of de facilitators wel (1) of niet (0) beschikbaar zijn. Hiermee wordt gecheckt of een facilitator niet met vakantie is bijvoorbeeld. Hiernaast moet ook nog gecheckt worden of trainingen die door dezelfde facilitator worden gegeven (zoals voor ERS005 en ERS018) niet op dezelfde dag worden ingepland. Hier worden extra checks voor ingebouwd:
   $x_{j^*k} + x_{j^*k} = 1 \qquad \forall\ j^* \in \{(ERS005, ERS018),\ (ERS023, ERS025),\ etc\},\ \ k \in K$

5. check of trainingen van twee dagen achter elkaar zijn gepland en in dezelfde week.
   Alle trainingen zijn in principe één of twee dagen lang. Voor alle trainingen geldt dus dat op combinaties van bepaalde dagen trainingen niet ingepland kunnen worden, zoals op de vrijdag in de eerste week en op maandag in de tweede week. Het checken of trainingen van twee

dagen op achtereenvolgende dagen gepland zijn is hetzelfde als nagaan of voor bepaalde combinaties van dagen, niet op beide dagen een training is ingepland:

$x_{jk^*} + x_{jk^*} = 1 \qquad \forall\, j \in J \quad k^* \in \{(Fr1, Mo2), (Mo1, We1), (Mo1, Th1), etc\}$

6. check dat er niet meer dan twee overlappende trainingen per dag per medewerker zijn:

$\sum_j w_{ij} x_{jk} \leq 2 \qquad \forall\, i \in I, j \in J, k \in K$

7. check of facilitators ook zelf trainingen volgen en geen overlap hebben.
Over het algemeen volgen facilitators zelf geen trainingen. Voor de veertien facilitators (veertien nummers van de desbetreffende medewerkers *i*) waar dit wel het geval is, is de volgende check ingebouwd:

$\sum_j w_{ij} x_{jk} \leq 1 \qquad \forall\, i \in \{31401834,...,31401726\}, j \in J, k \in K$

Bij het zien van de data bleken de overige voorwaarden niet nodig te zijn om te modelleren:

8. check of er meer dan acht aanmeldingen zijn.
Alle trainingen met minder dan acht aanmeldingen waren al uit de data gefilterd. Deze check is dus niet nodig om in te bouwen in het programma want zal nooit nodig zijn. Wanneer dit wel het geval zou zijn, zou het als volgt kunnen worden gecheckt.
Deze voorwaarde is een 'als-dan' voorwaarde:
als er meer dan acht aanmeldingen zijn: $\sum_i w_{ij} \geq 8 \qquad \forall\, j \in J, i \in I$,
dan mag de training ingepland worden: $\sum_k x_{jk} = a_j \qquad \forall\, j \in J, i \in I$.
Dit kan in een IP-model met de volgende truc worden gemodelleerd, voer hiertoe een 0-1 variabele δ met:

$$\delta = \begin{cases} 1 & als \sum_i w_{ij} \geq 8 \\ 0 & als \sum_i w_{ij} \leq 7 \end{cases} \quad \text{en} \quad \sum_k x_{jk} = \begin{cases} a_j & als\ \delta = 1 \\ 0 & als\ \delta = 0 \end{cases}$$

De volgende lineaire restricties zorgen ervoor dat aan deze eisen is voldaan:

$\sum_i w_{ij} \leq 7 + M\delta \qquad \forall\, j \in J, i \in I$

$\sum_i w_{ij} \geq 8\delta \qquad \forall\, j \in J, i \in I$

$\sum_k x_{jk} \leq \sum_i w \qquad \forall\, j \in J, i \in I, k \in K$

$\sum_k x_{jk} \geq \sum_i w_{ij} - M(1-\delta) \qquad \forall\, j \in J, i \in I, k \in K$

$\sum_k x_{jk} \leq M\delta \qquad \forall\, j \in J, k \in K$

$\delta \in \{0,1\}, \qquad \sum_k x_{jk} \geq 0$

met *M* een voldoende groot getal zodat altijd geldt dat $\sum_k x_{jk} \leq M$.

9. check of er meer dan twintig aanmeldingen zijn.
Wanneer er een training is met meer dan twintig aanmeldingen wordt deze vooraf gesplitst in twee trainingen om de controle te houden wie er in de twee groepen zitten. In de data zullen daarom ook geen trainingen met meer dan twintig aanmeldingen zijn. Deze check is dus niet

nodig om in te bouwen in het programma. Maar mocht deze check wel nodig zijn, dan kan dit ook gezien worden als een 'als-dan' voorwaarde, maar dan iets anders als de vorige check:
als training *j* ingepland is: $\sum_k x_{jk} > 0 \quad \forall j \in J, k \in K$,
dan zijn er minder dan 20 aanmeldingen: $\sum_i w_{ij} \leq 20 \quad \forall j \in J, i \in I$.
Dit kan in een IP-model met de volgende truc worden gemodelleerd. Voer een 0-1 variabele δ in en voeg de volgende lineaire restricties toe:

$$\sum_k x_{jk} \leq M\delta \qquad \forall j \in J, k \in K$$

$$\sum_i w_{ij} \leq 20 + M(1-\delta) \qquad \forall j \in J, i \in I$$

$$\delta \in \{0,1\}, \qquad \sum_k x_{jk} \geq 0$$

met M een voldoende groot getal zodat altijd geldt dat $\sum_k x_{jk} \leq M$ en $\sum_i w_{ij} \leq 20 + M$.

Doelfunctie:
De doelfunctie bevat de 'zachte' voorwaarden. Dit zijn twee voorwaarden: het minimaliseren van de overlap van trainingen ingepland op dezelfde dag op het wensenlijstje van de medewerker en het minimaliseren van de kosten aan zaalhuur, wat gelijk staat aan het aantal zalen per dag.
Het aantal trainingen dat ingepland is op dag k voor medewerking i kan verkregen worden met de volgende formule:

$$z_{ik} = \sum_j w_{ij} x_{jk} \qquad \forall i \in I, j \in J, k \in K$$

Wanneer $z_{ik} \geq 2$ is er sprake van overlap. Je wilt z_{ik} dus minimaliseren.
Het aantal zalen wat wordt gebruikt per dag k is gelijk aan:

$$\sum_j x_{jk} \qquad \forall j \in J, k \in K.$$

Dit wil je ook minimaliseren.

Omdat je de beide zachte voorwaarden wilt minimaliseren kan de doelfunctie (o = objective function) dan geschreven worden als een combinatie van beide:

$$\text{Min} \quad o = \sum_{ik} \left(\sum_j (w_{ij} x_{jk}) + \sum_j x_{jk} \right) \qquad \forall i \in I, j \in J, k \in K$$

3.4 Programma

Dit model heb ik geïmplementeerd in GAMS. GAMS is een pakket waarmee mathematische programmeringsmodellen kunnen worden opgelost, waaronder IP-problemen. GAMS bestaat uit een compiler en verschillende solvers. Voor dit model gebruik ik als solver *mip,* dit is de solver die gebruikt kan worden om (gemengde) geheeltallige problemen (MIP) op te lossen.
De code van het programma is in bijlage A toegevoegd.

3.5 Het verkrijgen van data

Alle medewerkers van Deloitte zijn via de mail gevraagd aan de hand van het huidige curriculum hun wensenlijstje voor de te volgen trainingen kenbaar te maken. Deze gegevens worden op een centraal

punt verzameld en zijn gemakkelijk te verkrijgen. De lengte per training, was afgezien van een enkele training, hetzelfde gebleven met vorig jaar. De beschikbare zalen in de eerste week (Amstelveen) en de tweede week (Noordwijkerhout) is beide gelijk aan vijf.
Het verkrijgen van de data betreffende de beschikbaarheid van de facilitators en de locatierestricties van de trainingen was een ander verhaal. De locatierestricties per training zijn nooit verzameld en vastgelegd. De lijst van trainingen met bijbehorende facilitators was flink verouderd, ongeveer de helft van de facilitators was alweer uit dienst. En vervolgens is de beschikbaarheid van de facilitators elk jaar anders, omdat dit afhangt van geplande projecten en/of vakantie. Dit was een tijdrovende klus om hier een overzicht van te krijgen omdat veel mensen persoonlijk benaderd moesten worden en er sprake was van veel tegenstrijdige en verouderde informatie.

3.6 Resultaten

Omdat waarschijnlijk niet aan alle voorwaarden voldaan kan worden zijn de voorwaarden in 'harde' en 'zachte' voorwaarden opgesplitst. Aan de harde voorwaarden moet voldaan worden (de constraints) en de zachte voorwaarden zijn de niet onoverkomelijke voorwaarden, en willen we in dit geval minimaliseren. De twee zachte voorwaarden zijn: zo weinig mogelijk overlap voor de medewerkers en het gebruik van zo min mogelijk kosten (zalen). Deze moeten beiden geminimaliseerd worden en komen daarom samen in de doelfunctie van het programma (zie sectie 3.3). De harde voorwaarden (beschikbaarheid facilitators, locatierestricties en overlap facilitators en eigen trainingen) komen terug in de constraints.
Wanneer we de verkregen data van de wensenlijstjes van de medewerkers vooraf bestuderen, kan er bijvoorbeeld bekeken worden hoeveel trainingsdagen per medewerker zijn 'gekozen' met de volgende formule:

$$\sum_j w_{ij} * a_j = r_i \ , \quad \forall i \in I,\ j \in J \ ,$$

waarbij r_i staat voor het aantal trainingsdagen voor medewerker i.
Dit blijkt al voor een aantal medewerkers boven de tien te liggen, terwijl er in totaal maar tien trainingsdagen zijn om in te roosteren (twee TechWeeks). Door de grote keuze van deze medewerkers zal er al noodzakelijk (extra) overlap zijn.

Er zijn 31 trainingen die ingeroosterd moeten worden, waarvan één training (ERS023) twee keer omdat deze meer dan twintig aanmeldingen had, namelijk 32 aanmeldingen. De medewerkers zijn verdeeld zodat twee trainingen met 16 deelnemers zijn gevormd (ERS123 en ERS223). Een extra check is toegevoegd dat deze twee trainingen niet op dezelfde dag gegeven kunnen worden.
Wanneer we met de lengte van de verschillende trainingen rekening houden komen we op in totaal 40 trainingsdagen die ingeroosterd moeten worden. Er is beschikbaarheid over tien dagen (twee werkweken) met elke dag vijf zalen, dat komt op 50 trainingsdagen en betekent dat er in principe plek is voor het inroosteren van alle trainingen.

Er zijn 121 werknemers die hun wensenlijstje hebben aangeleverd, waarvan 14 werknemers ook zelf één of meerdere trainingen faciliteren. Aan het wensenlijstje van deze 14 medewerkers wordt ook de training toegevoegd die ze zelf faciliteren en de checks worden toegevoegd dat deze medewerkers geen overlap in trainingen zullen hebben.

Na het invoeren van de data laat de output van GAMS laat het volgende zien:

Output GAMS

	Mo1	Tu1	We1	Th1	Fr1	Mo2	Tu2	We2	Th2	Fr2
ERS002	1.000									
ERS003						0.604			0.396	
ERS005			0.528		0.472					
ERS006		0.104	0.132		0.632				0.132	
ERS018				0.604	0.396					
ERS123	1.000									
ERS223		1.000								
ERS025						1.000				
ERS026							0.396			0.604
ERS028			0.472	0.396	0.132			0.472		0.528
ERS029		0.417		0.042			0.500	0.042		
ERS030		0.396		0.021	0.312			0.271		
ERS031			0.167	0.021	0.312		0.500	0.687		0.312
ERS032		0.083	0.833	0.917					0.083	0.083
ERS035						0.604		0.396		
ERS039				0.604	0.236	0.396	0.368			0.396
ERS041		0.500								0.5
ERS042		0.500	0.354	0.500		0.500				0.146
ERS057					0.687			0.271		0.042
ERS060			0.049						0.479	0.472
ERS064					0.368		0.632			
ERS077		1.000								
ERS079			0.479	0.479			0.479	0.042	0.521	
ERS086								0.542		0.458
ERS087	0.312			0.062	0.687	0.312		0.312	0.312	
ERS095						0.208	0.528	0.528	0.264	0.472
ERS096		0.896					0.104			
ERS097			0.396				0.604			
ERS100	0.701	0.104	0.299				0.299	0.299	0.299	
ERS106						0.396			0.604	
ERS107	0.687				0.312					

Omdat het programma met een *mip* (mixed integer program) solver wordt opgelost, wordt er een ‘gemengde’ output (geheeltallig en niet geheeltallig) gegeven wanneer er geen toegestane (feasible) oplossing bestaat in het geheeltallige geval. Dit is vaak het geval met roosterproblemen vanwege het enorme aantal voorwaarden waaraan het rooster moet voldoen. De eis $x_j \in \{0,1\}$ wordt dan bijvoorbeeld verzwakt tot $0 \leq x_j \leq 1$om een toegestane oplossing te krijgen. De verkregen toegelaten oplossing vervolgens is vaak een goede sub-optimale oplossing. In de praktijk is men vaak met zo’n oplossing al heel erg tevreden.

In ons geval is de eis $x_{jk} \in \{0,1\}$ verzwakt tot $0 \leq x_{jk} \leq 1$om een toegestane oplossing te krijgen. Deze sub-optimale oplossing geeft een goed idee waar het 'beste' trainingen gepland kunnen worden (waarden zo dicht mogelijk bij 1). En daarom is dit schema een basis om vast te leggen op welke dag welke training wordt ingepland. Voor de meeste trainingen kunnen de dagen met de hoogst berekende waarden doorslaggevend zijn. Maar wanneer alle trainingen gepland worden op de dagen met de hoogste waarden (waarden het dichts bij 1), kan er gezien worden dat aan sommige harde voorwaarden niet wordt voldaan. Anders zou dit natuurlijk ook de optimale oplossing in het geheeltallige geval zijn. Dit geldt bijvoorbeeld voor training ERS029 en ERS030. Wanneer we de waarden van GAMS volgen zouden deze beiden op de eerste dinsdag (Tu1) worden gepland, maar deze cursus heeft dezelfde facilitator dus dat is niet mogelijk (harde voorwaarde). De 'tweede beste' oplossing is ERS030 te plannen op de eerste vrijdag (Fr1).

Om tot een geheeltallige oplossing te komen zo dicht mogelijk bij de sub-optimale gemengde oplossing uit de vorige tabel, zijn alle trainingen op de volgende manier langsgelopen:
- Voldoet planning van training op dag met hoogste waarde aan alle harde constraints?
  → Ja? Plannen.
  → Nee? Voldoet planning op dag met tweede hoogste waarde aan alle harde constraints?
  → enz.

Na alle trainingen te zijn langsgelopen komen we tot de volgende geheeltallige oplossing:

Planning Techweek 2 & 3, 2008

	23jun	24jun	25jun	26jun	27jun	25aug	26aug	27aug	28aug	29aug
	Mo1	**Tu1**	**We1**	**Th1**	**Fr1**	**Mo2**	**Tu2**	**We2**	**Th2**	**Fr2**
ERS002										
ERS003										
ERS005										
ERS006										
ERS018										
ERS123										
ERS223										
ERS025										
ERS026										
ERS028										
ERS029										
ERS030										
ERS031										
ERS032										
ERS035										
ERS039										
ERS041										
ERS042										
ERS057										
ERS060										
ERS064										
ERS077										
ERS079										
ERS086										
ERS087										
ERS095										
ERS096										
ERS097										
ERS100										
ERS106										
ERS107										

Deze planning voldoet aan alle harde voorwaarden en aan zo min mogelijk overlap van gewenste trainingen voor de medewerkers (zachte voorwaarde). Toch is er voor een aantal medewerkers overlap aanwezig. Voor de medewerkers die te maken hebben met overlap van gekozen trainingen op bepaalde dagen is eerst bekeken wat de waarde van desbetreffende trainingen is (‘Mandatory’, ‘Recommended’ of ‘Value Added’) voor de desbetreffende medewerker. De keuze tussen een ‘Mandatory’ training en een ‘Recommended’ training is gemakkelijk gemaakt. Bij gelijke waarde hangt de keuze van de training van de groepsgrootte van beide trainingen af zodat een ingeplande training niet uiteindelijk minder dan acht deelnemers heeft.

Deze planning heeft uiteindelijk 42 overlappingen van de 490 aanmeldingen voor trainingen.

3.7 'Bottlenecks'

Het rooster in deze case-studie moet aan bepaalde voorwaarden voldoen, daarvoor worden het ook 'hard constraints' genoemd. Maar het is wel interessant om te bekijken welke harde voorwaarden nou de grootste boosdoeners, oftewel 'bottlenecks', zijn bij het minimaliseren van de doelfunctie.

Het optimale rooster is gebruikt om te 'spelen' met de verschillende voorwaarden, maar ook met de input. Door verschillende harde voorwaarden 'uit' te zetten of handmatig wat aanpassingen te doen in de input data (bijvoorbeeld in de tabel v_{jk} met de beschikbaarheid van de facilitators en locatierestricties) kan gekeken worden naar de invloed van bepaalde voorwaarden of inputdata op de optimale oplossing. Wat zijn de grootste boosdoeners?

Welk wensenlijstje is de grootste boosdoener?
Er is één persoon die zeven trainingen op zijn wensenlijstje heeft gezet, wat in totaal uitkomt op een vraag naar twaalf trainingsdagen, terwijl er maar tien zijn. Dat zijn al minimaal twee noodzakelijke overlappingen. Het moet wel gezegd worden dat deze medewerker ook wel een beetje pech heeft met de uiteindelijke planning, maar zijn wensenlijstje zorgt uiteindelijk voor zes overlappingen.

Welke facilitator is de grootste boosdoener?
Er is één facilitator die vier verschillende trainingen verzorgd, twee trainingen van één dag en twee trainingen van twee dagen, wat een totaal maakt van zes trainingsdagen. Deze facilitator is in totaal acht dagen beschikbaar, waar deze zes dagen training zonder overlap op ingepland moeten worden. In totaal zijn er twaalf overlappingen tijdens de trainingen van deze facilitator. Wanneer deze facilitator bijvoorbeeld twee van de vier trainingen aan iemand anders zou overdragen, zou dit al zo zes overlappingen kunnen schelen.

Welke training is de grootste boosdoener?
Training ERS023 kan alleen intern worden gegeven want beschikbaarheid over het intranet van Deloitte is noodzakelijk. Deze training moet dus in de eerste week worden ingepland. Verder moet deze training twee keer aangeboden worden, omdat er zoveel aanmeldingen voor ERS023 zijn binnengekomen. In combinatie met de beschikbaarheid van de facilitators in de eerste week kan deze training alleen op maandag of dinsdag in week 1 worden gegeven. Deze training moet dus dan ook op deze dagen ingepland worden. Deze training is de grootste bottleneck, wanneer voorwaarden niet zouden gelden, scheelt dat vijf overlappingen.

Welke algemene voorwaarde is de grootste boosdoener?
Trainingen die twee dagen duren moeten achter elkaar worden gepland in dezelfde week. Wanneer deze harde voorwaarde al wordt afgezwakt naar: plannen binnen dezelfde week, maar hoeft niet perse op achtereenvolgende dagen, dan scheelt dit al zo vijf overlappingen.

4. Conclusie en aanbevelingen

Lineaire programmering is een populaire methode om veel praktische problemen op te lossen. Een verrassend grote klasse van deze praktische problemen kan als een geheeltallig programmeringsprobleem (IP probleem) geformuleerd worden, waarbij soms een vindingrijke formuleringsopzet vereist is. Dit is ook één van de nadelen van IP, omdat er extra trucs nodig zijn om bepaalde 'als-dan' voorwaarden aan het model toe te voegen.

Geheeltallige programmering is ook toepasbaar op het Student Scheduling Problem. Aan de hand van een case-studie bij Deloitte is laten zien dat met behulp van geheeltallige programmering een basis voor een oplossing gevonden kan worden voor een roosterprobleem dat te omschrijven is als een Student Scheduling Problem. Omdat de optimale oplossing niet toelaatbaar (feasible) is in het geheeltallige geval, kan de gemengde (geheeltallig én niet-geheeltallig) basisoplossing vervolgens gebruikt worden om een uiteindelijk rooster te maken. Verschillende bottlenecks zijn besproken die de grootste invloed hebben op de uiteindelijke oplossing.

Het automatiseren van het roosterprobleem bij Deloitte verdient de voorkeur boven het handmatig zoeken naar een oplossing. Kijkend naar de bottlenecks zouden de aanbevelingen voor Deloitte (voor volgend jaar) kunnen zijn:

1. een cursus met locatierestricties moet gefaciliteerd worden door een facilitator die maximaal beschikbaar is
2. cursussen moeten door zoveel mogelijk verschillende mensen gefaciliteerd worden, bijvoorbeeld een maximum van twee trainingen met dezelfde facilitator
3. er moet een maximum zijn aan trainingsdagen die mogen worden gekozen door de medewerkers.

Literatuurlijst

[1] A. Schearf. *A survey of automated timetabling.* Artificial Intelligence Review 13(2), pag. 87-127, 1999.

[2] S. Even, A. Itai en A. Shamir. *On the complexity of timetable and multicommodity flow problems.* SIAM Journal of Computing 5, pag. 691-703, 1976.

[3] E. Cheng, S. Kruk en M. Lipman. *Flow formulations for the student scheduling problem.* In Practice and Theory of Automated Timetabling IV, Burke and De Causmacker, Lecture Notes in Computer Science, Vol. 2740 Springer-Verslag, pag. 299-309, 2003.

[4] J. van den Broek, C. Hurkens en G. Woeginger. *Timetabling Problems at the TU Eindhoven.* E.K. Burke, H. Rudová (Eds.): PATAT 2006, pag.141-156.

[5] M.W. Carter. *A comprehensive course timetabling and student scheduling system at the university of Waterloo.* In Practice and Theory of Automated Timetabling III, Burke and Erben, Lecture Notes in Computer Science, Vol. 2079 Springer-Verslag, pag. 64-84, 2003.

[6] M.L. Hall. *Optimal scheduling of army initial entry training courses.* Thesis, Naval Postgraduate School, Monterey (California), 1999.

[7] X. Qi, J.F. Bard en G. Yu. *Class scheduling for pilot training.* Operations Research, Vol.52, No.1, pag.148-162, 2004.

[8] G. Laporte en S. Desrochers. *The problem of assigning students to course sections in a large engineering school.* Computational Operations Research 13, pag. 387-394, 1986.

[9] A. Tripathy. *Computerised decision aid for timetabling – a case analysis.* Discrete Applied Mathematics 35(3), 313-323, 1992.

[10] I. Miyaji, K. Ohno en H. Mine. *Solution method for partitioning students into groups.* European Journal of Operations Research Society 37, pag. 689-693, 1981.

[11] H. Tijms. *Operationele Analyse, een inleiding in modellen en methoden.* Elipson Uitgaven Utrecht, 2002.

Appendix A – code GAMS

Sets

i persons / 31000140, 31000381, 31001514, …., 31406416, 00003590 /

j courses / ERS002, ERS003, ERS005, …..,, ERS106, ERS107 /

k days / Mo1, Tu1, We1, Th1, Fr1, Mo2, Tu2, We2, Th2, Fr2 /

Parameters

a(j) duration of course j in days

/ ERS002 1

ERS003 1 etc. /

b(k) available room on day k

/ Mo1 5

Tu1 5 etc. /

Table w(i,j) wishes on courses of persons

ERS002 ERS003 ERS005 ERS006 ERS018 ERS123 ERS223 ERS025 ERS026 ERS028 ERS029 ERS030 ERS031 ERS032 ERS035

31000140 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0

31000381 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 etc.

+ ERS039 ERS041 ERS042 ERS057 ERS060 ERS064 ERS077 ERS079 ERS086 ERS087 ERS095 ERS096 ERS097 ERS100 ERS106 ERS107

31000140 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

31000381 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
etc.

Table v(j,k) preferences of courses on days

Mo1 Tu1 We1 Th1 Fr1 Mo2 Tu2 We2 Th2 Fr2

ERS002 1 0 1 0 0 0 0 0 0 0

ERS003 0 0 0 0 0 1 1 1 1 1 etc.

Variables

x(j,k) planning of courses on daysp

o number of overlapped courses for persons

r(i) number of total coursedays of prefered courses of person i ;

```
Binary Variable x ;

Positive Variable r ;

Equations

        overlap       define objective function

        rooms(k)        observe class room limit at day k

        length(j)       check length of course j

        facili(j)      check preferences facilitators

        courseD(i)     number of prefered course days of person i

        check51(j)

        check52(j)              etc. t/m check88(j)

        checkf1(i,k)

        checkf2(i,k)            etc. t/m checkf14(i,k)

        check90(k)

        check92(k)              etc t/m check97(k) ;

overlap ..       o =e= sum((i,k), (sum((j),x(j,k))+sum((j), (w(i,j)*x(j,k))))) ;

facili(j) ..    sum(k, v(j,k)*x(j,k)) =e= a(j) ;

courseD(i) ..   sum(j, w(i,j)*a(j)) =e= r(i) ;

length(j) ..    sum(k, x(j,k)) =l= a(j) ;

rooms(k) ..     sum(j, x(j,k)) =l= b(k) ;

check51(j) ..   sum(k$(ord(k) EQ 5), x(j,k))+sum(k$(ord(k) EQ 6), x(j,k)) =l= 1 ;

check52(j) ..   sum(k$(ord(k) EQ 1), x(j,k))+sum(k$(ord(k) EQ 3), x(j,k)) =l= 1 ;etc. t/m check88(j)

checkf1(i,k)$(ord(i) EQ 26) ..          sum(j,w(i,j)*x(j,k)) =l= 1;

checkf2(i,k)$(ord(i) EQ 7) ..           sum(j,w(i,j)*x(j,k)) =l= 1;                          etc. t/m checkf14(i,k)

check90(k) ..     sum(j$(ord(j) EQ 8), x(j,k))+sum(j$(ord(j) EQ 24), x(j,k)) =l= 1;

check91(k) ..     sum(j$(ord(j) EQ 9), x(j,k))+sum(j$(ord(j) EQ 28), x(j,k))  =l= 1;             etc. t/m check97(k)

Model planning /all/ ;

Solve planning using mip minimizing o ;

Display x.l, x.m;
```