Klantgericht roosteren.

Gebruik van CPsolver binnen Rostar Eduflex

Simone van Balen
Denise van Brenk
Camiel Egbers

Begeleiders UT: Gerhard Post, Johann Hürink
Begeleiders Paralax: Paul Coldenhoff, Jelle de Vries

UNIVERSITEIT TWENTE.

Inhoudsopgave

1 Inleiding

Deze bacheloropdracht is afkomstig van het bedrijf Paralax, een aanbieder in software oplossingen en dienstverlening op het gebied van personeelsplanning, personeelsroosters en workforce management. Een van de producten die verkocht wordt, is een sofware pakket voor het maken van schoolroosters, Rostar EduFlex. Dit is een programma voor roosterplanning dat gemaakt is om zowel complexe als eenvoudige roosterproblemen automatisch op te lossen en het is daarom geschikt voor zowel grote als kleine onderwijsinstellingen.

Om het maken van een rooster zoveel mogelijk automatisch te kunnen doen, wordt binnen deze software gebruik gemaakt van een solver. Door hier gebruik van te maken kan, mits alle input goed ingegeven wordt, geheel automatisch een rooster gegenereerd worden. Hierbij kunnen tevens wensen meegegeven worden waaraan een dergelijk rooster moet voldoen. Rostar EduFlex, voortaan Eduflex genoemd, maakt gebruik van een combinatie van twee eigen ontworpen solvers. De eerste, de clusterautomaat, deelt alle leerlingen in geschikte klassen in. Vervolgens wordt met de tweede, de roosterautomaat, het rooster voor deze eerder gegroepeerde klassen gemaakt.

Op veel onderwijsinstellingen is het mogelijk voor leerlingen om tussen verschillende vakken en/of modules te kiezen. Dit heeft tot gevolg dat de keuze om bepaalde klassen te maken veel invloed kan hebben op de kwaliteit van het gegenereerde rooster. Om de roosterkwaliteit te kunnen verbeteren heeft Paralax de keuze gemaakt om een overstap te overwegen naar een andere solver, CPsolver genaamd. Deze veelbelovende solver is mede ontworpen voor de International Timetambling Competition 2007 door Tomáš Müller. Binnen deze competitie zijn drie verschillende categoriëen: tentamens inroosteren, curriculum gebaseerd roosteren en modulair roosteren. Bij de eerste twee categorieën heeft CPsolver een eerste plaats behaald, bij de laatste een vijfde plaats. Dit betekent dat CPsolver de functies van de twee huidige solvers, clusteren en roosteren, zou kunnen overnemen. Nu de solver openbaar gemaakt is, heeft Paralax vanwege de veelbelovende resultaten besloten de implementatie van CPsolver in Eduflex te overwegen.

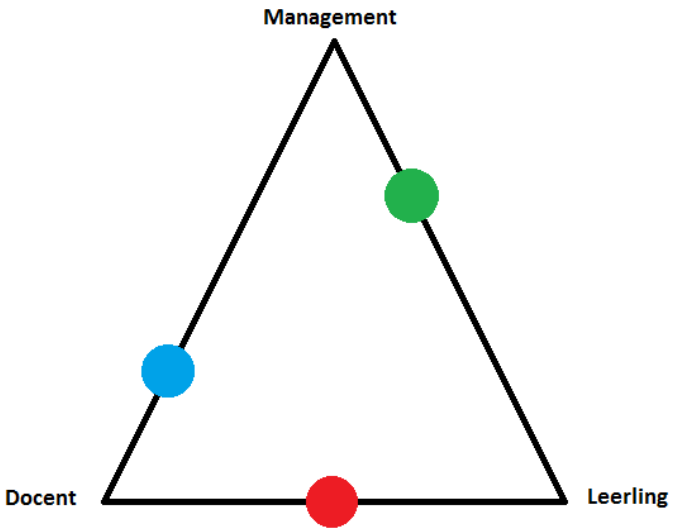
Een groot probleem dat bij deze implementatie om de hoek komt kijken, zijn de instellingen van CPsolver. De solver maakt namelijk gebruik van veel parameters, waarvan vooralsnog onbekend is hoe ze ingesteld moeten worden voor de klanten die gebruik maken van Eduflex. Het hoofddoel van dit onderzoek is dan ook om uit te zoeken hoe CPsolver het best kan worden ingesteld, afhankelijk van de eisen van de klant. Om tot een antwoord te komen, zal er allereerst onderzoek verricht moeten worden naar de eisen van de klant. Hiervoor zal uitgegaan worden van drie verschillende categorieën; de leerling, de docent en het management. Vervolgens wordt er gekeken naar de opzet van roosterproblemen in het algemeen, de werking van Eduflex en de werking van CPsolver. Ook de koppeling hiertussen zal worden besproken, waarbij tevens aanbevelingen gedaan zullen worden voor de implementatie van CPsolver in Eduflex.

Met deze informatie, die voornamelijk verkregen zal worden uit literatuuronderzoek, worden de parameters van CPsolver gefilterd en gegroepeerd. Hierna zullen de overgebleven parameters verder onderzocht worden. Om dit te automatiseren is er een programma geschreven die automatisch de parameters veranderd en onderzoekt. De hierbij verkregen roosters zullen worden vergeleken door middel van een doelfunctiewaarde, welke de wensen van de klant weergeeft. Met de gevonden resultaten uit het onderzoek kunnen vervolgens conclusies worden getrokken over de instellingen van CPsolver.

2 Probleemstelling

De vraag vanuit Paralax is om onderzoek te doen naar juiste instellingen van CPsolver voor verschillende voorkeuren van de klant. Om dit op een goede manier uit te kunnen voeren is gebruik gemaakt van enkele onderzoeksvragen en een onderzoeksdoel, deze zullen hieronder puntsgewijs worden toegelicht.

Het vooraf opgestelde doel van dit onderzoek is om juiste instellingen van CPsolver te kunnen vinden aan de hand van de eisen van de klant. Hiervoor krijgt de klant de keuze uit drie verschillende categorieën bij het roosteren; de leerling, de docent en het management. Het idee hierbij is dat de klant kan aangeven in hoeverre het belangrijk is dat een rooster voldoet aan eisen horend bij een bepaald categorie. Aan de hand van deze voorkeuren zouden dan de instellingen voor CPsolver gegenereerd moeten worden. Dit zou bijvoorbeeld geïmplementeerd kunnen worden aan de hand van schuifjes, waarmee wordt aangegeven wat belangrijk is. Dit idee is vanuit Paralax ontstaan aan de hand van [Tri12]. Wordt het schuifje precies tussen docent en leerling geplaatst, dan wordt bijvoorbeeld een tussenuur voor een docent als even kwalijk beschouwd als voor een leerling. In figuur 1 wordt door de klant als voorbeeld een kwalitatief rooster voor de docent en leerling even belangrijk gevonden. Het management wordt belangrijker geacht dan de leerlingen, terwijl de docenten hier weer boven worden gesteld. Het gaat hier dus niet om een transitiviteit van de instellingen, oftewel het derde schuifje is onafhankelijk van de andere twee schuifjes. Bij dit schuifjessysteem is meegenomen dat het geen toegevoegde waarde heeft om alle categorieën maximaal belang toe te kennen, aangezien dit uiteindelijk hetzelfde zou betekenen als weinig belang aan alle categorieën toekennen. Het gaat immers om de afweging tussen verschillende categorieën; een absoluut belang toegekend aan een bepaald categorie is zonder de andere toegekende belangen betekenisloos.



Figuur 1: Voorbeeld van schuifjes voor het doorgeven van de voorkeuren van de klant

Om vervolgens van deze doorgegeven voorkeuren een goede instelling van CPsolver te kunnen vinden, is het van belang om eerst enkele onderzoeksvragen beantwoord te hebben. De hoofdvraag die hierbij centraal staat is:

Hoe beïnvloeden de instellingen van de verschillende parameters van CPsolver de kwaliteit van de output van Eduflex?

Bij het nader bekijken van deze hoofdvraag komen een aantal aspecten naar boven waarover meer informatie gewenst is. Er is allereerst meer informatie nodig over wat kwaliteit is; wanneer is een roos-

ter beter dan een ander rooster? Hiernaast zijn de werking van zowel CPsolver als Eduflex een belangrijk aspect, alsmede de samenwerking tussen deze twee componenten. Tenslotte zijn de instellingen van de parameters van groot belang, dit is uiteindelijk hetgeen dat getracht wordt te verbeteren. Al deze vragen worden stapsgewijs onderzocht volgens de volgende deelvragen, waarbij aangegeven is in welk hoofdstuk de vraag behandeld wordt.

- Hoe kan een rooster gewaardeerd worden op kwaliteit? *(Hfst. 3)*
- Hoe wordt een roosterprobleem wiskundig geformuleerd? *(Hfst. 4)*
- Hoe werkt CPsolver? *(Hfst. 5)*
  - Hoe werkt het algoritme van CPsolver?
  - Wat is de in- en output van CPsolver?
  - Hoe werken de instellingen van CPsolver? Hierbij gelet op de algemene werking van de parameters alsmede de individuele invloed van een parameter op de output.
- Hoe werkt Eduflex? *(Hfst. 5)*
  - Waar wordt Eduflex voor gebruikt?
  - Wat zijn de instelmogelijkheden?
  - Hoe is CPsolver geïmplementeerd binnen Eduflex?

Aan de hand van de verkregen kennis zal vervolgens een onderzoeksplan opgesteld worden voor een onderzoek naar de invloed van verschillende parameterwaarden op de output, dit zal gebeuren in hoofdstuk 6. Allereerst is hiervoor een filtering van de parameters nodig, zoals te lezen in hoofdstuk 5. Vervolgens zullen de parameters die invloed hebben op de output onderzocht worden middels een programma, geschreven in C#. Meer over dit programma is te lezen in paragraaf 6.4. De bevindingen van het parameteronderzoek zullen besproken worden in hoofdstuk 7. Uiteindelijk worden de conclusies, de aanbevelingen voor Paralax en de suggesties voor vervolgonderzoek besproken in hoofdstuk 8.

3 Roosterkwaliteit

Bij het maken van een rooster, of dit nu met de hand of automatisch gebeurt door bijvoorbeeld CPsolver, is het van belang om het rooster te kunnen beoordelen op kwaliteit. Hiervoor is het goed om te weten hoe het er op MBO's in Nederland aan toe gaat en wat de roosterwensen bij verschillende onderwijsinstellingen zijn.

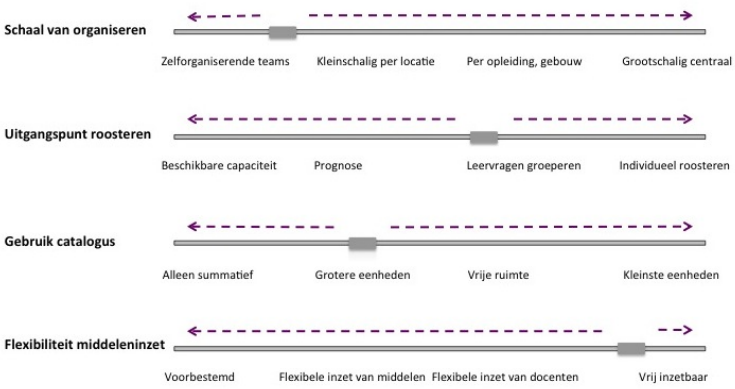
Het onderwijs op ROC's wordt vaak gegeven volgens modulair onderwijs. Hierbij staat de leervraag van de student centraal. De student krijgt voorafgaand aan een onderwijsperiode een keuze tussen verscheidene modules, waarna op basis van alle gemaakte keuzes door studenten een goed rooster moet worden gevormd. Hierbij wordt getracht zoveel mogelijk aan de keuzes van de studenten tegemoet te komen.

Er zijn verschillende meningen over hoe de roosters nu daadwerkelijk het beste te beoordelen zijn. Triple A en Paralax hebben hier beide onderzoek naar gedaan. In het vervolg zullen de uitkomsten van deze onderzoeken over de roosterkwaliteit kort ter sprake komen, waarbij de nadruk ligt op de verschillende factoren die mee worden genomen bij het beoordelen van een rooster.

3.1 Triple A

Een netwerk van ROC's en AOC's werkt aan een gezamenlijke visie op onderwijsvernieuwing. Onder deze onderwijsvernieuwing valt bijvoorbeeld het bovengenoemde modulair onderwijs, dat steeds intensiever is ingevoerd in de afgelopen jaren. Hiernaast komt er meer en meer vraag naar betaalbaar en toch goed georganiseerd onderwijs. De visie op onderwijs heeft zich inmiddels steeds meer ontwikkeld en functionele ontwerpen zijn onder de vlag van saMBO-ICT uitgewerkt en geactualiseerd.

In één van de functionele ontwerpen van saMBO-ICT, onder de naam Triple A [Tri12], worden de visie en uitgangspunten van verschillende MBO scholen uitgediept. Het geeft daarmee een richtlijn waar ICT systemen aan moeten voldoen om in te gaan op de wensen van de MBO scholen. De wensen worden aan de hand van vier scenario's gerangschikt. De scenario's omschrijven verschillende doelen: het individu staat centraal, er wordt gewerkt op teamniveau, het niveau van een opleiding of gebouw en een centraal opgesteld onderwijsaanbod. Om meer vat op de scenario's te krijgen, wordt er gekeken naar de belangrijkste onderwijslogistieke elementen, deze zijn in figuur 2 weergegeven. Het is voor een onderwijsinstelling gemakkelijker zichzelf te herkennen in de elementen dan in de scenario's. De ervaring dat scholen gemakkelijker hun voorkeuren door kunnen geven aan de hand van dergelijke schuifjes is dan ook gebruikt in dit onderzoek, zoals beschreven in hoofdstuk 2.



Figuur 2: Onderwijslogistieke elementen; aan de hand van de schuifjes kunnen onderwijsinstellingen doorgeven wat hun wensen zijn.

Hierbij staat de *schaal van organiseren* voor de mate waarin de onderwijsinstellingen het logistieke deel van het onderwijs wil organiseren. Het *uitgangspunt voor het roosteren* beschrijft de schaal waarop de

onderwijsinstelling wenst te roosteren. Hierbij wordt in feite het uitgangspunt voor het roosterproces gedefinieerd. Het derde schuifje, *gebruik van onderwijscatalogus*, geeft aan waarvoor de onderwijsinstelling de onderwijscatalogus wil gebruiken. Dit geeft aan in welke mate het onderwijsaanbod moet worden uitgewerkt. Tenslotte wordt het laatste schuifje, *flexibiliteit middeleninzet*, gebruikt om aan te geven op welke manier de onderwijsinstelling de middeleninzet in het logistieke proces wil flexibiliseren.

In combinatie met het onderzoek van Paralax zullen de elementen die hierboven genoemd zijn beoordeeld worden.

3.2 Onderzoek Paralax

Paralax heeft in samenwerking met ROC de Leijgraaf onderzoek gedaan naar de wensen van ROC's. In dit onderzoek stond vooral de vraag centraal welke indicatoren belangrijk zijn bij het beoordelen van een rooster. Dergelijke indicatoren worden *Key Perfomance Indicators* (KPI's) genoemd. In dit onderzoek is uitgegaan van vier verschillende doelen; ergonomie van de docent, ergonomie van de leerling, kwaliteit van het onderwijs en efficiënte bedrijfsvoering. Binnen deze doelen zijn dan KPI's gedefinieerd die tezamen een score kunnen geven aan een gemaakt rooster. Deze score wordt opgebouwd uit strafpunten, dus een lagere score staat voor een beter rooster. De KPI's die gevonden zijn aan de hand van dit onderzoek zijn weergegeven in tabel 1. Deze KPI's kunnen van belang zijn tijdens het onderzoek bij het waarderen van verschillende roosters.

De nadruk bij het onderzoek van Triple A ligt bij de elementen van het roosteren en de flexibiliteit van de middeleninzet. Hierbij wordt in het vervolg uitgegaan van individueel roosteren en vrij inzetbare middelen, aangezien de vraag vanuit Paralax met een dergelijke achtergrond gesteld is. Het idee van de schuifjes, die eerder in hoofdstuk 2 aan de orde kwamen, stamt af van het onderzoek van Triple A. De categorieën; leerling, docent en management, die verder in het onderzoek centraal komen te staan, zijn echter gebaseerd op het onderzoek van Paralax, gezien Paralax het beste zicht heeft op de wensen van hun klanten. Er is wel gekozen om de indicator kwaliteit van het onderwijs niet mee te nemen, omdat dit lastig te meten is aan het rooster.

Naam KPI	**Relevantie**	**Berekening van de beslissingsvariabele**
TheoriePraktijk-Balans	Meestal is het wenselijk om het aanbod van theorie- en praktijklessen rechtevenredig over een dag te verdelen	De gemiddelde afwijking van de verhouding tussen het aantal theorie lesuren per deelnemer per dag en het totale aantal lesuren per deelnemer per dag ten opzichte van de verhouding tussen theorie en praktijk lesuren van alle voor de leerling te plaatsen lessen
Deelnemer-LesurenPerDag	Op sommige onderwijsinstellingen is voor leerlingen is een minimaal en een maximaal aantal lesuren per dag bepaald	Het totale aantal uren dat een deelnemer per dag voor minder dan het minimum of meer dan het maximum aantal lesuren is ingeroosterd
DeelnemerDubbele-Roosteringen	Dubbele roosteringen (twee of meer te volgen lessen in hetzelfde tijdblok) zijn niet wenselijk vanuit deelnemer perspectief. Vanuit bedrijfsefficintie perspectief kan het echter aantrekkelijk zijn een beperkt aantal dubbel roosteringen toe te laten	De verhouding tussen het aantal dubbele roosteringen (twee of meer lessen geplaatst in zelfde lesuur) en het totale aantal geplaatste lessen
Deelnemer-Tussenuren	Tussenuren zijn sterk bepalend voor de ervaren kwaliteit van een rooster vanuit deelnemer perspectief	Het gemiddelde aantal tussenuren per dag over alle deelnemers
Docent-LesurenPerDag	Voor docenten is een maximaal aantal lesuren per dag bepaald	Het totaal aantal uren dat docenten per dag meer dan het maximum aantal lesuren zijn ingeroosterd
LokaalOverbezetting	Lokalen hebben een beperkte deelnemer capaciteit. In principe mag de maximum capaciteit van een lokaal niet worden overschreden, maar soms kan het vanuit efficiëntie overwegingen aantrekkelijk zijn om het toch toe te laten	De totale overschrijding van de maximum capaciteit van lokalen waarin lessen zijn geplaatst gedeeld door het totale aantal lessen waarin de maximum capaciteit wordt overschreden
DocentReistijd	De tijd benodigd voor verplaatsing tussen twee lokalen tussen aaneensluitende lesuren wordt bepaald. Het is wenselijk de totale tijd te minimaliseren	De totale reistijd voor een docent
DeelnemerReistijd	De tijd benodigd voor verplaatsing tussen twee lokalen tussen aaneensluitende lesuren wordt bepaald. Het is wenselijk de totale tijd te minimaliseren	De totale reistijd voor een deelnemer

Tabel 1: De KPI's uit het onderzoek van Paralax

4 Wiskundige probleemomschrijving

Nu de probleemstelling geformuleerd is en duidelijk is op wat voor soort aspecten een rooster beoordeeld kan worden, is het goed om te kijken naar de wiskundige formulering van roosterproblemen. Hiervoor zullen eerst de aspecten van het wiskundig model algemeen worden toegelicht, vervolgens zal het wiskundig model preciezer geformuleerd worden. Hierna zullen enkele voorbeelden van voorwaarden gegeven worden die extra ingevoerd kunnen worden in dergelijke wiskundige modellen en tenslotte wordt nog kort de complexiteit van het probleem beschreven.

4.1 Algemeen

Om uiteindelijk tot niet alleen een uitvoerbaar maar ook wenselijk rooster te komen, wordt gebruik gemaakt van hard en soft constraints in samenwerking met een doelfunctie. Er zal nu worden weergegeven hoe dit in zijn werk gaat, beginnend met tabel 2 waarin alle aspecten van het wiskundig probleem kort worden uitgelegd.

Doelfunctie	Functie waar voor elke mogelijke oplossing een waarde uitkomt, deze functie wordt tijdens het oplossen geprobeerd te minimaliseren. De functie bestaat uit gewogen soft constraints die overtreden worden en de doelfunctiewaarde geeft dus een indicatie van de kwaliteit van een rooster.
Beslissingsvariabelen	Keuzes die gemaakt worden, waardoor een rooster gevormd wordt. Een voorbeeld van zo'n beslissingsvariabele is x_{ik} uit het voorbeeld in 4.2. Deze variabele geeft aan of les i op tijdstip k wordt gegeven.
Hard constraints	Eisen waar de oplossing aan moet voldoen. Dit zijn restricties die opgelegd worden aan de beslissingsvariabelen. Denk hierbij aan het niet gelijktijdig inroosteren van twee lessen voor een persoon of lokaal. Het is niet mogelijk om beslissingen te nemen die voor een conflict zorgen doordat ze een hard constraint overtreden.
Soft constraints	Voorkeuren die men graag terugziet in het rooster. Mocht niet aan deze voorkeuren worden voldaan, worden er straffen toegekend aan de overschrijding. Denk hierbij aan de voorkeur om zo min mogelijk tussenuren voor de studenten te hebben. Het is mogelijk om deze soft constraint te schenden bij het nemen van beslissingen, echter stijgt hierdoor wel de waarde van de doelfunctie.

Tabel 2: Aspecten van wiskundig probleem

Om het systeem te verduidelijken, worden er in paragraaf 4.3 enkele voorbeelden gegeven van mogelijke hard en soft constraints. Maar allereerst wordt het verschil tussen een hard en soft constraint uitgelegd aan de hand van een voorbeeld. Stel er zijn twee scholen, deze hebben beide als doel om het aantal tussenuren van studenten zo laag mogelijk te houden. De uitvoering van deze wens is echter verschillend. School 1 heeft een maximum gezet op het aantal tussenuren per leerling; het maximum aantal tussenuren is vastgesteld op 5. Of een leerling nu 1 of 5 tussenuren in de week heeft maakt verder niet uit, zolang er maar geen enkele leerling is die het maximum overschrijdt. School 2 heeft de instelling dat tussenuren niet wenselijk zijn, maar dat het soms niet anders kan in een rooster. Ook zij vinden het niet erg als een leerling 5 of minder tussenuren in de week heeft, maar daarboven wordt het rooster wel echt als minder goed beschouwd.

Als we dit verschil in mening weergeven in het beschreven wiskundige model, zien we een verschil tussen de hard en soft constraints. School 1 wil echt niet dat er roosters komen waarbij het maximum

aantal tussenuren overschreden wordt, dit resulteert in een hard constraint. Als x_i staat voor het aantal tussenuren in de week van student i, wordt deze hard constraint als volgt:

$$x_i \leq x_{max} \quad \forall i.$$

In het model van school 2 wordt het aantal tussenuren per leerling in een soft constraint verwerkt. Deze soft constraint wordt meegenomen in de doelfunctie. Een rooster waarbij de wensen van de school genegeerd wordt, krijgt een hogere doelfunctiewaarde. Dit zou bijvoorbeeld als volgt gemodelleerd kunnen worden:

$$x_i \leq x_{max} + e_i \quad \forall i, \tag{1}$$
$$e_i \geq 0.$$

Vervolgens wordt in de doelfunctie de waarde van e_i meegenomen in combinatie met een gewicht.

4.2 Wiskundige formulering

Les	Een moment van samenkomen tussen leerlingen en een docent in een bepaald lokaal.
Vak	Een reeks van lessen met dezelfde leerlingen en dezelfde docent.
Modulen	Een pakket van vakken wat een leerling kan kiezen.

Tabel 3: Definities

Voordat het wiskundige model geformuleerd gaat worden, zullen er allereerst een aantal begrippen moeten worden gedefinieerd. Dit is gebeurt in tabel 3. Wanneer dan de leervraag van de student uit hoofdstuk 3 en de aspecten voor een wiskundig probleem worden gecombineerd, komt men tot een wiskundige formulering van het probleem [Sch99]. Om aan te sluiten op de leervraag wordt hierin gebruik gemaakt van modules; de leerlingen zitten dus niet in vaste groepen, maar worden zo gunstig mogelijk voor het rooster geclusterd. Het model ziet er als volgt uit:

Er zijn q verschillende vakken $K_1, ..., K_q$ en vak K_i bestaat uit k_i lessen, met $i = 1, \ldots, q$. Er zijn r modules $S_1, ..., S_r$, die groepen van vakken zijn die leerlingen gemeenschappelijk hebben. Dit betekend dat vakken in S_l op verschillende tijden gegeven moeten worden. Het aantal perioden is p. In dit geval wordt met een periode de lesuren bedoeld. Verder is l_k het maximale aantal lessen dat ingeroosterd kan worden in periode k (in andere woorden het maximale aantal lokalen dat beschikbaar is in een periode). Gebruikmakend van deze notatie, heeft het roosterprobleem de volgende vorm:

$$\min \sum_{i=1}^{q} \sum_{k=1}^{p} d_{ik} x_{ik} \tag{2}$$
$$s.t. \sum_{k=1}^{p} x_{ik} = k_i \qquad \forall i = 1, \ldots, q \tag{3}$$
$$\sum_{i=1}^{q} x_{ik} \leq l_k \qquad \forall k = 1, \ldots, p \tag{4}$$
$$\sum_{i \in S_l} x_{ik} \leq 1 \qquad \forall k = 1, \ldots, p \quad \forall l = 1, \ldots, r \tag{5}$$
$$x_{ik} \in \{0, 1\} \qquad \forall i = 1, \ldots, q \quad \forall k = 1, \ldots, p \tag{6}$$

In vergelijking (2) is x_{ik} de beslissingsvariabele geeft d_{ik} de wenselijkheid van vak K_i in periode k weer. Deze vergelijking is de doelfunctie van het probleem en maakt van het roosterprobleem een

optimalisatie probleem. Er wordt dus gezocht om zoveel mogelijk lessen in een wenselijke periode in te roosteren. Voorwaarde (3) zorgt ervoor dat elk vak het benodigd aantal lessen krijgt ingeroosterd. Bij (4) wordt ervoor gezorgd dat er niet meer lessen zijn dan dat er lokalen zijn en (5) zorgt ervoor dat er geen conflicten tussen lessen in eenzelfde modulen kunnen ontstaan. Tot slot geeft vergelijking (6) de waarde van de beslissingsvariabele x_{ik} weer, waarbij de variabele gelijk aan 1 is indien les i op tijdstip k plaatsvindt.

Het hierboven genoemde model is een versimpeld model. In werkelijkheid wordt er namelijk bij iedere les een aantal leerlingen, een docent en een lokaal toegewezen en worden tevens alle lessen in de tijd geplaatst. Hierdoor ontstaat er een multi-dimensionaal toewijzing probleem, wat vele malen complexer is dan bovenstaand probleem.

4.3 Enkele voorbeelden van voorwaarden

In paragraaf 4.1 werd er gesproken over hard en soft constraints. Allereerst zal er in deze paragraaf gekeken worden naar een aantal voorbeelden van deze constraints [Mül05]. Waarna er van een aantal naar de wiskundige interpretatie hiervan gekeken zal worden.

Hard constraints. Dit zijn fysieke beperkingen die opgelegd worden aan het roosterprobleem. Er zijn drie verschillende soorten hard constraints, namelijk:

1. De leerling kan op ieder tijdstip slecht één les volgen.
2. De docent kan slechts één les tegelijkertijd geven.
3. Een lokaal kan maar door één les tegelijkertijd in gebruik genomen worden.

Soft constrains. Deze bestaan uit opgegeven voorkeuren. Een aantal voorbeelden hiervan zijn:

- Tijdsvoorkeuren. Een vak kan op een specifiek tijdstip gegeven moeten worden of een docent kan voorkeur hebben voor zijn/haar werktijden.
- Voorkeur voor lokalen.
- Voorkeuren in volgorde van lessen van een bepaald vak. Er moeten bijvoorbeeld eerst theorielessen gegeven worden voor de praktijklessen.
- Het verspreiden van de lessen van een vak over de week.
- Continuïteit, dezelfde lessen moeten bijvoorbeeld in hetzelfde lokaal op hetzelfde tijdstip gegeven worden.

4.3.1 Beschikbaarheid lokalen

Een beperkende factor kan gevormd worden door de beschikbare lokalen. Er kunnen veel voorkeuren worden opgegeven voor lokalen, toch mag het aantal in te roosteren lessen per periode nooit groter zijn dan het aantal beschikbare lokalen. Vergelijking (4) geeft deze hard constraint weer.

Het kan ook gebeuren dat er voor bepaalde vakken speciale voorzieningen in een lokaal aanwezig moeten zijn. Indien hier niet van afgeweken mag worden, kan in het model een hard constraint worden toegevoegd. Deze voorwaarde lijkt op vergelijking (4), alleen worden in plaats van alle vakken, enkel de vakken meegenomen voor die voorziening en in plaats van alle lokalen wordt er een beperking gelegd op de lokalen met deze voorziening. Neem als voorbeeld het vak gym, dat moet altijd worden gegeven in een gymlokaal. Dit kan als volgt worden meegenomen:

$$x_{gk} \leq l_g \qquad \forall 1, \ldots, p$$

In deze voorwaarde zijn l_g het aantal gymlokalen en K_g is het van gym.

4.3.2 Gemeenschappelijke vakken

Vaak moet er worden voorkomen dat studenten twee lessen tegelijkertijd hebben, helemaal als dit twee lessen uit eenzelfde module zijn. Deze hard constraint is door middel van vergelijking (5) toegevoegd aan het model.

Daarnaast kan het ook zijn dat leerlingen nog vakken volgen uit andere modulen. Om door middel van een hard constraints ervoor te zorgen dat deze vakken niet gedurende dezelfde perioden worden gegeven, kan vergelijking (6) worden toegevoegd aan het model.

$$x_{ik} + x_{jk} \leq 1 \qquad i, j = 1, \ldots, q \quad i < j \quad \forall k = 1, \ldots, p \tag{7}$$

4.3.3 Soft constraints

Om als school invloed te kunnen uitoefenen op een te maken rooster, kunnen allerlei voorkeuren worden meegegeven, de soft constraints. In het model kunnen deze worden meegenomen in de doelfunctie door in vergelijk (2) de wenselijkheid (d_{ik}) op te geven. Hierbij wordt de wenselijkheid om de les op dat moment in te roosteren beïnvloed. Ook kunnen de voorkeuren worden meegenomen op de manier die in vergelijking (1) is gebruikt.

4.4 Verschillende probleemvarianten

Naast het toevoegen van voorwaarden kunnen er ook een aantal aanpassingen [Sch99] aan het model gedaan worden die meer mogelijkheden creëren. Op deze manier kan op meer wensen van een school ingespeeld worden.

4.4.1 Vooraf toegewezen lessen

Veelal is het bij het maken van een rooster wenselijk om een aantal lessen vooraf al in te plannen. Er zijn vele mogelijkheden te bedenken waarvoor het vooraf inroosteren van lessen nuttig kan zijn. Denk hierbij bijvoorbeeld aan een gastcollege dat enkel op een bepaald tijdstip gegeven kan worden. In het model wordt deze voorwaarde als een harde voorwaarde gezien, omdat hier niet van afgeweken mag worden. De les wordt namelijk niet voor niets vooraf ingeroosterd. Wiskundig gezien ziet de voorwaarde er dan als volgt uit:

$$x_{ik} = p_{ik} \qquad p_{ik} \in \{0, 1\} \qquad \forall i = 1, \ldots, q \quad \forall k = 1, \ldots, p$$

Hierbij is $p_{ik} = 0$, wanneer de les nog niet is toegewezen en $p_{ik} = 1$, wanneer les i is toegewezen is in periode k. Op dezelfde manier kan door middel van dummylessen de afwezigheid van docenten worden meegenomen.

4.4.2 Periodes van verschillende lengte

Vooralsnog is er altijd vanuit gegaan dat alle lessen dezelfde tijdslengte hebben. Echter is het denkbaar dat een school werkt met verschillende tijdschema's. Dit betekent dat er bijvoorbeeld lessen van 45 en 60 minuten in hetzelfde rooster verwerkt moeten worden of dat er een verschillend tijdschema is voor de onder- en bovenbouw om zodoende de pauzedrukte te verdelen. Indien er meerdere tijdschema's actief zijn, worden voorwaarden (3) en (4) vervangen door voorwaarden die het volgende meenemen: twee lessen l_i en l_j, die beginnen op tijdstip p en tijdstip $q > p$, geven een conflict wanneer $q - p < d_{l_i}$, waarbij d_{l_i} de lengte is van les l_i. De beslissingsvariabele x_{ik} staat nu voor les i dat op tijdstip k begint. De periodes duren nu korter dan de lessen, dus les i loopt in verschillende periodes.

4.4.3 Groepering probleem

Op een aantal scholen worden sommige vakken meerdere keren per week gegeven. Deze vakken moeten door veel leerlingen gevolgd worden en worden daarom vaak verdeeld over verschillende groepen. Het maken van deze verschillende groepen kan het aantal conflicten in de oplossing doen verminderen. Als voorbeeld: neem aan dat module S_1 de vakken K_1 en K_2 bevat en module S_2 de vakken K_1 en K_3. Stel nu dat de les van vak K_2 plaatsvindt op tijdstip p en les van K_3 op tijdstip q. In dit geval kunnen de lessen van vak K_1 niet plaatsvinden op tijdstip p en tijdstip q. Echter wanneer vak K_1 gegeven wordt in twee delen, kan het ene deel gegeven worden op tijdstip p en de andere les op tijdstip q. De volledige wiskundige omschrijving hiervan is terug te vinden in [LD84].

4.5 Complexiteit

Om voor het probleem dat eerder in deze paragraaf geformuleerd is een oplossing te vinden, wordt gebruik gemaakt van heuristieken. Het is gerechtvaardigd om een heuristiek toe te passen, gezien het probleem NP-compleet is. Om aan te tonen dat dergelijke problemen inderdaad NP-compleet zijn, wordt gebruik gemaakt van een representatie middels een graaf. Dit probleem komt namelijk overeen met het vinden van een k-puntkleuring. Om het probleem te versimpelen wordt gebruik gemaakt van de graaf $G = (V, E)$. Hierbij staat v_i voor een bepaald vak en (v_i, v_j)∈E als de vakken v_i en v_j gemeenschappelijke studenten hebben. Als nu punt v_i en v_j met elkaar verbonden zijn, betekent dit dat in de planning deze vakken niet tegelijkertijd ingeroosterd mogen worden. Stel er zijn nu p periodes waarin de vakken ingeroosterd moeten worden, dan komt dit overeen met het vinden van een p-puntkleuring van graaf G. Dit probleem hoort bij Karp's 21 NP-complete problemen [CK95]. Hiermee is dus aangetoond dat het roosterprobleem NP-compleet is.

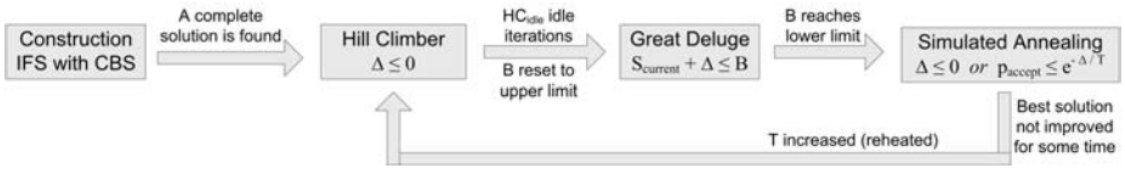
5 Implementatie CPsolver binnen Eduflex

In dit hoofdstuk zal de implementatie van CPsolver binnen Eduflex besproken worden. Eerst zullen daartoe de werking van zowel CPsolver als Eduflex los uitgelegd worden.

5.1 Algoritme CPsolver

Voor roosterproblemen zijn verschillende oplossingstechnieken uit de literatuur bekend. Een overzicht van de bekendste methodes is te vinden in appendix A. In dit hoofdstuk zal gekeken worden naar de werking van de oplosmethode die in Eduflex gebruikt zal worden: CPsolver. Het algoritme dat gebruikt wordt om roosterproblemen op te lossen zal besproken worden. Hierna zal aangegeven worden welke input CPsolver verwacht en hoe de gegenereerde output eruitziet. Voor een (uitgebreid) overzicht van de instelmogelijkheden van CPsolver kan appendix D geraadpleegd worden.

Om voor de eerder genoemde problemen tot een oplossing[1] te komen, maakt CPsolver gebruik van een algoritme. Bij dit algoritme worden een aantal heuristieken toegepast. Het algoritme van CPsolver bestaat uit vier fasen. In de eerste fase wordt een complete en uitvoerbare oplossing gegenereerd. Vervolgens wordt in een afwisseling van fase 2, 3 en 4 deze oplossing verbeterd. Omdat het een heuristiek betreft is nooit zeker of de gevonden oplossing ook echt de allerbeste is. Na een vooraf ingestelde tijd stopt het algoritme, waarna het de beste oplossing die tot dan toe gegenereerd is weergeeft. Een systematisch overzicht van de fasen is te vinden in figuur 3. Aan de hand van [Mül09] volgt een omschrijving van de vier fasen van het algoritme.



Figuur 3: Overzicht fasen CPsolver zodra Simulated Annealing wordt toegepast. (figuur afkomstig uit artikel [Mül09])

5.1.1 Constructiefase

In de constructiefase wordt een complete oplossing gegenereerd, waarbij het niet toegestaan is om hard constraints te schenden. Hierbij wordt gebruik gemaakt van een *Iterative Forward Search* algoritme (IFS). Het algoritme werkt als volgt: in het begin zijn aan alle variabelen nog geen waarden toegekend. In elke iteratie die nu volgt wordt getracht een variabele een toegestane waarde te geven, door bijvoorbeeld een bepaalde les een plaats en een tijd toe te wijzen. Mocht zo'n dergelijke toewijzing tot een conflict leiden, worden de conflicterende variabelen weer in de lijst van niet geplaatste variabelen geplaatst. In deze fase wordt alleen gekeken naar de hard constraints, soft constraints worden nu nog niet meegenomen. Een voorbeeld van een dergelijke hard constraint is de eis dat een docent niet op twee verschillende lessen ingeroosterd mag worden op hetzelfde tijdstip. Een voorbeeld van een soft constraint, die in deze fase dus nog genegeerd wordt, is de wens om het aantal tussenuren te minimaliseren. Een oplossing met schendingen van hard constraints zal nooit aangenomen worden. Het algoritme stopt als uiteindelijk alle variabelen geplaatst zijn en er dus een compleet en uitvoerbaar rooster is gegenereerd.

Om het algoritme sneller en slimmer te laten verlopen, wordt bij het kiezen van de te plaatsen variabelen in elke iteratie een bepaalde volgorde aangehouden. Allereerst wordt getracht de moeilijke variabelen vast te leggen, dit houdt bijvoorbeeld in dat een docent die alleen op maandag werkt eerder wordt

1. Hieronder wordt een rooster verstaan waarbij alle gewenste lessen zijn ingeroosterd.

ingepland dan een docent die de gehele week beschikbaar is. In de constructiefase wordt hiervoor gebruik gemaakt van de parameters in de *Lecture Selection* categorie, zoals in paragraaf 5.5.2 beschreven staat. Aan de hand van deze parameters bepaalt CPsolver telkens welke les als volgende ingeroosterd wordt.

Daarnaast wordt er tijdens deze fase gebruikt gemaakt van Conflict-based Statistics (CBS). CBS is een methode die wordt toegepast om eerder voorgekomen conflicten tussen bepaalde variabelen te vermijden. Aan de hand van een data lijst worden conflicten met de bijbehorende intensiteit bijgehouden. Deze lijst wordt gebruikt om cykels in het algoritme te voorkomen en het proces hierdoor sneller te laten verlopen. Ook voor de CBS methode staan in paragraaf 5.5.2 de parameters beschreven.

5.1.2 Hill Climbing fase

In de tweede fase wordt de gevonden oplossing verbeterd aan de hand van de vooraf ingegeven gewichten voor de soft constraints. In de praktijk komt dit neer op het verminderen van tussenuren, verdeling van uren over de werkweek en andere wenselijkheden binnen een onderwijsinstelling. De wensen kunnen bijvoorbeeld overeenkomen met de door Paralax gevonden KPI's, zie hoofdstuk 3. De gewichten kunnen ingesteld worden naar de wensen van de klant. Voor meer informatie over de instellingen verwijzen we naar paragraaf 5.5.2. Daar worden onder andere de parameters uit de groepen *Solution Comparator* en *Placement Selection* beschreven. Deze parameters worden in alle drie de fasen na de constructiefase gebruikt.
Het verbeteren van de KPI-waarden gebeurt middels een *Hill Climbing* algoritme, waarbij in elke iteratie een "willekeurige" verandering wordt aangebracht in de oplossing. De keuze van de verandering is willekeurig maar wordt wel beïnvloed door parameters die later beschreven zullen worden. De verandering wordt geaccepteerd als het de oplossingswaarde verbetert en het hierbij de hard constraints niet schendt. Na een vooraf ingesteld aantal iteraties HC_{idle} waarbij de oplossing niet verbetert, eindigt de fase. Er is nu een lokaal minimum gevonden, de hieropvolgende fasen worden gebruikt om eventueel nog een betere oplossing te vinden die verder weg ligt van dit lokale minimum.

5.1.3 Great Deluge fase

Na beëindiging van de *Hill-Climbing* fase wordt geprobeerd om nog een verbetering te vinden met het Great Deluge algoritme. In deze fase wordt gebruik gemaakt van een grens B. Deze grens zorgt ervoor dat het algoritme zich niet beperkt tot een lokaal deel van de oplossingsruimte. Een gemaakte keuze van een variabele wordt geaccepteerd als de nieuwe oplossing niet deze grens overschrijdt. Dat wil zeggen de doelfunctiewaarde moet lager zijn dan de grens B wil de oplossing geaccepteerd worden. Wiskundig wordt de grens in het begin als volgt gedefinieerd:

$$B = GD_{ub} \cdot S_{best}.$$

Hierin staat GD_{ub} voor een vooraf ingestelde bovengrens en S_{best} voor de waarde van de tot nu toe beste oplossing. Deze grens wordt vervolgens verlaagd na elke iteratie door vermenigvuldiging met een factor $GD_{cr} < 1$:

$$B = B \cdot GD_{cr}.$$

Dit wordt herhaald totdat de grens B een ondergrens gelijk aan $GD_{lb}^{at} \cdot Sbest$ bereikt, waarbij GD_{lb} een parameter is die de ondergrens van de coëfficient definieert. Na het bereiken van deze ondergrens, wordt de grens B weer gereset naar de bovengrens gelijk aan $GD_{ub}^{at} \cdot Sbest$.

$$B < GD_{lb}^{at} \cdot Sbest \Rightarrow B = GD_{ub}^{at} \cdot Sbest$$

Hierbij is de parameter at een teller die als beginwaarde 1 heeft. Deze teller wordt verhoogd met 1 als de ondergrens voor B bereikt is en B weer verhoogd wordt. De teller wordt gereset wanneer er een betere oplossing gevonden wordt. Deze teller zorgt er voor dat de grenzen steeds hoger worden zolang er geen verbetering gevonden wordt. Op die manier kan uit een diep lokaal minimum geklommen worden om een globaal beter minimum te vinden.

5.1.4 Simulated Annealing fase

Het *Simulated Annealing* algoritme wordt gebruikt nadat in de *Great Deluge* fase de lower bound bereikt is. Er wordt gebruik gemaakt van een temperatuurparameter T bij het al dan niet accepteren van een nieuwe oplossing. In de *Hill Climbing* fase wordt een nieuwe oplossing alleen geaccepteerd als de waarde van deze oplossing beter is dan de eerder als best beschouwde oplossing. In deze fase wordt er echter nog een mogelijkheid gegeven om een nieuwe oplossing te accepteren, ook al is deze slechter dan de tot nu toe gevonden beste oplossing, namelijk met kans

$$p_{accept} = e^{-\frac{\Delta}{T}}.$$

Hierbij is Δ de kostentoename van de huidige oplossing wanneer er een nadelige verandering plaatsvindt. De temperatuur T begint op de startwaarde SA_{it} en wordt na iedere $SA_{cc} \cdot TL$ iteraties vermenigvuldigd met een factor SA_{cr}. Hierin is SA_{cc} de verlagingfactor en TL een getal dat afhangt van het aantal mogelijke waarden van alle probleemspecifieke variabelen. Wanneer de best gevonden oplossing niet verbeterd is na $SA_{rc} \cdot SA_{cc} \cdot TL$ iteraties, wordt de temperatuur verhoogd naar

$$T = T \cdot SA_{cr}^{-1.7 \cdot SA_{rc}}.$$

De *Simulated Annealing* fase wordt gebruikt na de *Great Deluge* fase en gaat na de temperatuurverhoging weer over in de *Hill Climbing* fase, zoals te zien was in figuur 3. Op deze manier wordt afwisselend een lokaal minimum gezocht en geprobeerd "heel ergens anders" in de oplossingsruimte een nog lager punt te vinden.

5.2 User interactie CPsolver

Voor de gebruiker van het programma CPsolver is het zojuist omschreven algoritme meestal niet zichtbaar. De gebruiker krijgt voornamelijk te maken met de interface van het programma. Hierin kan hij zijn voorkeuren op een gebruiksvriendelijke manier instellen, waarna hiervan een inputfile voor CPsolver van gemaakt wordt. In deze inputfile, een xml file, wordt informatie gegeven, waarvoor CPsolver vervolgens een oplossing zoekt. Daarnaast kan een interface ook de oplossing, een outputfile, inlezen. Verschillende programma's kunnen dienen als interface voor CPsolver, veelgebruikt is Unitime. Unitime is vrij toegankelijke software gemaakt door de maker van CPsolver zelf, bedoeld voor gebruik bij het maken van roosters op voornamelijk universiteiten. In het volgende hoofdstuk komt Eduflex als interface aan bod, maar allereerst is het nuttig om input en output van CPsolver nader te bekijken.

5.2.1 Input

De input voor de CPsolver bevat alle gegevens die CPsolver nodig heeft om tot een oplossing te kunnen komen. Dit zijn onder andere de docenten, de lokalen met capaciteit en de leerlingen met de gemaakte vakkeuzes. Om het een en ander te verduidelijken zijn delen van de xml files terug te vinden in appendix C.2.

Allereerst wordt algemene informatie gegeven over de interface en het aantal dagen en uren dat wordt meegenomen in het rooster. Dit is het aantal uren dat gebruikt kan worden. Welke uren er daadwerkelijk gebruikt mogen worden, zal vooraf meegegeven moeten worden. Hierna volgen enkele declaraties:

- Alle lokalen die beschikbaar zijn voor het in te roosteren probleem krijgen een id. Hierbij wordt ook de capaciteit van het lokaal meegegeven. Verder is het mogelijk om een locatie toe te voegen, zodat de loopafstand tussen twee lesuren meegenomen kan worden in het oplossingsproces.
- Ook de docenten worden voorzien van een id.
- Alle in te roosteren lessen worden gedeclareerd. Dit is een combinatie van een vak, een klas, de beschikbare docenten en de beschikbare lokalen, waarbij wordt aangegeven hoe vaak per week deze combinatie ingeroosterd moet worden.
- Vervolgens bestaat er de mogelijkheid om groepsvoorwaarden in te voeren. Hier kunnen soft constraints die van toepassing zijn op een les worden meegegeven. De mogelijke instellingen zullen worden besproken in paragraaf 5.4.2.

- Tenslotte worden alle studenten weergegeven. Elke student krijgt een id toegewezen.Ook de modules die door een student gekozen zijn, worden hier aangegeven.

5.2.2 Output

Met bovenstaande input wordt een rooster gegenereerd in de vorm van een xml bestand (solution.xml). Hierin staat alle informatie die nodig is voor de interface om het rooster in te kunnen lezen. Daarnaast worden nog enkele bestanden gemaakt met gegevens over het gegenereerde rooster en het doorlopen proces. Deze bestanden zullen hier kort besproken worden.

Solution.xml.
In dit bestand is de oplossing weergegeven in xml formaat, deze kan worden ingelezen door de interface. De opbouw van het bestand is als volgt;

- Allereerst wordt er algemene gegevens over de oplossing gegeven.
- Hierna worden er een aantal standaard zaken gegeven, zoals de datum en tijd waarop de file aangemaakt is, maar ook het aantal dagen waarover het rooster gaat en het aantal inrooster momenten per dag staan gedefinieerd. Dit wordt op eenzelfde manier weergegeven als bij de input file.
- Daarnaast worden aan de hand van declaraties de ingeroosterde uren benoemd. De uren zijn voorzien van een leraar en lokaal en er wordt aangegeven op welk tijdstip de les plaatsvindt. Ook is er gekeken of het uur in conflict is met een ander uur voor bepaalde studenten, deze worden voorzien van een gewicht.
- Tot slot worden de studenten weergegeven met hun uren. Hierbij wordt ook het id van de modules vermeld.

Statistieken over oplossing.
Verschillende bestanden bevatten informatie over de oplossing. Hierbij valt te denken aan gegevens over input, zoals de hoeveelheid lokalen en docenten. Maar ook statistieken over de oplossing, zoals lokaalgebruik en aantal lessen per docent zijn hierin terug te vinden. In appendix C is voor de bestanden info.csv, info.txt, output.csv te vinden welke waarden er in de bestanden wordt weergegeven.

5.3 Rostar EduFlex

In dit hoofdstuk zal ingegaan worden op het software programma Rostar Eduflex, verderop alleen aangeduid met Eduflex. De onderzoeksvraag hoe Eduflex gebruik maakt van CPsolver wordt hier mede beantwoord. Om dit te kunnen doen is eerst algemene kennis van Eduflex nodig. Hierom zal allereerst omschreven worden voor welke roosterproblemen Eduflex verkocht wordt. Hierna zal worden uitgelegd hoe Eduflex werkt, waarna een overzicht gegeven zal worden van alle mogelijkheden die Eduflex biedt bij het maken van roosters.

5.3.1 Toepassing Eduflex

Eduflex is een software programma dat is ontwikkeld door het bedrijf Paralax voor roosterplanningen in het onderwijs. Eduflex is gemaakt om zowel complexe als eenvoudige roosterproblemen op te lossen en is daarom geschikt voor zowel grote als kleinere onderwijsinstellingen. In Nederland wordt Eduflex vooral gebruikt op middelbare scholen en ROC's. Bij het maken van het rooster kan gekozen worden om het rooster voor de gehele onderwijsinstelling in het geheel te genereren, maar het is ook mogelijk om dit op te delen in kleinere problemen. Hierbij kan bijvoorbeeld gedacht worden aan het maken van een rooster per afdeling of locatie. In het roosterscherm worden tegelijkertijd de klas-, lokaal-, docent- en leerlingroosters weergegeven en deze zijn 3D met elkaar gelinkt. Dit houdt in dat als in het ene rooster iets verandert wordt, dat alle andere roosters zich automatisch aanpassen. De nieuwste functie binnen Eduflex maakt het mogelijk dat studenten zich via internet inschrijven op vooraf vastgelegde

modules. Deze nieuwe functionaliteit komt overeen met het eerder beschreven modulair onderwijs, vooral ingevoerd op ROC's. Bij voldoende inschrijvingen kunnen deze modules dan vervolgens ingepland worden met Eduflex.

5.3.2 Hoe werkt Eduflex?

Het maken van een rooster binnen Eduflex kan op verschillende manieren; handmatig inplannen, semi-automatisch of volautomatisch. Het handmatig inplannen laat het inplannen geheel aan de roostermaker over, maar hierbij zijn wel extra handigheden ingebouwd. Zo is het onmogelijk om lokalen of personen dubbel in te roosteren, het programma geeft dan een duidelijke foutmelding dat er een conflict ontstaat dat niet is toegestaan. Bij het semi-automatisch roosteren wordt een lijst van in te plannen lessen afgegaan, die dan door de roostermaker een plaats in het rooster krijgen toegewezen. Ook hierbij wordt duidelijk aangegeven welke tijdstippen er nog beschikbaar zijn. Het volautomatisch roosteren kan op twee manieren: er kan gebruik gemaakt worden van de originele solver behorende bij Eduflex of de nieuwe geïmplementeerde CPsolver. Deze solvers worden voor verschillende deelproblemen apart aangeroepen, hoewel het ook mogelijk is om het hele rooster in een keer te genereren. Vervolgens worden alle resulterende roosters in Eduflex samengevoegd. De werking van CPsolver is in paragraaf 5.1 al besproken, in paragraaf 5.4 wordt ingegaan op de implementatie van deze solver binnen Eduflex.

Gegevens invoeren.
Om het rooster te kunnen maken, met welke manier van inplannen ook, is het allereerst nodig dat allerlei gegevens ingevoerd worden. Dit kan handmatig, met behulp van lijsten binnen Eduflex; docent-, leerling-, lokaal-, afdeling- en vaklijsten. In deze lijsten kunnen verschillende gegevens worden bijgehouden, de belangrijkste hiervan zijn verderop weergegeven in een overzicht van de mogelijkheden. Hiernaast is een belangrijk scherm de SelectieTabelTeDoen. Hierin staat welke lessen nog ingeroosterd moeten worden. Deze lessen worden door de solver automatisch ingeroosterd. Aangezien scholen vaak zelf ook al grote databases hebben met leerling- en docentgegevens is het gebruikelijk om de bestaande databases te koppelen aan Eduflex, hiervoor biedt Paralax ondersteuning.

5.3.3 Overzicht van mogelijkheden

Hieronder is een overzicht weergegeven van mogelijkheden die Eduflex biedt bij het inroosteren. Dit is gedaan aan de hand van veelgebruikte schermen binnen Eduflex. Om het overzichtelijk te houden zijn niet alle mogelijkheden hier benoemd, echter de meest relevante voor het onderzoek wel.

SelectieTabelTeDoen Dit is misschien wel het belangrijkste scherm indien Eduflex gebruikt wordt om automatisch een rooster te genereren. Alvorens de roosterautomaat te starten, is het nodig om een deel van deze lessen te selecteren, alleen de geselecteerde lessen worden door de roosterautomaat ingeroosterd. Zo is het mogelijk om bijvoorbeeld per afdeling het rooster te laten maken.

Docentlijst In deze lijst worden alle docenten bijgehouden. Verschillende gegevens zoals naam, e-mail, maximaal aantal uren per week zijn hier in te vullen. Ook wordt hier weergegeven welke vakken een docent allemaal kan geven en of er een vaste vervanger voor hem is.

Lokalenlijst Zoals de naam al doet vermoeden worden in deze lijst alle lokalen bijgehouden. Elk lokaal heeft hier numeriek een lokalensoort toegewezen, bijvoorbeeld theorie- of praktijklokaal. Zo is het mogelijk om in de SelectieTabelTeDoen niet een specifiek lokaal aan te geven, maar alleen een lokalensoort. Op deze manier wordt de solver minder beperkt. Tevens is hier aangegeven wat de capaciteit van elk lokaal is en wat geschikte uitwijklokalen zijn, mocht een lokaal niet beschikbaar zijn.

Leerlinglijst In de leerlinglijst zijn alle leerlingen met naam en studentnummer weergegeven. Tevens kunnen hier de keuzes voor webmodules staan, mocht de school hiermee werken. Ook als er verschillende studierichtingen binnen een onderwijsinstelling zijn is hier voor elke leerling weergegeven welke studie hij volgt.

KlasModuleLijst Een overzicht van alle vakken is gegeven in de KlasModuleLijst. Hierin is per vak aangegeven of dit een vak is dat gegeven wordt aan een stamklas, een klas die alle lessen met elkaar volgt, of dat het een module is. In het laatste geval is het mogelijk om (een deel van) verschillende klassen deel te laten nemen aan dit vak en moet ook aangeven worden welke vakken in deze module zitten. Hiernaast wordt hier aangegeven of voor dit vak ingeschreven moet worden via de online inschrijfmodule en wat het minimum en maximum aantal leerlingen is.

Hiernaast zijn er nog vier schermen die de roosters weergeven; **KlasModuleRooster**, **Lokaalrooster**, **Leerlingrooster** en **Docentrooster**. Hierin zijn respectievelijk de gegenereerde roosters voor de klassen, lokalen, leerlingen en docenten weergegeven. In deze schermen is het mogelijk om een extra code toe te voegen. Deze extra codes kunnen bijvoorbeeld handig zijn als een bepaald lokaal een bepaald tijdstip bezet is, door bijvoorbeeld verhuur aan een externe partij. In dat geval wordt in het Lokaalrooster een bezetcode toegevoegd op dat tijdstip. Op dezelfde manier is het mogelijk om afwezigheid van een docent door te geven. Er zijn hiervoor verschillende types extra codes in te voeren, deze zijn te beheren via InstellingenExtraCodes. Hierin kan worden aangegeven of een bepaalde code echt blokkeert, dan wordt het onmogelijk om op dat uur iets in te roosteren. Hiernaast kan een code niet blokkerend zijn, dan is het gewoon een melding in het rooster. De laatste mogelijkheid is dat er een vraag gesteld wordt als je op het betreffende tijdstip iets wilt inroosteren, dit is echter erg onhandig als automatisch geroosterd zal worden.

Voor het genereren van het rooster is het verder nog nodig om een tijdschema in te voeren. Dit is een vooraf ingesteld schema dat aangeeft hoelang de lesuren op een dag duren en hoeveel het er zijn. Ook is het mogelijk om vaste pauzetijden aan te geven. Dit is handig omdat niet elke onderwijsinstelling met dezelfde lesuren werkt. Tevens bestaat dan de mogelijkheid om verschillende tijdschema's aan te maken voor verschillende afdelingen, bijvoorbeeld onder- en bovenbouw. Dit kan met name praktisch zijn rond het middaguur, zodat niet iedereen op hetzelfde moment pauze heeft.

Extra's.
Naast bovengenoemde belangrijke dingen voor het genereren van het rooster, zijn er ook nog enkele extra handigheden ingebouwd in Eduflex. Zo is het mogelijk om een bepaalde groep studenten automatisch te laten mailen, bijvoorbeeld als een bepaald lesuur vervalt. Tevens bestaat de mogelijkheid om

de door de solver of handmatig gemaakte roosters te controleren op verschillende eisen, bijvoorbeeld de eisen van de CAO. Tenslotte is het mogelijk om het rooster uit te printen of te exporteren naar andere bestandsformaten, zodat het bijvoorbeeld ingelezen kan worden door de Outlookagenda.

Al met al zijn er veel verschillende manieren waarop Eduflex gebruikt kan worden. Het kan enkel gebruikt worden om ondersteuning te bieden bij het handmatig inroosteren of het programma kan grotendeels het werk van de roostermaker overnemen, mits alles goed is ingesteld. Ook bij het instellen zijn er veel verschillende mogelijkheden. Zo kunnen van tevoren veel restricties opgelegd worden aan het rooster, door bijvoorbeeld weinig vrij te kiezen te laten in de SelectieTabelTeDoen. De andere mogelijkheid is om de solver veel keuzes voor lokalen, klassensamenstelling en docenten te laten maken door gebruik te maken van lokaalsoorten en docenten te groeperen op vakgebied. Op deze manier kan elke onderwijsinstelling naar hun eigen voorkeuren op een flexibele manier gebruik maken van Eduflex.

5.4 Implementatie CPsolver in Eduflex

Om ervoor te zorgen dat CPsolver de nieuwe solver wordt voor het programma Eduflex, zal Paralax nog een aantal aanpassingen moeten uitvoeren. In paragraaf 5.1 zijn de mogelijkheden van CPsolver aan bod gekomen. Nu zal worden bekeken welke mogelijkheden op dit moment niet benut worden door Eduflex en hoe deze toch benut kunnen worden. Hiertoe zal allereerst een aantal algemene punten worden besproken die ervoor zorgen dat CPsolver gebruiksvriendelijker wordt binnen Eduflex. Daarnaast worden er een aantal gegevens niet meegenomen in de inputfile van CPsolver, hierdoor worden er ook nog een aantal functies niet optimaal benut. Wat deze functies inhouden en hoe ze te implementeren zijn in Eduflex, zal tot slot worden besproken.

5.4.1 Algemene punten

Doordat Eduflex al lange tijd werkt met een eigen gebouwde solver, is het programma hier volledig naar ingericht. Dit betekent dat wanneer CPsolver de huidige solver overneemt, er een aantal aanpassingen gedaan moeten worden. Welke aanpassingen in onze ogen gedaan kunnen worden om CPsolver gebruiksvriendelijker te maken binnen Eduflex, zullen hier puntsgewijs behandeld worden.

TabelTeDoen vernieuwen.
Uiteraard kan het gebeuren dat na het maken van een rooster deze toch opnieuw gemaakt dient te worden. In dat geval moet het oude rooster verwijderd worden. Onder het menu bewerken is de knop rooster wissen te vinden. Echter worden de dan weer in te roosteren lessen niet automatisch terug gezet in de TabelTeDoen. Ook is er geen vernieuwen knop te vinden. De gevonden mogelijkheid om de tabel zichtbaar te krijgen, is de functie kies scherm lay-out te gebruiken en opnieuw het opstartscherm te kiezen. Hiermee wordt het opstartscherm en daarmee ook de TabelTeDoen vernieuwd. Echter is dit niet de meest efficiënte manier om deze onhandigheid op te lossen.

De oude en nieuwe solver.
Binnen Eduflex is er lastig onderscheid te maken tussen de twee solvers, diegene die ontworpen is door Paralax en CPsolver. Doordat er op verschillende plekken instelmogelijkheden aangeboden worden, is het op dit moment moeilijk om te bepalen waar deze invloed op uitoefenen. Wanneer de CPsolver geïmplementeerd wordt, is het goed om duidelijk te hebben waar de klant zijn wensen kan invoeren.

Ander gebouw.
Er bestaat binnen Eduflex de functie om op te geven in welk gebouw een lokaal zich bevindt. Echter wordt dit niet meegenomen in de inputfile voor CPsolver. Ook kan er niet worden aangegeven hoe wenselijk het is dat bepaalde lessen in hetzelfde gebouw plaatsvinden. Wanneer er gekozen wordt deze functie ook binnen CPsolver in werking te stellen, kan dit meegenomen worden bij de afstanden tussen lokalen. Deze zullen in de volgende paragraaf omschreven worden.

Extra codes.
Eduflex biedt de mogelijkheid om voor de docent extra codes toe te voegen. De extra codes zijn beperkingen op de beschikbaarheid van de docent. Welke mogelijkheden hier voor zijn, staat omschreven in figuur B17. De codes kunnen drie verschillende doelen hebben; blokkeren, een melding geven of niet blokkeren. Bij de oude solver werken deze codes prima, echter zijn ze nog niet gekoppeld aan CPsolver. Op dit moment is het zelfs zo dat wanneer de extra codes staan ingesteld op blokkeren, de CPsolver vast loopt tijdens het proces. Gezien onderwijsinstellingen vaak beperkingen op de beschikbaarheid van hun docenten willen opleggen, zal deze koppeling nog gemaakt moeten worden.

Modules.
Daarnaast is er nog een instelling die goed moet staan om gebruik te kunnen maken van CPsolver. De in te roosteren lessen moeten namelijk in Eduflex staan als modules en daarnaast moet ook de inschrijving op ja staan. Dit betekent dat om CPsolver geïmplementeerd te krijgen in Eduflex alle lessen anders ingevoerd moeten worden, oftewel de datasets zullen anders ingedeeld moeten worden.

Roosteren op soort lokaal en soort docent.
In de SelectieTabelTeDoen is er de mogelijkheid om precies aan te geven welk vak aan welke klas door welke docent en in welk lokaal gegeven moet worden. Echter beperkt dit enorm de mogelijkheden van de CPsolver, aangezien er dan enkel de beslissing van het plaatsen in de tijd nog gemaakt moet worden. Daarom is het mogelijk om sets van docenten aan te maken. Dit gebeurt via de docentlijst, hierbij is voor elke docent weergegeven welke vakken hij kan geven. Ook is het mogelijk om de lokalen een lokalensoort mee te geven. Op deze manier kan het aantal beslissingen dat genomen moet worden door de CPsolver vergroot worden, hiervoor moet in de SelectieTabelTeDoen minder expliciet worden aangegeven welke docent en lokalen gebruikt moeten worden voor een les. Er moet dan gebruik gemaakt worden van de docent- en lokalensoort. Vervolgens zal CPsolver dan een geschikte combinatie uitkiezen, waarbij dus meer keuzemogelijkheid voor CPsolver overgelaten wordt.

5.4.2 Niet benut in inputfile

Eduflex en Unitime genereren beide een inputfile die meegegeven wordt aan CPsolver. Tussen deze files zijn echter nog een aantal verschillen. Dit heeft als gevolg dat Eduflex nog niet alle mogelijkheden van de CPsolver benut. De mogelijkheden die benut kunnen worden, zullen toegelicht worden.

Afstanden.
In CPsolver bestaat de mogelijkheid om de afstanden mee te nemen. Bij het roosteren wordt er dan rekening gehouden met het feit dat een docent niet achter elkaar in twee verschillende gebouwen hoeft les te geven. Ook bestaat er de mogelijkheid om maximale afstanden voor de leerlingen tussen lokalen op te geven. Deze afstanden kunnen worden opgegeven aan CPsolver door in figuur C19 bij locatie een coördinaat op te geven voor het betreffende lokaal. In Eduflex bestaat de mogelijkheid om de afstanden tussen lokalen te declareren nog niet. Het is denkbaar dat op Amerikaanse universiteiten de afstanden een belangrijkere rol spelen dan op de ROC's in Nederland. Het is dus de vraag of het wenselijk is om de afstanden mee te nemen in de inputfile. Hiervoor zal in paragraaf 5.5 bekeken worden welke parameters invloed hebben op de afstanden tussen de lokalen. Indien er door Paralax gekozen wordt om deze afstanden niet mee te nemen, heeft dit natuurlijk als gevolg dat niet alle mogelijkheden van CPsolver benut worden.

Group constraints.
In de inputfile kunnen verschillende voorkeuren worden opgegeven. Dit gebeurt onder het kopje groupConstraints. Welke mogelijkheden van voorkeuren er allemaal zijn, staat weergegeven in tabel C23. De instelmogelijkheden voor deze voorkeuren zijn; verplicht(R), voorkeur(1), sterke voorkeur(2), afkeur(-1) of sterke afkeur(-2). Een voorbeeld van zo'n constraint is te vinden in figuur C28. Helaas is het op

dit moment zo dat in de inputfile die door Eduflex gegenereerd wordt niets onder de groupConstraints staat. Er wordt door CPsolver dus geen rekening gehouden met voorkeuren. Indien dit geïmplementeerd wordt in Eduflex zal hier ook een plek toegevoegd moeten worden waar de voorkeuren opgegeven kunnen worden. Op deze manier verrijkt Paralax de mogelijkheden voor de klant. Het wordt hierdoor onder andere gemakkelijker om op te geven dat lessen in een bepaalde volgorde moeten worden gegeven. Denk bijvoorbeeld aan praktijklessen die gegeven dienen te worden na een theorieles of een vak dat op een vast tijdstip van de dag moet worden geven.

Ingeroosterde uren.
Wanneer er lessen al handmatig zijn ingeroosterd in Eduflex, worden in de inputfile de docent, het lokaal en de leerlingen op niet beschikbaar gesteld bij dat bepaalde uur. De les die al ingeroosterd was, komt hiermee niet in de inputfile en zodoende ook niet in de outputfile. Dit heef tot gevolg dat bij leerlingen die al ingeroosterd zijn bij dit lesuur dit niet wordt meegenomen, wat bijvoorbeeld tot extra tussenuren kan leiden. Er is echter een manier om deze al ingeroosterde lessen wel op te nemen in de input- en outputfile. Dit kan namelijk door bij het lesuur de variabele 'committed', zoals te zien in figuur C21, de waarde 'true' te geven. Hierna zal enkel het uur waarop de les gegeven wordt, worden weergegeven. Als een les dus al met de hand ingeroosterd worden van te voren, moet er dus worden opgegeven dat de les al toegewezen is.

Voorkeur lokalen.
In de inputfile bestaat er de mogelijkheid om per in te roosteren les een voorkeur of afkeur voor een bepaald lokaal op te geven. Het is bijvoorbeeld wenselijk dat scheikunde in een scheikunde lokaal wordt gegeven en het vak Nederlands verkiest een lokaal voor Frans boven een scheikundelokaal. De voorkeuren kunnen worden opgegeven door per les een gewicht mee te geven aan de beschikbare lokalen. Voor een voorbeeld, moet figuur C22 worden toegevoegd bij de declaratie van de in te roosteren uren.

Tijdsvoorkeuren.
Naast voorkeur voor lokalen, kunnen er ook per les voorkeuren voor tijdstippen worden opgegeven. Neem als voorbeeld het vak gym, het is prettiger om deze les aan het begin of aan het einde van een lesdag te hebben. Dit kan in de inputfile worden meegenomen door per les een voorkeur of afkeur mee te geven per mogelijk tijdstip, hiervoor moet figuur C23 aan de in te roosteren uren declaratie worden toegevoegd.

5.5 Instellingen CPsolver

5.5.1 Filtering van parameters

Zoals eerder kort benoemd zijn er veel parameters waar CPsolver gebruik van maakt die allen afzonderlijk in te stellen zijn middels een *configuratiebestand* (ook *configfile* genoemd). Dit bestand bevat 169 verschillende parameters, waarvan de waarde voor sommige een kommagetal is en andere op waar of niet waar gezet kunnen worden. Een lijst van alle parameters is te vinden in tabel D24. De parameters zijn door de ontwerper van CPsolver ondergebracht in 15 verschillende categorieën;

- General Settings
- Termination Conditions
- Implementations
- Default Time Preferences
- Distances
- Basic settings
- Lecture Selection
- Conflict-based Statistics
- Solution Comarator Weights
- Placement Selection
- Neighbour Selection
- Same Subpart Balancing
- Departmental Balancing
- Search Intensification
- Perturbation Penalty

Zoals de namen van de categorieën al doen vermoeden, zijn voor dit onderzoek niet alle parameters van belang. Zo was in paragraaf 5.3 te lezen dat Eduflex geen gebruik maakt van tijdsvoorkeuren en afstanden tussen lokalen, alle parameters die hier betrekking op hebben binnen CPsolver hebben dus geen invloed op de uitkomst. In tabel D24 is voor alle parameters aangegeven of ze worden gebruikt binnen Eduflex. Naast parameters die niet gebruikt worden zijn er algemene instellingen die sowieso hetzelfde moeten blijven, zoals bijvoorbeeld de parameters binnen de categorie *Implementations* die alleen verwijzen naar de juiste datalijsten. Om een filtering te kunnen maken voor het onderzoek, worden de volgende categorieën op basis van bovenstaande redenen niet meegenomen bij het onderzoek;

-General settings
-Termination Conditions
-Implementations
-Default time preferences
-Distances
-Basic settings

De parameters binnen deze categorieën zijn weergegeven in tabel D26 (in de appendix). Hierin staan ook de waarden van de parameters waarmee in het vervolgonderzoek gewerkt zal worden. Hiernaast zijn er nog enkele parameters die betrekking hebben op het meegeven van tijdsvoorkeuren en afstanden binnen de andere categorieën, ook deze hoeven niet meegenomen te worden in het onderzoek.

5.5.2 Groepering van parameters

Na het filteren van de parameters op relevantie voor het onderzoek, blijven uiteindelijk nog 112 parameters over die onderzocht moeten worden. Deze zijn al gegroepeerd zoals hierboven weergegeven, deze categorieën zullen nu puntsgewijs besproken worden. Hierbij wordt weergegeven welke parameters meegenomen worden binnen het onderzoek en waar de parameters binnen het algoritme van CPsolver gebruikt worden.

Lecture Selection.
Tijdens de constructiefase van het algoritme (zie paragraaf 5.1.1) wordt een volledige oplossing gegenereerd. In elke iteratie wordt hierin getracht een variabele een toegestane waarde te geven. Om een keuze te maken uit de variabelen die nog geen waarde toegekend hebben gekregen, worden de parameters uit tabel 4 gebruikt. Deze parameters worden gebruikt om te zoeken door alle variabelen die nog geen waarde hebben, alsmede parameters die gebruik maken van een deelverzameling van alle variabelen. Dit laatste zorgt voor een hogere snelheid van het algoritme.

Al deze parameters kunnen worden ingesteld op een waarde. Voor de meeste van deze parameters geldt dat de waarde moet liggen tussen de -2.0 en 2.0. Bijvoorbeeld 0.0 wanneer er geen voorkeur voor een bepaald lokaal bestaat en 2.0 wanneer er juist veel afkeur is voor een lokaal en -2.0 bij een voorkeur. Indien er niet aan een voorkeur voldaan wordt, zal de doelfunctiewaarde worden verhoogd met deze factors. Bij de parameters die niet tussen -2.0 en 2.0 moeten vallen is dit aangegeven.

Conflict-based Statistics.
De parameters in deze categorie hebben ook invloed in de constructiefase. In deze fase wordt er gebruik gemaakt van het IFS-algoritme voor het onthouden van eerdere tegengekomen conflicten. De onthouden conflicten kunnen gesorteerd worden op hoe lang geleden ze plaats hebben gevonden, hierbij worden de parameters uit tabel 5 gebruikt. De eerste parameter geeft aan met welke coëfficient het gewicht van een conflict verkleind moet worden na elke iteratie. De tweede parameter geeft aan na hoeveel iteraties een conflict de helft van zijn oorspronkelijke gewicht krijgt. Gebruikelijk is dat een van beide methodes gekozen wordt, dus één van deze parameters hoort gelijk aan nul te zijn.

Parameter	**Uitleg**
Lecture.RandomWalkProb	Random walk kans, tussen 0 en 1
Lecture.DomainSizeWeight	Gewicht van de domeingrootte
Lecture.NrAssignmentsWeight	Gewicht van aantal toewijzingen
Lecture.InitialAssignmentWeight	Gewicht begintoewijzingen
Lecture.NrConstraintsWeight	Gewicht aantal constraints
Lecture.UselessSlotWeight	Gewicht van een tussenuur
Lecture.DistanceInstructorPreferenceWeight	Gewicht lokaalafstand gebaseerde voorkeuren docent
Lecture.DeptSpreadPenaltyWeight	Gewicht van goede verdeling van uren over verschillende afdelingen in de tijd
Lecture.SelectionSubSet	Selectie tussen subsets van lessen ('true' of 'false')
Lecture.SelectionSubSetMinSize	Minimale grootte van de subsets (niet tussen -2.0 en 2.0)
Lecture.SelectionSubSetPart	Grootte van de subsets in percentage van aantal te kiezen lessen (niet tussen -2.0 en 2.0)
Lecture.SpreadPenaltyWeight	Gewicht van goede verdeling van uren binnen een afdeling
Lecture.CommitedStudentConflictWeight	Gewicht van studentenconflicten bij van tevoren toegekende waarden bij variabelen
Lecture.HardStudentConflictWeight	Gewicht van studentconflicten
Lecture.StudentConflictWeight	Gewicht van studentenconflicten
Lecture.TooBigRoomWeight	Gewicht van een te groot lokaal (meer dan 50% niet gebruikt)
Lecture.TimePreferenceWeight	Gewicht van tijdsvoorkeur
Lecture.ContrPreferenceWeight	Gewicht van voorkeur constraint
Lecture.RoomPreferenceWeight	Gewicht van lokaalvoorkeur

Tabel 4: Parameters Lecture selection

ConflictStatistics.Ageing
ConflictStatistics.AgeingHalfTime

Tabel 5: Parameters Conflict-bases statistics

Solution Comparator Weights.
In de constructiefase wordt er een complete oplossing gevonden, waarbij alleen gekeken wordt naar de hard constraints. De uiteindelijke output van deze fase is dus een rooster waarin geen schending van de hard constraints zitten. Vervolgens wordt de kwaliteit van het rooster verbeterd in de hieropvolgende fasen, zoals te lezen in paragraaf 5.1. Hiervoor is het nodig om verschillende roosters met elkaar te kunnen vergelijken. De parameters uit de categorie *Solution Comparator Weights* zijn hiervoor ontworpen. De kwaliteit van een oplossing wordt gegeven door de waarde van de doelfunctie, een gewogen som van alle soft constraints. Hierbij is het mogelijk om de relevantie van de verschillende soft constraints aan te geven door het meegeven van gewichten. Deze gewichten zijn weergegeven in tabel 6. Net als bij de *Lecture Selection* worden al deze parameters ingesteld op waarden tussen de -2.0 en 2.0.

Parameter	**Uitleg**
Comparator.UselessSlotWeight	Gewicht van een tussenuur
Comparator.PerturbationPenaltyWeight	Gewicht van het verschil tussen de initiële oplossing
Comparator.DeptSpreadPenaltyWeight	Gewicht van goede verdeling van uren over verschillende afdelingen in de tijd
Comparator.SpreadPenaltyWeight	Gewicht van goede verdeling van uren binnen een afdeling
Comparator.HardStudentConflictWeight	Gewicht van studentconflicten
Comparator.StudentConflictWeight	Gewicht van studentenconflicten
Comparator.CommitedStudentConflictWeight	Gewicht van studentenconflicten bij van tevoren toegekende waarden bij variabelen
Comparator.TooBigRoomWeight	Gewicht van een te groot lokaal (meer dan 50% niet gebruikt)
Comparator.ContrPreferenceWeight	Gewicht van voorkeur constraint
Comparator.RoomPreferenceWeight	Gewicht van lokaalvoorkeur
Comparator.StudentLecturePreferenceWeight	Gewicht van lesvoorkeuren van studenten

Tabel 6: Parameters Solution Comparator Weights

Placement Selection.
De roosters zullen op een slimme manier veranderd moeten worden. Dit wordt gedaan door de waarde van de variabele die de meeste verbetering op kan leveren bij verandering te veranderen. Het kiezen van deze variabele die de meeste soft constraints schendt, gebeurt aan de hand van de *Solution Comparator Weights*. Vervolgens moet een keuze gemaakt worden voor de nieuwe waarde van de desbetreffende variabele. Dit gebeurt aan de hand van drie verschillende levels die de mogelijke waardes voor de variabele lexicografisch ordenen. De uiteindelijke keuze voor de waarde heeft hierbij de laagste level1-som, de laagste level2-som binnen de andere waardekeuzes die tevens een lage level1-som hadden en weer de laagste level3-som onder deze waarden. De levels maken gebruik van een gewogen som, waarbij weer parameters invloed hebben op de gewichten binnen de som. De gewichten kunnen per level anders zijn. Zo kunnen belangrijke zaken in level 1 al bekeken worden terwijl minder belangrijke zaken dan nog buiten beschouwing gelaten worden. De parameters uit de categorie *Placement Selection* zijn in tabel 7 weergegeven. Van elk van deze parameters zijn er dus 3, behorend bij de levels.

Neighbour Selection.
De parameters in de categorie *Neighbour Selection* worden gebruikt tijdens de Hill Climbing fase en/of de Simulated Annealing fase. Voor het vinden van een nieuwe oplossing wordt er gekeken naar oplossingen die "in de buurt van" de oude oplossing liggen. Tijdens dit proces wordt er gebruik gemaakt van de parameters uit tabel 8.

Same Subpart Balancing en Departmental Balancing.
De volgende parameters, zie tabel 9 en 10, worden gebruikt om lesuren goed te verdelen over verschillende afdelingen binnen een onderwijsinstelling. Er wordt voorkomen dat favoriete uren helemaal volgepland kunnen worden door een bepaalde afdeling, door gebruik te maken van een maximaal aantal in te plannen lessen per tijdslot. Dit is in eerste instantie gelijk aan *Spread.Spreadfactor*, die gelijk is aan het aantal in te plannen lessen gedeeld door het aantal tijdsloten per dag. Eventueel kan tijdens het proces deze limiet verhoogd worden als het leidt tot conflicten middels de tweede parameter.

Parameter	**Uitleg**
Parameters die niet voorkomen in de levels	
Placement.RandomWalkProb	Kans dat een random waarde gekozen wordt voor de variabele
Placement.MPP_InitialProb	Kans dat de initiële waarde (mits bestaand) gekozen wordt
Parameters die voorkomen in level 1, 2 en 3	
Placement.NrAssignmentsWeight	Aantal variabelen waar al een waarde aan toegekend is
Placement.MPP_DeltaInitialAssignmentWeight	Verschil tussen de initiële oplossing
Placement.UselessSlotsWeight	Gewicht van tussenuren
Placement.DeptSpreadPenaltyWeight	Gewicht van goede verdeling van uren over verschillende afdelingen in de tijd
Placement.ThresholdKoef	Grens waarbij de oplossing wordt meegenomen naar het volgende level, niet gebruikt in level 3 dus
Placement.SpreadPenaltyWeight	Gewicht van goede verdeling van uren binnen een afdeling
Placement.NrConflictsWeight	Aantal conflicten
Placement.WeightedConflictsWeight	Aantal conflicten, gewogen door voorkoming aan de hand van CBS
Placement.NrPotentialConflictsWeight	Aantal conflicten dat ontstaat als deze waarde aan de variabele wordt toegekend
Placement.NrHardStudConfsWeight	Aantal conflicten
Placement.NrStudConfsWeight	Aantal studentconflicten
Placement.NrCommitedStudConfsWeight	Aantal studentenconflicten bij van te voren toegekende variabelen
Placement.TooBigRoomWeight	Gewicht van een te groot lokaal
Placement.DeltaTimePreferenceWeight	Gewicht van urenverdeling
Placement.ConstrPreferenceWeight	Gewicht van docentvoorkeur
Placement.RoomPreferenceWeight	Gewicht van lokaalvoorkeur

Tabel 7: Gewichten gebruikt bij lexicografische ordening uit categorie Placement Selection

Neighbour.SuggestionProbability
Neighbour.SuggestionTImeout
Neigbour.SuggestionDepth
Neighbour.SuggestionProbabilityAllAssigned

Tabel 8: Parameters Neighbour Selection

Spread.Spreadfactor
Spread.Unassignments2Weaken

Tabel 9: Parameters Same Subpart Balancing

DeptBalancing.SpreadFactor
DeptBalancing.Unassignments2Weaken

Tabel 10: Parameters Departmental Balancing

Search Intensification.
De parameters uit tabel 11 zijn interne instellingen voor het algoritme. Zo kan het aantal iteraties

worden opgegeven waarna terug gegaan wordt naar de beste tot dan toe gevonden oplossing zodat er niet te lang doorgezocht wordt vanaf een oplossing die in een "slecht" stuk van de oplossingsruimte ligt.

Parameter	**Uitleg**
SearchIntensification.IterationLimit	Aantal iteraties waarna een herstart plaatsvindt vanaf de tot nu toe best gevonden oplossing
SearchIntensification.ResetInterval	Aantal herstarts waarna het iteratielimiet verhoogd wordt
SearchIntensification.MultiplyInterval	Verhoging van het iteratielimiet
Searchintensification.Multiply	Aantal herstarts waarna het CBS geleegd wordt

Tabel 11: Parameters Search Intensification

Perturbation Penalty.
De parameters binnen deze categorie worden gebruikt om een al gemaakt rooster te kunnen wijzigen, naar aanleiding van externe veranderingen. Dit kan bijvoorbeeld een onverwachte lokaalwijziging zijn of een verandering in (de beschikbaarheid van) het personeel. Doel is nu om het rooster, dat opnieuw gemaakt moet worden door de verandering, zo min mogelijk te laten verschillen van het oude. Belangrijk hierbij is het feit of het rooster al gepubliceerd is of niet. Er zijn twee gevallen mogelijk;

- **Het rooster is al gepubliceerd.** Belangrijk is dat het aantal (extra) studentenconflicten geminimaliseerd wordt, maar dat daarnaast zo min mogelijk mensen last hebben van roosterwijzigingen. Dit houdt in dat wisselingen van lestijden en lokalen geminimaliseerd worden. Hiernaast blijft het belangrijk om het aantal schendingen van soft constraints te minimaliseren.
- **Het rooster is nog niet gepubliceerd.** Belangrijk is om het aantal studentenconflicten en schendingen van soft constraints te minimaliseren. Daarnaast kan het handig zijn om het verschil in lestijden te minimaliseren. In feite kan er gewoon een nieuw rooster gemaakt worden.

Deze parameters uit tabel 12 worden nu nog niet gebruikt in Eduflex en hoeven dus nog niet onderzocht te worden.

Perturbations.DeltaStudentConflictsWeight
Perturbations.NewStudentConflictsWeight
Perturbations.DifferentPlacement
Perturbations.AffectedStudentWeight
Perturbations.AffectedInstructorWeight
Perturbations.DifferentRoomWeight
Perturbations.DifferentBuildingWeight
Perturbations.DifferentTimeWeight
Perturbations.DifferentDayWeight
Perturbations.DifferentHourWeightv Perturbations.TooFarForInstructorsWeight
Perturbations.TooFarForStudentsWeight
Perturbations.DeltaInstructorDistancePreferenceWeight
Perturbations.DeltaRoomPreferenceWeight
Perturbations.DeltaTimePreferenceWeight
Perturbations.AffectedStudentByTimeWeight
Perturbations.AffectedInstructorByTimeWeight
Perturbations.AffectedStudentByRoomWeight
Perturbations.AffectedInstructorByRoomWeight
Perturbations.AffectedStudentByBldgWeight
Perturbations.AffectedInstructorByBldgWeight

Tabel 12: Parameters Perturbation Penalty

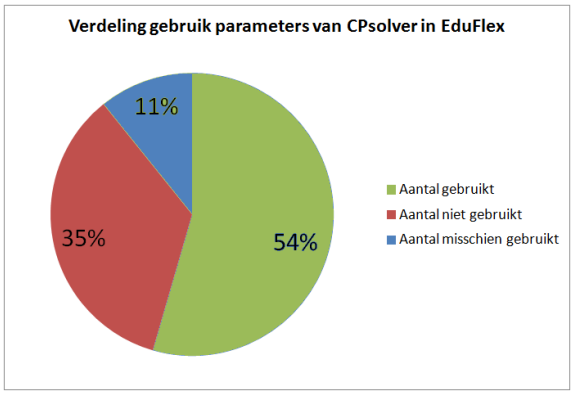
5.5.3 Onderzoek Müller naar parameters

CPsolver is ontworpen door Tomáš Müller en gebruikt bij een groot roosterprobleem aan de Purdue University. Ook hierbij was het nodig om de parameters goed in te stellen, hiervoor is dus ook onderzoek verricht door Müller. In eerste instantie zal in dit onderzoek ook uitgegaan worden van de resultaten die uit het onderzoek van Müller gekomen zijn. Een kanttekening die hierbij gemaakt moet worden is dat er uiteraard verschillen zitten tussen een roosterprobleem aan de Purdue University en roosterproblemen aan ROC's in Nederland. Hiernaast moet rekening gehouden worden met het feit dat Eduflex niet alle mogelijkheden van CPsolver benut, wat eventueel ook van invloed kan zijn op de resultaten. Hieronder volgt een opsomming van de resultaten van Müller:

- De solver geeft goede resultaten binnen redelijke tijd (30 minuten) als de waarde van *Placement.WeightedConflictsWeight* gelijk is aan de helft van de som van alle waarden van de parameters uit level 1.
- De waarden van de parameters uit level 2 komen vaak overeen met de gewichten die gebruikt worden om de oplossingen met elkaar te vergelijken. Hiermee worden de parameters bedoeld die gebruikt worden als nog geen complete oplossing gevonden is (categorie *Lecture Selection*) alsmede de parameters die gebruikt worden bij het verbeteren van de oplossing (categorie *Solution Comparator Weights*). Een overzicht van alle parameters die dezelfde waarde behoren te hebben is weergegeven in appendix D.4.

5.5.4 Benutting parameters

Niet alle parameters van CPsolver worden gebruikt binnen Eduflex. Aangezien onbekend is hoe vaak een bepaalde parameter aangeroepen wordt binnen CPsolver valt niet te zeggen welk deel van CPsolver door het wegvallen van dergelijke parameters onbenut blijft. Om toch een idee te krijgen is in de figuur hieronder een verdeling te zien waarin is aangegeven hoeveel parameters Eduflex al dan niet gebruikt. Belangrijk hierbij is dat deze figuur dus aangeeft óf een parameter gebruik wordt binnen Eduflex, het kan best zijn dat de desbetreffende parameters niet worden meegenomen in het onderzoek (bijvoorbeeld als het alleen de plaats van een datalijst aangeeft).



Figuur 4: Benutting parameters

6 Parameteronderzoek

6.1 Onderzoeksplan

Er is uiteengezet hoe het algoritme binnen CPsolver werkt en waar hierbij de parameters van toepassing zijn. Met deze kennis kan er nu gezocht worden naar de optimale instellingen van de parameters naar aanleiding van de wensen van de klant. De parameters zullen hiervoor opgedeeld worden naar aanleiding van hun functie binnen het algoritme. De oplossingen bij eenzelfde dataset zullen worden vergeleken aan de hand van een apart opgestelde doelfunctie.

6.1.1 Inputfile

Voor dit onderzoek wordt gebruik gemaakt van een inputfile gegenereerd door Unitime [Uni13], aangezien een inputfile gemaakt door Eduflex vanuit Paralax niet beschikbaar was. De dataset van Unitime betreft een deelprobleem van het roosterprobleem van de universiteit van Purdue. Purdue is een publieke universiteit met 39.000 studenten en biedt veel verschillende programma's aan. In totaal worden er 9.000 vakken gegeven in 570 lokalen. Dit betekent dat er over de gehele universiteit ongeveer 259.000 individuele verzoeken voor vakken ingeroosterd moeten worden. Het probleem is opgedeeld in kleinere roosterproblemen, namelijk via de verschillende afdelingen. De afdelingen werken vaak onafhankelijk van elkaar, de lessen worden in andere lokalen en door andere docenten gegeven.

Voor dit onderzoek zal gebruik gemaakt worden van de dataset van het Mathematics & Statistics Department. In tabel 13 staat een overzicht van de dataset gegevens. Er is gekozen voor een kleinere dataset om zodoende de tijd, die het runnen van het programma kost, te overzien.

Aantal lokalen	80
Aantal lessen	1.132
Al ingeroosterde lessen	692
Aantal studenten	11.192
Aantal docenten	161
Extra voorwaarden	457

Tabel 13: Gegevens dataset Unitime

6.1.2 Doelfunctie

Voor dit onderzoek is het van belang om de juiste parameterinstellingen te vinden van CPsolver. Hierbij moeten de wensen van de klant worden meegenomen in de instellingen, om zodoende een rooster te genereren dat zo dicht mogelijk bij de wensen van de klant ligt. Om nu de verschillende instellingen met elkaar te vergelijken, zullen de bijbehorende roosters gewaardeerd moeten worden. Dit kan door middel van een doelfunctie, die aan de hand van de output van CPsolver een waarde toekent aan elk rooster. Over de doelfuncties zal meer te lezen zijn in paragraaf 6.2. Belangrijke kanttekening die hierbij gemaakt moet worden is dat de daar genoemde doelfuncties niet op basis van literatuur gevormd zijn. De doelfunctie in het uiteindelijk ontwerp zal moeten worden aangepast aan de wensen van de klant. Er zal waarschijnlijk verder onderzoek gedaan moeten worden naar de manier waarop dit goed gedaan kan worden.

6.1.3 Programma

Het vinden van de juiste instellingen zal geautomatiseerd worden in een programma geschreven in C#. Het doel van het programma is om een "perfecte" configfile te creëren, waarmee CPsolver de roosters voor de klant genereert. De insteek van het programma zal zijn om een willekeurig gekozen parameter te verhogen of te verlagen met een van te voren opgegeven stapgrootte. Indien dit een verbetering oplevert van de opgegeven doelfunctiewaarde, zal deze verandering worden aangenomen.

Zo niet, dan wordt deze bewerking tijdelijk uit de set van mogelijkheden gehaald. In beide gevallen zal hierna opnieuw geprobeerd worden om andere parameters te verhogen of te verlagen. Wanneer er geen mogelijkheden meer zijn om een betere parameterinstelling te vinden, zal het programma eindigen en is de geoptimaliseerde configfile gevonden. De precieze werking van het programma zal worden uitgediept in paragraaf 6.4.

6.1.4 Runtijd

Er zijn twee verschillende instellingsmogelijkheden om CPsolver te laten stoppen, namelijk op basis van een opgegeven tijdslimiet of indien er een complete oplossing gevonden is. De complete oplossing wordt gevonden in de eerste fase, waarbij enkel de parameters uit de categorie *Lecture Selection* van toepassing zijn. Om deze parameters te onderzoeken, zal CPsolver dus zo ingesteld worden dat het stopt wanneer er een complete oplossing gevonden is.

Om onderzoek te verrichten naar de parameters die invloed hebben op de verbeterfase, zal CPsolver moeten termineren na een bepaald tijdslimiet. Het is wenselijk voor de bewerkbaarheid van het onderzoek om dit tijdslimiet behapbaar te houden. Om die reden staat in tabel 14 het aantal verbeteringen dat CPsolver nog vindt na bepaalde tijdsbestekken weergegeven. Hieruit blijkt dat na 30 minuten niet veel verbeteringen meer behaald worden. Een runtijd van 30 minuten zal een goed rooster opleveren binnen een behapbare tijd.

Runtijd (in minuten)	**Aantal verbeteringen vanaf 15 minuten**	**Aantal verbeteringen vanaf 30 minuten**	**Aantal verbeteringen vanaf 60 minuten**	**Tijdstip laatste verbetering (min. na start)**
30	0	-	-	11.86
30	0	-	-	13.78
30	1	-	-	19.28
60	8	0	-	20.16
60	0	0	-	13.92
60	0	0	-	9.36
120	4	3	1	89.27
120	6	4	3	115.19
120	7	4	3	79.67
180	1	0	0	18.01
180	8	2	2	133.86
180	5	4	2	147.44

Tabel 14: Tijdstatistieken

6.1.5 Te onderzoeken parameters

In paragraaf 5.5 is er inzichtelijk gemaakt welke verschillende categorieën parameters er zijn en heeft er een filtering plaatsgevonden welke parameters onderzocht gaan worden. Om deze overgebleven parameters te kunnen onderzoeken, zal er onderscheid gemaakt moeten worden naar de functies die de parameters hebben binnen het algoritme. Dit betekent dat er drie groepen ontstaan, namelijk;

1. **Parameters die betrekking hebben op de constructiefase.** Dit zijn de parameters behorende bij de categorie *Lecture Selection*.
2. **Parameters die betrekking hebben op de verbeterfase van het algoritme.** Deze parameters komen uit de categorieën *Solution Comparator Weights en Placement Selection*.
3. **Overige parameters.** Dit zijn parameters die geen directe betrekking hebben op de doelfunctie, er kan dus geen gewicht opgegeven worden. Hierbij behoren bijvoorbeeld de categorieën *Conflict-bases Statistics, Neighbour Selection, Same Subpart Balancing en Search Intensification.*

Aan de hand van deze groepering zullen er drie onderzoeken opgezet worden.

6.1.6 Oriëntatie onderzoek

Allereerst gaan er een aantal parameters, de overige parameters, onderzocht worden. De verwachting is dat deze parameters niet probleemafhankelijk zijn. Om hier meer vat op te krijgen, zullen de parameters met verschillende waardes gerund worden, waarna de bijbehorende oplossingen met elkaar zullen worden vergeleken aan de hand van de doelfunctiewaarde. Een overzicht van de parameters die op deze manier onderzocht worden, staat in tabel 16.

6.1.7 Onderzoek constructiefase

In de eerste fase van het algoritme wordt een complete oplossing gemaakt, dit wordt ook wel de constructiefase genoemd. Het doel van het onderzoek naar deze constructiefase is het vinden van de instellingen waarbij deze complete oplossing zo dicht mogelijk bij de wensen van de klant ligt. Hierbij wordt gekeken naar de parameterinstellingen voor de parameters uit de categorie *Lecture Selection*, de exacte lijst is te vinden in tabel E29.

Optimalisatie.
Om het zojuist opgestelde doel te behalen, wordt er allereerst aan de hand van de doelfunctie gezocht naar optimale parameterinstellingen voor de parameters uit de *Lecture Selection*. Het vinden van deze optimale instellingen gebeurt door middel van het geschreven programma. Dit proces vindt plaats met twee verschillende initiële configfiles, allereerst met de configfile met standaardinstellingen en vervolgens met een configfile waarbij alle te onderzoeken parameters op nul staan ingesteld. De verwachting is dat er uiteindelijk twee soortgelijke configfiles overblijven.

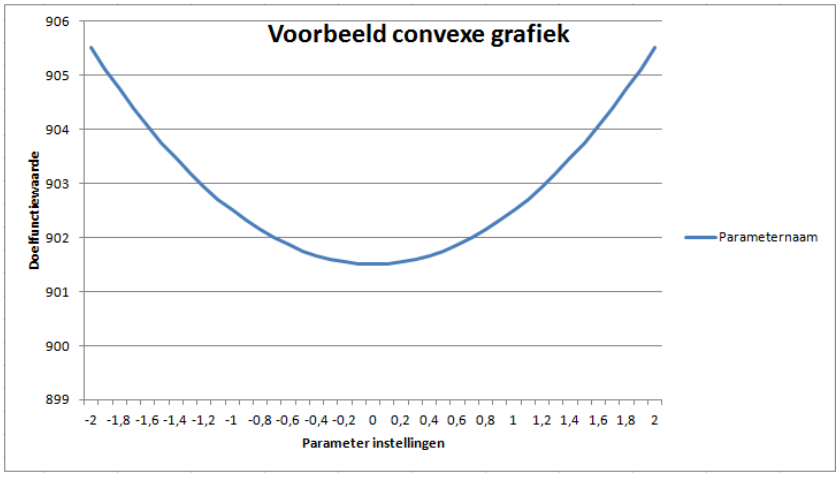
Tijdens het vinden van goede instellingen wordt bijgehouden hoe vaak een parameter voor verbetering zorgt. Indien er parameters zijn waarbij de kans op verbetering klein is, zullen deze uit de lijst in te stellen parameters verwijderd worden, om zodoende het programma sneller te laten werken.

Daarnaast kwam er uit het onderzoek van Müller dat een aantal parameters hetzelfde ingesteld moesten worden. Hieronder zaten ook de *Lecture Selection* parameters. Er zal dus tot slot worden gekeken of de andere parameters hieraan aangepast moeten worden. Dit gebeurt aan de hand van de doelfunctiewaarde van beide oplossingen.

Parameteranalyse.
Wanneer er een goede waarde voor alle parameters is gevonden in het optimalisatieproces, worden deze parameters per stuk onder de loep gelegd. Dit betekent dat een aantal waarden van een parameter doorlopen worden en de bijbehorende doelfunctiewaarden wordt bekeken. Deze waarden zijn gelijkmatig gekozen tussen -2.0 en 2.0 met een stapgrootte van 0.1. De verwachting is dat voor de parameters een beste instellingswaarde te zien is en dat de grafiek er convex uit zou moeten zien. In figuur 5 staat een voorbeeld van een dergelijke convexe grafiek.

6.1.8 Onderzoek verbeterfase

Nadat er een complete oplossing gevormd is, wordt er getracht de oplossing te verbeteren tijdens de verbeterfase. Voor deze fase staan de parameters uit de categorieën *Solution Comparator Weights* en *Placement Selection* centraal. Dit betekend dat er voor de verbeterfase nog 64 parameters van belang zijn. Het onderzoek naar de parameterinstellingen van deze fase zal veel tijd kosten, gezien het testen van een parameterinstelling in deze fase een half uur duurt. Om die reden zal er voornamelijk onderzoek verricht worden naar de overeenkomst tussen de parameter uit de constructiefase en de parameters uit de verbeterfase. Dit zal weer gebeuren in twee stappen, allereerst wordt er gekeken naar de optimalisatie en daarna naar de parameteranalyse.



Figuur 5: Voorbeeld van een convexe grafiek

Optimalisatie.
Voordat er onderzoek gedaan gaat worden naar de instellingen van de verbeterfase, zal er allereerst gekeken worden of het optimaliseren van de constructiefase zin heeft voor het verdere verloop van het proces. Hierbij wordt er gebruik gemaakt van de gevonden configfile uit het onderzoek constructiefase. In dit bestand wordt weer gezocht naar de optimale instellingen van de parameters uit tabel E31. Nu wordt echter de tijd voor het vinden van een oplossing op 30 minuten ingesteld in CPsolver. Het zoeken gebeurt, net als bij de constructiefase, met behulp van het ontwikkelde programma. Vervolgens worden de uitkomsten van de configfile met standaard instellingen, de configfile uit het onderzoek constructiefase en de met de zojuist beschreven methode gevonden configfile vergeleken.

Parameteranalyse.
Voor de onderzochte parameters uit de groep *Solution Comparator Weights* zal er per parameter apart bekeken worden of er een beste instellingsmogelijkheid voor is. Dit gebeurt door weer een aantal mogelijk parameterinstellingen te doorlopen en de bijbehorende doelfunctiewaarden te bekijken. Hierbij worden twee gevallen bekeken; de rest van de parameterwaarden op hun standaardwaarden of allemaal op 0. Verwacht is dat de ene parameter meer invloed zou kunnen hebben op de doelfunctie als andere gewichten op 0 gezet worden. De uitkomsten van dit deelonderzoek worden vergeleken met de uitkomsten van de parameteranalyse van de constructiefase. Hierdoor kan worden bekeken of de parameters uit beide groepen (*Lecture Selection* en *Solution Comparator Weights*) dezelfde eigenschappen hebben.

6.2 Doelfuncties

Zoals gezegd is het nodig voor de klant om zijn eisen door te geven aan het programma. Dit kan het best gebeuren aan de hand van schuifjes, aangezien uit het onderzoek van Triple A is gebleken dat onderwijsinstellingen dit een fijne manier vinden om hun voorkeuren door te geven. Binnen dit onderzoek wordt uitgegaan van drie verschillende aspecten bij het roosteren; de docent, de leerling en het management. De klant kan vervolgens middels deze schuifjes aangeven welke aspecten het belangrijkst gevonden moeten worden. Belangrijk is om op te merken dat het hierbij gaat om het relatieve belang van bijvoorbeeld het aspect leerling *ten opzichte van* de docenten en het management. Dit houdt in dat als er ingesteld wordt dat docent, leerling en management alle drie als niet erg belangrijk worden beschouwd dit op hetzelfde neerkomt als dat deze drie aspecten allemaal erg belangrijk worden geacht.

Allereerst zullen nu de mogelijke statistieken uit de outputfile van CPsolver benoemd worden waarmee gewerkt kan worden, hierbij zal ook worden aangegeven bij welke aspect ze horen. Hierna zullen drie

verschillende scenario's omschreven worden, waarbij ook een voorbeeld van een mogelijke doelfunctie gegeven wordt. Belangrijk hierbij is dat het een voorbeeld is, betere doelfuncties voor verschillende klanteisen zullen aan de hand van verder onderzoek opgesteld moeten worden.

6.2.1 Output CPsolver

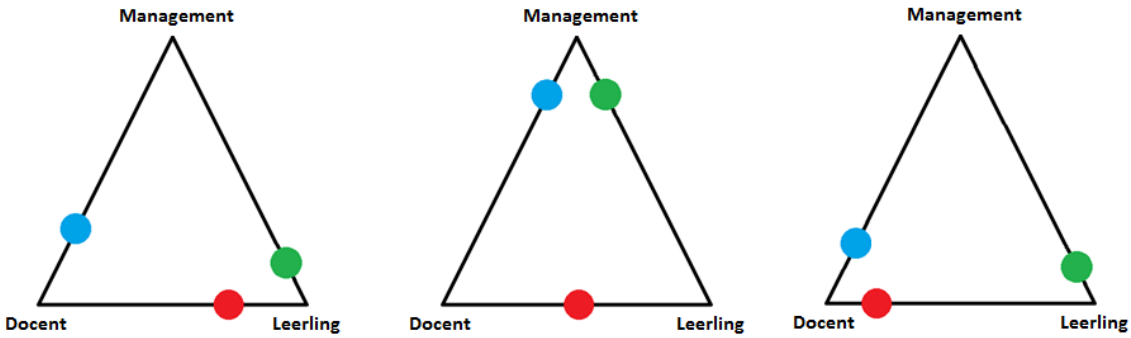
Voor de doelfunctie kan gebruik gemaakt worden van enkele gegevens die CPsolver over de gemaakte oplossing geeft. Dit is weergegeven in een Excel-bestand, dat gemakkelijk in te lezen is in C#. In tabel 15 zijn de te gebruiken waarden weergegeven. Belangrijk hierbij is dat niet alle waarden van belang zijn bij het onderzoek met de Eduflex-data, aangezien sommige aspecten niet door Eduflex meegenomen worden. Een voorbeeld hiervan zijn de waarden die horen bij *time preferences*. Wat op zal vallen is dat de indeling van de outputgegevens in de drie aspecten niet altijd eenduidig is. Zo wordt bij het tweede scenario bijvoorbeeld het aantal tussenuren ook vanuit managementoverwegingen zo laag mogelijk geprobeerd te houden, terwijl dit volgens de indicatietabel vooral voor de student belangrijk is.

Naam variabele	**Outputwaarde**	**Heeft invloed op aspecten**
	Assigned variables	n.v.t.
	Time [sec]	n.v.t.
A	Hard student conflicts	Student
B	Distance student conf	Student
C	Student conflicts	Student
D	Committed student conflicts	Student
E	All Student conflicts	Student
F	Time preferences	Docent/Management
G	Room preferences	Docent/Management
H	Useless half-hours	Student
I	Too big room Management	Docent/Management
J	Distribution preferences	Management
K	Back-to-back instructor pref.	Docent
L	Same subpart balancing penalty	Management
	Assigned variables max	n.v.t.
	Student enrollments	n.v.t.
	Student commited enrollments	n.v.t.
	All student enrollments	n.v.t.
	Time preferences min	n.v.t.
	Time preferences max	n.v.t.
	Room preferences min	n.v.t.
	Room preferences max	n.v.t.
	Useless half-hours max	n.v.t.
	Too big room max	n.v.t.
	Distribution preferences min	n.v.t.
	Distribution preferences max	n.v.t.
	Back-to-back instructor pref max	n.v.t.

Tabel 15: Te gebruiken outputgegevens vanuit CPsolver

6.2.2 Scenario 1 - Leerling centraal

Binnen dit scenario is het aspect waarop ingegaan wordt bij het roosteren de leerling. Getracht wordt om het rooster zo goed mogelijk aan te laten sluiten bij de wensen van de leerlingen. Dit houdt in dat als er een keuze gemaakt moet worden tussen een tussenuur voor een leerling of voor een docent, de docent het tussenuur krijgt toegewezen. Het management wordt minder belangrijk geacht. Dus eisen



Figuur 6: Voorbeeldscenario's 1, 2 en 3

voor goede lokaalbezetting kunnen in dit geval overboord gegooid worden als het tegenstrijdig is met de wensen van docent(en) en/of leerling(en).

$$\textbf{Mogelijke doelfunctie} = A + E + H + 0.1F + 0.5G + 0.5I$$

Hierin zijn de aspecten voor de leerling meegenomen (A,E,H) en daarna de aspecten voor de docenten (F,G,I), gewogen omdat dit als minder belangrijk wordt gezien.

6.3 Scenario 2 - Management centraal

Binnen dit scenario staat het management centraal. Dit houdt in dat tussenuren zoveel mogelijk vermeden worden en geprobeerd wordt om alle lessen rechtevenredig over de week te verdelen. Lokalen mogen niet onderbezet worden, zodat het eventueel mogelijk is om lege lokalen bijvoorbeeld te verhuren. Docent en leerling worden binnen dit scenario als gelijkwaardig beschouwd.

$$\textbf{Mogelijke doelfunctie} = 0.5F + G + I + J + L + 0.6H + 0.5E$$

6.3.1 Scenario 3 - Docent centraal

Bij dit laatste scenario staat de docent centraal. Geprobeerd wordt om bij het maken van het rooster de docent zoveel mogelijk tegemoet te komen. Dit houdt in dat tussenuren voor docenten zoveel mogelijk geminimaliseerd worden en dat afstanden tussen opeenvolgende lessen ook zo klein mogelijk gemaakt worden.

$$\textbf{Mogelijke doelfunctie} = 0.1F + G + I + 0.3E + 0.3H$$

6.3.2 Gebruik van de doelfuncties

In het onderzoek zal aanvankelijk de voorbeeld doelfunctie behorend bij het eerste scenario gebruikt worden om de gemaakte roosters te beoordelen. Later kunnen andere doelfuncties gebruikt worden, waarbij de methodes verder niet veranderen. Zoals eerder al genoemd zal de te gebruiken doelfunctie uiteindelijk met behulp van verder onderzoek bepaald moeten worden.

6.4 Instellingenoptimalisator

Hier zal de werking van de instellingenoptimalisator - het voor dit onderzoek geschreven programma om een optimum van de parameterinstellingen te vinden - uitgelegd worden. De instellingenoptimalisator voert CPsolver automatisch uit met verschillende configuratie bestanden. Door de configfiles op een slimme manier aan te passen kan in niet al te lange tijd een lokaal optimale verzameling parameters

```
while doorgaan met verbeteren do
    kies een parameter uit de lijst met aan te passen parameters;
    pas de gekozen parameter aan in de huidige configfile;
    run CPsolver met huidige configfile;
    lees statistiekenbestand uit en genereer doelfunctiewaarde;
    if doelfunctiewaarde beter then
        update beste configfile naar huidige versie;
        zet alle parameters in lijst aan te passen parameters;
    else
        ga terug naar vorige beste versie;
        haal aangepaste parameter uit lijst aan te passen parameters;
    end
end
```

Codeblok 1: Basis van de instellingenoptimalisator

verkregen worden bij een gegeven doelfunctie. In codeblok 1 is het programma kort weergegeven in pseudo code.

Hierbij worden de aan te passen parameters in het te optimaliseren configuratiebestand herkenbaar gemaakt door er "##**"voor te zetten, waarbij de sterretjes voor cijfers staan zodat telkens een tweecijferig getal de index van de parameter representeert. De optimalisator kiest een random index en richting en past de bijbehorende parameter aan in de configfile. Dit wordt gedaan door de parameter met een vooraf ingestelde stapgrootte te verhogen of verlagen. Vervolgens wordt na het runnen van CPsolver gekeken naar de statistiekbestanden die CPsolver automatisch aanmaakt. De statistieken uit deze bestanden worden met behulp van de vooraf opgestelde doelfunctie gewogen en opgeteld, waarmee één waarde aan het gegenereerde rooster toegekend wordt.

Als deze waarde beter is dan het vorige beste resultaat gaat de instellingenoptimalisator verder vanaf de nieuwe configfile. Mocht er geen verbetering zijn dan wordt teruggegaan naar de vorige configuratie-instellingen en wordt onthouden dat de net aangepaste parameter in deze richting geen verbetering opgeleverd heeft. Dit gebeurt aan de hand van een interne lijst met daarin twee keer alle parameterindexen: één keer voor verhogen en één keer voor verlagen. Telkens als een parameter geen verbetering oplevert wordt deze parameter met de desbetreffende richting uit de lijst gehaald. Wanneer een verbetering bereikt wordt zou het kunnen dat parameters die eerder geen verbetering opleverden nu wél voor een betere doelfunctiewaarde kunnen zorgen. De instellingenoptimalisator zal daarom na elke gevonden verbetering alle parameterindexes terugzetten in de lijst.

Tijdens bovenstaand beschreven proces worden twee tekstbestanden aangemaakt. In het eerste bestand wordt voor elke parameter bijgehouden hoe vaak wel en niet een verbetering gevonden is door deze parameter te verhogen dan wel te verlagen. Dit bestand blijft tussen verschillende runs van de instellingenoptimalisator behouden en kan apart op nul gezet worden. Het tweede bestand is een logbestand waarin bijgehouden wordt welke parameters veranderd zijn en welke doelfunctiewaarden met deze veranderingen bereikt worden. Vlak voordat de instellingenoptimalisator stopt wordt in dit tekstbestand nog aangegeven welke oplossing als beste gevonden is, met vermelding van bijbehorende configfile en doelfunctiewaarde.

Er kunnen twee redenen zijn waardoor het programma stopt. Indien geen enkele parameterverandering meer een verbetering oplevert aan de huidige configuratefile (de lijst met aan te passen parameters is leeg), is een lokaal minimum bereikt en zal niet verder gezocht worden. Er wordt ook gestopt wanneer een vooraf ingesteld aantal pogingen gedaan is. De eerste stopconditie is eigenlijk de gewenste omdat dan echt een lokaal minimum gevonden is. Als door de tweede stopconditie gestopt wordt is eigenlijk alleen nog maar een stap in de richting van een lokaal minimum gezet; het programma zou nog een keer gerund kunnen worden met het gegenereerde configuratiebestand om meer verbetering te boeken.

N.B.

Na gebruik van dit programma en na het vaker gebruiken van CPsolver is gebleken dat de output van CPsolver erg stochastisch is. Hierover is meer te lezen in paragraaf 7.4. Door deze bevindingen kan in twijfel worden getrokken of het verantwoord is om op basis van een enkele outputwaarde die beter is de instellingen in de configfile te behouden. Het programma zou op dit punt dus verbeterd kunnen worden, zie tevens paragraaf 7.4.

7 Resultaten

In dit hoofdstuk zullen de resultaten van de drie deelonderzoeken zoals beschreven in het onderzoeksplan worden gegeven. Hiertoe zal eerst kort beschreven worden wat er precies is gedaan, welke instellingen gebruikt zijn waarna de resultaten worden weergegeven. Tenslotte zullen per deelonderzoek de resultaten kort besproken worden en enkele conclusies worden getrokken. Uiteindelijk zal nog kort iets worden gezegd over de stochastiek van de output van CPsolver en zullen enkele aanbevelingen worden gedaan om dit mee te nemen in de instellingenoptimalisator.

7.1 Oriëntatieonderzoek

Er zijn enkele parameters die kort onderzocht worden op hun invloed binnen het algoritme. Een overzicht van deze parameters is te vinden in tabel 16. Hierbij is tevens de waarde gegeven van de parameter in de standaard configfile van CPsolver en de waardes die gebruikt zijn om de invloed te onderzoeken. In het onderzoek is steeds van de standaard configfile uitgegaan, waarbij één parameterwaarde veranderd is.

Hierbij is de parameter *ConflictStatistics.AgeingHalftime* niet veranderd, aangezien een keuze gemaakt moet worden tussen de twee verschillende methoden voor het CBS. Hierbij is ervoor gekozen om de waarden in het CBS te laten verouderen per iteratie, dus om gebruik te maken van *ConflictStatistics.Ageing*.

In totaal zijn er dus veertig verschillende configfiles gebruikt bij hetzelfde probleem. Bij de test is gebruik gemaakt van de inputfile van Unitime en is er steeds een runtijd van een half uur gebruikt. De inhoud van deze file is beschreven in paragraaf 6.1. Hierbij is gebruik gemaakt van drie verschillende computers. De reden hiervoor is tijdsefficiëntie, het kost erg veel tijd om CPsolver te runnen. De verkregen resultaten zijn te vinden in figuur E29 in de appendix.

7.1.1 Bespreking resultaten

Wat als eerste opvalt is dat de doelfunctiewaarde die verkregen wordt bij de standaard configfile erg verschillend is bij de drie verschillende computers. Dit kan twee oorzaken hebben; allereerst is het mogelijk dat de runtijd toch te kort is voor de grootte van het probleem, waardoor een snellere computer een veel betere oplossing zal vinden. Dit komt doordat extra iteraties minder verbetering op gaan leveren naarmate dichter bij de optimale configuratie gezocht wordt. Deze invloed zal bij langere runtijden een kleinere invloed gaan hebben. Een tweede oorzaak kan de stochastiek binnen CPsolver zijn; hierover is meer te vinden in paragraaf 7.4.

De verwachting was dat de onderzochte parameters weinig verbetering op zouden leveren bij verandering van waarde, omdat de parameters minder probleemspecifiek zijn en al uitvoerig onderzocht waren door Müller. In de output is dit ook terug te zien, slechts één van de 40 variaties heeft een significante verbetering opgeleverd ten opzichte van de standaard configfile op *dezelfde* computer. Dit was het verhogen van de parameter *SearchIntensification.Multiply*. Omdat naast het feit dat de parameters dus blijkbaar al redelijk goed zijn ingesteld het ook lastiger is om deze parameters met behulp van een programma aan te passen, is gekozen eerst andere groepen parameters op een automatische manier te optimaliseren. Vanuit de uitkomst uit dat onderzoek zal uiteindelijk nog extra gekeken kunnen worden of de hierboven besproken parameters nog verbetering op kunnen leveren.

7.2 Constructiefase

In dit deelonderzoek is gekeken naar de eerste fase van het algoritme van CPsolver, waarbij een complete oplossing wordt gegenereerd. Het doel was om de eerstgevonden complete oplossing aan de hand van verschillende instellingen van de parameters zo goed mogelijk te maken, gewaardeerd aan de hand van doelfunctie 1. De instellingen die gebruikt zijn tijdens het onderzoek zijn in tabel E30 in de appendix weergegeven. In de appendix is tevens een overzicht gegeven van de onderzochte parameters.

Parameter	**Standaard waarde**	**Verandering 1**	**Verandering 2**
Lecture.NrAssignmentsWeight	10.0	0	20
Placement.NrConflictsWeight1	10.0	0	20
Placement.WeightedConflictsWeight1	20.0	10	30
Lecture.DomainSizeWeight	30.0	20	40
Lecture.InitialAssignmentWeight	20.0	10	30
Lecture.SelectionSubSetMinSize	10	5	15
Lecture.SelectionSubSetPart	0.2	0.8	-
ConflictStatistics.Ageing	1.0	0.5	-
ConflictStatistics.AgeingHalfTime	0	-	-
Spread.SpreadFactor	1.2	2	-
Spread.Unassignments2Weaken	50	20	100
SearchIntensification.IterationLimit	100	50	200
SearchIntensification.ResetInterval	5	10	-
SearchIntensification.MultiplyInterval	2	1	5
SearchIntensification.Multiply	2	1	5
Neighbour.SuggestionProbability	0.1	0.5	-
Neighbour.SuggestionTimeout	500	100	800
Neighbour.SuggestionDepth	4	3	5
Neighbour.SuggestionProbabilityAllAssigned	0.5	0,1	0,9
DeptBalancing.SpreadFactor	1.2	2	-
DeptBalancing.Unassignments2Weaken	0	1	-
Lecture.RandomWalkProb	1.0	0,5	0,9
Placement.RandomWalkProb	0.00	0,1	0,5
Placement.MPP_InitialProb	0.20	0,1	0,5

Tabel 16: Parameters die handmatig onderzocht zijn

7.2.1 Resultaten

In tabel 17 en 18 zijn de resultaten weergegeven van het runnen van CPsolver met het probleem van Unitime en met de gevonden configuratiefile. De 'standaard' configuratiefile die hierin genoemd wordt, is de file zoals die op de site van Unitime te vinden is. De rij 'lecture' geeft de waarden weer behorend bij het door de instellingenoptimalisator gegenereerde configbestand. Bij 'Müller' tenslotte zijn de waarden uit dat configuratiebestand eerst volgens de "uitkomsten" van het onderzoek van Müller aangepast en is vervolgens daarmee CPsolver gedraaid. Weergegeven zijn de waarden van doelfunctie 1, dit is de doelfunctie waarop geoptimaliseerd is, alsmede het aantal hard student conflicts, dat natuurlijk zo laag mogelijk zou moeten zijn. Verder is in de tabel te zien hoeveel verandering de aangebrachte veranderingen opleveren aan de doelfunctie, telkens relatief ten opzichte van de standaardfile. In figuur 7 zijn van alle runs die gedaan zijn met de instellingenoptimalisator[2] statistieken weergegeven. Hierin is te zien hoe vaak het verlagen of verhogen van een parameter een positieve invloed heeft gehad op het resultaat, uitgedrukt als deel van het aantal pogingen tot die verandering.

De uitkomsten van de parameteranalyse zijn voor twee parameters weergegeven in figuren 8 en 9. De resultaten zagen er bij aanpassing van alle elf de parameters soortgelijk uit.

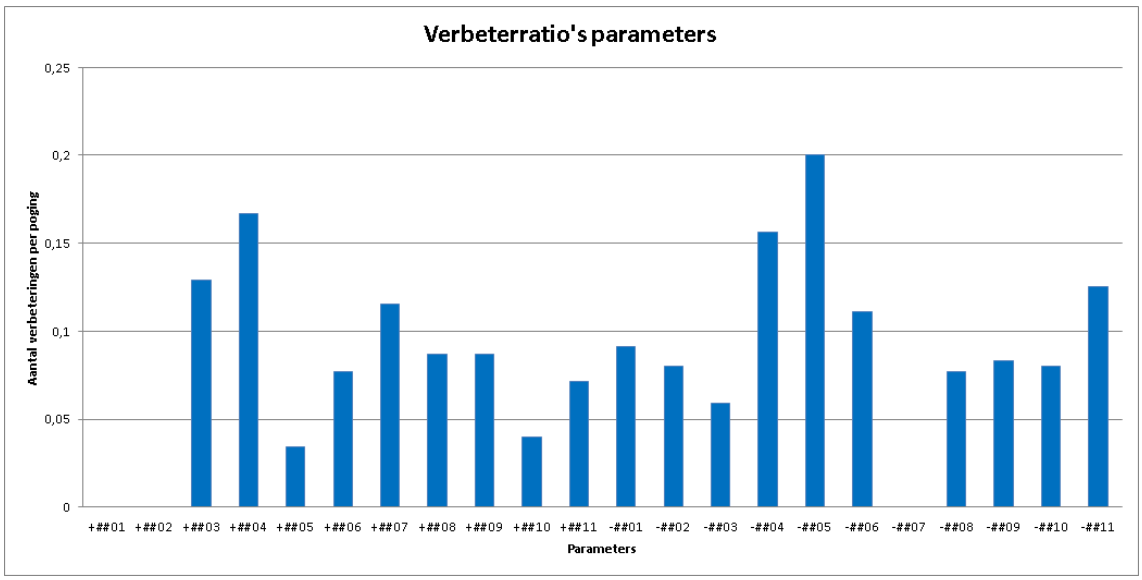
2. Dit zijn de runs zoals hierboven beschreven maar ook nog enkele testruns

	Computer 1			Computer 2			Computer 3		
Gebruikte configfile	Df. waarde	# hard student con-flicts	Δ	Df. waarde	# hard student con-flicts	Δ	Df. waarde	# hard student con-flicts	Δ
Standaard	980.4	17		1027.1	14		936.1	13	
Lecture	996.3	13	+2%	970	9	-4%	943.9	11	+1%
Müller	1099.8	24	+12%	1167.1	19	+1%	1247.6	40	+33%

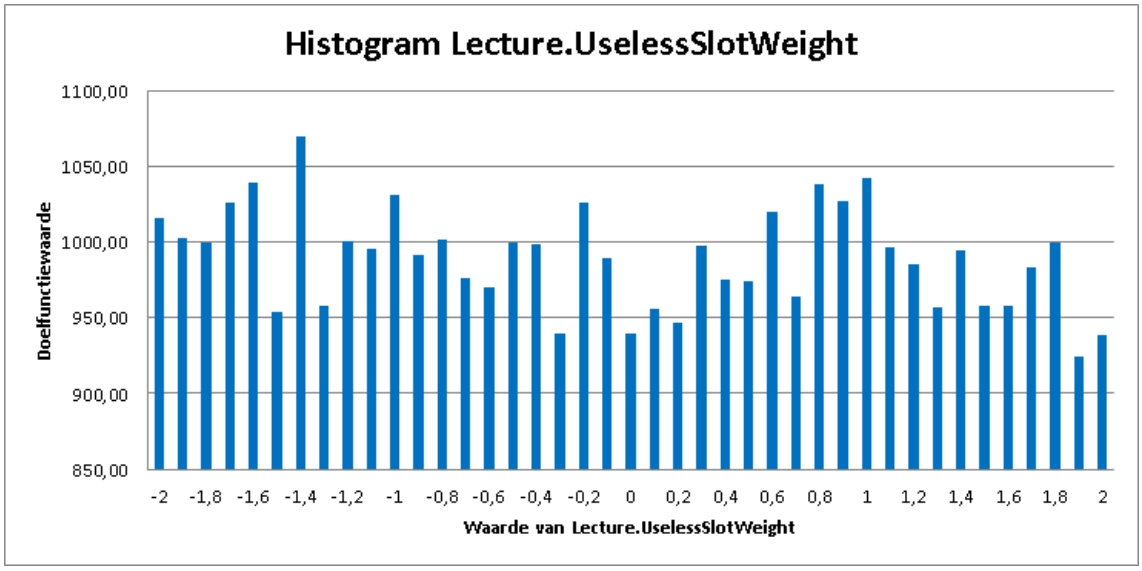
Tabel 17: Output CPsolver met output van instellingenoptimalisator gerund met standaard configfile waarbij Lecture parameters op nul zijn gezet. Hierbij is Δ het verschil in doelfunctiewaarde met de standaard configfile.

	Computer 1			Computer 2			Computer 3		
Gebruikte configfile	Df. waarde	# hard student con-flicts	Δ	Df. waarde	# hard student con-flicts	Δ	Df. waarde	# hard student con-flicts	Δ
Standaard	980.4	17	-	1027.1	14	-	936.1	13	-
Lecture	903.3	14	-8%	955	14	-7%	921.7	13	-2%
Müller	970.9	26	-1%	1190	14	+16%	1005	12	+7%

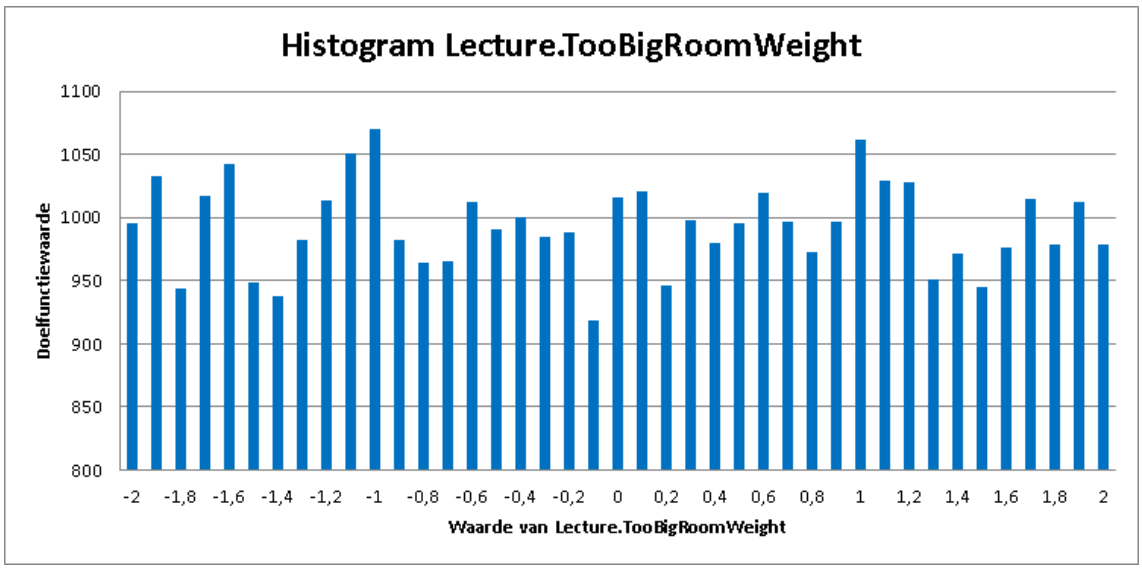
Tabel 18: Output CPsolver met output van instellingenoptimalisator gerund met standaard configfile. Hierbij is Δ het verschil in doelfunctiewaarde met de standaard configfile.



Figuur 7: Bijgehouden verbeteringen van parameters tijdens verloop instellingenoptimalisator



Figuur 8: Gemiddelde doelfunctiewaarden over 3 runs bij verschillende waarden van UselessSlotWeight



Figuur 9: Gemiddelde doelfunctiewaarden over 3 runs bij verschillende waarden van TooBigRoomWeight

7.2.2 Bespreking resultaten

Uit bovenstaande resultaten volgt dat het uitmaakt of bij gebruik van de instellingenoptimalisator begonnen wordt met de waarden vanuit de standaardfile of vanaf nul. Als vanaf de nulwaarden wordt begonnen geeft de instellingenoptimalisator een configfile die ofwel leidt tot slechtere resultaten of tot een erg geringe verbetering. Wel is duidelijk te zien dat als de outputfile wordt aangepast aan de suggesties van Müller, dit een nog slechter resultaat geeft.

Als echter wordt uitgegaan van de standaard configfile, waarbij de parameters al redelijk bij de goede waarden starten omdat deze zijn geconfigureerd door Müller, geeft de instellingenoptimalisator een gemiddelde verbetering van vijf procent. Dit zou een groot verschil kunnen maken op grotere onderwijsinstellingen. Wel moet rekening gehouden worden dat het roosterproces van CPsolver stochastisch van aard is en dat deze vijf procent gemiddeld dus wat hoger of lager zou kunnen liggen. Zelfs als van de standaardwaarden wordt uitgegaan, is het echter slecht om vervolgens de parameters aan te passen aan de literatuureisen. Dit levert een verslechtering op ten opzichte van de output van de instellingen optimalisator maar ook nog ten opzichte van de standaardwaarden.

Uit het histogram in figuur 7 blijkt niet overduidelijk dat een deel van de parameters nooit verbetering op zal leveren bij verandering. Parameters 1, 2 en 7 geven weliswaar geen verbeteringen als ze in één richting veranderd worden, echter omdat ze de andere kant op wél soms een verbetering opleveren blijken ze wel invloed te hebben op de doelfunctie en kunnen ze dus niet in hun geheel weggelaten worden. Aan de hand van deze resultaten kunnen dus geen parameters uitgesloten worden om zo het zoekproces sneller te maken. Wel valt te zien dat sommige parameters meer invloed hebben dan anderen. Bij een uitgebreidere steekproef zou dit gebruikt kunnen worden om eerst deze parameters te optimaliseren en vervolgens pas te kijken naar de minder invloedrijke parameters.

Uit de grafieken van de parameteranalyse komt de convexe structuur waarop gehoopt was niet duidelijk naar voren. Aangezien in de resulterende grafieken geen overtuigende trend te vinden is, is voor één parameter (*Lecture.UselessSlotsWeight*) gekeken of verhoging van het aantal runs samenhangender resultaat oplevert. De resulterende grafiek bij gebruik van twintig runs in plaats van drie is te zien in figuur E30, te vinden in de appendix. Ook is gekeken of gebruik van doelfunctie 2, zoals genoemd in paragraaf 6.3, een meer gewenst resultaat oplevert (figuur E31) en of de benodigde tijd tot een eerste oplossing gevonden is nog afhangt van de parameterinstelling (figuur E32). De waarden lijken, zelfs bij een gemiddelde over twintig runs, helemaal geen structuur te hebben maar springen "willekeurig" heen en weer. Het gebruik van een andere doelfunctie geeft weliswaar kleinere afwijkingen maar de grafiek blijft "heen en weer springen". Ook de tijd tot een eerste complete oplossing gevonden is lijkt niet logisch af te hangen van de parameterinstelling.

7.3 Verbeterfase

In dit deelonderzoek is gekeken naar de parameters uit de categorie *Solution Comparator Weights*. Deze worden gebruikt tijdens de fase waarin CPsolver de oplossing probeert te verbeteren. Allereerst zijn deze parameters geprobeerd goed in te stellen met de instellingenoptimalisator, waarna gekeken is of dit ook daadwerkelijk verbetering oplevert voor de uiteindelijke roosteroplossing. Hiervoor is de standaard configfile vergeleken worden met de configfile waarbij de Lecture Selection parameters al geconfigureerd zijn met de instellingenoptimalisator (*Lecture configfile*) en met de output waarbij tevens de *Solution Comparator Weights* goed zijn ingesteld(*Comparator configfile*). Op deze manier is onderzocht of het nut heeft om eerst de parameters uit de constructiefase goed in te stellen.

Tevens is een parameter uit deze categorie nader onderzocht, waarbij gekeken is of de parameter wel een lokaal optimum heeft als de rest van de parameters constant wordt gehouden. Er is hierbij gekozen voor de parameter *Comparator.UselessSlotWeight*. De hoop hierbij was dat dit een convexe grafiek op zou leveren, zoals ook aangegeven in het onderzoeksplan. Hierbij is onderscheid gemaakt tussen een configfile waarbij de standaardinstellingen zijn gebruikt en de waarde van *Comparator.UselessSlotWeight* is gevarieerd en een configfile waarbij alle gewichtenparameters gelijk aan nul zijn gesteld. Dit betreft

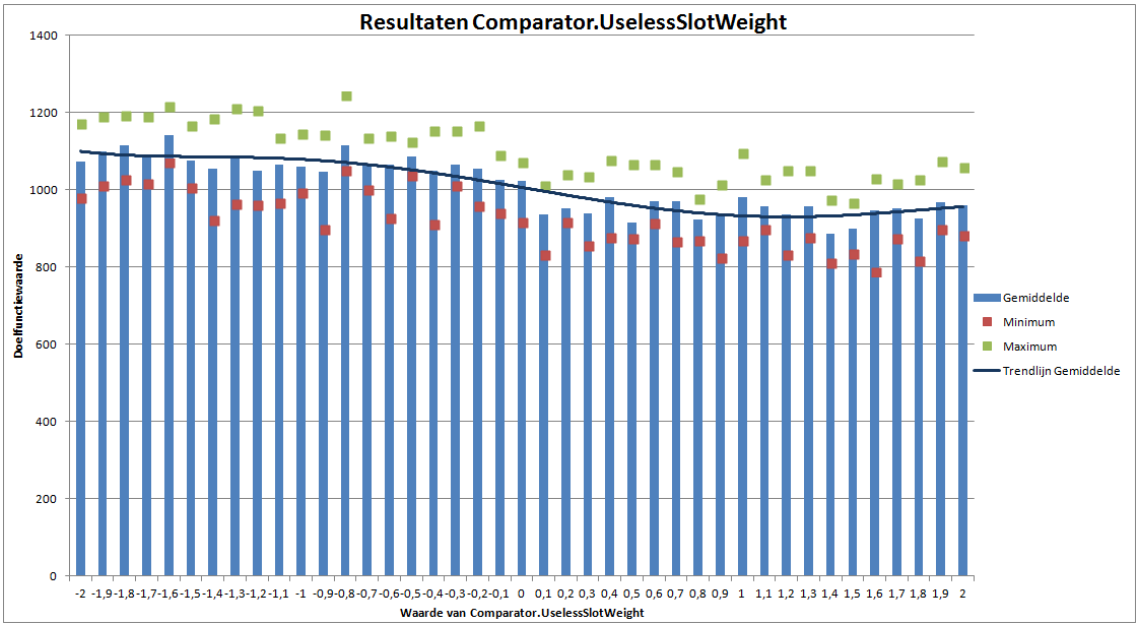
de parameters uit de categorieën *Lecture* en *Placement Selection* alsmede de *Solution Comparator weights*.

7.3.1 Resultaten

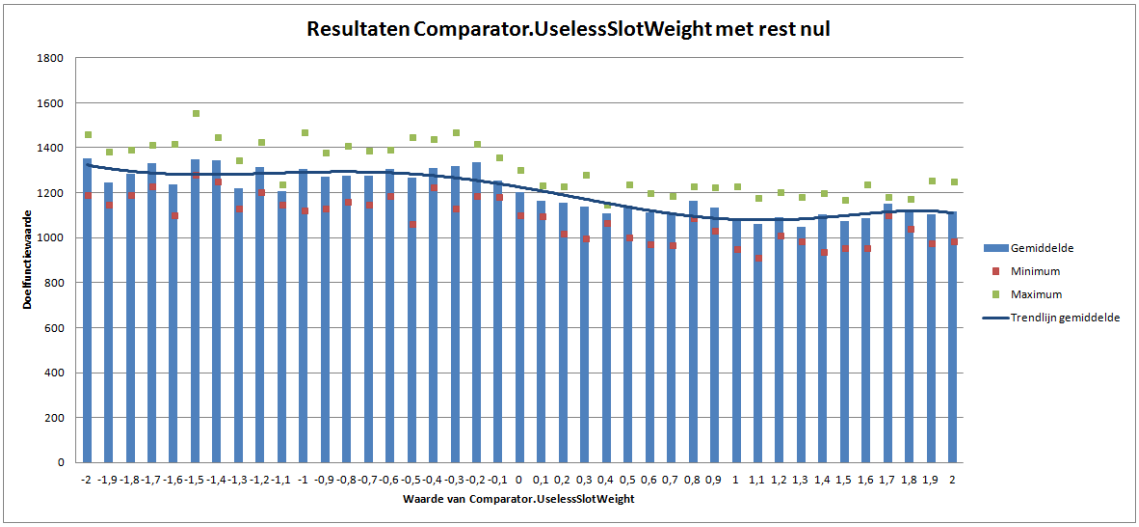
In tabel 19 zijn de resultaten van de drie verschillende configfiles terug te zien. De doelfunctiewaarde van de output van CPsolver is hierbij gegeven, dit is na een runtijd van 2.5 uur. Hierbij is onderscheid gemaakt tussen de drie verschillende computers. In figuren 10 en 11 zijn de resultaten van het variëren van de waarde van de parameter *Comparator.UselessSlotWeight* binnen de twee verschillende configuratiefiles te zien. Hierbij is gebruik gemaakt van negen verschillende runs per waarde. In de figuur is naast de gemiddelde doelfunctiewaarde ook het minimum en maximum aangegeven. De trendlijn die is toegevoegd is een zesde graads polynoom die met behulp van de kleinste kwadratenmethode is geproduceerd.

	Standaard configfile	**Lecture configfile**	**Comparator configfile**
Computer 1	967.9	985.9	1049.2
Computer 2	913.7	913.1	967.5
Computer 3	1033.7	1020.2	912.4

Tabel 19: Doelfunctiewaarden van de resultaten van CPsolver met de drie verschillende configfiles



Figuur 10: Resultaten variatie waarde Comparator.UselessSlotWeight met standaard configfile



Figuur 11: Resultaten variatie waarde Comparator.UselessSlotWeight met gewichtenparameters gelijk aan nul

7.3.2 Bespreking resultaten

In tabel 19 is te zien dat het gebruik van de instellingenoptimalisator niet perse verbetering oplevert. In de drie runs die gemaakt zijn heeft uiteindelijk 1 run een verbetering qua doelfunctiewaarde opgeleverd ten opzichte van de standaard configfile. Verder valt er uit deze tabel niet de conclusie af te leiden dat het nuttig is om eerst de parameters uit de *Lecture Selection* goed in te stellen. Maar bij twee van de drie computers heeft dit tot een betere uitkomst geleid. Deze uitkomst is niet verrassend, aangezien al gebleken is dat de uitkomst van CPsolver erg stochastisch lijkt. Dit heeft tot gevolg dat een steekproef van drie te klein is om echt conclusies uit te kunnen trekken.

Wel valt uit de resultaten op te maken dat de parameter *Comparator.UselessSlotWeight* wel degelijk een lokaal optimum heeft. Dit is te zien in de figuren 10 en 11. Hierin is een licht convexe grafiek te zien, wat impliceert dat de parameter een ideale instelwaarde van rond de 1.1 heeft. Hierbij is weinig verschil te zien tussen de verschillende configfiles waar vanuit is gegaan, behalve dat de doelfunctiewaarde bij de configfile waarbij alle gewichtparameters op nul worden gezet een stuk hoger is. Dit is logisch, aangezien deze instelling verre van goed is. Deze uitkomst is gunstig, omdat dit betekent dat het wel mogelijk kan zijn om een optimum te vinden voor in ieder geval deze parameter via een automatische zoekactie. De verwachting is dat dit ook geldt voor de resterende parameters uit deze categorie.

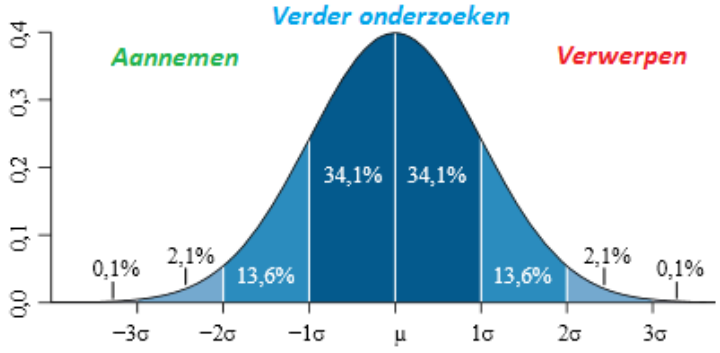
7.4 Variantie

Uit de resultaten is gebleken dat de variantie van met name de output van de constructiefase erg hoog is. Hierdoor is de aanname in de instellingenoptimalisator dat als een bepaalde configfile een lagere doelfunctie uitgeeft, deze configfile beter is dan de vorige niet helemaal gefundeerd meer. Het zou namelijk zo kunnen zijn dat als deze configfile vaker gebruikt wordt, het gemiddelde alsnog slechter is. Hierom is het wenselijk om een bepaalde grens te gebruiken in de instellingenoptimalisator om een bepaalde verandering te accepteren. Dit zou als volgt kunnen werken, een visuele toelichting is weergegeven in figuur 12;

- Doelfunctiewaarde $\leq G_o \rightarrow$ Accepteer de verandering
- Doelfunctiewaarde $\geq G_b \rightarrow$ Verwerp de verandering
- $G_o \leq$ doelfunctiewaarde $\leq G_b \rightarrow$ Vaker runnen om vast te stellen of de verandering gemiddeld verbetering of verslechtering oplevert

Om deze ondergrens G_o en bovengrens G_b voor elke configfile te vinden is het van belang om te weten hoe de uitkomst van CPsolver stochastisch verdeeld is.

Het vermoeden en de hoop bestaat nu dat deze uitkomst bij benadering normaal verdeeld is, of op zijn minst een symmetrische verdeling heeft. Dit zou inhouden dat voor het berekenen van de onder- en bovengrens gebruik gemaakt zou kunnen worden van de standaardafwijking. Om te kijken of dit vermoeden juist is, is op drie verschillende computers met de standaard configfile en de dataset van Unitime verschillende keren CPsolver gerund. Hierbij is onderscheid gemaakt tussen enkel het vinden van de eerste complete oplossing, hiermee wordt dus enkel de constructiefase onderzocht, en het doorlopen van het gehele programma. Bij dit laatste is steeds een runtijd van een half uur aangehouden.



Figuur 12: Voorbeeld normaalverdeling

7.4.1 Resultaten constructiefase

In figuur 13 zijn de resultaten te zien van de constructiefase. Hierbij is tevens het gemiddelde en de variantie aangegeven. In appendix E.4 zijn de resultaten van de drie afzonderlijke computers terug te vinden waaruit blijkt dat de gebruikte computer weinig invloed heeft op het resultaat.

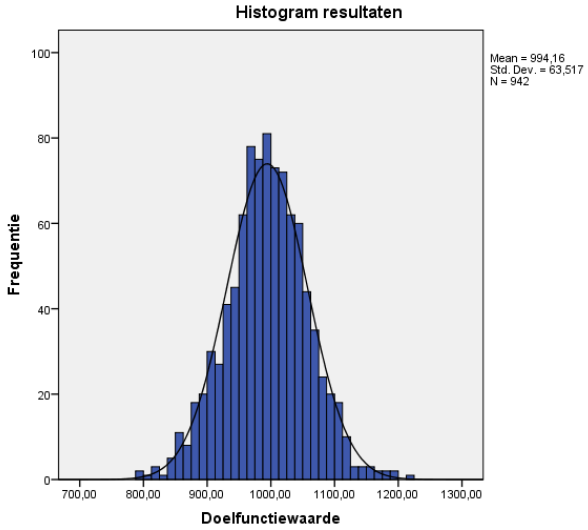
Zoals in figuur 14 te zien is, lijken de resultaten normaal verdeeld. De normaalverdeling met hetzelfde gemiddelde en dezelfde variantie is bij de resultaten geplot ter vergelijking. Voor de zekerheid is de Kolmogorov-Smirnov test in SPSS gebruikt om te kijken of de resultaten inderdaad normaal verdeeld zijn. Hierbij is gebruik gemaakt van de volgende hypotheses, waarbij a het significantieniveau is;

H_0: De uitkomst van CPsolver binnen de constructiefase is niet normaal verdeeld, p<a.
H_1: De uitkomst van CPsolver binnen de constructiefase is normaal verdeeld, p>a.

Het resultaat van de Kolmogorov-Smirnov test is hierbij 0.023 en heeft een p-waarde van 0.200 en ligt boven het significantieniveau van 0.05. Dus de nulhypothese wordt niet verworpen met a=0.05. De output van CPsolver in de constructiefase is dus niet aantoonbaar normaal verdeeld. Bij een significantieniveau van 0.20 zou de nulhypothese pas verworpen worden.

7.4.2 Resultaten verbeterfase

De resultaten voor de verbeterfase zijn weergegeven in figuur 15. Hierbij is weer het gemiddelde en de variantie aangegeven. De uitkomst lijkt nu minder normaal verdeeld te zijn als het resultaat uit de constructiefase, hierom is weer de Kolmogorov-Smirnov test uitgevoerd in SPSS. De resultaten hiervan zijn weergegeven in figuur 16. Met dezelfde nulhypothese als zojuist beschreven en als testresultaat 0.035 met p-waarde 0.200 leidt dit tot het niet verwerpen van de nulhypothese met a=0.05. De output van CPsolver is dus ook niet aantoonbaar normaal verdeeld als het gehele algoritme doorlopen wordt.



Figuur 13: Resultaten variantie constructiefase

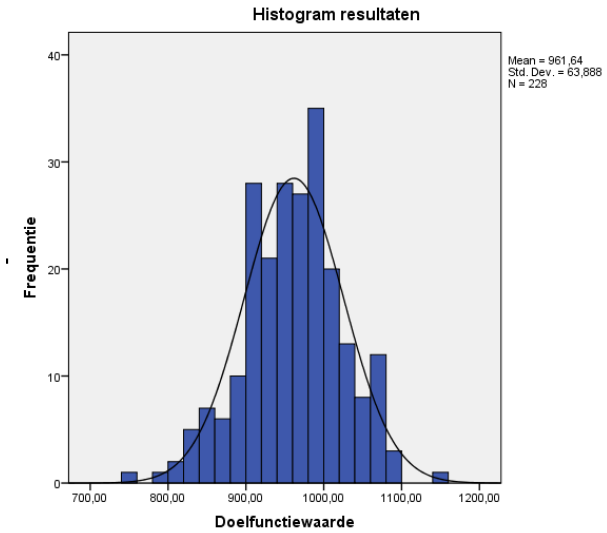
Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
TotaalResultaten	,023	942	,200*	,998	942	,254

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Figuur 14: Output SPSS normaaltest voor de constructiefase



Figuur 15: Resultaten verbeterfase

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
VAR00001	,035	228	,200*	,995	228	,737

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Figuur 16: Output SPSS normaaltest voor de verbeterfase

7.4.3 Bespreking resultaten

Uit de resultaten blijkt niet aantoonbaar dat de output van CPsolver normaal verdeeld is. De uitkomsten zijn echter wel redelijk symmetrisch verdeeld. Hierdoor is het alsnog mogelijk om het gemiddelde plus/min de standaardafwijking als grens voor G_o en G_b te gebruiken om een bepaalde configfile aan te nemen. De mogelijke programmaverbetering zoals in het begin van dit hoofdstuk lijkt dus een goede oplossing om de instellingenoptimalisator te verbeteren. Het advies hierbij is om als een verbetering van een configfile nader onderzocht moet worden, de originele configfile en de verbetering hiervan beiden x keer te runnen met CPsolver. De configfile waarbij gemiddeld de doelfunctiewaarde het laagst is, wordt dan als het best aangenomen. Deze x moet niet te groot genomen worden, aangezien dit veel rekentijd vergt.

Deze verbetering van de instellingenoptimalisator moet nog geïmplementeerd en verder onderzocht worden. Hiernaast moeten ook de te gebruiken grenzen nog verder onderzocht worden, misschien werkt het nog wel veel beter als grenzen het gemiddelde plus/min tweemaal de standaardafwijking wordt gebruikt.

8 Conclusie

In dit hoofdstuk zullen de conclusies en aanbevelingen die uit het literatuuronderzoek en parameteronderzoek zijn voortgevloeid besproken worden. Hierbij zullen eerst de conclusies volgend op de resultaten van het parameteronderzoek besproken worden. Daarna volgen enkele aanbevelingen voor verbetering van de instellingenoptimalisator en vervolgonderzoek. In de daaropvolgende paragraaf zullen de aanbevelingen die voor Paralax van belang kunnen zijn worden besproken. Hierbij wordt ingegaan op de literatuurstudie naar CPsolver, de implementatie van CPsolver en algemeen gebruik van EduFlex. In dit laatste stuk zal tevens de nadruk liggen op de terugblik naar de probleemstelling en onderzoeksvragen.

8.1 Conclusies en aanbevelingen - parameteronderzoek

Aan de hand van de verkregen resultaten bij het parameteronderzoek is gebleken dat het waarschijnlijk mogelijk is om op een automatische wijze een lokaal optimum te vinden voor de instellingen in de configuratiefile van CPsolver. Hierbij is naar voren gekomen dat het geen nut heeft om eerst de parameters die enkel in de eerste fase (*Lecture Selection*) van het algoritme gebruikt worden goed in te stellen. Dit komt doordat de uitkomst van de constructiefase erg stochastisch is, waarschijnlijk omdat er veel random gekozen wordt tijdens deze fase. Het eerst goed instellen van deze parameters levert hierdoor geen verbetering op in het uiteindelijke resultaat, het gevormde rooster.

De parameters uit de resterende fases van CPsolver hebben waarschijnlijk wel een lokaal optimum dat automatisch gevonden zou kunnen worden, dit is in ieder geval het geval bij de onderzochte parameter. Als alle parameters constant worden gehouden en deze parameter gevarieerd wordt, levert dit een licht convexe grafiek op. De verwachting is dat de resultaten die bij deze parameter behaald zijn doorgetrokken kunnen worden naar de resterende parameters uit dezelfde categorie. Tevens blijkt uit de resultaten dat in het algemeen geldt voor de parameters uit de categorie *Solution Comparator Weights* dat betere roosters verkregen worden als de parameters positief ingesteld zijn. Hiernaast is gebleken dat de output van CPsolver erg stochastisch is, ook als het gehele algoritme wordt uitgevoerd. Dit heeft geleid tot het inzicht dat het gebruikte programma (zie paragraaf 6.4) sterk verbeterd zou kunnen worden door de variantie van de output mee te nemen. De verdeling van de output van CPsolver is onderzocht en deze leek normaal verdeeld, maar testen hebben dit niet met hoge significantie bevestigd. Aangezien de output wel redelijk symmetrisch verdeeld is, is de aanbeveling dan ook voor verbeteren van het programma om een boven- en ondergrens in te stellen voor het al dan niet aannemen van een verandering in de configuratiefile. Voor resultaten die tussen deze grenzen liggen, is het verstandig om CPsolver vaker te laten runnen.

Naast dit verbeterpunt qua programmatuur is het nodig om op te merken dat in dit onderzoek de parameters steeds één voor één zijn aangepast tijdens het zoekproces. Denkbaar is echter dat de parameters veel invloed op elkaar uitoefenen en dat betere resultaten bereikt kunnen worden als deze invloed mee zou worden genomen, door parameters tegelijk te veranderen. Door dit in te voegen zou tevens onderzocht kunnen worden of het tegelijkertijd variëren van de parameters leidt tot een meer convexe grafiek. Tenslotte kan het handig zijn om een extra eigenschap in het programma in te bouwen om uit een lokaal optimum te kunnen komen, door bijvoorbeeld een configfile die een slechter resultaat uit heeft gegeven toch met een bepaalde kans aan te nemen. Aangezien het wel wenselijk is dat het programma uiteindelijk termineert zal dit een lastig proces zijn om te implementeren.

8.2 Conclusies en aanbevelingen - CPsolveronderzoek

In hoofdstuk 2 werd de hoofdvraag van dit onderwerp als volgt geformuleerd: **Hoe beïnvloeden de instellingen van de verschillende parameters van CPsolver de kwaliteit van de output van Eduflex?** Het beantwoorden van deze vraag is echter niet zo gemakkelijk gebleken. Het doel om de parameters van CPsolver automatisch in te stellen aan de hand van een doelfunctie, die gemaakt wordt nadat de klant met schuifjes zijn voorkeuren heeft doorgegeven, is dan ook niet behaald. Dit heeft voornamelijk als oorzaak dat er nog veel onderzoek gedaan moet worden, voordat dit mogelijk is. Het beantwoorden van de hoofdvraag is ook niet aan bod gekomen, voornamelijk doordat CPsolver nog niet daadwerkelijk in gebruik is genomen. Hierdoor is het onmogelijk gebleken om resultaten te verkrijgen vanuit Eduflex. Alle verkregen resultaten zoals beschreven in dit verslag zijn behaald uit datasets van Unitime en met de originele versie van CPsolver. Aan de hand van de conclusies die getrokken zijn, wordt het echter wel mogelijk geacht om deze hoofdvraag en het doel te kunnen bereiken. Hiervoor zal verder onderzoek moeten worden verricht, en wel op twee verschillende domeinen;

Allereerst heeft de **implementatie** van CPsolver binnen Eduflex nog enkele punten waar verbetering mogelijk is. Zo is gebleken dat bepaalde instelmogelijkheden binnen Eduflex (bijv. extra codes) leiden tot het niet kunnen runnen van CPsolver. Ook zijn er veel mogelijkheden die CPsolver biedt, die nog niet optimaal worden benut door Eduflex. Dit komt voornamelijk omdat de mogelijkheid er (nog) niet is binnen Eduflex om verscheidene voorkeuren door te geven. Aangezien veel instelmogelijkheden binnen CPsolver juist invloed uitoefenen op deze doorgegeven voorkeuren, zijn hiermee ook veel parameters ongebruikt. De belangrijkste voorbeelden hiervan zijn het al dan niet meenemen van afstanden tussen lokalen en het doorgeven van voorkeuren voor docenten, lokalen en tijdstippen voor een bepaalde les. Belangrijk hierbij is dat de afweging gemaakt wordt of het ook wenselijk is dat deze aspecten meegenomen worden. Mocht de keuze gemaakt worden dat dit niet van belang is voor de Nederlandse onderwijsinstellingen, zal overwogen moeten worden of CPsolver dan wel de juiste solver is voor gebruik binnen Eduflex. CPsolver heeft namelijk bewezen een goede solver te zijn in vergelijking met anderen bij het gebruik van alle mogelijkheden. De vraag is of deze prestaties ook bovengemiddeld zijn als niet alles wordt benut.

Hiernaast is in het onderzoek steeds gebruik gemaakt van een doelfunctie die de wensen van de klant representeert. Om dit realistisch te laten zijn is het nodig dat er **marktonderzoek** gedaan wordt, waaruit blijkt wat de wensen binnen onderwijsinstellingen in Nederland zijn. Er zijn al verscheidene prestatie-indicatoren uit verschillende onderzoeken naar voren gekomen, maar het belang van elke factor is hiermee nog niet bepaald. Om automatisch een doelfunctie op te kunnen stellen met behulp van schuifjes is dit essentiële informatie die eerst verkregen zal moeten worden.

Referenties

[CK95] Tim B Cooper en Jeffrey H Kingston. *The complexity of timetable construction problems*. Basser Department of Computer Science, University of Sydney, 1995.

[LD84] Gilbert Laporte en Sylvain Desroches. „Examination timetabling by computer". In: *Computers & operations research* 11.4 (1984), p. 351–360.

[Mül05] Tomáš Müller. „Constraint-based timetabling". Proefschrift. Charles University in Prague, 2005.

[Mül09] Tomáš Müller. „ITC2007 solver description: a hybrid approach". In: *Annals of Operations Research* 172.1 (2009), p. 429–446.

[Sch99] Andrea Schaerf. „A survey of automated timetabling". In: *Artificial intelligence review* 13.2 (1999), p. 87–127.

[Tri12] *Onderwijslogistiek Roosteren en Beheren middelen*. 2012. URL: `http://triplea.samboict.nl/`.

[Uni13] UniTime. *Dataset UniTime*. Mei 2013. URL: `http://www.unitime.org/uct_datasets.php/`.

A Oplosmethoden Roosterprobleem

De varianten op een roosterprobleem, zoals in paragraaf 4.4 omschreven, zijn op verschillende manieren op te lossen. Over de jaren heen zijn verschillende methoden en algoritmen ontwikkeld om roosterproblemen automatisch op te kunnen lossen, hiervan gebruikt CPsolver er een aantal. Om een overzicht te krijgen van de bekendste mogelijkheden volgt hieronder een korte uitleg van deze oplossingstechnieken, op volgorde van ontstaan.

A.1 Reductie tot graafkleuring

Een roosterprobleem kan gereduceerd worden tot een graafkleuring probleem. Definieer hiervoor voor elke les l_i van vak V_j een punt m_{ij}. Met hierbij i=1,...,q en q het aantal lessen dat gegeven wordt van vak V_j. Zorg er hierna voor dat voor elk vak V_j de punten m_{ij} een clique vormen. Introduceer nu lijnen tussen de punten van clique V_{j_1} en V_{j_2} als deze twee vakken conflicteren, bijvoorbeeld als er een leerling is die beide vakken wil volgen. In het geval van tijdsvoorkeuren voor bepaalde vakken kan een extra set van p punten worden toegevoegd, waarbij p het aantal tijdsperioden is. Deze punten zijn allemaal met elkaar verbonden, zodat in de puntkleuring elke tijdsperiode een andere kleur krijgt toegewezen. Als een vak niet op een bepaalde periode gegeven mag worden, worden alle lessen van dit vak verbonden met het punt behorende bij de desbetreffende tijdsperiode. Als een vak op een bepaalde tijdsperiode gegeven moet worden, worden juist alle lessen verbonden met alle tijdsperioden behalve de gewenste. Op deze manier wordt het probleem weergegeven als een puntkleuring probleem.

A.2 ILP (Integer Linear Programming)

Om het roosterprobleem op te lossen maken verscheidene auteurs gebruik van al bekende technieken voor Integer Lineaire Programmering. Aangezien, zoals in paragraaf 4.2 omschreven is, het probleem als integer lineair programma op te schrijven is, kunnen deze uit de literatuur bekende technieken direct toegepast worden.

A.3 Network flow technieken

Een andere representatie van het probleem is met behulp van netwerkmodellen. Deze netwerkmodellen kunnen in polynomiale tijd opgelost worden. Het probleem is echter dat hierbij niet alle voorwaardes meegenomen worden. Zo kan een docent bijvoorbeeld dubbel ingeroosterd worden. Daarom kan het nodig zijn de gewichten van de doelfunctie handmatig aan te passen om zo de voorwaardenschending aan te pakken. Hierdoor moet de procedure een aantal keer uitgevoerd worden alvorens een echte oplossing gevonden wordt. Door deze aanpassingen blijkt echter wel dat deze aanpak een heuristiek is, er wordt dus geen optimale oplossing geleverd.

A.4 Tabu search

Bij Tabu search wordt telkens gesprongen naar een "buuroplossing", een oplossing die dicht bij de vorige in de buurt ligt. Ongeacht de doelfunctiewaarde van de huidige oplossing wordt de buuroplossing met de laagste doelfunctiewaarde gekozen, dus ook als deze hoger is dan de huidige. Om hierbij niet dezelfde oplossingen meerdere keren na elkaar te proberen wordt gebruik gemaakt van een Tabu lijst. Deze lijst houdt een vast aantal oplossingen bij, die dus niet meer gekozen kunnen worden. Als er een nieuwe oplossing aan de lijst wordt toegevoegd, wordt daarmee de oudste oplossing in de Tabu lijst weer beschikbaar. De Tabu lijst kan bovendien nog genegeerd worden als de verbetering van de doelfunctiewaarde groot genoeg is. Er zal dan tóch een oplossing uit de lijst gekozen worden, ondanks dat dit normaalgesproken niet toegestaan is.

Bij deze heuristiek moet nog gedefinieerd worden wat onder een buuroplossing verstaan wordt. Een voorbeeld voor deze keuze zou kunnen zijn een oplossing die gemaakt kan worden door één les te verplaatsen in de huidige oplossing. Hierbij kan dan nog een restrictie meegegeven worden voor de

buuroplossingen die meegenomen worden, bijvoorbeeld dat de buuroplossing een les met minstens 1 conflict moet verplaatsen, zodat alleen slimme keuzes van lessenverplaatsing overwogen worden.

A.5 Rule-based approach

Rule-based approaches maken, zoals de naam al doet vermoeden, gebruik van een aantal regels om resources (hulpbronnen) te verdelen. De gebruiker geeft een aantal regels op aan de hand waarvan resources verdeeld worden. Zo zijn er regels voor het toewijzen van lessen aan tijden. Regels voor het behouden van de constraints worden bij elke toewijzing van een les aan een tijdslot bekeken om zo conflicten direct op te sporen. Verder zijn er regels voor het veranderen van vorige toewijzingen voor als het toewijzen van een nieuwe les niet lukt.

Deze aanpak wordt gebruikt voor algemene resource toewijs problemen, waarvan course timetabling enkel een voorbeeld is. De regels hangen af van de gebruiker en de heuristiek ligt dus nog niet vast, maar hangt sterk af van wat de gebruiker kiest voor (met name de toewijs-) regels.

A.6 Constraint Logic Programming approach

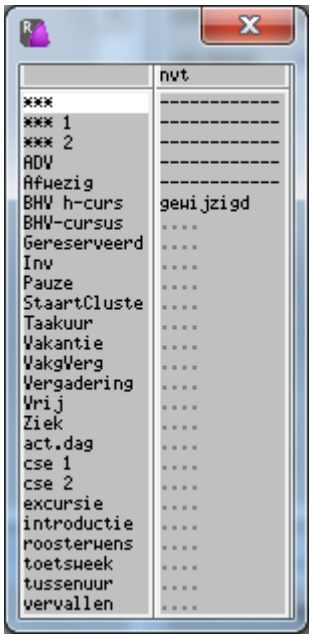
Bij Constraint Logic Programming (CLP) wordt geprobeerd een oplossing te vinden door vooruit te kijken en delen van de oplossingsruimte die voor problemen zorgen vroegtijdig weg te halen uit de zoektocht. Mogelijke oplossingen worden in diepte afgegaan om een boel soortgelijke oplossingen met een minder goede doelfunctiewaarde snel af te kunnen schrijven. Deze aanpak gaat uit van een bepaalde structuur die de oplossingsruimte heeft. De gebruiker zal aan de hand van de structuur van het probleem een selectiecriterium op kunnen geven zodat "soortgelijke" oplossingen snel afgeschreven kunnen worden.

A.7 Overige technieken

Er zijn nog veel meer methodes, elk met hun eigen sterktes en valkuilen. Een voorbeeld is bijvoorbeeld de groep van genetische algoritmes die het idee van natuurlijke evolutie gebruiken om betere oplossingen te vinden, door vorige oplossingen te "mixen". Meer voorbeelden, alsmede een diepgaandere uitleg van de hierboven beschreven methodes, zijn te vinden in artikel [Sch99].

B EduFlex

De keuzes die binnen EduFlex mogelijk zijn voor extra codes zijn te zien in figuur B17



Figuur B17 ExtraCodes

C In- en outputbestanden

C.1 Statistieken

Door CPsolver worden enkele bestanden gegenereerd die bekeken zijn tijdens het onderzoek. Hierbij is vooral output.csv een veelgebruikt bestand geworden, waarin veel statistieken over de oplossing worden gegeven. Hieronder volgt een overzicht van alle bestanden die gemaakt worden en welke informatie hierin wordt gegeven.

Instance	Average number of classes per student
Assigned variables	Average number of committed classes per student
Back-to-back instructor preferences	Average number of meetings per class
Distribution preferences	Average number of hours per class
Overall solution value	Number of rooms
Room preferences	Room size min/max
Same subpart balancing penalty	Room frequency (size≥0 used/available times)
Speed	Room utilization (size≥0 used/available seats)
Student conflicts	Number of rooms
Time preferences	Min/max room size
Too big rooms	Number of instructors
Useless half-hours	Number of class-instructor assignments
Averae classes per instruction	Average classes per instructor
Classes with single value	Average number distribution constraints per class
Average domain size	Joint enrollment constraints
Average number of available times/rooms	Number of students

Tabel C20 Overzicht gegevens in info.csv

Gegevens over het doorlopen proces.
Het bestand debug.log geeft de verschillende stappen die CPsolver heeft doorlopen weer. Dit houdt achtereenvolgend in het inlezen van de inputfile, het creëren van de benodigde gegevens voor CPsolver aan de hand van deze inputgegevens en het initialiseren van de solver. Vervolgens wordt een initiële oplossing gegenereerd met een hierbij behorende waarde. Deze waarden zijn hier ook in terug te vinden. Vervolgens is de fase van het verbeteren van de oplossing weergegeven, waarbij voor elke gevonden oplossing de bijbehorende waarde wordt weergegeven. Dit zijn de Hill Climber en Great Deluge fase zoals beschreven in hoofdstuk 5.1.Er is in het debug.log bestand ook te zien wanneer het iteratielimiet en de teller at van de Great Deluge fase worden verhoogd. Tenslotte wordt na de vooraf ingestelde tijd weergegeven dat er een outputfile wordt gegenereerd en wordt het aantal uitgevoerde iteraties weergeven met bijbehorende maximale snelheid.

Total number of classes	Total number of instructional offerings
Total number of configurations	Total number of scheduling subparts
Average number classes per subpart	Average number classes per config
Average number classes per offering	Total number of classes with only one value
Total number of classes with only one time	Total number of classes with only one room
Total number of classes requesting no room	Total number of classes requesting one room
Total number of classes requesting one or more rooms	Average number of requested rooms
Average minimal class limit	Average maximal class limit
Average number of placements	Average number of time locations
Average number of room locations	Average minimal requested room size
Average maximal requested room size	Average requested room sizes
Average requested room size	Average maximum normalized time preference
Average minimum normalized time preference	Average normalized time preference
Average maximum room preferences	Average minimum room preferences
Average room preferences	Total number of students
Number of inevitable student conflicts	Total amount of student enrollments
Number of student enrollments	Total amount of joined enrollments
Number of joint student enrollments	Average number of students
Average number of enrollements (per student)	Total amount of inevitable student conflicts
Average number of meetings (per class)	Average number of hours per class
Total number of rooms	Minimal room size
Maximal room size	Average room size
Maximal distance between two rooms	Average distance between two rooms
Average distance between two rooms [m]	Maximal distance between two rooms [m]
Average hours available	Total number of instructors
Total class-instructor assignments	Average classes per instructor
Total number of group constraints	Total number of hard group constraints
Average classes per group constraint	

Tabel C21 Overzicht gegevens in info.txt

Assigned variables	Time [sec]
Hard student conflicts	Distance student conf
Student conflicts	Committed student conflicts
All Student conflicts	Time preferences
Room preferences	Useless half-hours
Too big room	Distribution preferences
Back-to-back instructor pref.	Same subpart balancing penalty
Assigned variables max	Student enrollments
Student committed enrollments	All student enrollments
Time preferences min	Time preferences max
Room preferences min	Room preferences max
Useless half-hours max	Too big room max
Distribution preferences min	Distribution preferences max
Back-to-back instructor pref max	

Tabel C22 Overzicht gegevens in output.csv

C.2 XML files

De input die door Eduflex wordt ingegeven in CPsolver, alsmede de output die weer vanuit CPsolver door Eduflex worden ingelezen met hierin het gevormde rooster, worden in xml-formaat geproduceerd. Hieronder volgen enkele figuren met inhoud van deze bestanden. Bij deze voorbeeldbestanden zijn de docent en het lokaal al vooraf gekoppeld.

```
<timetable version="2.4" initiative="Paralax" term="11-02-2013_45" year="2013" created="2013" nrDays="7" slotsPerDay="7" >
```

Figuur C18 Algemene informatie.

```
<room id="3" name="VM207a" capacity="2" location="0,0" ignoreTooFar="true" discouraged="false" />
```

Figuur C19 Declaratie lokalen.

```
<instructor id="6" name="6_BASTIAA" />
```

Figuur C20 Declaratie docenten.

```
<class id="1" offering="1" name="2_INT" config="1" subpart="1" minClassLimit="1" maxClassLimit="4" nrRooms="1" committed="false" >
<instructor id="6" />
<room id="5" />
<time days="1000000" start="0" length="1"/>
<time days="1000000" start="1" length="1"/>
<time days="1000000" start="2" length="1"/>
<time days="1000000" start="3" length="1"/>
<time days="1000000" start="4" length="1"/>
<time days="1000000" start="5" length="1"/>
<time days="0100000" start="0" length="1"/>
<time days="0100000" start="1" length="1"/>
<time days="0100000" start="2" length="1"/>
```

Figuur C21 Declaratie in te roosteren uren.

```
<room id="90" pref="4"/>
<room id="91" pref="0"/>
<room id="92" pref="-4"/>
<room id="93" pref="1"/>
<room id="94" pref="0"/>
<room id="95" pref="4"/>
```

Figuur C22 Lokaal voorkeur

```
<time days="1010100" start="102" length="12" pref="0.0"/>
<time days="1010100" start="114" length="12" pref="-40.0"/>
```

Figuur C23 Tijd voorkeur

```
<student id="1" name="leijgraven, Inneke van" minSlots="1" maxSlots="45" >
<offering id="2" reward="0" />
<offering id="2" reward="0" />
<offering id="2" reward="0" />
<offering id="2" reward="0" />
<offering id="2" reward="0" />
```

Figuur C24 Declaratie studenten.

```
<!--University Course Timetabling-->
<!--Solution Info:
    Assigned variables: 100.00% (15/15)
    Back-to-back instructor preferences: 100.00% (0)
    Distribution preferences: 100.00% (0)
    Iteration: 30
    Memory usage: 4.17M
    Overall solution value: 0.00
    Room preferences: 100.00% (0)
    Same subpart balancing penalty: 0.00
    Speed: 192.31 it/s
    Student conflicts: 0 [committed:0, distance:0, hard:0]
    Time: 0.16 sec
    Time preferences: 100.00% (0.00)
    Too big rooms: 0.00% (0)
    Useless half-hours: 0.00% (0 + 0)
```

Figuur C25 Informatie algemene oplossing.

```
<class id="4" ord="3" config="1" committed="false" subpart="4" minClassLimit="1" maxClassLimit="4" name="3_MAR1" datePatternName="12/31-01/01" dates="11">
  <!--5 students assigned (0 are in a conflict), reward 5-->
  <instructor id="7" solution="true"/>
  <room id="6" pref="0" solution="true"/>
  <time days="1000000" start="0" length="1" breakTime="0" pref="0" npref="0.0"/>
  <time days="1000000" start="1" length="1" breakTime="0" pref="0" npref="0.0"/>
  <time days="1000000" start="2" length="1" breakTime="0" pref="0" npref="0.0" solution="true"/>
  <time days="1000000" start="3" length="1" breakTime="0" pref="0" npref="0.0"/>
  <time days="1000000" start="4" length="1" breakTime="0" pref="0" npref="0.0"/>
  <time days="1000000" start="5" length="1" breakTime="0" pref="0" npref="0.0"/>
  <time days="0100000" start="0" length="1" breakTime="0" pref="0" npref="0.0"/>
```

Figuur C26 Het ingeroosterde uur.

```
<student id="1" name="leijgraven, Inneke van" minSlots="1" maxSlots="45">
  <offering id="2"/>
  <class id="10"/>
  <class id="11"/>
  <class id="12"/>
  <class id="13"/>
  <class id="14"/>
  <class id="15"/>
```

Figuur C27 De uren per leerling.

```
 <groupConstraints>
   <constraint id="121" type="BTB" pref="R">
   <constraint id="122" type="BTB" pref="R">
     <class id="1481"/>
     <class id="1482"/>
     <class id="1483"/>
   </constraint>
```

Figuur C28 Groupconstraints

C.2.1 Voorkeuren

In deze xml files kunnen nog verschillende voorkeuren opgegeven worden voor in te plannen lessen. In tabel C23 zijn deze voorkeuren weergegeven.

SAME_TIME	Lessen die op dezelfde tijd van de dag gegeven moeten worden
SAME_START	Lessen die op hetzelfde moment moeten starten
SAME_DAYS	Lessen die op dezelfde dag gegeven moeten worden
BTB_TIME	De lessen moeten gegeven worden in aangrenzende lesuren.
BTB	De lessen moeten gegeven worden in aangrenzende lesuren en in hetzelfde lokaal.
NHB_GTE(1)	Tussen de gegeven lessen zit 1 uur of meer.
NHB_LT(6)	De gegeven lessen moeten minder dan 6 uur tussen het einde van de eerste en het begin van de volgende hebben.
NHB(x)	De gegeven lessen moeten precies x uur tussen het einde van de eerste en het begin van de volgende hebben.
DIFF_TIME	De gegeven lessen mogen niet overlappen in tijd.
SPREAD	De gegeven lessen moeten verdeeld worden over de tijd.
BTB_DAY	Lessen moeten gegeven worden op opvolgende dagen.
CAN_SHARE_ROOM	Als het lokaal groot genoeg is, mogen de gegeven lessen een lokaal delen.
SAME_INSTR	De gegeven lessen moeten behandeld worden alsof ze dezelfde leraar hebben.
SAME_STUDENTS	Lessen met dezelfde leerlingen mogen niet overlappen.
MIN_GRUSE(10x1h)	Minimaliseer het aantal tijdgroepen dat gebruikt wordt bij de gegeven lessen.
MIN_GRUSE(5x2h)	Minimaliseer het aantal tijdgroepen dat gebruikt wordt bij de gegeven lessen.
MIN_GRUSE(3x3h)	Minimaliseer het aantal tijdgroepen dat gebruikt wordt bij de gegeven lessen.
MIN_GRUSE(2x5h)	Minimaliseer het aantal tijdgroepen dat gebruikt wordt bij de gegeven lessen.
MEET_WITH	Van de gegeven lessen zijn de ontmoetingen tegelijkertijd.
PRECEDENCE	De gegeven lessen moeten gegeven worden in een bepaalde volgorde.
MIN_ROOM_USE	Minimaliseer het aantal lokalen bij de gegeven verzameling lessen.

Tabel C23 Group Constraints

D Parameters CPsolver

In figuur D24 is een overzicht van alle parameters van CPsolver gegeven. Alle parameters met een asterisk worden niet gebruikt binnen Rostar Eduflex, dit heeft bijvoorbeeld te maken met afstanden tussen lokalen en het doorgeven van verschillende voorkeuren. Als de asterisk tussen haakjes is weergegeven wordt de parameter naar waarschijnlijk niet gebruikt binnen Eduflex, dit is bijvoorbeeld het geval bij de parameters die over urenverdeling tussen verschillende afdelingen gaan.

D.1 Overzicht alle parameters

	Categorie	**Parameter**
	Basic settings	General.SwitchStudents
	Basic settings	Xml.ExportStudentSectioning
	Basic settings	XML.ShowNames
(*)	Basic settings	Comparator.StudentHoursRewardWeight
(*)	Basic settings	Placement. StudentHoursRewardWeight1
(*)	Basic settings	Placement. StudentHoursRewardWeight2
(*)	Basic settings	Placement. StudentHoursRewardWeight3
	Basic settings	General.SaveAsStudentSct
	Basic settings	Basic.Disobeyhard
	Conflictbased-statistics	ConflictStatistics.Ageing
	Conflictbased-statistics	ConflictStatistics.AgeingHalfTime
	Conflictbased-statistics	ConflictStatistics.Print
	Conflictbased-statistics	ConflictStatistics.PrintInterval
*	Default time preferences	Timepreferences.Weight
*	Default time preferences	TimePreferences.Pref
(*)	Departmental Balancing	DeptBalancing.SpreadFactor
(*)	Departmental Balancing	DeptBalancing.Unassignments2Weaken
*	Distances	Student.DistanceLimit
*	Distances	Student-DistanceLimit75min
*	Distances	Instructor.NoPreferenceLimit
*	Distances	Instructor.discouragedlimit
*	Distances	Instructor.ProhibitedLimit
	General settings	General.MPP
	General settings	General.spread
	General settings	General.AutoSameStudents
(*)	General settings	General.DeptBalancing
	General settings	General.AutoSameStudentsConstraint
*	General settings	General.UseDistanceConstraints
	General settings	General.Sae
	General settings	General.SaveBestUnassigned
	General settings	General.NormalizedPrefDecreaseFactor
	General settings	General.IgnoreRoomSharing
	Implementations	PerturbationCounter.Class
	Implementations	Termination.Class
	Implementations	Solver.Class
	Implementations	Comparator.Class
	Implementations	Variable.Class
	Implementations	Value.Class
	Implementations	TimetableLoader
	Implementations	TimetableSaver
	Implementations	Neighbour.Class

	Categorie	**Parameter**
	Lecture selection	Lecture.RouletteWheelSelection
	Lecture selection	Lecture.RandomWalkProb
	Lecture selection	Lecture.DomainSizeWeight
	Lecture selection	Lecture.NrAssignmentsWeight
	Lecture selection	Lecture.InitialAssignmentWeight
	Lecture selection	Lecture.NrConstraintsWeight
	Lecture selection	Lecture.UselessSlotWeight
*	Lecture selection	Lecture.DistanceInstructorPreferenceWeight
(*)	Lecture selection	Lecture.DeptSpreadPenaltyWeight
	Lecture selection	Lecture.SelectionSubSet
	Lecture selection	Lecture.SelectionSubSetMinSize
	Lecture selection	Lecture.SelectionSubSetPart
(*)	Lecture selection	Lecture.SpreadPenaltyWeight
*	Lecture selection	Lecture.CommitedStudentConflictWeight
	Lecture selection	Lecture.HardStudentConflictWeight
	Lecture selection	Lecture.StudentConflictWeight
	Lecture selection	Lecture.TooBigRoomWeight
*	Lecture selection	Lecture.TimePreferenceWeight
*	Lecture selection	Lecture.ContrPreferenceWeight
*	Lecture selection	Lecture.RoomPreferenceWeight
	Neighbour selection	Neighbour.SuggestionProbability
	Neighbour selection	Neighbour.SuggestionTImeout
	Neighbour selection	Neigbour.SuggestionDepth
	Neighbour selection	Neighbour.SuggestionProbabilityAllAssigned
*	Perturbation Penalty	Perturbations.DeltaStudentConflictsWeight
*	Perturbation Penalty	Perturbations.NewStudentConflictsWeight
*	Perturbation Penalty	Perturbations.DifferentPlacement
*	Perturbation Penalty	Perturbations.AffectedStudentWeight
*	Perturbation Penalty	Perturbations.AffectedInstructorWeight
*	Perturbation Penalty	Perturbations.DifferentRoomWeight
*	Perturbation Penalty	Perturbations.DifferentBuildingWeight
*	Perturbation Penalty	Perturbations.DifferentTimeWeight
*	Perturbation Penalty	Perturbations.DifferentDayWeight
*	Perturbation Penalty	Perturbations.DifferentHourWeight
*	Perturbation Penalty	Perturbations.TooFarForInstructorsWeight
*	Perturbation Penalty	Perturbations.TooFarForStudentsWeight
*	Perturbation Penalty	Perturbations.DeltaInstructorDistancePreferenceWeight
*	Perturbation Penalty	Perturbations.DeltaRoomPreferenceWeight
*	Perturbation Penalty	Perturbations.DeltaTimePreferenceWeight
*	Perturbation Penalty	Perturbations.AffectedStudentByTimeWeight
*	Perturbation Penalty	Perturbations.AffectedInstructorByTimeWeight
*	Perturbation Penalty	Perturbations.AffectedStudentByRoomWeight
*	Perturbation Penalty	Perturbations.AffectedInstructorByRoomWeight
*	Perturbation Penalty	Perturbations.AffectedStudentByBldgWeight
*	Perturbation Penalty	Perturbations.AffectedInstructorByBldgWeight
	Placement selection	Placement.RandomWalkProb
	Placement selection	Placement.MPP_InitialProb
	Placement selection	Placement.MPP_Limit
	Placement selection	Placement.MPP_PenaltyLimit
	Placement selection	Placement.NrAssignmentsWeight1
	Placement selection	Placement.MPP_DeltaInitialAssignmentWeight1

	Categorie	**Parameter**
	Placement selection	Placement.UselessSlotsWeight1
*	Placement selection	Placement.DistanceInstructorPreferenceWeight1
(*)	Placement selection	Placement.DeptSpreadPenaltyWeight1
	Placement selection	Placement.ThresholdKoef1
	Placement selection	Placement.NrAssignmentsWeight2
	Placement selection	Placement.MPP_DeltaInitialAssignmentWeight2
	Placement selection	Placement.UselessSlotsWeight2
*	Placement selection	Placement.DistanceInstructorPreferenceWeight2
(*)	Placement selection	Placement.DeptSpreadPenaltyWeight2
	Placement selection	Placement.ThresholdKoef2
	Placement selection	Placement.NrAssignmentsWeight3
	Placement selection	Placement.MPP_DeltaInitialAssignmentWeight3
	Placement selection	Placement.UselessSlotsWeight3
*	Placement selection	Placement.DistanceInstructorPreferenceWeight3
(*)	Placement selection	Placement.DeptSpreadPenaltyWeight3
(*)	Placement selection	Placement.SpreadPenaltyWeight1
(*)	Placement selection	Placement.SpreadPenaltyWeight2
(*)	Placement selection	Placement.SpreadPenaltyWeight3
	Placement selection	Placement.CanUnassingSingleton
	Placement selection	Placement.NrConflictsWeight1
	Placement selection	Placement.WeightedConflictsWeight1
	Placement selection	Placement.NrPotentialConflictsWeight1
	Placement selection	Placement.NrHardStudConfsWeight1
	Placement selection	Placement.NrStudConfsWeight1
	Placement selection	Placement.NrConflictsWeight2
	Placement selection	Placement.WeightedConflictsWeight2
	Placement selection	Placement.NrPotentialConflictsWeight2
	Placement selection	Placement.NrHardStudConfsWeight2
	Placement selection	Placement.NrStudConfsWeight2
	Placement selection	Placement.NrConflictsWeight3
	Placement selection	Placement.WeightedConflictsWeight3
	Placement selection	Placement.NrPotentialConflictsWeight3
	Placement selection	Placement.NrHardStudConfsWeight3
	Placement selection	Placement.NrStudConfsWeight3
*	Placement selection	Placement.NrCommitedStudConfsWeight1
*	Placement selection	Placement.NrCommitedStudConfsWeight2
*	Placement selection	Placement.NrCommitedStudConfsWeight3
	Placement selection	Placement.TooBigRoomWeight1
	Placement selection	Placement.TooBigRoomWeight2
	Placement selection	Placement.TooBigRoomWeight3
*	Placement selection	Placement.TimePreferenceWeight1
*	Placement selection	Placement.DeltaTimePreferenceWeight1
*	Placement selection	Placement.ConstrPreferenceWeight1
*	Placement selection	Placement.RoomPreferenceWeight1
*	Placement selection	Placement.TimePreferenceWeight2
*	Placement selection	Placement.DeltaTimePreferenceWeight2
*	Placement selection	Placement.ConstrPreferenceWeight2
*	Placement selection	Placement.RoomPreferenceWeight2
*	Placement selection	Placement.TimePreferenceWeight3
*	Placement selection	Placement.DeltaTimePreferenceWeight3
*	Placement selection	Placement.ConstrPreferenceWeight3

	Categorie	Parameter
*	Placement selection	Placement.RoomPreferenceWeight3
	Same subpart balancing	Spread.Spreadfactor
	Same subpart balancing	Spread.Unassignments2Weaken
	Search intensification	SearchIntensification.IterationLimit
	Search intensification	SearchIntensification.ResetInterval
	Search intensification	SearchIntensification.MultiplyInterval
	Search intensification	Searchintensification.Multiply
	Solution comparator weights	Comparator.UselessSlotWeight
*	Solution comparator weights	Comparator.DistanceInstructorPreferenceWeight
(*)	Solution comparator weights	Comparator.PerturbationPenaltyWeight
(*)	Solution comparator weights	Comparator.DeptSpreadPenaltyWeight
(*)	Solution comparator weights	Comparator.SpreadPenaltyWeight
	Solution comparator weights	Comparator.HardStudentConflictWeight
	Solution comparator weights	Comparator.StudentConflictWeight
*	Solution comparator weights	Comparator.CommitedStudentConflictWeight
	Solution comparator weights	Comparator.TooBigRoomWeight
*	Solution comparator weights	Comparator.TimePreferenceWeight
*	Solution comparator weights	Comparator.ContrPreferenceWeight
*	Solution comparator weights	Comparator.RoomPreferenceWeight
*	Solution comparator weights	Comparator.StudentLecturePreferenceWeight
	Termination Conditions	Termination.MinPerturbances
	Termination Conditions	Termination.MaxIters
	Termination Conditions	Termination.TimeOut
	Termination Conditions	Termination.StopWhenTimeOut
	Termination Conditions	Termination.StopWhenComplete

Tabel D24 Overzicht van alle parameters

D.2 Categorieën van parameters uitgesloten bij het onderzoek

Aangezien niet alle parameters meegenomen zijn in het onderzoek met de dataset van Unitime, omdat deze naar alle waarschijnlijkheid goed zijn ingesteld, is hieronder een overzicht gegeven van de waarden waarop deze parameters zijn vastgesteld.

Categorie	Parameter	Waarde
Basic settings	General.SwitchStudents	true
Basic settings	Xml.ExportStudentSectioning	
Basic settings	XML.ShowNames	true
Basic settings	Comparator.StudentHoursRewardWeight	0.0
Basic settings	Placement. StudentHoursReward-Weight1	0.0
Basic settings	Placement. StudentHoursReward-Weight2	0.0
Basic settings	Placement. StudentHoursReward-Weight3	0.0
Basic settings	General.SaveAsStudentSct	true
Basic settings	Basic.Disobeyhard	false
Default time preferences	Timepreferences.Weight	0.0
Default time preferences	TimePreferences.Pref	oneindig
Distances	Student.DistanceLimit	67.0
Distances	Student-DistanceLimit75min	100.0
Distances	Instructor.NoPreferenceLimit	0.0

Distances	Instructor.discouragedlimit	5.0
Distances	Instructor.ProhibitedLimit	20.0
General settings	General.MPP	false
General settings	General.spread	true
General settings	General.AutoSameStudents	true
General settings	General.DeptBalancing	false
General settings	General.AutoSameStudentsConstraint	SAME_STUDENTS
General settings	General.UseDistanceConstraints	true
General settings	General.Sae	true
General settings	General.SaveBestUnassigned	-1
General settings	General.NormalizedPrefDecreaseFactor	0.77
General settings	General.IgnoreRoomSharing	false
Implementations	PerturbationCounter.Class	net.sf.cpsolver.coursett.-heuristics.Universal-PerturbationsCounter
Implementations	Termination.Class	net.sf.cpsolver.ifs.-termination.MPP-TerminationCondition
Implementations	Solver.Class	net.sf.cpsolver.coursett.-TripleATimetableSolver
Implementations	Comparator.Class	net.sf.cpsolver.coursett.-heuristics.Timetable-Comparator
Implementations	Variable.Class	net.sf.cpsolver.coursett.-heuristics.LectureSelection
Implementations	Value.Class	net.sf.cpsolver.coursett.-heuristics.Placement-Selection
Implementations	TimetableLoader	net.sf.cpsolver.coursett.-TimetableXMLLoader
Implementations	TimetableSaver	net.sf.cpsolver.coursett.-TimetableXMLSaver
Implementations	Neighbour.Class	net.sf.cpsolver.coursett.-heuristics.Neighbour-SelectionWithSuggestions
Termination Conditions	Termination.MinPerturbances	-1
Termination Conditions	Termination.MaxIters	-1
Termination Conditions	Termination.TimeOut	60
Termination Conditions	Termination.StopWhenTimeOut	true
Termination Conditions	Termination.StopWhenComplete	false

Tabel D25 Parameters behorende bij een categorie die niet meegenomen wordt

D.3 Overige parameters die uitgesloten zijn bij het onderzoek

Omdat in Rostar Eduflex afstanden niet worden meegenomen, is dit in het onderzoek met de dataset van Unitime ook niet gedaan. Hieronder zijn de waarden weergegeven waarop deze parameters zijn vastgesteld. Hierbij is tevens het meenemen van de afstanden uitgezet.

Categorie	**Parameter**	**Waarde**
Conflictbased-statistics	ConflictStatistics.Print	true
Conflictbased-statistics	ConflictStatistics.PrintInterval	-1
Lecture selection	Lecture.RouletteWheelSelection	true
Lecture selection	Lecture.DistanceInstructorPreferenceWeight	1.0
Lecture selection	Lecture.SelectionSubSet	true
Lecture selection	Lecture.	0.3
Placement selection	Placement.MPP_Limit	-1
Placement selection	Placement.MPP_PenaltyLimit	-1.0
Placement selection	Placement.DistanceInstructorPreferenceWeight1	0.1
Placement selection	Placement.DistanceInstructorPreferenceWeight2	1.0
Placement selection	Placement.DistanceInstructorPreferenceWeight3	0.0
Placement selection	Placement.TimePreferenceWeight1	0.0
Placement selection	Placement.TimePreferenceWeight2	0.3
Placement selection	Placement.TimePreferenceWeight3	0.0
Solution comparator weights	Comparator.DistanceInstructorPreferenceWeight	1.0
Solution comparator weights	Comparator.TimePreferenceWeight	0.3

Tabel D26 Parameters die niet meegenomen hoeven te worden doordat de mogelijkheden hiervan niet worden benut in Eduflex

D.4 Onderzoek Müller

Volgens Müller worden de beste resultaten verkregen als de volgende parameters gelijk ingesteld worden:

Placement level 2	**Lecture**	**Comparator (W van Müller)**
Placement.UselessSlotsWeight2	= Lecture.UselessSlotWeight	= Comparator.UselessSlotWeight
Placement.DistanceInstructorPreferenceWeight2	= Lecture.DistanceInstructorPreferenceWeight	= Comparator.DistanceInstructorPreferenceWeight
	= Placement.MPP_DeltaInitialAssignmentWeight2	= Comparator.PerturbationPenaltyWeight
Placement.DeptSpreadPenaltyWeight2	= Lecture.DeptSpreadPenaltyWeight	= Comparator.DeptSpreadPenaltyWeight
Placement.SpreadPenaltyWeight2	= Lecture.SpreadPenaltyWeight	= Comparator.SpreadPenaltyWeight
Placement.NrHardStudConfsWeight2	= Lecture.HardStudentConflictWeight	= Comparator.HardStudentConflictWeight
Placement.NrStudConfsWeight2	= Lecture.StudentConflictWeight	= Comparator.StudentConflictWeight
Placement.NrCommitedStudConfsWeight2	= Lecture.CommitedStudentConflictWeight	= Comparator.CommitedStudentConflictWeight
Placement.TooBigRoomWeight2	= Lecture.TooBigRoomWeight	= Comparator.TooBigRoomWeight
Placement.TimePreferenceWeight2	= Lecture.TimePreferenceWeight=	= Comparator.TimePreferenceWeight
Placement.ConstrPreferenceWeight2	= Lecture.ContrPreferenceWeight=	= Comparator.ContrPreferenceWeight
Placement.RoomPreferenceWeight2	= Lecture.RoomPreferenceWeight=	= Comparator.RoomPreferenceWeight

Tabel D27 Parameters die dezelfde waarde moeten hebben volgens onderzoek Müller

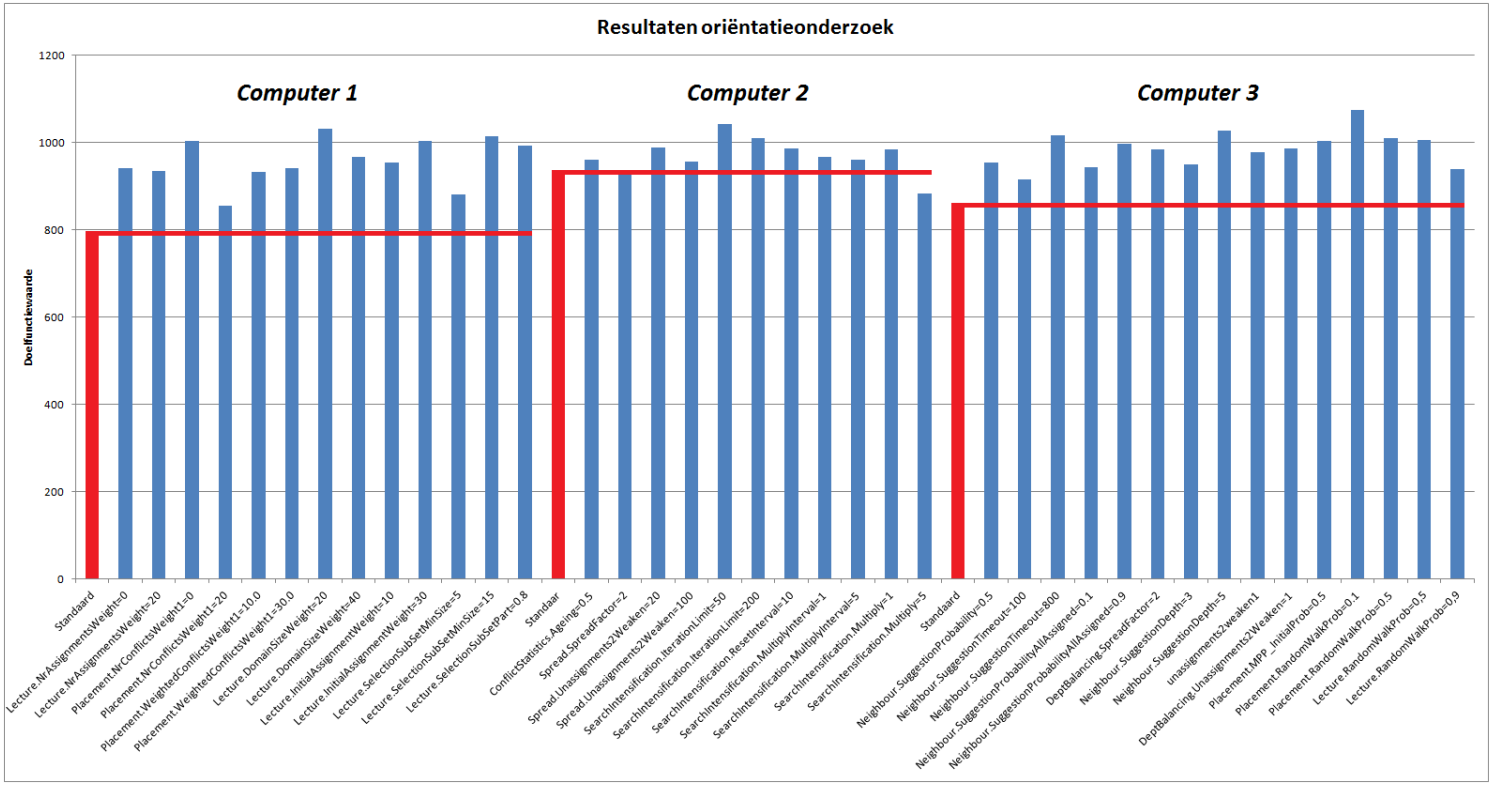
E Resultaten

E.1 Oriëntatieonderzoek

ConflictStatistics.Ageing	ConflictStatistics.AgeingHalfTime
DeptBalancing.SpreadFactor	DeptBalancing.Unassignments2Weaken
Lecture.NrAssignmentsWeight	Lecture.DomainSizeWeight
Lecture.InitialAssignmentWeight	Lecture.SelectionSubSetMinSize
Lecture.SelectionSubSetPart	Lecture.RandomWalkProb
Neighbour.SuggestionProbability	Neighbour.SuggestionTimeout
Neighbour.SuggestionDepth	Neighbour.SuggestionProbabilityAllAssigned
Placement.NrConflictsWeight1	Placement.WeightedConflictsWeight1
Placement.RandomWalkProb	Placement.MPP_InitialProb
Spread.SpreadFactor	Spread.Unassignments2Weaken
SearchIntensification.IterationLimit	SearchIntensification.ResetInterval
SearchIntensification.MultiplyInterval	SearchIntensification.Multiply

Tabel E28 De parameters voor het oriënterende onderzoek

Hieronder zijn in een histogram de resultaten van het oriëntatieonderzoek weergegeven. Langs de verticale as is de doelfunctiewaarde gegeven die met een keer runnen van CPsolver is gevonden. Langs de horizontale as is aangegeven welke configfile er bij het runnen gebruikt is. Hierbij is steeds aangegeven welke parameter er veranderd is naar welke waarde ten opzichte van de standaard configfile.



Figuur E29 Resultaten oriëntatieonderzoek

E.2 Onderzoek constructiefase

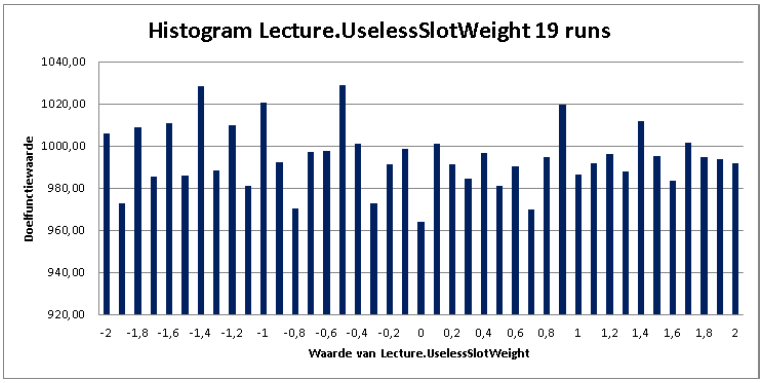
Lecture.NrAssignmentsWeight	Lecture.HardStudentConflictWeight
Lecture.StudentConflictWeight	Lecture.ContrPreferenceWeight
Lecture.RoomPreferenceWeight	Lecture.UselessSlotWeight
Lecture.TooBigRoomWeight	Lecture.DeptSpreadPenaltyWeight
Lecture.CommitedStudentConflictWeight	Lecture.SpreadPenaltyWeight
Lecture.NrConstraintsWeight	

Tabel E29 De parameters voor het onderzoek constructiefase

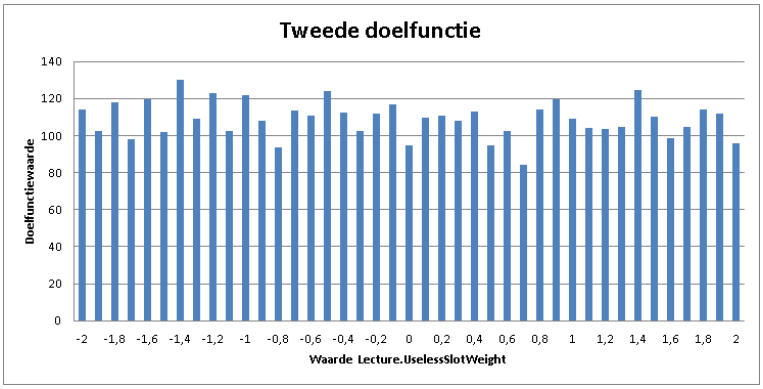
Terminatieconditie: Stop wanneer complete oplossing is gevonden	waar	CPsolver is zodanig ingesteld dat deze termineert zodra een complete oplossing gevonden is. Dit kan omdat de parameters uit de *Lecture Selection* enkel invloed hebben op deze eerste fase
Maximaal iteraties instellingenoptimalisator	500	Aangezien in de meeste gevallen de eerste complete oplossing binnen 1.5 minuten gevonden wordt, levert dit een maximale verwachtte looptijd van de instellingenoptimalisator van 13 uur op. Dit was praktisch voor het runnen
Maximale looptijd CPsolver tijdens instellingenoptimalisator	300 sec	Mocht CPsolver om wat voor reden dan ook geen of niet snel een complete oplossing genereren, is deze tijd ingesteld
Stapgrootte van parameters	0.1	De parameterinstellingen in de configfile worden met een stapgrootte van 0.1 veranderd door de instellingenoptimalisator
Grenzen van parameters	[-2,2]	De waarden van de parameters blijven te allen tijde tussen deze waarden
Maximale looptijd CPsolver bij vergelijken configfiles	9000 sec	Deze tijd is ruim genomen, zodat zeker is dat CPsolver een optimum vindt

Tabel E30 Instellingen voor programma en CPsolver bij onderzoek naar Lecture Selection parameters

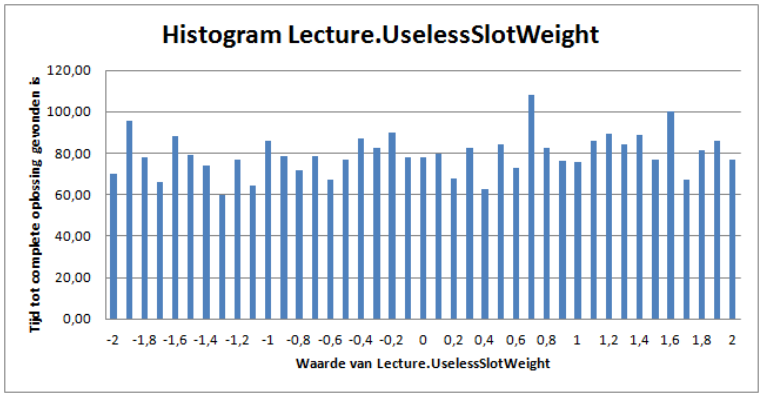
Hier zijn in histogramvorm de resultaten weergegeven waar in het hoofdstuk zelf geen ruimte voor was. Figuren E30, E31 en E32 geven de invloed weer die de parameter *Lecture.UselessSlotsWeight* heeft op de waarde van twee doelfuncties en op de tijd tot de eerst gevonden complete oplossing.



Figuur E30 Gemiddelde doelfunctiewaarden(1) over twintig runs bij verschillende waarden van UselessSlotsWeight



Figuur E31 Gemiddelde doelfunctiewaarden(2) over twintig runs bij verschillende waarden van UselessSlotsWeight



Figuur E32 Gemiddelde tijd tot eerste complete oplossing bij verschillende waarden van UselessSlotsWeight

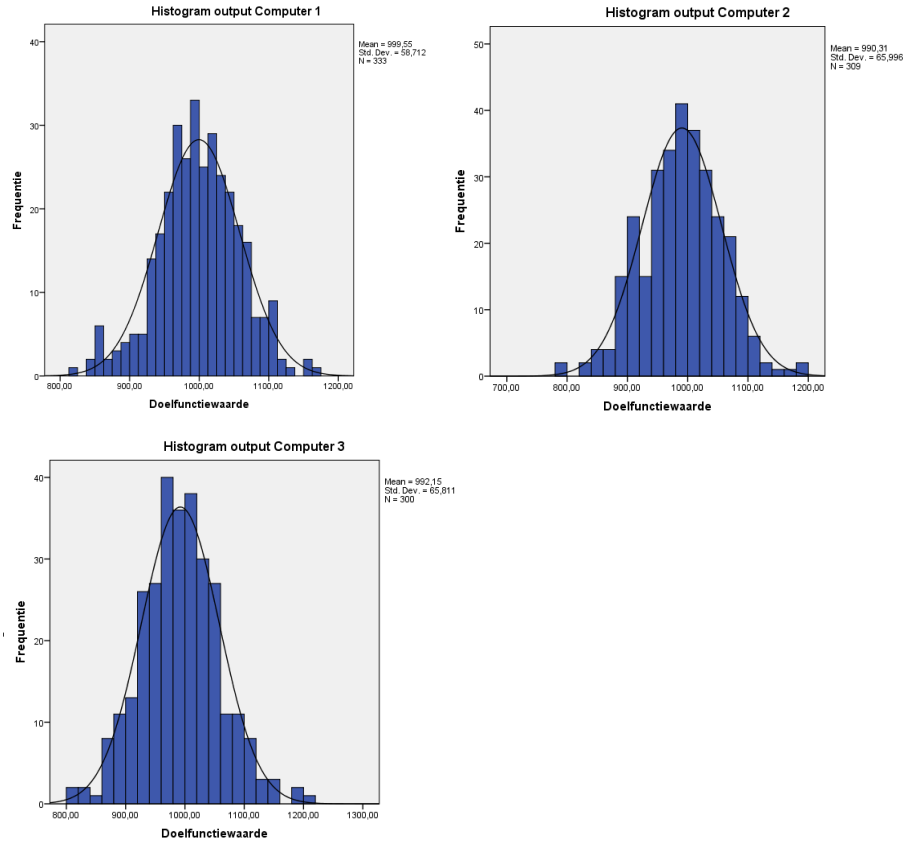
E.3 Onderzoek verbeterfase

Comparator.HardStudentConflictWeight	Comparator.StudentConflictWeight
Comparator.ContrPreferenceWeight	Comparator.RoomPreferenceWeight
Comparator.UselessSlotWeight	Comparator.TooBigRoomWeight
Comparator.DeptSpreadPenaltyWeight	Comparator.CommitedStudentConflictWeight
Comparator.SpreadPenaltyWeight	Comparator.PerturbationPenaltyWeight
Comparator.StudentLecturePreferenceWeight	

Tabel E31 De parameters voor het onderzoek verbeterfase

E.4 Onderzoek variantie output

In figuur E33 zijn de afzonderlijke resultaten per computer weergegeven. Deze lijken allemaal normaal verdeeld met ongeveer hetzelfde steekproefgemiddelde en variantie. Hierom zijn alle resultaten samengevoegd bij de test op normaliteit in paragraaf 7.4. In de histogrammen zijn tevens de normale verdelingen met gelijk steekproefgemiddelde en variantie geplot ter vergelijking.



Figuur E33 Resultaten variantie constructiefaseoutput per computer