Een optimaal rooster

Philip Roeleveld, Gido Limperg en Roshan Mahes

Universiteit van Amsterdam, Faculteit der Natuurwetenschappen, Wiskunde en Informatica,
Science Park 904, 1098 XH Amsterdam, Nederland
{phi_lip, glimperg}@live.nl, roshan-1@live.com

Samenvatting In deze paper proberen we een zo goed mogelijk lesrooster te maken. Een rooster noemen we goed wanneer deze een hoge score heeft, o.a. afhankelijk van diverse aantallen conflicten. Verder willen we dat zo'n rooster consistent binnen afzienbare tijd bereikt kan worden. Dit is een onoplosbaar probleem, daarom is deze case zeer interessant. Er zijn vele manieren om toch zeer goede roosters te vinden. Eerst berekenen we hiervoor een bovengrens voor onze toestandsruimte. Vervolgens bekijken we de methoden Random Sampling, Random Fit, Hill Climbing, en Simulated Annealing en bespreken we uitvoerig de resultaten. We kiezen de best gevonden methode, Simulated Annealing, en proberen deze methode door diverse aanpassingen te optimaliseren voor ons probleem. Tot slot trekken we enkele conclusies en geven we ons best gevonden rooster.

1 Introductie

Roosters zijn van uiterst belang in het hedendaags leven. Ze geven veel structuur aan verscheidene zaken. De kleinste planning die je in je hoofd hebt, is als een rooster te beschouwen. Vaak is zo'n planning niet één die je gemakkelijk zal ontschieten, voor de reismogelijkheden van Science Park naar Berkhout, een track die je niet dagelijks afreist, is een visueel rooster zeer interessant. Niet alleen voor jou als gebruiker is een goed rooster noodzakelijk, maar vooral voor de makers, opdat je reis naar elke bestemming spoedig kan verlopen.

Wegens hun enorme belang in het hedendaags leven, is het evident dat er uitgebreid onderzoek wordt gedaan naar het bepalen van optimale roosters[1]. In onze case zijn we ook hiernaar op zoek voor een specifiek geval. Helaas hebben we te maken met een onoplosbaar probleem, wat betekent dat er geen algemene oplossing bekend is om voor elke toestand binnen afzienbare tijd een optimaal rooster te creëren.

Natuurlijk bestaat de optie om alle mogelijke roosters af te gaan en het beste rooster te selecteren. We doorzoeken dan de zogenaamde *toestandsruimte*, ook wel *state space* genoemd[2]. Helaas is de toestandsruimte vaak erg groot, waardoor het vinden van een optimaal rooster met deze methode zelfs met de modernste computers al gauw tientallen jaren kan duren.

2 Beschrijving van het probleem

In onze case hebben we te maken met 609 studenten. Elk van de studenten volgt tussen de één en vijf vakken. Er zijn 29 vakken. Verder zijn er zeven zalen, waarin de studenten les hebben. Het is aan ons om de studenten in te delen in groepen en een weekrooster te maken. Het rooster moet aan de volgende eisen voldoen:

- Alle studenten dienen ingedeeld te worden;
- Vakken bestaan uit hoorcolleges en/of werkcolleges en/of practica. Elke activiteit van elk vak dient ingeroosterd te worden;
- Elke zaal is geschikt voor elk type college (hoorcollege, werkcollege, practicum);

[1] We refereren hier naar het zogenaamde *University Course Timetabling Problem, UCP*. Meer informatie is te vinden op http://www.unitime.org/uct_description.php.

[2] Zie Appendix A.1 voor een berekening van onze toestandsruimte.

– Bij werkgroepen en practica moeten de studenten, afhankelijk van de capaciteit, worden opgedeeld in zo min mogelijk groepen;
– Een college duurt van 9:00-11:00, 11:00-13:00, 13:00-15:00 of 15:00-17:00 op een werkdag[3]. Eén zo'n periode van twee uur wordt een *tijdsslot* genoemd. Elke activiteit bestaat na roostering uit een paar tijdsslot-zaal, zogenaamd een *zaalslot*.

Er zijn enorm veel roosters te maken die voldoen aan bovenstaande criteria. Een student wil dat hij/zij zo veel mogelijk opsteekt bij de vakken. Sommige van deze roosters zijn dan ook beter dan andere. Om te onderzoeken welke roosters beter zijn, is een scorefunctie nodig.

2.1 Scorefunctie

Een scorefunctie is noodzakelijk om roosters met elkaar te vergelijken. Zo kunnen we onderzoeken welke roosters beter zijn, en dan ook tot minder klachten leiden. We definiëren de scorefunctie aan de hand van de volgende bonus- en maluspunten:

– *Capaciteitpunten:* Voor elk zaalslot valt één maluspunt per student die wel in dit slot ingedeeld is, maar niet meer in de zaal past;
– *Conflictpunten:* Voor iedere student die meerdere activiteiten in één tijdsslot heeft (een roosterconflict), geldt 1 maluspunt per conflict;
– *Configuratiepunten:* Elk vak van twee tot vier maximaal over de week verdeelde activiteiten[4] levert 20 bonuspunten op. Voor ieder vak van x activiteiten geldt dat ze 10 maluspunten opleveren als ze op $x-1$ dagen geroosterd zijn, 20 voor $x-2$ en 30 voor $x-3$.

Een rooster dat aan eerder genoemde criteria voldoet, geven we 1000 punten. We vinden met de zojuist gedefinieerde bonus- en maluspunten de volgende scorefunctie:

$$score := 1000 + capaciteitpunten + conflictpunten + configuratiepunten. \tag{2.1}$$

2.2 Doel

Het doel van dit onderzoek is een algoritme vinden waarmee de case zo goed mogelijk wordt opgelost. Hiermee bedoelen we dat dit algoritme consistent binnen afzienbare tijd een rooster vindt met een zo hoog mogelijke score. We testen hiervoor verschillende algoritmen, die we door kleine aanpassingen voor onze case proberen te optimaliseren. Allereerst berekenen we voor dit onderzoek een bovengrens van onze toestandsruimte[2]. Vervolgens bestuderen we de methoden *Random Sampling*, *Hill Climbing* en *Simulated Annealing*. Op basis van onze resultaten trekken we tot slot enkele conclusies.

3 Materialen

Bij dit onderzoek gebruiken we de volgende data, allen in csv-formaat:

– Een lijst van studenten met hun vak-inschrijvingen;
– Een lijst van vakken en het aantal hoorcolleges, werkcolleges en practica per vak (en het maximum aantal studenten per werkcollege-/practicumgroep);
– Een lijst van zalen met hun capaciteit.

[3] In onze case heeft de grootste zaal, *C0.110*, ook nog een tijdsslot van 17:00-19:00. Het gebruik hiervan kost 50 maluspunten. We hebben besloten deze optie niet te gebruiken; Zie Appendix A.4 voor een motivatie.
[4] We noemen twee activiteiten *maximaal verdeeld* als ze op ma-do of di-vr ingeroosterd zijn. Voor drie activiteiten is dit op ma-wo-vr en voor vier activiteiten op ma-di-do-vr.

We gebruiken Python-versie 3.6. Verder is maken we gebruik van de Python module *Reportlab 3.4.0* voor de visualisatie van het rooster. We hebben drie verschillende computers tot onze beschikking[5]. Echter, om enige meetfouten te voorkomen, zijn alle resultaten in dit verslag gegenereerd door de *HP ENVY 13-d010nd*.

4 Methoden

4.1 Random Sampling

Onze eerste methode is *Random Sampling*. We bekijken voor elk vak per activiteit hoeveel groepen er moeten zijn. Elk van deze groepen krijgt een eigen les. In totaal zijn er 129 lessen. Aangezien er vier tijdssloten per dag per zaal zijn, en vijf dagen en zeven zalen, zijn er in totaal $4 \cdot 5 \cdot 7 = 140$ plekken op te vullen in het rooster. Er zullen dus elf lege plekken in het rooster zijn.

Allereerst bekijken we per vak de studenten en plaatsen we elk van hen per vakactiviteit, indien mogelijk, willekeurig in een groep. Uiteindelijk hebben we dan 129 lessen met bijbehorende studenten. Nu geven we elk van deze lessen willekeurig een plek in een leeg rooster. Deze methode, waarin we een volledig willekeurig rooster creëren, voldoet aan de vereisten.

Door dit algoritme een aantal keer toe te passen en een histogram te maken van de bijbehorende scores, krijgen we een indruk van de *toestandsruimte*[2].

4.2 Random Fit

Merk op dat we hogere scores kunnen krijgen door de *capaciteitpunten* (zie (2.1)) te vermijden. We kunnen namelijk ook willekeurig in plaats van uit alle zalen, uit de zalen die groot genoeg zijn een zaal kiezen. Op deze wijze ontstaat een nieuw algoritme, dat we *Random Fit* noemen.

4.3 Hill Climbing

Een ander algoritme dat we gaan gebruiken is Hill Climbing. Dit algoritme heeft het volgende verloop:

- Neem een beginrooster (met behulp van Random Sampling of Random Fit) en bepaal de score.
- Breng een 'kleine wijziging' aan in het rooster en neem het nieuwe rooster als beginrooster.
- Herhaal totdat er een lokaal maximum is bereikt.

We kunnen op twee manieren een kleine wijziging aanbrengen in een rooster. Ten eerste kunnen we twee lessen omwisselen in het rooster. We zoeken dan de verwisseling van lessen die de score het meest verhoogt. Daarnaast kunnen we twee studenten tussen verschillende werk- of practicumgroepen omwisselen. Hierbij kiezen we willekeurig een vak uit en accepteren we de eerste verwisseling die we tegenkomen die de score verhoogt.

Merk op dat een verwisseling van studenten in het algemeen minder invloed op de score heeft dan een verwisseling van lessen, aangezien een studentenswap enkel roosterconflicten op kan lossen. Het zou dus kunnen lonen om naar verhouding meer studentenswaps dan lessenswaps uit te voeren. Om redenen die later duidelijk zullen worden, gaan we hier niet mee experimenteren bij het Hill Climbing algoritme (zie sectie 4.4).

Als experiment voeren we een aantal keer het algoritme uit om zo de eindscores te vergelijken met het volgende algoritme.

[5] Zie Appendix A.2 voor meer informatie.

4.4 Simulated Annealing

De methode Simulated Annealing lijkt erg op Hill Climbing en verloopt als volgt:

- Neem een beginrooster.
- Voer een willekeurige swap (studenten- of lessenswap) uit om een nieuw rooster te vinden.
- Accepteer het nieuwe rooster met de zogenaamde acceptatiekans.
- Verander de temperatuur volgens het koelingsschema.

Een lessenswap is eenvoudig: kies willekeurig twee colleges in het rooster en verwissel ze. Een studentenswap is niet veel moeilijker. Kies namelijk willekeurig een vak, en kies hiervan willekeurig twee werkcollege- of practicumgroepen. Kies nu uit de eerste groep willekeurig een student en uit de tweede groep een willekeurige plaats (ongeacht of er al een student is op de gekozen plaats). Zet de gekozen student op de gekozen plaats. Als er al een student op die plaats was, wordt deze in de groep van de eerste student gezet.

Voor de acceptatiekans nemen we $P_{acceptatie} = e^{(S_{\text{nieuw}} - S_{\text{oud}})/T}$, afhankelijk van de scores S_{oud} en S_{nieuw}, en van de temperatuur T. Indien het nieuwe rooster geen lagere score geeft dan het oude, is $S_{\text{nieuw}} - S_{\text{oud}} \geq 0$, dus $P_{acceptatie} \geq e^0 = 1$; dan accepteren we het rooster altijd.

In een Simulated Annealing algoritme is er meer vrijheid om te optimaliseren dan in het geval van Hill Climbing. Er is niet alleen de keuze uit verhoudingen van swaps, maar ook starttemperatuur, eindtemperatuur en het koelingsschema. Allereerst zoeken we de optimale starttemperatuur. We kiezen een vaste swapverhouding en koelingsschema en testen een aantal verschillende starttemperaturen.

Vervolgens testen we verschillende koelingsschema's. Eerst kijken we naar de volgende enkelvoudige temperatuurfuncties, daarna zullen we ook kijken naar wat complexere schema's. Hierbij is T_i de temperatuur na i iteraties en N het totaal aantal iteraties.

$$\text{Lineair:} \quad T_i = T_{i-1} - \frac{T_0 - T_N}{N} \qquad \text{Exponentieel:} \quad T_i = T_{i-1} \left(\frac{T_N}{T_0}\right)^{\frac{1}{N}}$$

$$\text{Sigmoïde:} \quad T_i = \frac{T_0 - T_N}{1 + e^{c\left(\frac{i}{N} - \frac{1}{2}\right)}} + T_N \qquad \text{Geman:} \quad T_i = \frac{T_0}{1 + \log(i+1)}$$

De sigmoïde functie hangt af van een constante c. De waarde van deze constante bepaalt hoe snel de temperatuur afneemt. We kiezen voor de waarde $c = 20$ in dit onderzoek[6].

Naast de bovenstaande enkelvoudige koelingsschema's beschouwen we ook vier samengestelde schema's. Deze maken we door de bovenstaande schema's te combineren en/of de temperatuur opnieuw te verhogen. We gebruiken de volgende temperatuurfuncties[7]:

- *Lineair-Sigmoïde*: De eerste helft van de iteraties de lineaire en vervolgens de sigmoïde functie.
- *Exponentieel-Lineair*: De eerste helft van de iteraties de exponentiële, daarna de lineaire functie.
- *Exponentieel-Sigmoïde*: De eerste helft de exponentiële functie, de tweede de sigmoïde functie.
- *Exponentieel met naverhitting*: Een exponentiële functie die een aantal keer, regelmatig verdeeld, opnieuw verhit naar een deel van de starttemperatuur.

We laten per koelingsschema het algoritme vijf keer een uur lang draaien met dezelfde vijf beginroosters voor elk schema. We nemen steeds $T_0 = 1$ °C, $T_N = 0,00001$ °C en $N = 400.000$. Als het

[6] De aandachtige lezer zal hebben gemerkt dat deze sigmoïde functie niet precies uitkomt op de waarde van T_0 (resp. T_N) door $i = 0$ (resp. N) in te vullen. Dit is echter geen probleem, aangezien de waardes wel altijd in het interval (T_N, T_0) liggen en bovendien zeer dichtbij (relatief t.o.v. de gebruikte T_0 en T_N) de eigenlijke waardes.

[7] Zie Appendix A.3 voor een exacte notatie van deze temperatuurfuncties. Zie [2] voor een onderzoek waarin dit effectief bleek te zijn.

uur voorbij is voordat de N iteraties zijn bereikt stopt het algoritme. Als swapverhouding nemen we een $1:1$ verhouding. Voor de exponentiële functie met naverhitting herverhitten we op een kwart, halverwege en op driekwart van de iteraties.

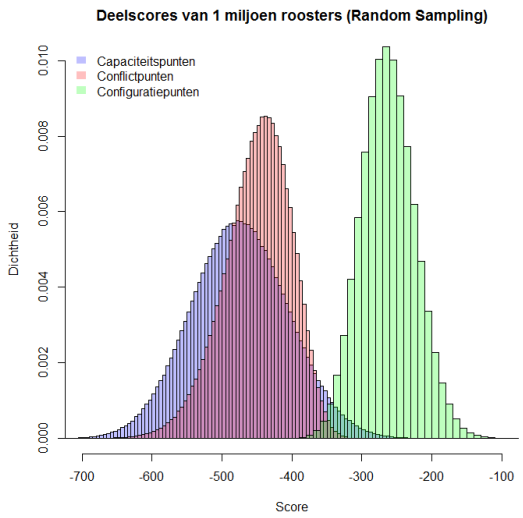
Na het beste koelingsschema te hebben gevonden testen we ook nog verschillende swapverhoudingen. Dit gaat op dezelfde manier. We nemen T_0, T_N en N zoals hierboven. Voor het koelingsschema gebruiken we de lineair-sigmoïde functie. We testen een aantal verhoudingen; $1:10$, $1:5$, $1:2$, $1:1$ en $2:1$.[8] Uiteindelijk combineren we alle resultaten (inclusief Hill Climbing resultaten) om het beste algoritme te vinden.

5 Resultaten

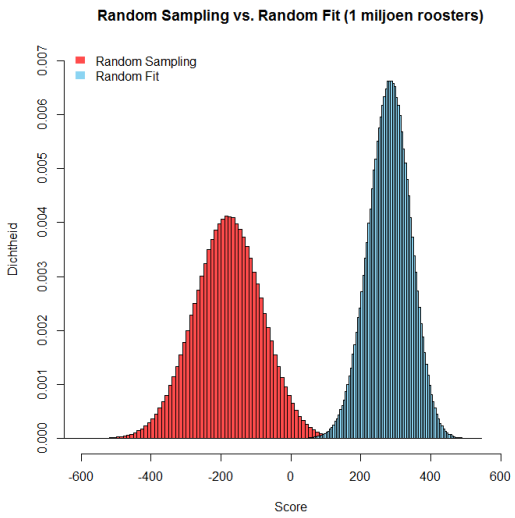
5.1 Random Sampling en Random Fit

We starten deze sectie met de resultaten over het Random Sampling. In figuur 1 zien we de deelscores van één miljoen roosters, zoals gedefiniëerd in (2.1). De capaciteitspunten hebben zowel het laagste gemiddelde als de grootste variantie. De configuratiepunten hebben het hoogste gemiddelde en de kleinste variantie.

In figuur 2 is te zien dat het Random Fit algoritme niet alleen gemiddeld een score brengt van ongeveer 500 punten hoger, maar ook een lagere variantie heeft dan het Random Sampling. Merk op dat Random Fit-roosters nagenoeg allemaal een score boven de nul hebben, terwijl Random Sampling-roosters hier niet relatief vaak boven liggen.



Figuur 1: Deelscores van roosters gegenereerd door Random Sampling.



Figuur 2: Scores van roosters gegenereerd door Random Sampling (rood) en Random Fit (blauw).

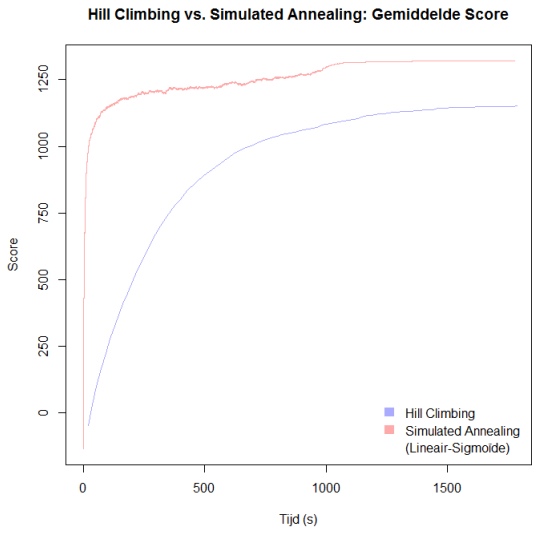
5.2 Hill Climbing en Simulated Annealing

5.2.1 Hill Climbing vs. Simulated Annealing

In figuur 3 zien we de gemiddelde score van roosters gemaakt met behulp van de Hill Climbing- en Simulated Annealing-algoritmes. Het valt op dat de score van het Simulated Annealing-algoritme veel

[8] De verhouding $a:b$ geeft aan dat er per a studentenswaps b lessenswaps worden gedaan.

sneller stijgt dan de score van Hill Climbing. Verder zien we dat de score bij Simulated Annealing vanaf het begin veel hoger is. In het bijzonder is de eindscore dus ook hoger.



Figuur 3: Gemiddelde score van de Hill Climbing- en Simulated Annealing-algoritmes. Deze algoritmes zijn gestart met vijf verschillende beginroosters, gegenereerd door Random Sampling.

5.2.2 Temperatuurfuncties

We bekijken de resultaten van de temperatuurfuncties beschreven in sectie 4.4. In figuur 4 is te zien dat de Geman functie uiteindelijk de laagste score oplevert. Door Simulated Annealing toe te passen met deze temperatuurfunctie bereiken we zeer snel een relatief goede score, maar doordat de score op een gegeven moment niet meer hard stijgt, wordt deze ingehaald door alle overige versies van Simulated Annealing. Op hetzelfde moment daalt de temperatuur ook niet meer hard.

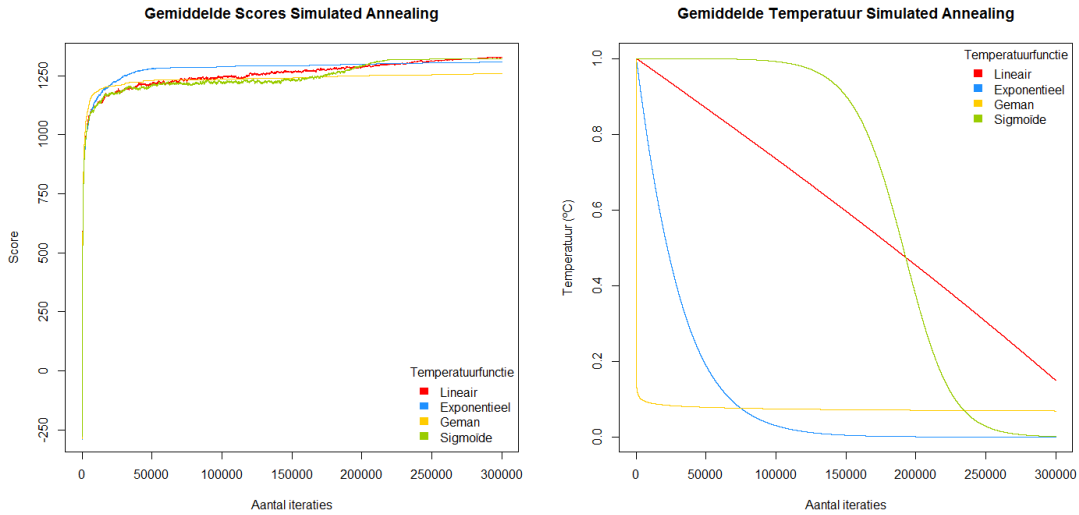
De overige eindscores liggen wat dichter bij elkaar. Er is te zien dat de exponentiële functie na 50.000 iteraties de hoogste score heeft. De temperatuur is dan gezakt naar 0,2° C. Vanaf dat moment gaat de score niet meer ver omhoog, na 150.000 iteraties is het nulpunt ook nagenoeg bereikt. Toch wordt pas na ongeveer 200.000 iteraties de exponentiële functie weer ingehaald.

De sigmoïde en lineaire temperatuurfuncties lijken voor het grootste deel gelijkwaardige scores te geven. De sigmoïde functie lijkt het later te gaan verliezen, maar wanneer de temperatuur van de sigmoïde functie hard begint te dalen, komt de score van de sigmoïde functie boven die van de lineaire functie. Op het laatste moment, na ongeveer 280.000 iteraties, blijkt de lineaire functie de eindstrijd te winnen.

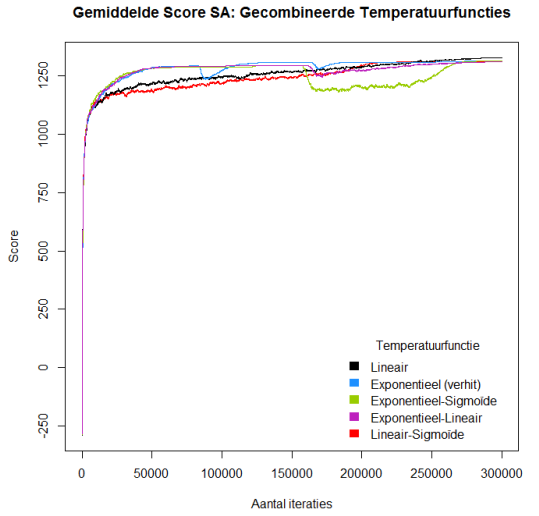
Vervolgens bekijken we de resultaten van de complexere temperatuurfuncties. We zien in figuur 5 dat het verhitten en het combineren van functies hoge scores oplevert. Echter, op het moment van veranderen van functie of bij het verhitten daalt de temperatuur enorm. Uiteindelijk komen de complexere temperatuurfuncties niet boven de gemiddelde eindscore van de lineaire functie uit.

5.2.3 Verhoudingen van swaps

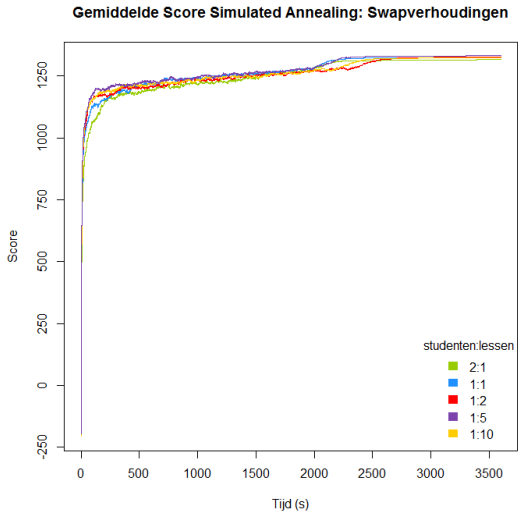
We presenteren de grafiek in figuur 6 van een algoritme voor verschillende verhoudingen van swaps van studenten en lessen. Zoals al vermeld in sectie 4.4 hebben wij dit uitsluitend voor een Simulated Annealing algoritme gedaan, met een lineair-sigmoïde temperatuurfunctie. We zien dat meerdere verhoudingen dusdanig dicht bij elkaar liggen dat de krommen elkaar meermaals snijden. Wel is het duidelijk dat de $1:5$ verhouding uiteindelijk de hoogste score oplevert en zowel de $2:1$ als de $1:10$ verhouding significant lager liggen dan de andere drie.



Figuur 4: De gemiddelde scores voor verschillende complexe temperatuurfuncties met hun bijbehorende temperatuur volgens Simulated Annealing. Bij elke temperatuurfunctie wordt gestart met hetzelfde beginrooster, waarvan er vijf vooraf d.m.v. Random Sampling zijn gekozen. Het gemiddelde wordt per temperatuurfunctie genomen over deze vijf metingen.



Figuur 5: De gemiddelde scores voor verschillende complexere temperatuurfuncties berekend met hun bijbehorende temperatuur volgens Simulated Annealing zoals bij figuur 4.



Figuur 6: De gemiddelde scores voor verschillende verhoudingen studenten:lessen volgens Simulated Annealing, berekend zoals bij figuur 4. De bijbehorende temperatuurfunctie is de lineaire-sigmoïde.

6 Discussie

6.1 Conclusie

Ten eerste beschouwen we beide random algoritmes. Aangezien roosters gegenereerd door het Random Fit algoritme nagenoeg uniform hogere scores hebben dan die gevonden door Random Sampling, concluderen we dat het Random Fit algoritme ook beter is.

Verder kunnen we uit figuur 3 concluderen dat Simulated Annealing met elke door ons gekozen temperatuurfunctie een veel hogere eindscore levert dan Hill Climbing. Uiteraard geeft Simulated Annealing ook hogere scores dan de random algoritmes, dus Simulated Annealing is de beste methode (van alle algoritmes die we hebben bekeken).

Zoals de resultaten in sectie 5.2.2 tonen, levert de lineaire temperatuurfunctie de hoogste eindscores op. Hieruit concluderen we dat dit koelingsschema de beste is van alle geteste schema's. Deze uitspraak moet echter wel met een korrel zout genomen worden, omdat bijvoorbeeld de scores van het lineair-sigmoïde algoritme zeer dicht op die van het lineaire algoritme volgen.

De enige verhoudingen waarvoor het Simulated Annealing algoritme bijzonder veel afwijkt van de andere zijn de verhoudingen $2:1$ en $1:10$. Door veel runs uit te voeren, kan deze uitspraak wiskundig bewezen worden. De drie verhoudingen $1:1$, $1:2$ en $1:5$ zijn beter dan de overige twee. Het is intuïtief duidelijk dat verhoudingen buiten dit gebied (zoals bijvoorbeeld $100:1$ of $1:100$) niet beter zullen zijn dan de verhoudingen tussen $1:1$ en $1:5$ omdat de 'randverhoudingen' $2:1$ en $1:10$ al slechter zijn dan de verhoudingen daartussenin.

Door alle bovenstaande resultaten te combineren vinden we dat het Simulated Annealing algoritme volgens het lineaire koelingsschema met een swapverhouding tussen $1:1$ en $1:5$ de beste combinatie is van de geteste variabelen. Hier nemen we wel aan dat elke temperatuurfunctie dezelfde optimale swapverhouding heeft.

Terugkijkend op het gestelde doel in sectie 2.2, kunnen we zeker zeggen dat we ons doel hebben bereikt. We hebben namelijk het beste algoritme gevonden uit de verzameling algoritmes waarmee we zijn begonnen. Dit betekent natuurlijk niet dat we ook het best mogelijke algoritme hebben gevonden.

6.2 Limitaties en verbeteringen

De eerste duidelijke limitatie is dat we veel experimenten slechts vijf keer hebben uitgevoerd. Hierdoor zijn onze conclusies minder betrouwbaar. Het zou namelijk kunnen dat de verdeling van de kwaliteit van een algoritme een hele grote variantie heeft en we voor bepaalde algoritmes 'geluk' hebben gehad. Het zou dus beter zijn om alle tests vaker te doen, maar vanwege tijdsgebrek was dit niet mogelijk.

Verder zal de aandachtige lezer hebben gemerkt dat er geen concrete resultaten zijn over de optimale start- en eindtemperatuur. De reden hiervoor is dat het betrekkelijk moeilijk is om dit paar te bepalen, aangezien deze variabelen overaftelbaar[9] veel waardes aan kunnen nemen. Hoewel het moeilijk is, is het niet onmogelijk om deze waardes tot op zekere significantie te benaderen.

Zoals eerder vermeld is het zeer waarschijnlijk dat het beste algoritme dat wij hebben gevonden niet het beste mogelijke algoritme is. Dit onderzoek kan te allen tijde aangescherpt worden door meer temperatuurfuncties en swapverhoudingen te testen. Bovendien kan natuurlijk ook een compleet ander soort algoritme worden gebruikt, zoals bijvoorbeeld een Fireworks of Genetisch algoritme.

Dankwoord

Wij danken een ieder die op enige wijze het onderzoek positief heeft beïnvloed. Onze dank gaat uit naar Quinten van der Post, die ons door het hele onderzoeksproces heeft geleid. Verder bedanken wij onze docenten Bart van Baal en Daan van den Berg, die ons, naast onze begeleider, uitstekende feedback hebben gegeven tijdens de werkcolleges. Met hen bedanken wij ook onze medestudenten voor hun positieve kritiek. Laatstgenoemde willen wij ook bedanken voor zijn duidelijke hoorcolleges, en zijn enthousiasme bij het doceren. De overige hoorcollegedocenten, Bas Terwijn en Edwin Steffens, verdienen uiteraard ook enige annotatie in dit dankwoord.

Referenties

1. Luke, B.: Simulated Annealing Cooling Schedules. http://www.btluke.com/simanf1.html (2002)
2. Abramson, D., Krishnamoorthy, M. Dang, H.: Simulated Annealing Cooling Schedules for the School Timetabling Problem. Monash University (1997)

[9] Dit wil zeggen dat er niet over alle mogelijke waardes geïtereerd kan worden.

A Appendix

A.1 Toestandsruimte

We bepalen het aantal slots M waarin lessen worden geroosterd en berekenen het aantal lessen n dat ingeroosterd moet worden en schrijven

$$\frac{M!}{(M-n)!}$$

voor het aantal manieren om lessen in te roosteren. Voor het aantal manieren om studenten in groepen te zetten berekenen we een bovengrens. Per vak nemen we de capaciteit van een college en het aantal studenten s dat het vak volgt, en berekenen we per college het aantal groepen g dat nodig om alle studenten in te roosteren. Dan is de waarde

$$\prod_{\text{lessen}} g^s$$

een bovengrens voor het aantal manieren waarop studenten kunnen worden ingeroosterd. Vermenigvuldigen we de twee bovenstaande getallen nu met elkaar, dan vinden we de bovengrens

$$2,0674 \cdot 10^{940}.$$

A.2 Gebruikte Computers

Tabel 1: De computers die wij voor dit onderzoek gebruiken.

	HP ENVY 13-d010nd	ASUS F555LA-DM2095T	MSI GE62-6QD Apache Pro
Processor	Intel® Core™ i5-6200U 2.3 GHz (2.8 GHz)	Intel® Core™ i5-5200U 2.2 GHz (2.7 GHz)	Intel® Core™ i7-6700HQ 2.6 GHz (3.5 GHz)
Opslag	128 GB SATA M.2 SSD	256 GB SATA III SSD	128 GB PCIe 3.0 M.2 SSD
Werkgeheugen	2x2GB 1600 MHz DDR3L	1x4GB 1600 MHz DDR3L	2x8GB 2133 MHz DDR4
Besturingssysteem	Windows® 10 Home 64	Windows® 10 Home 64	Windows® 10 Home 64

A.3 Samengestelde Temperatuurfuncties

$$\text{Lineair-Sigmoïde:} \quad T_i = \begin{cases} T_{i-1} - \frac{T_0 - T_N}{N} & i < \frac{N}{2} \\ \frac{T_0 - T_N}{1 + e^{c\left(\frac{i}{N} - \frac{1}{2}\right)}} + T_N & i \geq \frac{N}{2} \end{cases}$$

$$\text{Exponentieel-Lineair:} \quad T_i = \begin{cases} T_{i-1} \left(\frac{T_N}{T_0}\right)^{\frac{1}{N}} & i < \frac{N}{2} \\ T_{i-1} - \frac{T_0 - T_N}{N} & i \geq \frac{N}{2} \end{cases}$$

$$\text{Exponentieel-Sigmoïde 2:} \quad T_i = \begin{cases} T_{i-1} \left(\frac{T_N}{T_0}\right)^{\frac{1}{N}} & i < \frac{N}{2} \\ \frac{T_0 - T_N}{1 + e^{c\left(\frac{i}{N} - \frac{1}{2}\right)}} + T_N & i \geq \frac{N}{2} \end{cases}$$

$$\text{Exponentieel (met naverhitting):} \quad T_i = \begin{cases} \alpha_j \cdot T_0 & i = \alpha_j N \\ T_{i-1} \left(\frac{T_N}{T_{\alpha_j N}}\right)^{\frac{1}{N}} & \text{elders} \end{cases}$$

De α_j zijn een rij getallen tussen 0 en 1. Neemt men bijvoorbeeld de rij van lengte 1 met $\alpha_1 = \frac{1}{2}$, dan wordt er één keer verhit bij $i = \frac{N}{2}$. In dit onderzoek nemen we $(\alpha_j) = (\frac{1}{4}, \frac{1}{2}, \frac{3}{4})$.

A.4 Beste rooster

Hieronder ons best gevonden rooster, met een score van 1351:

SCHEDULE	A1.08	A1.06	A1.04	B0.201	A1.10	C1.112	C0.110
Monday							
9-11	Moderne Databases - practical - Group A	Reflectie op de digitale cultuur - seminar - Group A	Autonomous Agents 2 - lecture	Interactie-ontwerp - lecture	Heuristieken 1 - seminar - Group A	Moderne Databases - practical - Group C	Collectieve Intelligentie - lecture
11-13	Reflectie op de digitale cultuur - seminar - Group B	Netwerken en systeembeveiliging - practical - Group B	Webprogrammeren en databases - lecture	Informatie- en organisatieontwerp - lecture	Bioinformatica - practical - Group B	Lineaire Algebra - lecture	Machine Learning - lecture
13-15	Project Genetic Algorithms - practical - Group A	Architectuur en computerorganisatie - lecture	Calculus 2 - seminar - Group C	Technology for games - lecture	Bioinformatica - practical - Group A	Bioinformatica - practical - Group C	Algoritmen en complexiteit - lecture
15-17	Data Mining - practical - Group A	Data Mining - practical - Group B	Databases 2 - seminar - Group B	Calculus 2 - seminar - Group B	Software engineering - seminar - Group A	Data Mining - practical - Group C	Compilerbouw - lecture
Tuesday							
9-11	Webprogrammeren en databases - seminar - Group A		Calculus 2 - seminar - Group A	Bioinformatica - seminar - Group A	Reflectie op de digitale cultuur - lecture	Data Mining - seminar - Group C	Collectieve Intelligentie - lecture
11-13	Moderne Databases - seminar - Group A	Autonomous Agents 2 - lecture	Data Mining - seminar - Group A	Moderne Databases - seminar - Group C	Compilerbouw practicum - practical - Group A	Advanced Heuristics - practical - Group A	Kansrekenen 2 - lecture
13-15		Advanced Heuristics - practical - Group B	Heuristieken 2 - lecture	Bioinformatica - seminar - Group B	Moderne Databases - seminar - Group B	Heuristieken 1 - seminar - Group B	Compilerbouw - lecture
15-17	Webprogrammeren en databases - seminar - Group B	Data Mining - seminar - Group B	Software engineering - practical - Group B	Algoritmen en complexiteit - seminar - Group A	Software engineering - practical - Group A	Informatie- en organisatieontwerp - lecture	Bioinformatica - seminar - Group C
Wednesday							
9-11	Collectieve Intelligentie - practical - Group B	Netwerken en systeembeveiliging - practical - Group C	Collectieve Intelligentie - practical - Group A	Programmeren in Java 2 - practical - Group B	Technology for games - lecture	Zoeken sturen en bewegen - practical - Group B	Programmeren in Java 2 - practical - Group F
11-13	Compilerbouw practicum - practical - Group C	Collectieve Intelligentie - practical - Group C	Collectieve Intelligentie - practical - Group D		Zoeken sturen en bewegen - practical - Group A	Analysemethoden en -technieken - lecture	Project Numerical Recipes - practical - Group C
13-15		Programmeren in Java 2 - practical - Group A	Programmeren in Java 2 - practical - Group C		Programmeren in Java 2 - practical - Group E	Programmeren in Java 2 - practical - Group D	Bioinformatica - lecture
15-17	Compilerbouw practicum - practical - Group B	Netwerken en systeembeveiliging - practical - Group D	Project Numerical Recipes - practical - Group A			Zoeken sturen en bewegen - practical - Group C	Databases 2 - lecture
Thursday							
9-11	Autonomous Agents 2 - seminar - Group A	Autonomous Agents 2 - seminar - Group B	Interactie-ontwerp - lecture	Project Genetic Algorithms - practical - Group B		Machine Learning - lecture	Collectieve Intelligentie - lecture
11-13		Architectuur en computerorganisatie - lecture	Informatie- en organisatieontwerp - seminar - Group B	Compilerbouw - seminar - Group B	Netwerken en systeembeveiliging - practical - Group A	Algoritmen en complexiteit - practical - Group B	Lineaire Algebra - lecture
13-15	Webprogrammeren en databases - practical - Group A	Project Numerical Recipes - practical - Group B		Webprogrammeren en databases - practical - Group B	Moderne Databases - lecture	Project Genetic Algorithms - practical - Group C	Data Mining - lecture
15-17	Autonomous Agents 2 - seminar - Group C	Informatie- en organisatieontwerp - seminar - Group A	Software engineering - seminar - Group B	Bioinformatica - lecture	Compilerbouw - seminar - Group A	Reflectie op de digitale cultuur - seminar - Group C	Algoritmen en complexiteit - practical - Group A
Friday							
9-11		Algoritmen en complexiteit - seminar - Group B	Heuristieken 2 - seminar - Group A	Heuristieken 1 - lecture	Bioinformatica - lecture	Compilerbouw - practical - Group A	Compilerbouw - practical - Group B
11-13	Autonomous Agents 2 - practical - Group B	Technology for games - seminar - Group A	Autonomous Agents 2 - practical - Group C	Collectieve Intelligentie - seminar - Group B	Collectieve Intelligentie - seminar - Group C	Reflectie op de digitale cultuur - lecture	Calculus 2 - lecture
13-15	Technology for games - seminar - Group B	Advanced Heuristics - lecture	Databases 2 - seminar - Group A	Collectieve Intelligentie - seminar - Group D	Heuristieken 2 - seminar - Group B	Informatie- en organisatieontwerp - practical - Group B	Software engineering - lecture
15-17	Autonomous Agents 2 - practical - Group A	Moderne Databases - practical - Group B	Webprogrammeren en databases - lecture	Data Mining - lecture	Collectieve Intelligentie - seminar - Group A	Informatie- en organisatieontwerp - practical - Group A	Kansrekenen 2 - lecture

Aangezien 1400 het theoretisch maximum is, en het avondslot van 17:00-19:00 per inzet 50 maluspunten oplevert, kunnen we met zekerheid zeggen dat het optimale rooster voor deze case geen avondslot gebruikt indien de score van het beste rooster hoger dan $1400 - 50 = 1350$ is. Ons beste rooster heeft deze score net aan bereikt. Hiermee kunnen we concluderen dat het avondslot de case niet het best oplost, aangezien er hogere scores te vinden zijn wanneer we deze optie niet gebruiken.