Upperbound statespace
De upperbound zonder dat er naar de constraints wordt gekeken is heel hoog. Namelijk slots!/(slots - sessions)! voor slots = 140 is en sessions = 72 geeft dit 5.42815304e144. Nu nemen zeggen we dat er 72 sessions zijn, maar als werkcolleges en practica ook worden ingedeeld zal het aantal sessions meer worden en de upperbound dus hoger.

Echter zitten er vele constraints aan het indelen van de vakken dus het werkelijke aantal roosters is een stuk kleiner. Als we aannemen dat we de vakken indelen op volgorde van het aantal colleges dat ze hebben, waarbij we beginnen met de vakken die de meeste colleges hebben ziet de formule er al anders uit.

Aantal colleges	**Vakken met dat aantal colleges**
1	7
2	10
3	5
4	5
5	2

Het begin van de formule is simpel: het aantal opties voor het indelen van de vakken met 5 colleges. Deze vakken hebben namelijk elke dag een college:
140 * (140 - 28) * (140 - 2 * 28) * (140 - 3 * 28) * (140 - 4 * 28) * 135 * (135 - 27) * (135 - 2 * 27) * (135 - 3 * 27) * (135 - 4 * 27).

Omdat voor de vakken met 2 t/m 4 colleges per vak meer aannames nodig zijn, en we die op het moment nog niet hebben gedaan. Zullen we hierbij nog niet naar de verspreiding constraint kijken. Er zijn 2 * 10 + 3 * 5 + 4 * 5 = 55 sessies in totaal bij deze vakken. Voor deze sessies zijn er na het indelen van de vakken met 5 colleges nog 130 slots vrij. Dus dan komt de term 130!/(130-55)! bij de formule.

Het laatste deel van de formule is wel weer simpel: de vakken met 1 college hebben namelijk (gekeken naar alleen de verspreiding constraint) geen constraints. Wanneer we deze aan het einde indelen zijn er nog 140 - 72 + 7 = 75 slots vrij. Dus dan is het laatste deel van de formule 75 * 74 * 73 * 72 * 71 * 70 * 69.

Een lagere upperbound is dus:
140 * (140 - 28) * (140 - 2 * 28) * (140 - 3 * 28) * (140 - 4 * 28) * 135 * (135 - 27) * (135 - 2 * 27) * (135 - 3 * 27) * (135 - 4 * 27) * 130!/(130-55)! * 75 * 74 * 73 * 72 * 71 * 70 * 69 = 9.272711e141.
Dit is dus duidelijk een kleiner dan de oorspronkelijke upperbound maar nog steeds een enorm groot getal.

Upperbound maluspunten

- Voor het gebruik van het avondslot kunnen er max 5 * 20 = 100 maluspunten zijn.
- Voor studenten die niet in de zaal passen kunnen er max 1332 maluspunten zijn. Dit is berekend met de aanname dat alle colleges in het kleinste lokaal geroosterd zijn dat een capaciteit heeft van 20 mensen. Wel is er rekening mee gehouden met het maximum aantal studenten dat mag deelnemen aan een werkcollege en practicum. Voor de verschillende type colleges zijn dus verschillende formules die er als volgt uitzien:
    - lecture: (E(studenten) - 20) * #lectures
    - tutorial: (max_studenten_tutorial - 20) * ceiling(E(studenten) / max_studenten_tutorial
    - practical: (max_studenten_practical - 20) * ceiling(E(studenten) / max_studenten_practical

  Als voor alle vakken de som van deze waarden wordt genomen en dit bij elkaar wordt opgeteld kom je dus op 1332 maluspunten.
- Voor de verspreiding van vakken kunnen er max 430 maluspunten zijn. Dit is berekend door het aantal vakken dat 1 activiteit heeft * 0 + het aantal vakken dat 2 activiteiten heeft * 10 + het aantal vakken dat 3 activiteiten heeft * 20 etc.
- Voor de vakconflicten van individuele studenten kunnen er max 11417 maluspunten zijn. Dit is berekend door ervan uit te gaan dat alle activiteiten van de studenten op hetzelfde moment zijn geroosterd. Waarbij de definitie van een vakconflict is dat bij x activiteiten die tegelijk zijn gepland er x + (x - 1) + (x - 2) + … + 2 + 1 conflicten zijn.

De upperbound van het aantal maluspunten is dus 100 + 1332 + 430 + 11417 = 13279

Upperbound bonuspunten

- Voor de verspreiding van vakken kunnen er max (29 - 7) * 20 = 440 bonuspunten zijn. Waarbij 29 - 7 het aantal vakken met meer dan 1 college is.
- Voor het niet hebben van vakconflicten van individuele studenten kunnen er max 609 * 1 = 609 bonuspunten zijn.

De upperbound van het aantal bonuspunten is dus 440 + 609 = 1409.

<u>Complexiteit</u>

Wat maakt deze case moelijk:

Verschillende aspecten van dit probleem zorgen voor complexiteit in het toepassen en implementeren van onderdelen. In onze versie van een poging een oplossing te vinden voor dit probleem wordt gewerkt met verschillende classes, zoals ook te zien in het UML-diagram.

Rekening houden met constraints

Er zijn verschillende manieren om rekening te houden met de hard constraints. Er wordt bij deze versie van het oplossen van het probleem tijdens het inroosteren alleen een rooster gemaakt als deze valide is. Wanneer tijdens het inroosteren blijkt dat een rooster niet langer valide kan worden, wordt het inroosteren afgekapt en begonnen aan een nieuw rooster. Het zou hierdoor wel kunnen dat roosters die dichtbij een goed rooster zaten worden overgeslagen.

Wanneer door middel van algoritmes nieuwe roosters worden gecreëerd, moet ook worden gecontroleerd of deze nog aan de hard constraints voldoet. Dit wordt aan de hand van een functie gedaan.

Soft constraints worden gemeten en geïmplementeerd in algoritmes.

Avondslot
Een avondslot gebruiken in het rooster kan maximaal 100 maluspunten opleveren. Om dit te voorkomen is besloten dit avondslot buiten de mogelijkheden te laten. Dit zou echter wel betekenen dat mogelijk bonuspunten worden misgelopen voor bijvoorbeeld spreiding van de vakken. Het zou daarom wellicht beter zijn dit avondslot toch te implementeren. Op dit moment wordt gebruikt gemaakt van lijsten in lijsten om een schema te creëren. Op een positie in deze lijsten wordt één sessie geplaatst, dit representeert één sessie op één moment en één locatie in de tijd. Omdat er voor alle dagen en tijdsloten dezelfde iteraties worden gedaan en alleen voor één lokaal het avondslot beschikbaar is, kan het moeilijkheden geven dit avondslot mee te nemen in het schema zoals het nu is gecodeerd.

Groepen studenten en capaciteit
In eerste instantie werd tijdens het inroosteren geen rekening gehouden met de capaciteit van bijvoorbeeld werkcolleges of practica, maar in een later stadium wel. Dit zorgde ervoor dat er meer werkcolleges en practica van één specifiek vak ingeroosterd moesten worden. Omdat de sessies van één vak niet mogen overlappen met elkaar (en niet met die van bepaalde vakken die tegelijk te volgen moeten zijn), zorgde dat voor een probleem bij het creëren van een valide rooster. Om dit op te lossen is gekeken naar de studentgroepen die deze sessies volgen. Wanneer een werkcollege van een vak uit twee groepen bestaat, bijvoorbeeld groep A en groep B, en een practicum van ditzelfde vak ook uit twee groepen bestaat, weer groep A en groep B, is de aanname gedaan dat deze groepen uit dezelfde individuele studenten bestaan. Zo kan werkcollege van groep A overlappen met die van groep B, maar ook met het practicum van groep B. Hierdoor wordt meer ruimte gecreëerd voor oplossingen.

Individuele studenten
Zoals eerder beschreven wordt er tijdens het inroosteren vanuit gegaan dat een groep studenten die ingeroosterd wordt in een sessie dezelfde groep individuele studenten blijft voor het gehele verdere vak. Dit zorgt echter wel dat, wanneer algoritmes mogelijk worden ingesteld om verschillende individuele studenten te wisselen van sessie, óf de hele groep moet wisselen, óf de kans op het krijgen van een invalide rooster groter wordt. Groepen zouden namelijk tegelijk een sessie kunnen hebben waardoor de student één sessie zou moeten missen.
Dit zou kunnen worden opgelost door tijdens het indelen van het rooster geen uitzondering meer te maken voor verschillende vakactiviteiten binnen één vak. Dit houdt in dat verschillende groepen van één vakactiviteit, bijvoorbeeld de groepen A en B van een werkcollege, wel tegelijk kunnen worden ingeroosterd, maar dat niet langer werkcollege groep A ook tijdens practicum groep B mag worden ingeroosterd.

Complexiteit: Analyse

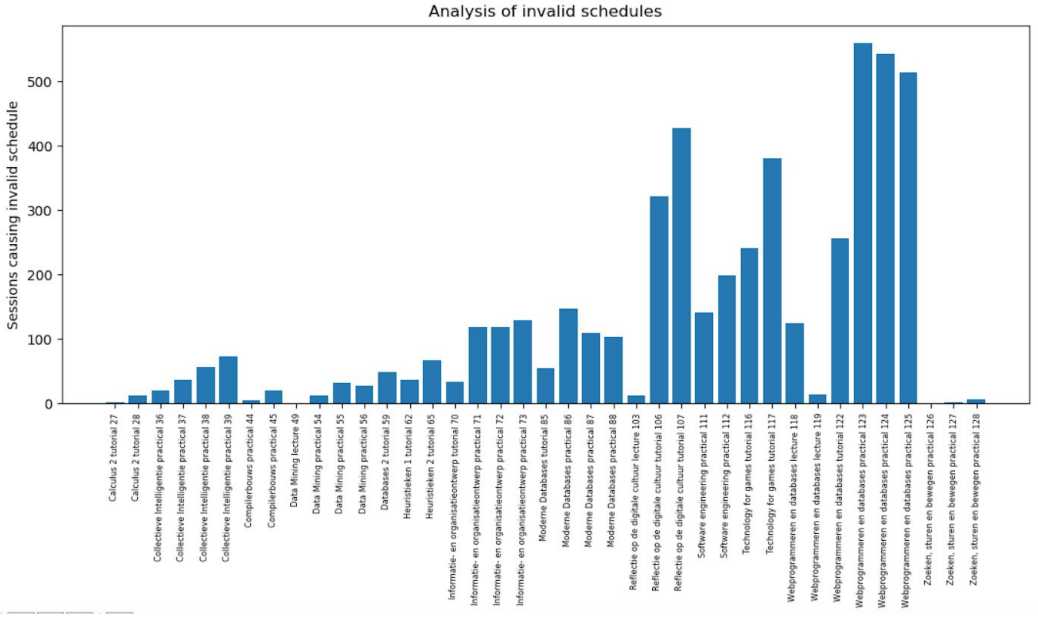
Om een analyse uit te voeren over de complexiteit is uitgezocht welke vakken zorgen voor problemen tijdens het indelen.

In eerste instantie wordt een random valide rooster gecreëerd. Na het analyseren van 5000 roosters bleek dat vooral een aantal sessies van de vakken Webprogrammeren, Technology for Games, Reflectie op de digitale cultuur, Moderne databases en Software engineering moeilijkheden veroorzaken in het indelen van een rooster (zie figuur 1). Daarnaast is te zien dat voor bepaalde sessies sommige problemen vaker voorkomen dan anderen (zie figuur 2). De problemen zijn opgedeeld in drie soorten:

- een hoorcollege had voor de huidige sessie gemoeten
- de huidige sessie heeft problematische overlap met het eigen vak
- de huidige sessie heeft problematische overlap met een vak dat ook gevolgd moet kunnen worden.

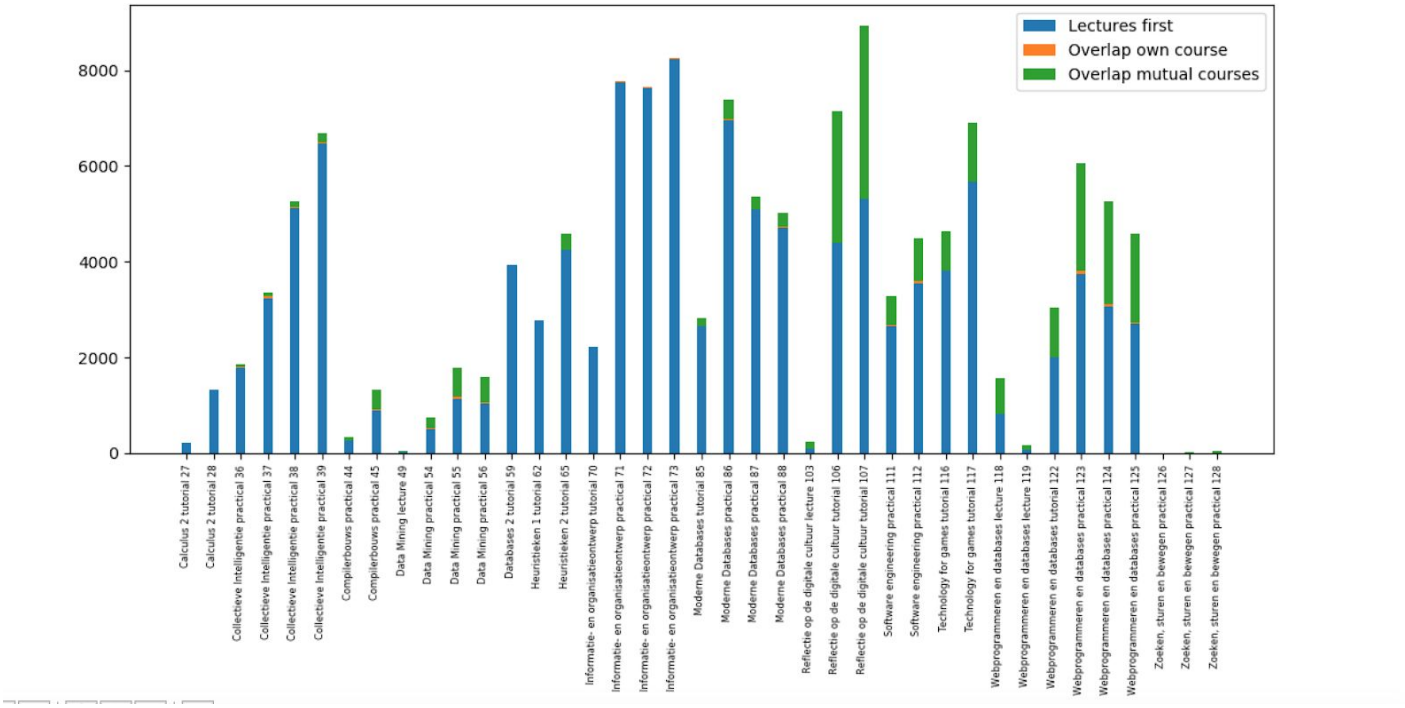
Een probleem wordt gezien als het moment dat de sessie, die op dat moment wordt ingedeeld, wordt beperkt in de mogelijke locaties waar de sessie zou worden kunnen geplaatst door een van de hierboven genoemde vereisten. Iedere keer wanneer een locatie van de locatielijst wordt verwijderd, veroorzaakt door een van de vereisten, wordt een punt opgeteld bij dat specifieke probleem.

Er is te zien dat voor bijna alle vakken de grootste beperking wordt veroorzaakt door de



Figuur 1. Staafdiagram van de frequentie dat een vak zorgde voor een stop op de indeling van een valide random rooster wanneer er 5000 keer een invalide rooster werd gecreëerd.

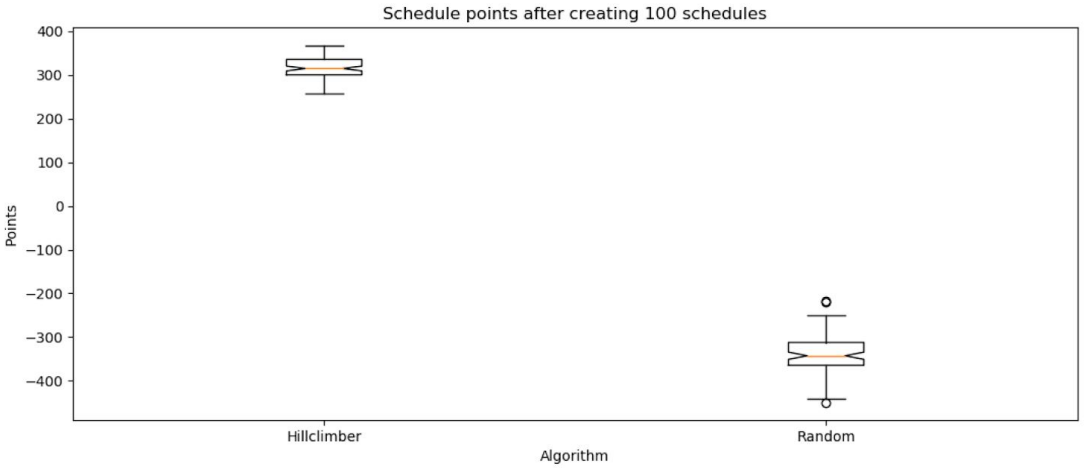
Analysis of invalid schedules



Figuur 2. Staafdiagram van de frequentie dat een bepaald probleem binnen een vak zorgde voor een stop op de indeling van een valide random rooster wanneer er 5000 keer een invalide rooster werd gecreëerd.

Relativering van resultaten

Om te bepalen welke score een ‘goede’ score is, is gekeken naar de gemiddelde scores van een hillclimber algoritme als het 100 keer heeft gerund. Er is bepaald dat een goed score de grens is van het derde kwartiel. Dit betekent dat dit voor ons 355 punten is. Wanneer zo’n plot voor alle algoritmes is gemaakt kan met meer zekerheid worden gezegd dat dit een goed rooster.



Figuur 3. Boxplots van roosterpunten na het analyseren van 100 roosters gecreëerd met een hillclimber of met alleen het random algoritme.