

dd

T

Odisee Technologiecampus Gent

Gebroeders de Smetstraat 1, 9000 Gent

|  |  |
| --- | --- |
| Optaplanner metaheuristieken examen roostering | |
| Louis D’Hont | |
| Opleiding: Elektronica – ICT | Academisch jaar 2021-2022 |

Inhoudsopgave

[Inhoudsopgave 1](#_Toc91801165)

[Codefragmentenlijst 2](#_Toc91801166)

[Figurenlijst 3](#_Toc91801167)

[Tabellenlijst 4](#_Toc91801168)

[1 Literatuurstudie 5](#_Toc91801169)

[1.1 Toronto dataset 5](#_Toc91801170)

[1.2 Multi-Phase Hybrid Metaheuristics 6](#_Toc91801171)

[1.2.1 Tabu Search 7](#_Toc91801172)

[1.2.2 Simulated Annealing 8](#_Toc91801173)

[2 Framework 9](#_Toc91801174)

[2.1 Waarom 9](#_Toc91801175)

[3 Uitwerking oplossing 10](#_Toc91801176)

[3.1 Klassen 10](#_Toc91801177)

[3.1.1 Examen 11](#_Toc91801178)

[3.1.2 Student 11](#_Toc91801179)

[3.1.3 Tijdslot 12](#_Toc91801180)

[3.1.4 Examen tabel solution 12](#_Toc91801181)

[3.2 Implementatie 13](#_Toc91801182)

[3.3 Beperkingen (constraints) 14](#_Toc91801183)

[3.3.1 Harde beperking 14](#_Toc91801184)

[3.3.2 Zachte beperking 15](#_Toc91801185)

[4 Resultaten 16](#_Toc91801186)

[5 Literatuurlijst 17](#_Toc91801187)

Codefragmentenlijst

[Codefragment 1: Examen klasse 11](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789877)

[Codefragment 2: Student klasse 11](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789878)

[Codefragment 3: Tijdslot klasse 12](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789879)

[Codefragment 4: Examen tabel oplossing 12](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789880)

[Codefragment 5: ExamTableApp klasse 13](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789881)

[Codefragment 6: Constraints met harde beperking 14](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789882)

[Codefragment 7: Zachte beperking 15](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789883)

Figurenlijst

[Figuur 1: Resultaten Multi-Phase Hybrid Metaheuristics 6](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789872)

[Figuur 2: Flowchart Tabu Search [10] 7](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789873)

[Figuur 3: Flowchart Simulated Annealing [11] 8](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789874)

[Figuur 4: Diagram klassen structuur 10](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789875)

[Figuur 5: OptaPlanner resultaten 16](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789876)

Tabellenlijst

[Tabel 1: Toronto datasets [4] 5](file:////Users/louisdhont/Downloads/Verslag-Metaheuristieken-takehome-2.docx#_Toc91789864)

# Literatuurstudie

Het plannen van examens of werkschema’s is een veelvoorkomend probleem dat bekend staat als een onderwerp met complexe beperkingen, beter bekend als NP-moeilijke problemen. Voornamelijk scholen hebben nood aan het oplossen van dit soort problemen om bijvoorbeeld examenroosters te maken zonder conflicten.

Een NP-moeilijk (niet-deterministisch poly-nominaal hard) probleem is een zeer complex probleem waarbij het niet mogelijk is om door middel van een volledige zoekactie op de zoekruimte een oplossing te vinden binnen een aanvaardbare tijd. In de meest zuiverste vorm is het roosteren van examens een allocatieprobleem dat ook wel eens een combinatorisch optimalisatieprobleem genoemd wordt. [1]

Het oplossen van een NP-moeilijk probleem binnen universiteiten kan verschillen van universiteit tot universiteit. In veel gevallen kunnen de beperkingen in twee soorten verdeeld worden, namelijk harde en zachte beperkingen. Het vinden van een gepaste oplossing is daardoor geen evidente opdracht. Door de jaren heen is er al heel wat onderzoek gebeurd naar NP-moeilijke problemen.

## Toronto dataset

In 1996 introduceerden Carter, M.W., Laporte, G. en Lee, S.Y. in een artikel genaamd “Examination Timetabling: Algorithmic Strategies and Applications” [2] een reeks van 13 datasets met data van 13 middelbare scholen en universiteiten. In het artikel werd onderzocht hoe ze het aantal benodigde tijdsloten en het spreiden van examens binnen de tijdsloten konden optimaliseren. [3]

Deze datasets worden doorgaans gebruikt als test dataset om de effectiviteit van nieuwe benaderingen op toe te passen en te evalueren. Er bestaan een groot aantal gepubliceerde papers over deze datasets, wat aantoont dat benaderingsmethoden erin slagen betere resultaten te produceren in vergelijking met andere methoden. De onderstaande tabel toont de probleem instanties die in de dataset aanwezig zijn. In deze studie werd gebruik gemaakt van de lse-91 dataset. [4]

Afbeelding met tafel

Automatisch gegenereerde beschrijving

Tabel 1: Toronto datasets [4]

## Multi-Phase Hybrid Metaheuristics

Het oplossen van een meta-heuristisch probleem kan op verschillende manieren worden opgelost. In een paper geschreven door A. Hmer en M. Mouhoub wordt er een aanpak beschreven waarmee het examenroosteringsprobleem kan worden opgelost. [5] Deze aanpak wordt beschreven in drie fasen:

* Voorbewerkingsfase
* Constructiefase
* Verbeteringsfase

Zij stellen een Multi-Phase Hybrid Metaheuristics aanpak voor waarin de 3 fasen van belang zijn om het probleem op een efficiënte manier op te lossen. In deze aanpak gebruiken ze 4 verschillende algoritmen om tot een oplossing te komen, namelijk:

* Tabu Search (TS)
* Hill Climbing (HC)
* Simulated Annealing (SA)
* Great Deluge Algoritme

Als eerste fase wordt er in de voorbewerkingsfase examens gesorteerd volgens de meest beperkte variabelen, door middel van het dynamische pad consistentie (DPC)-algoritme worden de eerste heuristische problemen en beperkingen ontdekt. [5]

Tijdens de constructiefase wordt een eerste oplossing gevonden met behulp van Tabu Search samen met een lijst van conflicten om het zoeken te vereenvoudigen en ook minder lang te laten duren. In de verbeteringsfase wordt een gekozen metaheuristiek gebruikt. Zodra een oplossing niet langer kan worden verbeterd of een dal wordt bereikt, treedt een ander algoritme in werking om zo tot de meest optimale oplossing te komen.

In Figuur 2 worden de resultaten van de 4 algoritmen (MTH1, MTH2, MTH3 en MTH4) weergegeven en vergeleken.

Afbeelding met tafel

Automatisch gegenereerde beschrijving

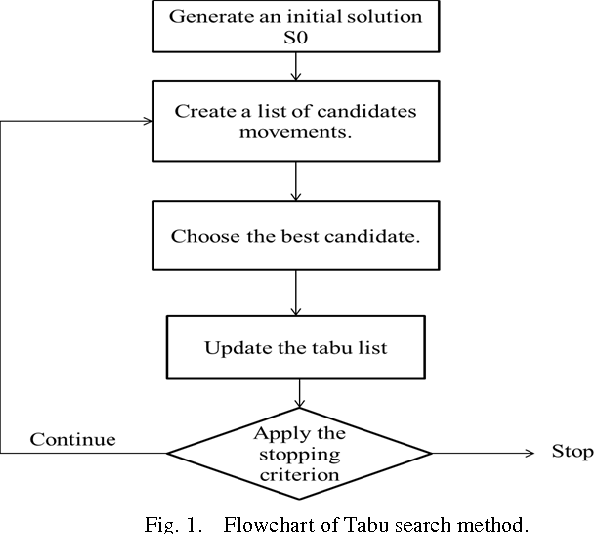
Figuur 1: Resultaten Multi-Phase Hybrid Metaheuristics

### Tabu Search

Tabu Search is een meta-heuristieke procedure gebaseerd op het werk van Fred Glover. Het proces is ontworpen om combinatorische optimalisatieproblemen op te lossen. Het algoritme heeft het vermogen om voorbij de grenzen van lokale optimaliteit te geraken, doordat het gebruik maakt van een taboelijst met recente zetten. Het woord Tabu (taboe) betekent in dit algoritme dat de recente zetten opgeslagen worden in een vaste lijst, zodat deze beschouwd worden als oude oplossing. [6]

Het voordeel van Tabu Search is dat het relatief weinig code vergt om het probleem te beschrijven en uit te werken, zo is het ook eenvoudig om goed te schalen op grotere problemen en bekomt het vaak een oplossing van hoge kwaliteit na een relatief korte zoektijd. Eén van de nadelen van Tabu Search is dat het probleem programmatisch voorstellen moeilijk kan zijn naar mate er meerdere beperkingen en vereisten zijn.

Om een probleem op te lossen start Tabu Search steeds met willekeurig een oplossing te kiezen en deze te valideren. De validatie stap berekent dan een score op de oplossing om zo de kwaliteit van de oplossing uit te drukken. De beste oplossing wordt steeds bewaard tot er een betere oplossing gevonden wordt. De vorige oplossing wordt daarna toegevoegd aan de taboelijst zodat deze niet meer kan worden uitgevoerd, dit helpt om uit de lokale minima te geraken. Figuur 1 toont een stroomdiagram van de stappen die Tabu Search definiëren.



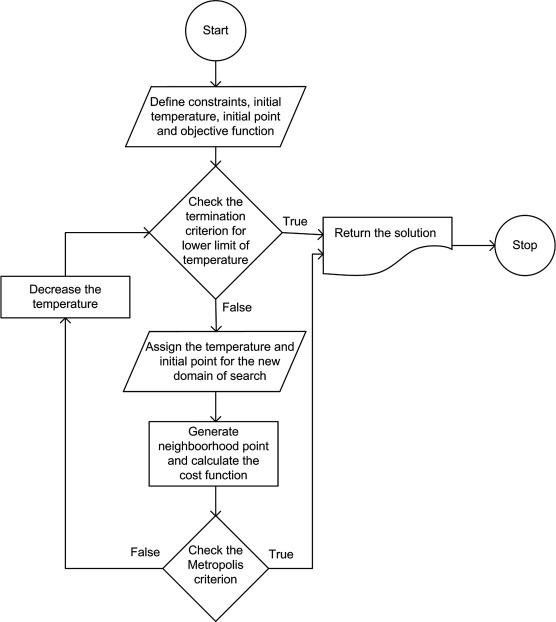
Figuur 2: Flowchart Tabu Search [11]

### Simulated Annealing

Simulated Annealing is een heuristische methode voor het oplossen van optimalisatieproblemen met beperkingen en grenzen. De methode modelleert het fysieke proces van het verwarmen van een materiaal en het vervolgens langzaam verlagen van de temperatuur om defecten te verminderen, waardoor de systeemenergie wordt geminimaliseerd.

Bij elke iteratie van het gesimuleerde algoritme wordt willekeurig een nieuw punt gegenereerd. De afstand van het nieuwe punt tot het huidige punt, of de omvang van de zoekopdracht is gebaseerd op een kansverdeling met een schaal die evenredig is aan de temperatuur. Het algoritme accepteert alle nieuwe punten die de doelstelling verlagen, maar ook, met een zekere waarschijnlijkheid, punten die de doelstelling verhogen.

Door punten te accepteren die het doel verhogen vermijdt het algoritme vast te zitten in lokale minima en kan het wereldwijd zoeken naar meer mogelijke oplossingen. Er wordt een uitgloeischema geselecteerd om de temperatuur systematisch te verlagen naarmate het algoritme vordert. Naarmate de temperatuur daalt vermindert het algoritme de omvang tussen beide punten naar een minimumwaarde. In onderstaande figuur wordt het Simulated Annealing proces in een stroomdiagram weergegeven. [7]



Figuur 3: Flowchart Simulated Annealing [12]

High quality

solution

• Low run time

• It can work on

large scheduling

problems

High quality

solution

• Low run time

• It can work on

large scheduling

problems

High quality

solution

• Low run time

• It can work on

large scheduling

problems

# Framework

OptaPlanner is een open-source planningsengine (constraint solver) ontwikkeld door Geoffrey De Smet in 2006 met als naam Taseree. Enkele jaren later heeft Red Hat OptaPlanner overgenomen om verder professioneel uit te kunnen werken. Het is ontwikkeld in Java en is ook in Python, Kotlin en Scala te gebruiken. Intern maakt OptaPlanner gebruik van geavanceerde algoritmen met kunstmatige intelligentie (zoals Tabu Search, Simulated Annealing, Late Acceptance en andere metaheuristieken) met zeer efficiënte scoreberekening voor het oplossen van beperkingen voor NP-complete of NP-moeilijke problemen. Het oplossen van een NP-moeilijk probleem is vaak moeilijker dan verwacht waardoor een Brute-force benadering vaak enorm lang kan duren zonder goede resultaten te leveren. [8], [9]

Het stelt ontwikkelaars in staat om optimalisatieproblemen efficiënt op te lossen. Het is object georiënteerd en functioneel vriendelijk voor ontwikkelaars. In OptaPlanner wordt de data en het probleem voorgesteld in klassen en structuren. Op die manier kan OptaPlanner met de geavanceerde algoritmen een oplossing zoeken. Het uitdrukken van beperkingen wordt ook op een zeer abstracte manier toegepast wat het voor de ontwikkelaar een betere leesbaarheid biedt. [10]

Door de veelzijdigheid van OptaPlanner kunnen er heel wat verschillende problemen worden opgelost, zoals:

* Voertuigroutering optimaliseren
* Personeelsroosters opmaken
* Taaktoewijzing optimalisaties
* Onderhoudsplanningen opmaken
* Examenplanningen opmaken

Elk probleem heeft een specifiek doel dat zo optimaal mogelijk moet worden benaderd en verwacht een oplossing volgens beperkte middelen onder specifieke beperkingen. Het uitwerken van een oplossing wordt vaak gedaan met een bepaalde reden, zoals:

* Maximale winst - het optimale doel resulteert in de hoogst mogelijke winst.
* Minimale ecologische voetafdruk - het optimale doel heeft de minste impact op het milieu.
* Maximale tevredenheid voor werknemers of klanten - het optimale doel geeft prioriteit aan de behoeften van werknemers of klanten. [10]

## Waarom

Om problemen zoals het opstellen van een examenrooster te implementeren is een metaheuristieken framework van belang waarbij kwaliteit en betrouwbaar belangrijk is. Persoonlijk vind ik OptaPlanner een interessant framework om een dergelijk probleem in op te lossen, omdat het op een eenvoudige manier toelaat om met klassenstructuren de data te beschrijven.

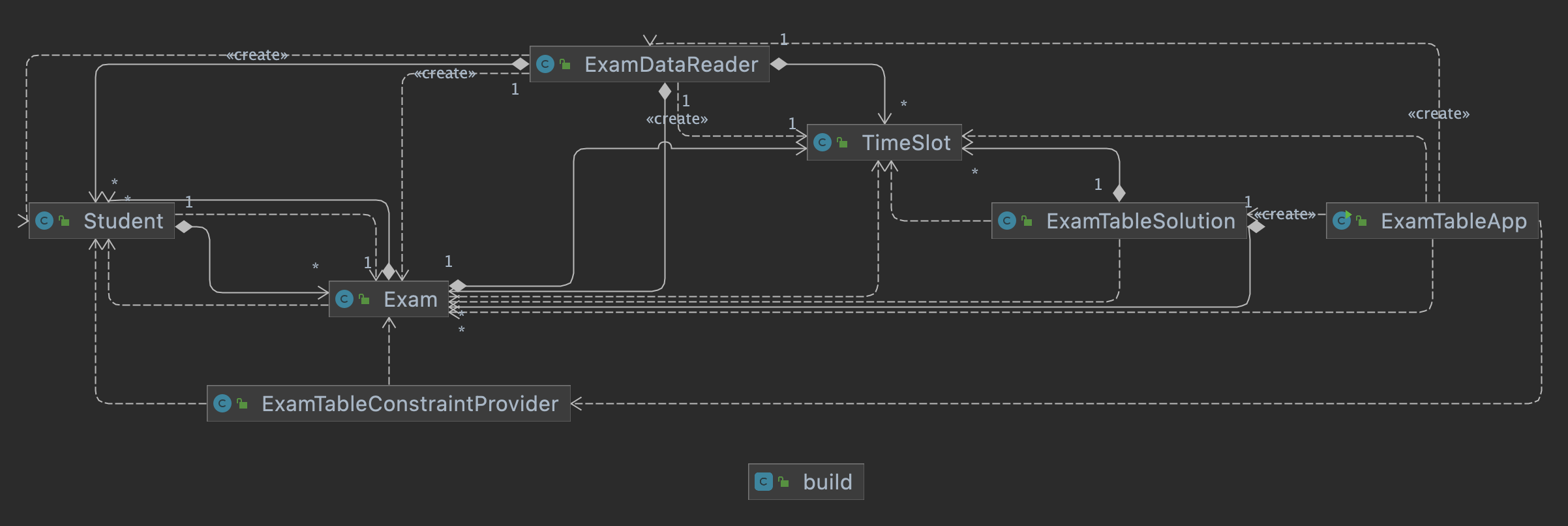
Het opstellen van een oplossingsklasse met beperkingen is ook op een relatief eenvoudige manier te beschrijven. Het gebruiken van een framework dat wereldwijd gebruikt wordt, is ook een zekere garantie dat het algoritme correct werkt volgens de implementatie van de beperkingen en datastructuren.

# Uitwerking oplossing

## Klassen

Om een probleem te beschrijven kan het data object georiënteerd, ingelezen en verwerkt worden door OptaPlanner. Figuur 2 geeft een structuur weer met de onderlinge connecties tussen de verschillende klassen. De OptaPlanner applicatie start bij de ‘ExamTableApp’ klasse waarin de oplossing aangemaakt wordt en de data van de ‘ExamDataReader’ binnenkomt, om vervolgens aan de solver mee te geven. De ‘ExamTableApp’ klasse laadt ook de beperkingen (constraints) uit de ‘ExamTableConstraintProvider’ klasse in.

De ‘ExamDataReader’ klasse haalt uit 3 bestanden (lse-f-91.crs en lse-f-91.stu) de examens, studenten en lijst van tijdsloten op, vervolgens worden er lijsten van objecten aangemaakt om de data bruikbaar te maken.

Nadat OptaPlanner met een oplossing komt kan de gekozen oplossing visueel worden weergegeven door in de ‘ExamTableApp’ klasse het resultaat van de ‘ExamTableSolution’ te printen.

Figuur 4: Diagram klassen structuur

### Examen

Een examen (Exam) klasse bevat een ‘id’ dat voor ieder examen uniek is. Het ‘id’ krijgt een ‘PlanningId’ annotatie dat ieder examen identificeert. Het in te vullen tijdslot wordt bijgehouden in een ‘TimeSlot’ klasse en krijgt als annotatie de ‘PlanningVariabele’ van OptaPlanner om aan te geven dat de tijdsloten moet worden verspreid over de tijdsloten van de examens. Verder is er nog een studentenlijst voorzien dat bijhoudt welke studenten het examen moeten bijwonen. De examenklasse in Codefragment 1 wordt als ‘PlanningEntity’ geannoteerd om duidelijk te maken dat deze klasse moet worden verdeeld over de tijdsloten.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Codefragment 1: Examenklasse

### Student

Een studenten (Student) klasse bevat een ‘id’ en een examenlijst om elke student bij te houden. In deze uitwerking wordt de examenlijst niet gebruikt maar kan wel eventueel van pas komen bij uitbreidingen. Om de lijsten op te slaan wordt er gebruik gemaakt van LinkedLists, omdat deze sneller en beter zijn om data te manipuleren, omdat er geen bitshifting nodig is in het geheugen.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Codefragment 2: Student klasse

### Tijdslot

Afbeelding met tekst

Automatisch gegenereerde beschrijvingDe tijdslot klasse houdt het ‘id’ bij van elk tijdslot met bijhorende getter en setter methodes. De (toString) methode wordt gebruikt om tijdens het uitprinten het tijdslot weer te geven.

Codefragment 3: Tijdslot klasse

### Examen tabel solution

OptaPlanner heeft een klasse die geannoteerd is met de ‘PlanningSolution’ tag. Deze klasse bevat de tijdslotenlijst en examenlijst. De tijdslotlijst bevat een ‘ValueRangeProvider’ met een ‘ProblemFactCollectionProperty’ om aan te geven dat deze lijst in verbinding staat met de variabele ‘timeSlot’ in de examenklasse. Met deze gegevens gaat OptaPlanner de lijst van examens verdelen over de tijdslotenlijst. Om de uiteindelijke score te zien kan er aan de solution klasse een score opgevraagd worden dat het aantal harde en zachte overtredingen weergeeft.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Codefragment 4: Examen tabel oplossing

## Implementatie

In de main methode wordt de OptaPlanner oplossing aangemaakt en de data ingelezen. De ‘SolverFactory’ maakt een solution aan met enkele configuratie instellingen. De ‘ExamTableSolution’ wordt hierin aangeduid als solution klasse, de entiteit klasse (Exam) wordt aangeduid en de ‘ExamTableConstraintProvider’ met de beperkingen waarmee OptaPlanner rekening dient te houden, worden gekoppeld.

De ingelezen data dat in de ‘ExamTableSolution’ zit, wordt doorgegeven aan de OptaPlanner ‘SolverFactory’ waarna de solver aan het werk gaat. Na een verloop van 5 seconden kan de beste oplossing die OptaPlanner gevonden heeft, worden uitgeprint door de tijdsloten en examens op te vragen aan de solution variabele.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Codefragment 5: ExamTableApp klasse

## Beperkingen (constraints)

Zonder beperkingen worden alle examens op hetzelfde tijdslot geplaatst, wat geen goede oplossing vormt. Door harde en zachte beperkingen in te bouwen zal het algoritme een bepaalde kost vastleggen aan overtredingen tijdens het uitzoeken van een oplossing, zodat deze afgestraft worden, wat leidt tot een betere oplossing.

Hoewel deze beperkingen variëren voor verschillende universiteiten, zijn de volgende harde beperkingen de meest voorkomende:

* Een student kan slechts 1 examen per examensessie afleggen;
* Het aantal studenten dat een examen afleggen moet kleiner zijn dan de maximumcapaciteit van het examenlokaal;
* Elk examen moet toegewezen worden aan exact 1 tijdslot.

Zachte beperkingen kunnen ook sterk variëren afhankelijk van de noden die gevraagd worden. Zo kan er gevraagd worden om bijvoorbeeld voor docent A zo weinig mogelijk lessen op maandag te plannen of rekening te houden met verlofdagen en dergelijke. In deze opgave werden er 2 beperkingen voorgelegd aan het probleem, één harde en één zachte beperking.

### Harde beperking

De harde beperking stelt dat een student maximaal één examen per tijdslot kan afleggen. Wanneer deze beperking verbroken wordt, krijgt OptaPlanner voor deze actie een harde score die deze actie afstraft in het algoritme.

De ‘studentExamTimeslotConflict’ functie controleert elke uniek examen en neemt daarvan elk tijdslot, vervolgens worden er twee examens genomen per keer en vergeleken. Hierna wordt er gecontroleerd of de student van examen1 voorkomt in de lijst van examen2. Wanneer dit voorkomt moet er gezorgd worden dat deze studenten niet in hetzelfde tijdslot zitten. De variabele ‘found’ wordt dan op true gezet, wat zal resulteren in een beperking waarna het algoritme opnieuw een poging moet doen om de examens anders te ordenen.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Codefragment 6: Constraints met harde beperking

### Zachte beperking

De zachte beperking stelt voor om zo veel mogelijk ruimte (tijdsloten) te laten tussen de examens van een student. De scores per tijdslot zijn als volgt:

* Een score van 16 wanneer een student twee aaneensluitende examens heeft;
* Een score van acht wanneer er één tijdslot zit tussen twee aaneensluitende examens;
* Een score van vier wanneer twee tijdsloten zitten tussen twee aaneensluitende examens;
* Een score van twee wanneer drie tijdsloten zitten tussen twee aaneensluitende examens;
* Een score van één wanneer vier tijdsloten zitten tussen twee aaneensluitende examens.

De ‘timeSlotsBetweenExamsConflict’ functie controleert het aantal tijdsloten dat tussen twee opeenvolgende examens ligt. Om een soft score te kunnen teruggeven aan de constraint methode kan in de penalize functie eenderde parameter worden meegeven, in dit geval een getal. Er worden twee examens afgenomen per keer die met elkaar worden vergeleken, zoals de harde beperking dat doet. Wanneer beide examens een ander tijdslot hebben kan de functie controleren of beide examens dezelfde student bevatten, hierna wordt het absolute verschil tussen beide tijdsloten berekend en vervolgens in een switch case overlopen om een gepaste score in te stellen. In onderstaand codefragment wordt de score weergegeven.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Codefragment 7: Zachte beperking

# Resultaten

Na het uitvoeren van OptaPlanner wordt de lijst met tijdsloten en examens uitgeprint. Na het uitprinten wordt de score uitgeprint. De score geeft aan hoeveel harde en zachte beperkingen er gebroken zijn. In dit geval zijn er 15 harde beperkingen gebroken wat wil zeggen dat er 15 keer een overlapping heeft plaatsgevonden en deze niet heeft doorgevoerd.

Afbeelding met tekst

Automatisch gegenereerde beschrijvingEen (negatieve) zachte beperking mag niet verbroken worden, als deze kan worden vermeden. Wanneer een zachte beperking verbroken wordt, is het volgens OptaPlanner niet mogelijk om met de beperking rekening te houden. In dit geval zijn er 5553 zachte beperkingen niet doorgevoerd. De resultaten kunnen terug worden gevonden in het bestand met de naam optaplanner-results.txt.

Figuur 5: OptaPlanner resultaten

# Literatuurlijst

|  |  |
| --- | --- |
| [1] | H. behiels, „Het opstellen van examenroosters in universiteiten:een geautomatiseerde aanpak,” 2009. [Online]. Available: https://adoc.pub/het-opstellen-van-examenroosters-in-universiteiten-een-geaut.html. [Geopend 28 December 2021]. |
| [2] | S. R. Publishing, „University Timetable Generator Using Tabu Search,” Scientific Research Publishing, 1996. [Online]. Available: https://www.scirp.org/(S(351jmbntvnsjt1aadkposzje))/reference/ReferencesPapers.aspx?ReferenceID=1938977. [Geopend 28 December 2021]. |
| [3] | „Benchmark Data Sets in Exam Timetabling,” Univercity of Nottingham, 1 Juni 2021. [Online]. Available: http://www.cs.nott.ac.uk/%7Epszrq/data.htm. [Geopend 28 December 2021]. |
| [4] | P. Alefragis, „A multiple metaheuristic variable neighborhood search framework for the Uncapacitated Examination Timetabling Problem,” 2021. [Online]. Available: http://patatconference.org/patat2020/proceedings/papers/12.%20PATAT\_2020\_paper\_42.pdf. [Geopend 29 December 2021]. |
| [5] | A. Hmer en M. Mouhoub, „A Multi-Phase Hybrid Metaheuristics Approach for the Exam Timetabling,” 29 November 2016. [Online]. Available: http://www2.cs.uregina.ca/~mouhoubm/res/=postscript/=papers/aliIJCIA2016PublishedVersion.pdf. [Geopend 28 December 2021]. |
| [6] | R. Gashgari, „A Survey on Exam Scheduling Techniques,” April 2018. [Online]. Available: https://www.researchgate.net/publication/327191072\_A\_Survey\_on\_Exam\_Scheduling\_Techniques. [Geopend 28 December 2021]. |
| [7] | „What Is Simulated Annealing?,” 2021. [Online]. Available: https://nl.mathworks.com/help/gads/what-is-simulated-annealing.html. [Geopend 28 December 2021]. |
| [8] | G. D. Smet, „Optaplanner,” [Online]. Available: https://www.optaplanner.org. [Geopend 28 December 2021]. |
| [9] | „OptaPlanner,” 25 Mei 2021. [Online]. Available: https://en.wikipedia.org/wiki/OptaPlanner. [Geopend 28 December 2021]. |
| [10] | „Optaplanner User Guide,” 27 Juli 2021. [Online]. Available: https://docs.optaplanner.org/8.9.0.Final/optaplanner-docs/html\_single/. |
| [11] | F. O. Z. Kaddouri, „Application of the Tabu Search Algorithm to Cryptography,” 2017. [Online]. Available: https://www.semanticscholar.org/paper/Application-of-the-Tabu-Search-Algorithm-to-Kaddouri-Omary/de741741dcf174cb864fbe47a0d6c2a084eb5bfd. [Geopend 28 December 2021]. |
| [12] | „Simulated Annealing Algorithm,” 2019. [Online]. Available: https://www.sciencedirect.com/topics/engineering/simulated-annealing-algorithm. [Geopend 28 December 2021]. |