# Verslag Grote Opdracht

Jesper Kuiper (6203299)
Leander van Boven (6215637)
*'Het Wozcek duo'*

22 december 2019

## 1 Introductie

Voor deze opdracht is het de bedoeling om een schema voor het ophalen van afval bij verschillende bedrijven te optimaliseren. Dit is een ingewikkelde opdracht aangezien sommige bedrijven slechts één keer per week een vuilniswagen verwachten terwijl andere bedrijven soms wel 4 of 5 keer per week een vuilniswagen verwachten. Daar komt nog eens bij dat een vuiliniswagen niet meer dan 12 uur per dag onderweg mag zijn en de laadcapaciteit gelimiteerd is.
Vanwege de complexiteit van het probleem is de oplossingsruimte enorm groot. Dit betekent dat het heel veel tijd kost om van elke mogelijke planning de tijd te berekenen om op deze manier een planning te vinden die optimaal is. Om toch een redelijk goede planning te vinden in overzienbare tijd gaan we gebruik maken van lokaal zoeken. Een lokaal zoekalgoritme kan namelijk redelijk snel convergeren naar een lokaal optimum zodat veel sneller een aantal 'goede' oplossingen gevonden kunnen worden. Het enigse nadeel aan het gebruik van lokaal zoeken voor dit probleem is dat we nooit kunnen controleren of het gevonden lokale optimum het globale optimum is. Echter kunnen we een doelscore stellen die we willen bereiken.

### 1.1 Definities

In dit verslag en maken we gebruik van een aantal definities:

- `Tour`. Elk schema bevat voor elke dag voor elke vuilniswagen een route. In deze route kan de vuilniswagen meerdere keren langs de stort komen om tussentijds te legen. Elke route van stort naar stort op een dag noemen wij een `Tour`.
- `Buuroperator`. Een buuroperator is een manier voor het lokaal zoekalgoritme om een buurstaat te genereren.

Definities gebruikt in de code worden ter plekke toegelicht.

## 2 Buuroperators

Wij maken op dit moment gebruik van vier buuroperators:

1. `RandomAdd`. Van alle orders die nog niet zijn ingepland, voeg een willekeurige order toe aan een willekeurige tour van een willekeurige truck op een willekeurige dag, op zo'n manier dat geen constraints worden overschreden.

2. `RandomDelete`. Voor een willekeurige tour van een willekeurige truck op een willekeurige dag, verwijder een willekeurige order. Hierbij moet rekening worden gehouden dat de truck nu van de vorige order in één keer naar de order erna gaat rijden. Het kan zo voorkomen dat dit langer duurt dan ertussen stoppen bij de te verwijderen order, dus moet er worden gecontroleerd of hiermee niet de totale diensttijd wordt overschreden.

3. `RandomTransfer`. Voor een willekeurige tour van een willekeurige truck op een willekeurige dag, verwijder een willekeurige order. Voeg deze order nu willekeurig *ergens anders* in. Hierbij houden we natuurlijk rekening met eerder genoemde constraints.

4. `RandomSwap`. Vind twee (van elkaar verschillende) willekeurige orders, en verwissel deze. Hierbij moet natuurlijk weer op de constraints worden gelet.

Voor deze operators gelden, afgezien van de "normale"constraints, ook wat extra kanttekeningen. Ten eerste, orders die een frequentie hebben van 2 of hoger. Er zijn twee manieren om hiermee om te gaan, ofwel je ziet zo'n order als één geheel, in die zin dat de order meteen voor elke dag wordt ingepland en verwijderd, maar ook getransfert en geswapt. De andere mogelijkheid is dat je elke dag waarop je de order zou kunnen inplannen ziet als eigen identiteit. Dit betekent dat je soms werkt met een toestand die niet volledig voldoet aan de constraints. Wij hebben gekozen voor de eerste manier, en dit hebben we geïmplementeerd voor de `RandomAdd`. Echter hebben de rest van de operators nog geen ondersteuning voor orders met een hogere frequentie dan 1, oftewel, op dit moment, als een hogerfrequente order wordt ingepland, dan staat hij vast. Dit is uiteraard niet optimaal.
Het tweede punt gaat over het toe/invoegen en verwijderen van storten. Op het moment dat je hiervoor ondersteuning wilt bieden, moet voldaan worden aan extra constraints. Dit hebben we nog niet geïmplementeerd, het aantal keer dat de stort wordt aangedaan is bij ons dus nog constant. Ook dit is verre van optimaal.

## 3 Oplossingsmethode

Ons programma zet per keer meerdere solvers aan. Deze solvers gaan vervolgens in parallel, onafhankelijk van elkaar, bezig met het zoeken van een oplossing.

Tegelijk wordt om de zoveel tijd bijgehouden welke solver op dat moment de beste score heeft gevonden, die real-time wordt weergeven in de console. Op het moment dat alle solvers zijn getermineerd (of de gebruiker het proces interrumpeert door middel van het indrukken van `Esc` of `X`), wordt de beste oplossing aangenomen. Deze oplossing wordt ook meteen in de root directory opgeslagen als `result.txt` die rechtsreeks in de checker kan worden gekopieerd.
In elke iteratie van één zo'n solverinstantie genereert de solver een verzameling van operaties, gekozen op basis van onze vier operators in een kansdistributie. Deze verzamling kan groter zijn dan het aantal operators die we hebben, dus dan wordt minstens één operator meerdere keren geëvalueerd. Evalueren betekent hier: kiezen op welke order(s) de operatie wordt uigevoerd, en kijken hoe dit de score zal beïnvloeden. Aangezien alles willekeurig wordt gekozen, kan inderdaad één operator verschillende keren wordt geëvalueerd met andere uitkomst. Uit deze verzameling van geëvalueerde operaties wordt dan al dan niet een operatie gekozen die daadwerkelijk wordt uitgevoerd, en deze keuze wordt gedaan op basis van onze zoekmethode.
De evaluatiefunctie, die de zoekmethoden gebruiken om een operatie te kiezen, is op dit moment de som van de verandering in penalty met de verandering in reistijd. Dit is dus gelijk met het verschil in score. Als die negatief is, dan weten we dat deze operatie een goede kandidaat zal zijn om aan te nemen. Wij maken op dit moment gebruik van een combinatie van *Simulated Annealing* met *Random Hillclimb* en ook *Random Walk* (een willekeurige buurtoestand aannemen zelfs als deze slechter is). Wij beginnen met een begintoestand waar beide trucks elke dag twee lege tours maken (van de stort meteen weer naar de stort). Aangezien onze operators op dit moment nog geen storten kan toevoegen, invoegen of verwijderen moeten we deze nu nog handmatig invoegen. We zijn tot de conclusie gekomen dat twee tours per dag per route min of meer de juiste hoeveelheid is. Eén solverinstantie doorloopt de volgende twee fasen:

1. `Doing_Add`-fase. Eerst gaan we het schema zo veel mogelijk vullen, hier gebruiken we daarom alleen maar de `Add`- en `Delete`-operators. Voor deze fase maken we gebruik van zowel *Simulated Annealing* en *Random Walk*, waartussen we elke 15000 iteraties wisselen. *Simulated Annealing* heeft hier een begintemperatuur van 0.7 en een koelschema van 0.9999. Aangezien het hier vooral gaat om het vullen van ons schema en nog wat minder om het vinden van een *goed* schema, gebruiken we hier de *Random Walk* om te zorgen voor meer exploratie. Op dit moment doen we dit 30000 iteraties lang, of totdat er 3000 iteraties geen operator is uitgevoerd. Hiermee komen we altijd uit rond een score van 7100-7300. We hebben nu meestal meer dan 1000 orders ingepland, waardoor onze penalty erg laag is.

2. `Doing_All`-fase. Vanaf nu gaan we veel minder toevoegen, en meer verwijderen en omwisselen. We maken nu vooral gebruik van de `Swap`- en `Transfer`-operators en in mindere mate de `Delete`-operator en (in nog mindere mate) de `Add`-operator. Hier stappen we volledig over op *Simulated Annealing*, met een begintemperatuur van 0.9 en een koelschema van

0.9999. We beginnen dus met een hoge mate van willekeurigheid, en die willekeurigheid gaat ook redelijk langzaam naar beneden. Echter zijn we tot de conclusie gekomen dat dit probleem, met de operators zoals we die op dit moment hebben geïmplementeerd, daar baat bij heeft. Dit houden we vol voor 500000 iteraties, of totdat er voor 75000 iteraties geen verbetering plaatsvindt. In realiteit is dit meestal de reden dat het algoritme termineert.

Daarbij doen we elke zoveel iteraties een aantal `2.5-opt` operaties. Dit zorgt ervoor dat elke tour min of meer optimaal is, in de zin dat er geen andere volgorde van *dezelfde* orders binnen een tour is zodat de reistijd korter is dan de huidige reistijd. Doordat we dit doen hebben we geen `RandomSwap`-equivalent nodig voor swaps binnen een tour.

# 4 Resultaten

Zoals voor een deel hierboven als is vermeld, gebruiken we nu de volgende parameters:
Aantal threads/solverinstanties: 10
Aantal operaties per iteratie: 20
Maximaal aantal iteraties voor `Doing_Add`-fase: 30000
Maximaal iteraties zonder positieve verandering in `Doing_Add`-fase: 3000
Kansdistributie operaties `Doing_Add`-fase:

1. `RandomAdd`: $\frac{7}{8}$
2. `RandomDelete`: $\frac{1}{8}$
3. `RandomTransfer`: 0
4. `RandomSwap`: 0

Maximaal aantal iteraties voor `Doing_All`-fase: 500000
Maximaal iteraties zonder positieve verandering in `Doing_All`-fase: 75000
Kansdistributie operaties `Doing_All`-fase:

1. `RandomAdd`: $\frac{1}{12}$
2. `RandomDelete`: $\frac{1}{12}$
3. `RandomTransfer`: $\frac{5}{12}$
4. `RandomSwap`: $\frac{5}{12}$

Hiermee hebben we een beste score van 6180.93 gehaald.

# 5 Evaluatie

Zoals we de solver nu hebben geïmplementeerd, komen we uit op een score die net boven de score valt waaronder je minstens moet komen zodat er niets is fout gegaan. Dit komt voor ons niet als een verassing; er zijn gewoon een aantal dingen die volgens ons een vereiste zijn om tot een goed resultaat te komen, maar die we nog niet hebben geïmplenteerd. De grootste dingen zijn:

- Het voor alle buuroperators mogelijk maken om om te gaan met orders die een frequentie van hoger dan 1 hebben.
- Het voor alle buuroperators mogelijk maken om om te gaan met storten. Hierbij zullen we ook een andere vorm voor de evaluatiefunctie moeten aannemen, aangezien verandering in score niks zegt over de mate van wenselijkheid van het toe/invoegen of verwijderen van een stort. Hier zullen we dus bijvoorbeeld moeten kijken naar hoeveel laadruimte er overblijft en/of vrijkomt.

Wij denken dat we, zodra we dit hebben geïmplenteerd, in ieder geval onder de 6000 zouden moeten kunnen duiken. Helaas zijn we daar simpelweg niet aan toe gekomen – niet vanwege gebrek aan inzet.
Daarbij hebben we nog wat andere ideeën en observaties, die naar ons idee onze score nog verder naar beneden zouden kunnen helpen.

- We zien in de visualizer van de checker dat de routes die onze solver geeft, nogal 'all over the place zijn' – letterlijk. De grotere routes hebben nog al eens de neiging om het hele gebied af te gaan en in elke plaats een klein aantal orders mee te pakken. Dit is natuurlijk niet optimaal wat betreft reistijd, het is beter om een route te hebben die naar één gebied gaat met een relatief grote dichtheid van orders. Om dit voor elkaar te krijgen, willen we de orders clusteren door middel van k-means en een heuristiek introduceren die gebruik maakt van die clusters.
- Op dit moment zijn de operaties volledig willekeurig in welke order ze selecteren en wat ze ermee doen. Wij denken dat we deze operaties misschien een klein beetje kunnen sturen (niet te veel natuurlijk), eventueel met de clusterheuristiek als hierboven beschreven.
- We zien op dit moment dat ons algoritme termineert doordat er een hele tijd geen positieve verandering meer plaatsvindt. Dit komt waarschijnlijk doordat de temperatuur van *Simulated Annealing* te laag is en we dus vastzitten in een lokaal optimum. We willen dus een manier vinden om daaruit te kunnen ontsnappen. Één manier zou zijn om om de zoveel tijd het *SA*-traject opnieuw te initiëren, en dus weer te beginnen met een hoge mate van willekeurigheid. Een tweede manier is het introduceren van *Tabu Search*. Deze zoekmethode is erg effectief in het ontsnappen van lokale optima. Echter is natuurlijk de grootste moeilijkheid van *Tabu Search* het representeren van een toestand op een manier die vergelijking tussen toestanden mogelijk maakt.

- Wij laten op dit moment meerdere solverinstanties in parallel naar een opossing zoeken. Op dit moment doen ze dit volledig onafhankelijk van elkaar. Dit betekent dat onze methode op dit moment equivalent is aan één solverinstantie meerdere keren achter elkaar aanzetten, en van deze instanties het beste resultaat te pakken. Echter willen wij uiteindelijk implementeren dat de solverinstanties met elkaar kunnen communiceren. Zo zouden we tijdens het itereren een top $n$ resultaten bij kunnen houden, en elke zoveel iteraties elke solverinstantie verder laten werken met één van die $n$ huidige beste resultaten. Hier is het ook mogelijk dat het aantal solverinstanties groter is dan $n$, want, aangezien de operators altijd een mate van willekeurigheid gaan houden, zal elke instantie iets anders met een resultaat doen, ookal beginnen ze bij hetzelfde startpunt. We zouden ook een *Divide and Conquer*-techniek kunnen toepassen, waar elke instantie een deelprobleem van het geheel optimaliseert (waar dus niet per se aan alle constraints hoeft worden voldaan), maar dat je door die deelresultaten te combineren een volledige valide oplossing kan genereren.

Het was van origine natuurlijk de bedoeling dat we de bovengenoemde punten al hadden geïmplementeerd voordat we dit verslag moesten inleveren. Dit is niet gelukt, simpelweg vanwege gebrek aan tijd. Wij hebben, deels omdat het echt nodig was en deels omdat we toch net niet tevreden waren, grote delen van onze code moeten herschrijven, en misschien hebben we aan het begin een aantal dingen geïmplementeerd die niet een heel hoge prioriteit hebben in vergelijking met vooral de bovenste twee punten. Echter hebben we een duidelijk beeld van wat er moet gebeuren, en verwachten we eind januari in ieder geval onder de 6000 te komen, en hopelijk natuurlijk een stuk lager.