Zoekprobleem autosharing: Verslag

Pieter Dilien, Lucas Van Laer

29 maart 2023

1 Oplossingsvoorstelling

De oplossing wordt voorgesteld door een object dat 3 attributen heeft. De eerste is een req_to_car array. Hierbij stelt elke index een reservatie voor en elke waarde een wagen. De tweede is een car_to_zone array. Hierbij stelt elke index een wagen voor en elke waarde een zone. De laatste is een int die de cost voorstelt.

Deze voorstelling is gebaseerd op de output-file dat gegenereerd moet worden. Daardoor is het wegschrijven naar deze file erg gemakkelijk.

2 Datavoorstelling

Om berekeningen te doen werken we met een object dat volgende attributen heeft:

- 1D Array of ints: reservation id as index, get corresponding car
- 1D Array of ints: car id as index, get corresponding zone
- 1D Array of ints: reservation id as index, get corresponding cost
- 2D Array of ints: car id as index, get list of corresponding reservations
- 2D Array of bools: row for every reservation, every row has length = #cars
- 2D Array of bools: row for every zone, every row has length = #cars
- 1D Array of RequestStructs: reservation id is the index
- 1D Array of ZoneStructs: zone id is the index
- int: cost

Dit in combinatie met twee "structs", een RequestStruct en ZoneStruct. De RequestStruct houdt alle informatie bij over 1 reservatie. Dit gaat dan over de zone, dag, starttijd, tijd, lijst van mogelijke wagens, penalty 1 en penalty 2 kost. De ZoneStruct houdt dan weer informatie bij over 1 zone. Hiermee wordt bedoeld: een lijst van aanliggende zones, een booleaanse lijst die de relatie geeft t.o.v. elke andere zone.

3 Initiële Oplossing

Om de initiële oplossing te berekenen gaan we sequentieel over de lijst van reservaties. Bij elke reservatie gaan we de verschillende wagens af. Eerst wordt er nagegaan of de wagen reeds aan een zone gelinkt is, indien dit nog niet het geval is, plaatsen we de wagen in de zone van de reservatie.

Indien de wagen al aan een zone gelinkt is checken we of deze in de (naburige) zone van de reservatie ligt. Indien dit het geval is wordt er gekeken of deze wagen dan ook beschikbaar is, zo ja linken we deze wagen aan de reservatie.

Als er over alle wagens (die mogelijk zijn voor de specifieke reservatie) gegaan is en er geen wagen beschikbaar is, wordt er naar de volgende reservatie gegaan en blijft de reservatie ontoegewezen.

4 Zoekomgeving

Onze zoekomgeving werd gedefinieerd door twee operatoren. Een kleine die eerder kleine aanpassingen maakt en een grote die grotere sprongen maakt.

4.1 Kleine Operator

De kleine operator itereert random over de verschillende reservaties en per reservatie random over de verschillende wagens. Er wordt telkens gecheckt of er geen (betere) wagen kan gevonden worden voor een reservatie. Met beter wordt hier voornamelijk bedoeld een wagen die in de eigen zone van een reservatie ligt en niet in een naburige. De resultaten van enkel deze operator zijn te zien als v1 op 1.

4.2 Grote Operator

Deze werd van begin af aan reeds gecombineerd met de kleine operator. Dit door eerst de grote operator uit te voeren en wanneer deze geen verbetering meer gaf de kleine verder uit te voeren.

Versie 1: verplaatst een random wagen naar een random zone en gaat vervolgens alle reservaties af om te kijken of er een wagen aan toegewezen kan worden. Resultaten te zien als v2 op 1.

Versie 2: iteert random over een lijst met wagens en per wagen random over een lijst met zones. Vervolgens wordt dan hetzelfde gedaan als bij Versie 1. Een groot verschil is dat hierbij de beste wordt gezocht. Alle combinaties worden dus geprobeerd en de beste wordt gekozen. De resultaten zijn te zien als v3 op 1. Het is duidelijk te zien op de grafiek dat deze manier slecht werkt op korte tijd, omdat de operator zo veel computing time nodig heeft.

Versie 3: i.p.v. zoals in Versie 2 de beste te zoeken, wordt dit keer de eerste beste genomen. We zien duidelijk op 1, dat v4 die hiermee correspondeerd een beter resultaat geeft.

Versie 4: hierbij werd geen aanpassing gedaan aan de operator zelf, maar aan wanneer deze wordt uitgevoerd in combinatie met de kleine operator. In het begin blijven we de grote operator uitvoeren totdat deze geen verbetering meer geeft. Hierna voeren we de kleine operator 5 keer uit, waarna we de grote nog een keer uitvoeren. Als de grote operator in totaal 2 keer gefaald heeft, slaan we de huidige oplossing op en beginnen we met een nieuwe initiële oplossing. Iets dat we ook pas sinds deze versie doen. We zien op 1 dat het resultaat (v5) voornamelijk beter is bij een langere runtime.

5 Metaheuristiek

In deze sectie gaan we het kort hebben over de verschillende metaheuristieken die wij geprobeerd hebben.

5.1 Simulated Annealing (v6)

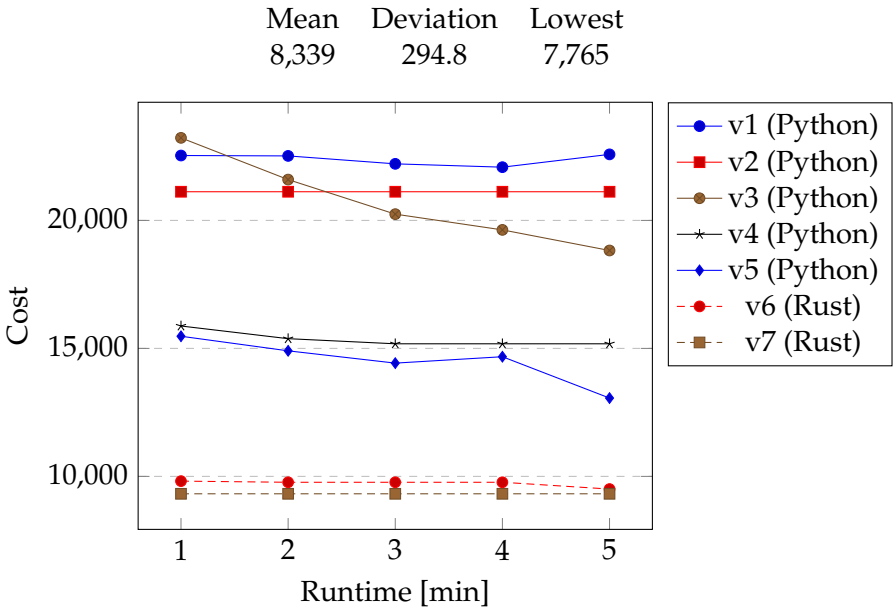
Hierbij beginnen we met een start temperatuur van 300 na 1000 iteraties wordt deze gedeeld door 1,3 totdat we een minimum temperatuur 5 of kleiner bereiken of we voor 500 iteraties niet verbeteren. Nadat de local search stopt herbeginnen we met een andere initiele oplossing. Deze versie bevat ook een simplistische threshold die niet van waarde verandert behalve als we vast komen te zitten, hetzelfde gebeurt bij de temperatuur. Deze verhoogt als we falen om een betere oplossing te vinden. De resultaten hiervan zijn te zien op grafiek 1 als v6.

5.2 Thresholding (v7)

In deze versie hebben we alleen maar voor thresholding gekozen, hierdoor ging de cost sneller naar beneden en hebben we onze beste cost gevonden. Deze versie houdt ook geen rekening met of het vast komt te lopen of niet. We zagen dat als we dit er wel bij staken op dezelfde manier als bij versie 6 dat dit eigenlijk niks deed. Het verschil met deze thresholding en met de simplistische thresholding die we in versie 6 deden is dat deze wel daalt. De threshold daalt met 2 na 2000 iteraties en start op 40. Als de threshold 20 of lager gaat proberen we opnieuw met een andere initiele oplossing.

6 Resultaten

Hieronder kan u de resultaten zien voor de finale versie (v7), dit is de versie met thresholding. De resultaten zijn voor 360_5_71_25.csv met telkens een runtime van 5 minuten en een random seed.



Figuur 1: Results for 360_5_71_25.csv with seed 123

7 Reflectie en suggesties

We konden ons programma nog verbeteren bij thresholding door wat te spelen met parameters om zo proberen te achterhalen of we al dan niet op een lokaal minimum zitten. Dit zou onze resultaten evenuteel nog verbeterd kunnen hebben.

Ten slotte is het ook nog mogelijk om multithreading te implementeren, ook al is dit niet vanzelfsprekend in Rust.