Project AI Cambio

Ruben Kindt, Jeroen Vastmans en Christophe Huybrechts

Probleem omschrijving

Het cambio probleem bestaat uit het optimaal plannen van verhuurauto's a.d.h.v. een lijst met verzoeken voor bepaalde automodellen, die beschikbaar moeten zijn in een bepaalde zone en tijdslot. De kostenfunctie wordt bepaald door de onderstaande som.

totK= $\sum_{v}^{aantalV} (p * p_1 + q * p_2)$

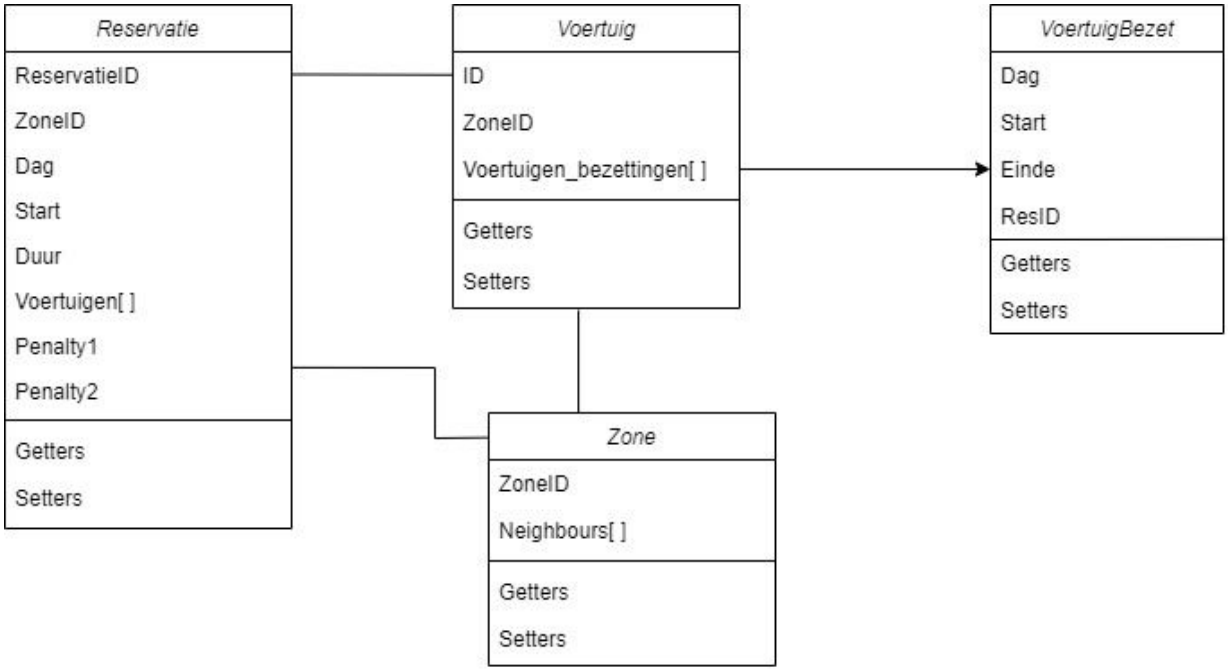
- *totK* is de totale kost;
- *aantalV* het totale aantal verzoeken;
- *P* is 1 als het verzoek niet vervuld is en is 0 in de andere gevallen;
- *Q* is 1 indien het verzoek een auto toewijst aan een naburige zone en Q is 0 in alle andere gevallen;
- *P1* is de kost bij het niet vervullen van een verzoek, zijnde 'penalty1'
- *P2* is de kost bij het vervullen van het verzoek door middel van het plaatsen van een auto in een naburige zone, zijnde 'penalty2'.

Het doel is om de kost zo laag mogelijk te maken door een optimale plaatsing voor alle auto's te bepalen om aan zoveel mogelijk verzoeken te voldoen en de bijhorende kost te minimaliseren.
Een reservatie is voldaan als één van de gewenste auto's in deze zone beschikbaar is en dat deze voor het gevraagde tijdslot kan worden toegewezen. Een reservatie is half voldaan als één van de gewenste auto's aan een naburige zone is toegewezen voor het gevraagde tijdslot.

Aanpak

In de eerste fase hebben we een gedeelde code basis aangemaakt met de onderstaande klassenstructuur: Reservatie, Voertuig, Zone, VoertuigBezet.

De inhoud van de klassen Reservatie, Voertuig en Zone zijn een representatie van de betreffende concepten. Het object VoertuigBezet omvat een lijst met tijdsloten wanneer het voertuig bezet is en aan welk verzoek het gekoppeld is.
Om een breed bereik van implementaties te bekomen, hebben we geopteerd voor het splitsen van de opdracht, waarbij iedereen van een gemeenschappelijke basis vertrekt en een eigen solver probeert te maken. We zijn begonnen van de initiële oplossing waarbij elke auto een random zone toegewezen kreeg en waarbij dat er nog geen verzoeken voldaan waren.



De eerste solver is een Random benadering, die zoekt zolang er nog niet-toegewezen auto's zijn. Daarnaast wordt er random bepaalt of er een auto toegewezen gaat worden uit de gegeven lijst of bij aangrenzende zones te gaan kijken, want het is mogelijk dat daar al een auto kan staan die het verzoek vervult. Dit algoritme kan de hele zoekruimte aflopen maar heeft door de grootte en complexiteit van het probleem oneindig veel tijd nodig om het optimum te vinden. Deze functie heet zoekRuben.

In de functie zoekJeroen, wordt geopteerd om eerst op zoek te gaan naar de zones die het meest gebruikt worden in de huidige reservaties. Aan de hand van deze resultaten (meest voorkomende zones) zal er aan de voertuigen een nieuwe zone worden toegewezen. In tegenstelling tot de random benadering wordt er voor het toewijzen van de zones aan de voertuigen gebruikgemaakt van de meest voorkomende zones. Vervolgens wordt er een solver gebruikt die random zoekt zolang dat er nog niet toegewezen auto's zijn. Hierbij wordt er telkens een voertuig aan een reservatie gekoppeld. Er zijn twee mogelijkheden voor het toewijzen van de voertuigen. Er wordt een voertuig met een toegewezen zone aan een reservatie toegewezen, of er staat een voertuig in een naburige zone van de reservatiezone.
Met dit algoritme wordt de hele zoekruimte afgelopen, maar wanneer een voertuig reeds toegewezen is aan een reservatie, kan dit voor een beperking zorgen voor mogelijke oplossingen. Een andere mogelijke oplossing is om een voertuig, dat aan een reservatie toegewezen is, te verplaatsen naar een naburige zone. Dit kan ervoor zorgen dat we een lagere penalty score bekomen.

Voor het ontwerpen van de oplossingsmethode zoekChristophe zijn we begonnen met het probleem uit te werken in een flowchart waarvan de implementatie begon. Het algoritme begint met het random toewijzen van zones aan auto's, waaruit volgende stappen volgen. Er wordt geïtereerd over de lijst met reservaties. Indien een reservatie nog geen auto heeft toegekend,

wordt er geïtereerd over alle auto's. Wanneer er een auto gevonden is die geldig kan toegewezen worden, wordt er geverifieerd of de wagen zich in dezelfde zone of naburige zones van de reservatie bevindt. Er wordt een voorkeur geven aan auto's die zich in dezelfde zone bevinden als de reservatie om de penalty score zo laag mogelijk te houden. Wanneer alle reservaties doorlopen zijn, is er een initiële oplossing gevonden. Hierna wordt er een willekeurige auto verplaatst naar een willekeurige nieuwe zone. Indien dit een betere oplossing is, wordt deze behouden en anders wordt deze verworpen. Dit process gaat door totdat de tijd om is.

Het algoritme werkt correct op de Toy1 dataset maar wegens een bug zijn de resultaten incorrect op grotere datasets. Daarom is het deel van het algoritme dat zoekt in naburige zones uitgeschakeld, zodat de resultaten correct zijn.

Resultaten

zoekRuben	**sec-seed- thread**		**sec-seed- thread**		**sec-seed- thread**		**sec-seed- thread**	
input\ argument	**10-123-2**	**30-123-2**	**10-130-2**	**30-130-2**	**10-120-2**	**30-120-2**	**60-123-2**	**60-75-2**
toy1.csv	700	700	700	700	700	700	700	700
100_5_14_25.csv	30250	30250	30540	30430	30430	30230	30250	30790
100_5_19_25.csv	30740	30450	30630	30240	30625	30625	30310	30250
210_5_33_25.csv	65090	65040	65210	65020	65320	65130	64960	64875
210_5_44_25.csv	67640	67640	67830	67610	67740	67740	67670	67800
360_5_71_25.csv	115530	115530	115730	115500	115700	115470	115530	115200

We zien hier dat ons programma een kleine verbetering toont als het meer rekentijd krijgt, maar de verbeteringen zijn niet noemenswaardig. Ons programma lijkt de extra tijd niet nuttig te gebruiken, dit is wat we ook verwachten van een random zoek programma. We zien ook dat onze kost 3 tot 10 keer slechter zijn dan de gegeven niet-optimale waarden.

zoekJeroen	**sec-seed-thread**	**sec-seed-thread**	**sec-seed-thread**	**sec-seed-thread**

Input \ argument	**10-123-2**	**30-123-2**	**10-130-2**	**30-130-2**	**10-120-2**	**30-120-2**	**60-123-2**	**60-75-2**
toy1.csv	700	700	700	700	700	700	700	700
100_5_14_25.csv	30610	30510	30500	30480	30750	30750	30500	30180
100_5_19_25.csv	30740	30700	30780	30700	30500	30360	30700	30495
210_5_33_25.csv	65400	65110	65050	65000	65420	65350	64880	64710
210_5_44_25.csv	68020	67860	67720	67630	67640	67610	67860	67900
360_5_71_25.csv	115500	115460	115470	115090	115170	115230	115060	115100

Uit de bovenstaande tabel kunnen we afleiden dat wanneer we het programma meer tijd geven voor het berekenen, we geen noemenswaardige verbetering krijgen. We zien echter maar een kleine verbetering t.o.v. het resultaat dat we bekwamen bij het uitvoeren van het programma met een korte uitvoeringstijd. Aangezien dat we hier gebruik maken van een random functie voor de toewijzingen, is dit echter wel te verwachten. Om een betere penality score te bekomen, kan men best opteren om een voertuig aan een naburige zone toe te wijzen. Dit kan als resultaat hebben dat we een lagere penalty score bekomen.

zoekChristophe	Sec- seed- thread		Sec - seed- thread		Sec - seed - thread		Sec - seed - thread	
input\ argument	**10-123-8**	**30-123-8**	**10-130-8**	**30-130-8**	**10-120-8**	**30-120-8**	**60-123-8**	**60-75-8**
toy1.csv	700	700	800	800	700!(800)	700!(800)	700	800
100_5_14_25.csv	30850	30850	30490! (30820)	30490! (30820)	31010	31010	30850	31050
100_5_19_25.csv	31330	31330	31200	31200	30980	30980	31330	31090
210_5_33_25.csv	65090	65090	65030	65030	65320	65320	65090	65580
210_5_44_25.cs	67760	67760	68070!	68070!	67740	67740	67760	67740

v			(68500)	(68501)				
360_5_71_25.csv	115370	115370	115430	115430	115710	115710	115370	115480

Ook hier zien we dat het algoritme weinig tot geen verbeteringen geeft indien er meer tijd wordt voorzien. Bij de getallen waar een uitroepteken bijstaat, wordt er een betere kost gevonden, maar wordt de oplossing van deze kost niet behouden. De betere oplossing wordt weggeschreven maar bij de volgende iteratie is deze teniet gedaan. Het getal tussen haakjes geeft de effectieve kost van de teruggegeven oplossing weer. Dit algoritme scoort in de meeste gevallen echter slechter dan de versies van Ruben en Jeroen.

Aangezien onze programma's weinig verbeteren als we ze meer tijd geven, hebben we besloten om de versie van Ruben en Jeroen na elkaar uit te voeren en de beste oplossing terug te geven. De versie van Christophe heeft vreemde bugs en dit is ook de reden waarom het niet bij de finale versie is opgenomen.

Reflectie

Doordat onze objecten in lijsten zitten die niet geoptimaliseerd zijn om efficiënt te kunnen doorzoeken, wordt er relatief veel tijd gestoken in het itereren over de lijsten. Deze zou beter besteed kunnen worden naar het zoeken van andere oplossingen. Dit zou een verbetering geven bij grotere opdrachten. Een grotere winst van de kost kan echter behaald worden door het implementeren van Hill climbing, TabuSearch en/of het Greedy algoritme. Graag hadden we ook Hill climbing willen implementeren, maar we bleven bij gebruik van deze code incorrecte oplossingen bekomen. Als deze fouten uit de code opgelost zijn, dan zouden we het hill climbing algoritme hierop toepassen. We blijven dit doen totdat er geen betere kost gevonden wordt voor x aantal epochs op een rij. Om de kans te verkleinen dat het algoritme vast komt te zitten in een lokaal optimum, laten wij de hill climbing random herstarten indien er geen verbetering waargenomen worden voor enkele opeenvolgende stappen. Hierbij laten we het algoritme uitvoeren gedurende een beperkt tijdbudget en retourneren we de beste oplossing die we gevonden hadden.

Besluit

Ons algoritme vindt geldige oplossingen, maar het resultaat kan verbeterd worden door het implementeren van extra zoekalgoritmes zoals Hill climbing, TabuSearch en/of het Greedy algoritme. Hierdoor zullen er betere oplossingen kunnen gevonden worden. Aangezien we van de extra zoekalgoritmes geen gebruik maken, heeft ons programma maar een beperkte nood aan meer tijd om betere oplossingen te vinden.