Space Freight

Heuristieken
Minor Programmeren 2017

Wij:
Jennifer Buur, Emma Hokken, Sem Sturkenboom

Abstract

Het ISS is een ruimtestation wat bemand wordt door meerdere ruimtevaartorganisaties. Om vracht naar het ISS te krijgen willen we een transportvloot opstellen. Hiervoor hebben we drie algoritmes opgesteld: Greedy, *Brute Force*, en *Random Optimization*. Uit de resultaten blijkt dat het Greedy algoritme een optimale oplossing geeft voor het inladen op gewicht. Voor inladen op volume werkt het *Brute Force* algoritme beter. Het algoritme dat het beste werkt om de transportvloot samen te stellen is het *Random Optimization* algoritme.

Introductie

Het International Space Station (ISS) is een groot ruimtevaartstation wat rond de aarde cirkelt. In dit station wordt veel onderzoek uitgevoerd door astronauten vanuit verschillende landen. Dit station heeft natuurlijk materiaal nodig: eten voor de astronauten, materiaal om onderzoek mee uit te kunnen voeren, en nog veel meer. Dit materiaal, of vracht, moet op de een of andere manier naar boven komen. Dit gebeurt met raketten. Hiervoor werken vier ruimtes organisaties samen: de Amerikaanse NASA, de Japan Aerospace Exploration Agency, de European Space Agency, en de Russian Federal Space Agency. Deze hebben ieder hun eigen raket met hun eigen gewicht capaciteit (in kg) en ruimte capaciteit (in m3). De precieze specificaties zijn in onderstaande tabel te vinden. Om goed te kunnen beoordelen of vracht goed in een ruimteschip past is ook de dichtheid (gewicht/volume) van de schepen berekend.

Ruimteschip	**Land**	**Organisatie**	**Capaciteit KG**	**Capaciteit m3**	**Dichtheid**
Cygnus	VS	NASA	2000	18.9	105.8
Verne ATV	EU	ESA	2300	13.1	175.6
Progress	Rusland	RFSA	2400	7.6	315.8
Kounotori	Japan	JAEA	5200	14	371.4

Later in dit verslag zullen er nog twee andere ruimteschepen in het spel komen. De specificaties van deze schepen staan hieronder.

Ruimteschip	**Land**	**Organisatie**	**Capaciteit KG**	**Capaciteit m3**	**Dichtheid**
TianZhou	China	CNSA	6500	15	433.3
Dragon	VS	SpaceX	3400	42	81.0

Wij kregen de opdracht om deze schepen op verschillende manieren te vullen. Als eerste moesten we alleen letten op het gewicht, daarna moesten we ook het volume in ogenschouw nemen. Als laatste moesten we nog rekening houden met politieke *constraints*. De specifieke opdrachten worden hieronder verder uitgelegd.

Over de toestandsruimte kunnen we al zeggen dat deze een bovengrens heeft van:

$$\#\ spacecrafts^{\#cargoitems}$$

Materiaal

Voor dit project zijn de drie aangeleverde *cargolists* gebruikt.

Verder hebben we voor dit project gebruik gemaakt van Python versie 3.5. We hebben dit geschreven in Atom, wat geïnstalleerd is op MacBooks met de laatste versie van OS-X: versie 10.12.5.

Methode

Tijdens dit project hebben we 3 algoritmes gebruikt. Een uitleg van alle algoritmes is in onderstaande sectie te vinden.

Greedy

Voor dit algoritme hebben we het algoritme zo gemaakt dat als een *cargoitem* in het schip past deze er ook ingezet wordt. In dit geval itereren we over alle cargo in de lijst en vullen we de schepen op volgorde. Aangezien de schepen op 24 (4!) verschillende volgorde kunt zetten, krijgen we 24 verschillende mogelijke oplossingen voor een *cargolist*. Deze lijst kan random in de schepen geladen worden, maar het is wellicht een goed idee om de lijst te sorteren. Als iemand gaat verhuizen worden de verhuisdozen eerst met de zwaarste spullen ingeladen. De lichtere items worden bovenop gelegd. Als we hier rekening mee houden lijkt het ons een goed idee om de lijst te sorteren van zwaar naar licht, en van grote dichtheid naar lage dichtheid. We hebben gekozen voor de dichtheid omdat een schip het best gevuld kan worden als de gemiddelde dichtheid van de ingeladen *cargoitems* gelijk is aan de dichtheid van het desbetreffende schip. Daarom laden we een *cargoitem* in een schip als het verschil tussen de dichtheid van het schip en de nieuwe gemiddelde dichtheid van alle ingeladen *cargoitems* het kleinst is ten opzichte van de andere schepen.

Brute Force algoritme

In dit algoritme is de cargo niet gesorteerd op dichtheid maar worden wel geplaatst in de *spacecrafts* op basis van hun dichtheid. De score functie voor het bepalen van de optimale indeling die we hierbij gebruiken is als volgt: *wasted weight* + *unloaded items*. In de score functie is de *wasted space* weggelaten omdat het totale gewicht van *cargolist* 1 gelijk is aan 11.907 kilo en het totale volume aan 72.05 m^3. De beschikbare *spacecrafts* hebben in totaal 11.900 kilo en 53.6 m^3 beschikbaar. Dit betekent dat we zoveel mogelijk zware spullen moeten meenemen om tot zo weinig mogelijk *wasted weight* te komen. Door de *Spacecraft* te vullen zal er nooit een grote *wasted space* ontstaan met *cargolist* 1.

Onze prioriteit ligt in deze opdracht bij het minimaliseren van de *wasted weight*. Daarnaast zorgen we ervoor middels deze score functie dat een oplossing met eenzelfde *wasted weight* maar meer ingepakte cargo's de voorkeur heeft boven een oplossing met eenzelfde *wasted weight* maar minder ingepakte cargo's.

Random Optimization algoritme

In het derde algoritme willen we kijken of het optimaliseren van de *cargolist* gevolgen heeft voor de resultaten. We hebben hier twee soorten geoptimaliseerde lijsten voor gemaakt: een *optimized list* en een *shortened list*. Bij de *optimized list* wordt de originele *cargolist* gesorteerd van grote dichtheid naar kleine dichtheid. Vervolgens wordt de maximale ruimte en gewicht van de *spacecrafts* berekend. Hierna wordt er over de originele *cargolist* geïtereerd en worden alle *cargoitems* die een dichtheid hebben die lager of gelijk is aan de dichtheid van de *spacecrafts* in een nieuwe, geoptimaliseerde cargolist gestopt. Deze wordt vervolgens met het bovenstaande *Brute Force* algoritme in de schepen geladen.

Bij de *shortened list* gebeurt in principe hetzelfde, alleen wordt de originele *cargolist* van tevoren niet gesorteerd. Dit is omdat de derde cargolist heel groot is en er anders veel te veel lege schepen de lucht in worden gestuurd.

Resultaten

Inladen op gewicht

Greedy algoritme: *Cargolist* 1 en 2

De eerste opdracht was om de eerste 4 ruimteschepen in te laden, en hierbij alleen te letten op het gewicht. Voor zowel *cargolist* 1 als 2 kwamen door deze manier meteen tot een optimale oplossing:

	***Cargolist* 1**	***Cargolist* 2**
Unloaded cargo	0	1
Wasted weight	5	4

Dit zijn optimale oplossingen omdat het totale gewicht van *cargolist* 1 11895 kilo is. Dat is vijf kilo minder dan het totale gewicht dat meegenomen kan worden door de schepen, namelijk 11900. Als alles mee kan, dan is er een optimale oplossing gevonden.

Bij *cargolist* 2 is duidelijk dat dit de optimale oplossing is, omdat bekend is dat het totale gewicht van de lijst gelijk is aan 11907 kilo. Daarnaast is bekend dat het lichtste item 11 kilo is, waardoor duidelijk is dat als alleen dit item niet mee zou gaan de optimale oplossing gevonden is. Het totale meegenomen gewicht is gelijk aan 11896, wat gelijk staat aan 4 kilo *wasted weight*.

Inladen op dichtheid

Greedy algoritme: *Cargolist* 1 en 2

Naast dat we rekening moeten houden met het gewicht dat meekon in de schepen, moeten we natuurlijk ook rekening houden met het volume van de schepen.
Voor *cargolist 1* hebben we voor dit *Greedy* algoritme het volgende resultaat gevonden:

Wasted space	**Wasted weight**	**Unloaded cargo**
9.62	3819	21

Voor *cargolist* 2 hebben we met dit *Greedy* algoritme het volgende resultaat gevonden:

Wasted space	**Wasted weight**	**Unloaded cargo**
0.2	6224.0	45

Brute Force algoritme: *Cargolist* 1 en 2

Om te testen of bovenstaand resultaat daadwerkelijk een goed resultaat was hebben we met het *Brute Force* algoritme nogmaals naar deze lijsten te kijken. Dit algoritme hebben we 100.000.000 keer laten lopen in een tijdsspanne van +- 10 uur. Het beste resultaat van dit *Brute Force* algoritme is in onderstaande tabel weergegeven. Een meer uitgebreide samenvatting van de resultaten van dit algoritme is te vinden in de appendix.

Cargolist 1

Wasted space	**Wasted weight**	**Unloaded cargo**
4.22	3230.0	21

Cargolist 2

Wasted space	**Wasted weight**	**Unloaded cargo**
0.08	4389.0	28

<u>*Random Optimization* algoritme: *Cargolist* 1 en 2</u>

Alhoewel het resultaat beter is, willen we onderzoeken of het nog beter kan. Dit willen we doen door het optimaliseren van de *cargolist*. Voor *cargolist* 1 en 2 hebben we gebruik gemaakt van de *optimized cargolist*, en dus niet de *shortened cargolist*.

Voor *cargolist* 1 heeft de geoptimaliseerde *cargolist* totaal volume van 53.5 m^3 en totaalgewicht van 9765 KG Het is dus niet mogelijk om met de geldende restricties meer dan 9765 KG mee te nemen, wat neerkomt op een minimale *wasted space* van 11900 – 9765 = 2135 KG, als alle *cargoitems* meegenomen kunnen worden. Met deze geoptimaliseerde *cargolist* hebben we opnieuw het *Brute Force* algoritme uitgevoerd omdat bleek dat dit beter werkt dan het *Greedy* algoritme. Het resultaat van dit *Brute Force* algoritme is beduidend beter dan de vorige, namelijk:

Cargolist 1

Wasted space	**Wasted weight**	**Unloaded cargo**
5.69	3082.0	12

Een meer uitgebreide samenvatting van de resultaten is te vinden in de appendix.

Het resultaat uit dit *Random Optimization* algoritme middels de geoptimaliseerde *cargolist* is beduidend beter en sneller dan de eerder genoemde *Greedy* en *Brute Force* algoritmes. Het optimale resultaat van 2135 KG weten we echter nog niet te benaderen.

Voor cargolist 2 hebben we met het derde algoritme een geoptimaliseerde *cargolist* bepaald waarbij we een totaal volume van 53.52 en totaal gewicht van 7960 hebben. Wat neerkomt op een minimale *wasted weight* van 3940 kilo. Als we hier het *Brute Force* algoritme op toepassen krijgen we het volgende resultaat.

Wasted space	**Wasted weight**	**Unloaded cargo**
0.08	3940.0	28

Uit bovenstaande resultaten blijkt dat de beste oplossing uit de resultaten gelijk is aan onze vooraf vastgezette optimale oplossing, namelijk een minimale *wasted weight* van 3940 kilo. Een meer uitgebreide samenvatting van de resultaten is te vinden in de appendix.

Meerdere spelers

Zowel *cargolist* 1 als *cargolist* 2 waren niet heel groot, maar het ISS heeft waarschijnlijk veel vracht nodig. Daarvoor is *cargolist* 3 in het leven geroepen. Deze bestaat uit maar liefst 1250 *cargoitems*. Al deze *cargoitems* kunnen uiteraard niet in een keer naar boven. Hiervoor moet een transportvloot samengesteld worden. Hierbij moeten we ook rekening houden met extra spelers: China en SpaceX uit de VS. In eerste instantie worden er politieke constraints opgezet: als Rusland 2 schepen naar het ISS stuurt dan moeten de rest van de landen er minimaal 1 en maximaal 3 sturen. Dit betekent voor de VS die 2 vaartuigen hebben we een keuze kunnen maken tussen Cygnus en Dragon.

Op deze derde *cargolist* hebben we alleen het *Random Optimization* algoritme toegepast omdat uit de vorige secties bleek dat dit algoritme het beste werkt.

Random Optimized algoritme: met politieke constraints

De restricties voor deze opdracht waren dat ieder land één schip meer of minder naar boven mag sturen dan de andere landen. Dit betekent dat de VS, die twee schepen hebben, één schip op de grond moet laten staan. We hebben ervoor gekozen om het schip Cygnus van NASA niet te gebruiken omdat dit schip de meest ongunstige dichtheid had van de twee schepen van de VS.

Met het inladen van de overige vijf schepen hebben we het *Brute Force* algoritme toegepast. Dit algoritme is telkens 1000 keer over de *shortened cargolist* heen gegaan. Uit de resultaten bleek dat er in totaal minimaal 72 schepen naar boven moeten om alle cargo uit *cargolist* 3 naar het ISS te krijgen. Specificaties van hoe vaak elk schip naar boven moet staan in onderstaande tabel.

Schip	**Land**	**Aantal keer omhoog**
Cygnus	VS	0
Verne	EU	14
Progress	Rusland	14
Kounotori	Japan	14
TianZhou	China	15
Dragon	VS	15
Totaal	**-**	**72**

Random Optimized algoritme: zonder politieke constraints

Wat nou als die politieke constraints niet golden? Zou je dan minder schepen naar boven moeten sturen? Om dit te testen hebben we besloten om in eerste instantie alleen de twee grootste schepen te gebruiken: de TianZhou van China en de Dragon van de VS. We hebben hier weer hetzelfde *brute force* algoritme op laten lopen en we zijn weer gaan werken met de *shortened lists*. Uit de resultaten bleek dat zowel de TianZhou als de Dragon ieder 28 keer naar boven moeten, wat neerkomt op 56 schepen in totaal.

Conclusie en Discussie

Uit bovenstaande resultaten kan geconcludeerd worden dat als we alleen rekening moeten houden met het gewicht van de cargo en het gewicht dat de schepen aankunnen het *Greedy* algoritme meteen de optimale oplossing vindt. Als we rekening houden met het volume van de cargo en de schepen werkt dit algoritme echter minder goed. Het *Brute Force* algoritme komt hier dichterbij een optimale oplossing, maar dit algoritme bereikt deze nog niet. Het *Random Optimization* algoritme komt het dichtste bij de optimale oplossing.

Uit de resultaten van het *Random Optimization* algoritme kunnen we concluderen dat *cargolist* 1 moeilijker is in te delen dan *cargolist* 2. Want ook al is bij beide lijsten de oplossing met de ingekorte *cargolist* beter, er is ook te zien dat het resultaat bij *cargolist* 1 wel ten koste gaat van de *wasted space*. De reden hiervoor heeft te maken met de diversiteit van de dichtheid van de *cargo*. Bij *cargolist* 1 varieert de dichtheid van 107.4 tot 347.1, terwijl de schepen variëren van 105.8 tot 371.4. Dit betekent dat de *spacecrafts* nooit optimaal te vullen zijn aangezien de *range* van de dichtheid van de *spacecrafts* buiten die van de *cargo* vallen. Bij *cargolist* 2 is dit niet het geval en daarom zijn de *spacecrafts* niet optimaal te vullen.

Voor de derde *cargolist* hebben we alleen het *Random Optimization* algoritme toegepast. Uit de resultaten is gebleken dat de toevoeging van politieke constraints er voor zorgt dat er veel meer schepen naar boven moeten, namelijk 72 - 56 = 16 schepen. Dit zal ervoor zorgen dat alle cargo omhoog krijgen veel duurder zal zijn dan als er geen politieke constraints zijn. De politieke constraints zijn in het leven geroepen omdat men niet wil dat één of twee landen in hun eentje moeten betalen voor het naar boven brengen van de vracht die alle landen gaan gebruiken. Een idee voor het oplossen van dit probleem is dat de kosten voor de twee gebruikte ruimteschepen worden verdeeld over de vijf landen.

In conclusie kunnen we zeggen dat het *Random Optimization* algoritme het beste werkt voor het inladen van de vracht als er rekening wordt gehouden met het volume van de vracht. Als alleen naar het gewicht wordt gekeken blijkt het Greedy algoritme beter te zijn.

Appendix: Gedetailleerde resultaten algoritmes.

Brute force algoritme, hele lijst:

Cargolist 1

Iterations	Score	Wasted space	Wasted weight	Unloaded cargo
0	4278.0	6.11	4214.0	36
4	4151.0	5.88	4092	31
877	3651.0	5.39	3595.0	28
1609	3614.0	6.32	3557.0	29
4576	3520.0	3.9	3467.0	25
26204	3466.0	5.39	3407.0	31
427678	3410.0	4.61	3354.0	28
628386	3337.0	4.3	3284.0	25
2188208	3311.0	4.9	3256.0	27
10741731	3300.0	4.87	3247.0	25
29028980	3281.0	4.22	3230.0	21

Cargolist 2

Iteration	Score	Wasted space	Wasted weight	Unloaded cargo
0	6002.0	0.22	5951.0	51
4	5381.0	0.28	5340.0	41
540	5050.0	1.29	5013.0	37
715	5015.0	0.15	4977.0	38
14186	4645.0	0.29	4610.0	35
613900	4546.0	0.11	4513.0	33
13024308	4443.0	0.59	4408.0	35
37107424	4429.0	0.13	4398.0	31
54826208	4417.0	0.08	4389.0	28

Random Optimization algoritme, geoptimaliseerde lijst.

Cargolist 1

Iteratie	Score	Wasted space	Wasted weight	Unloaded cargo
0	3527.0	7.67	3515.0	12
3	3484.0	7.17	3472.0	12
522	3368.0	7.13	3356.0	12
1714	3304.0	5.79	3289.0	15
2074	3260.0	5.98	3249.0	11
6382	3249.0	6.1	3238.0	11
342270	3099.0	5.86	3090.0	9
816945	3094.0	5.69	3082.0	12

Cargolist 2

Iterations	Score	Wasted space	Wasted weight	Unloaded cargo
0	4879.0	4.32	4845.0	34
1	4616.0	4.56	4584.0	32
83	4131.0	0.92	4102.0	29
280	4031.0	0.76	4002.0	29
2564	3968.0	0.08	3940.0	28