Algoritmes

Nearest neigbhour

Het eerste algoritme is het **nearest neighbour** algoritme die aan de hand van de volgende constraints een lijnvoering maakt:

- Startstation = random.
- Kies altijd nearest neighbour tenzij de nearest neighbour al bereden is, kies dan 1. Het kortste spoor naar een onbereden station. 2. De kortste van de onbereden sporen.
- Alle sporen en stations bereden vanuit het huidige station? Kies willekeurig een spoor.
- Als alle sporen & alle stations al zijn geweest: break.
- Traject stopt bij 120 minuten. (Maak een nieuw traject als dit overschreden wordt en herhaal bovenstaande stappen)
- Pas scorefunctie toe, als het niet nodig was een extra traject te maken, maar het wel gebeurd is scorefunctie: trajecten(t) -1.

Farest neighbour

Farest neighbour algoritme: *Het nearest neighbour algoritme, maar in plaats van het nearest spoor, kiest het het farest spoor.*

Random neighbour

Random neighbour algoritme*: Het nearest neighbour algoritme, maar in plaats van het nearest spoor, kiest het een random spoor.*

Hill climber

Het tweede algoritme dat wordt toegepast is een **'hill climber'**: het verschil met het nearest neighbour algoritme is het start punt:

- De hill climber begint met de oplossing, x1, van het nearest neighbour algoritme en berekent de scorefunctie f(x1).
- Deze oplossing, x1, wordt aangepast door het nearest neighbour algoritme opnieuw te laten runnen, maar dan met willekeurig gekozen beginstations. Dit levert ons oplossing x2.
- Reken de scorefunctie f(x2) uit voor de nieuwe oplossing, x2.
- Bewaar de oplossing die de hoogste scorefunctie oplevert.
- Itereer x aantal keer.

Simulated annealing

Het derde algoritme dat wordt toegepast is **'simulated annealing'**: het verschil met het hill climber algoritme is dat niet altijd de beste oplossing bewaard wordt:

- Het SA algoritme begint met de oplossing, x1, van het nearest neighbour algoritme en berekent de scorefunctie f(x1).
- Deze oplossing, x1, wordt aangepast door het nearest neighbour algoritme opnieuw te laten runnen, maar dan met willekeurig gekozen beginstations. Dit levert ons oplossing x2.
- Reken de scorefunctie f(x2) uit voor de nieuwe oplossing, x2.
- Bewaar de oplossing afhankelijk van de weging van de kans functie, P(T). Als f(x2)(nieuw) > f(x1)(oud) : accepteer x2(nieuw) als oplossing. Als f(x2)(nieuw) < f(x1)(oud), accepteer MISSCHIEN x2(nieuw).

P(T) is een functie die schaalt tussen 0 en 1 en bepaalt of, van x1 en x2, de hoogste of de laagste oplossing wordt aangenomen of niet. Deze kans functie is afhankelijk van T, de temperatuur. Deze temperatuur is een analogie met het proces van het koelen van metalen.

Wij kiezen voor verschillende koelschema´s om vervolgens te kijken welke koeling techniek ons de beste score geeft:

Lineair: $Ti = To – i(To-Tn)/ N$

Exponentieel: $Ti = To(Tn/To)^{(i/N)}$

Sigmoidal: $Ti = Tn + (To-Tn)/(1 + e^{(0.3(i-N/2))})$

Geman: $Ti = c / \log(i+d)$

Ti is de huidige temperatuur.

To is de begin temperatuur.

Tn is de eindtemperatuur.

I is de huidige iteratie.

N is het totale aantal iteraties.

$Ti = a * T(i-1)$ (a is een constante tussen 0.8 en 0.99)

Met Ti kan de kans berekend om bepaalde oplossingen aan te nemen berekend worden met

$P(Ti) = e^{(verkorting/Ti)}$. Waar de verkorting =scorefunctie(ni) - scorefunctie(ni-1).

- Itereer N aantal keer.

Verklaring nearest neighbour

De reden waarom wij gekozen hebben voor het nearest neighbour algoritme is dat dit algoritme zeer goed tot de verbeelding spreekt als het ter visualisatie gecombineerd wordt met een graaf. Het was een efficiënte manier om op spelende wijze kennis te maken met de case. Onze case is een variatie van een travelling salesman problem. De overeenkomst is dat alle “nodes” in de graaf bezocht moeten worden, echter zijn in ons geval niet alle “nodes” met elkaar verbonden. Een simpele versie van de travelling salesman problem kan met behulp van een nearest neighbour algoritme 25 procent onder de optimale score van het probleem komen. (Johnson, D.S and Mcgeoch, L.A. (1997)). Omdat wij te maken hebben met

een variatie van een travelling salesman problem, hoopten wij dat het nearest neighbour algoritme voor ons probleem ook een redelijke eerste score (25 tot 50 procent naast de optimale score) zou krijgen. Uiteindelijk bleek uit experimentatie dat het algoritme erg goed scoorde, vandaar dat we het algoritme hebben gehouden in onze top 3.

Onze eerste score met het nearest neighbour* algoritme was : **8880**. De hoogst mogelijke score die gehaald kan worden voor ons probleem is: **9819.84**. We zitten dus ongeveer **10 procent** van de hoogst mogelijke score af.

*voor de vergelijking met farest neighbour/random neighbour/nearest neighbour, zie **Heuristieken**/RailNL/Resultaten/**Algoritmes**/*resultaten experiment Holland en nl.pdf/*

Verklaring hill climber

Het hill climber algoritme kan binnen korte tijd een goede score opleveren voor de scorefunctie van het op te lossen probleem, en het is niet moeilijk te programmeren. Een nadeel van het algoritme is dat er geen zekerheid is voor het vinden van een maximum van de scorefunctie. In tegendeel, de hill climber staat bekend om het bereiken van een lokaal maximum van de scorefunctie (Russell et al, *2003)*. Het hill climber algoritme hebben wij gekozen omdat we hiermee onze nearest neighbour kunnen optimaliseren. We zullen niet de beste oplossing hiermee vinden, maar we zullen zeker een verbetering van het nearest neighbour algoritme vinden. Bovendien is de stap van een hill climber naar een simulated annealing algoritme relatief makkelijk gezet, lees verder voor meer informatie daarover.

Ons nearest neighbour algoritme begint op willekeurige stations, echter weten wij niet welke beginstations de beste score zullen opleveren. Daarom hebben wij een hill climber ontworpen; we laten stapsgewijs de beginstations veranderen om uit te zoeken wat de meest optimale startvoorwaarden zijn van ons nearest neighbour algoritme. De resultaten hiervan hebben ons niet teleurgesteld.

Nearest neighbour algoritme score:	**8880**
Hill climber score:	**9780**
Maximale score:	**9820**

De hill climber is een verbetering van ongeveer **10 procent** ten opzichte van het nearest neighbour algoritme. We zitten nu nog maar **40 scorepunten** af van de maximaal haalbare score. Dit is minder dan een verschil van **0.5 procent**.

Verklaring simulated annealing

Het simulated annealing algoritme staat erom bekend dat het in staat is om het maximum van een scorefunctie te vinden. Waar de hill climber blijft haken op een lokaal maximum, kan een goed geïmplementeerd simulated annealing algoritme gegarandeerd de beste oplossing vinden als het oneindig lang zou kunnen draaien (Cerny, 1985). Het levert ons dus niet gegarandeerd de beste oplossing, maar het biedt wel de mogelijkheid om de beste score te vinden. Omdat we al een goedwerkende hill climber hebben leek het ons het makkelijkste om over te stappen op simulated annealing, met de hoop dat we nog dichter bij de maximale score kunnen komen.

Nearest neighbour algoritme score:	**8880**
Hill climber score:	**9780**
Simulated annealing score:	**xxxx**
Maximale score:	**9820**

Citaties:

• • *Russell, Stuart J.; Norvig, Peter* (2003), *Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, pp. 111–114, ISBN 0-13-790395-2*

• • Černý, V. (1985). "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". *Journal of Optimization Theory and Applications*. **45**: 41– 51. doi:10.1007/BF00940812

• Johnson, D. S.; McGeoch, L. A. (1997). "The Traveling Salesman Problem: A Case Study in Local Optimization" (PDF). In Aarts, E. H. L.; Lenstra, J. K. *Local Search in Combinatorial Optimisation*. London: John Wiley and Sons Ltd. pp. 215–310.