Handelsreizigersprobleem

**Concepten:** complexiteit en praktisch onoplosbare problemen

**Doel:** Voor sommige problemen is het vinden van de optimale (d.w.z. beste) oplossing praktisch ondoenlijk. Vaak bestaan er oplossingsstrategieën die zulke optimale oplossingen benaderen en die wel praktisch doenlijk zijn. In dit geval willen we het kortst mogelijke pad bepalen om een aantal locaties te bezoeken en terug te keren naar het startpunt (het handelsreizigersprobleem).

**Leerdoelen:**

Na afloop kun je:

* Redeneren over de complexiteit van een algoritme in relatie tot de invoer.
* Herkennen of er een efficiënte oplossing voor een probleem bestaat, of dat het praktisch onoplosbaar is.
* Een heuristiek toepassen om een oplossing te vinden voor een onhandelbaar probleem.

**Algoritmes en technieken:** Brute force algoritmes, gretige algoritmes, heuristieken, handelsreizigersprobleem.

**Voorkennis**: minimaal opspannende bomen, praktisch onoplosbare problemen

**Praktische context**: Oplossen van problemen die gebaseerd zijn op het handelsreizigersprobleem.

# Inleiding

Er zijn problemen die te moeilijk zijn voor een computer, bijvoorbeeld een gesprek voeren. Dit is te moeilijk omdat wij zelf niet snappen hoe we het doen. Maar er zijn ook dingen die een computer makkelijk kan uitvoeren, maar waar de computer te lang over doet. Dit gebeurt als er bijvoorbeeld heel veel berekeningen moeten gebeuren. Zo’n probleem noemen we dan ‘onhandelbaar’ of ‘niet efficiënt’.

**Opdracht 1. Krantenwijk**

Tom heeft een krantenwijk. Hij woont bij A en moet langs drie flatgebouwen (B,C en D) en daarna weer terug naar huis. Bepaal voor het kortste route vanuit zijn huis, langs alle andere flats en terug naar huis. Hij wil geen twee keer langs dezelfde flat fietsen. Dit is een variant van het handelsreizigersprobleem.



Antwoord: De kortste route, zonder een locatie meerdere keren te bezoeken is:

A-> B -> D -> C -> A. De afstand is dan 2+1+4+4 = 11

**Opdracht 2. Op bezoek**

Kies 5 verschillende plekken waar jij met enige regelmaat naar toe gaat (bijvoorbeeld school, voetbaltraining, huis of het huis van je vriendin). Zoek met Google maps op hoe lang het duurt om van elk van die plekken naar elk ander plek te fietsen. Maak een eenvoudige versie van een kaart (een graaf dus) met de 5 plekken en elke reisduur. Merk op dat Google maps rekening houdt met welke kant je op fietst: als je bergop moet doe je daar natuurlijk langer over dan als je de andere kant op zou gaan en bergaf gaat. Als de reistijd de ene kant op anders is dan de ander, gebruik dan pijltje om de richting aan te geven (je krijgt dan een gerichte graaf).

Gebruik het kortste-pad algoritme van Dijkstra om te bepalen wat de snelste route is van een van die plekken (bijvoorbeeld jouw huis) naar alle andere plekken.

*Teken hier je graaf en bepaal de snelste route langs alle locaties*.

*Antwoord:*

*Let op dat er pijltjes in de graaf voorkomen als de ene kant op sneller is dan de ander.*

# Praktisch onoplosbare probleem

Om de beste oplossing te vinden voor het handelsreizigersprobleem moet je eerst alle mogelijke paden uitrekenen en dan kijken welke de kortste is. Door een locatie toe te voegen wordt het aantal mogelijke paden die je af moet gaan steeds veel groter wordt (je moet alle combinaties afgaan). Op een gegeven moment zal de snelste computer er honderden jaren over doen om dit probleem voor maar een relatief klein aantal locaties op te lossen.

Bekijk dit filmpje over praktisch onoplosbare problemen: <https://www.youtube.com/watch?time_continue=3&v=xWQ7D08eUVw>

# Het handelsreizigersprobleem**[[1]](#footnote-1)**

In dit opdracht kijken we naar het Handelsreizigersprobleem (in het Engels: Travelling Salesman Problem). De handelaarsreiziger moet langs een aantal locaties (steden, huizen, luchthavens, ...), maar wil natuurlijk zo snel mogelijk weer thuis zijn. Het doel is om de kortste weg te vinden die precies een keer langs iedere stad komt, en weer in de thuisbasis terugkeert.

Merk op dat dit een andere probleem is dan de kortste pad tussen twee plekken te vinden (zoals een navigatiesysteem doet).

In dit opdracht onderzoek je waarom dit een onhandelbaar probleem is. Je ziet dan in dat alle mogelijkheden afgaan (brute-force) een inefficiënte oplossing is, en je bekijkt hoe je toch op een handige manier een oplossing kunnen vinden, al is het misschien niet de aller beste (meest optimale) oplossing.

# Deel A: Onhandelbare groei

Op de volgende website worden kortste routes berekend tussen steden met behulp van snelle computers. Ga naar <http://www.csfieldguide.org.nz/releases/1.9.9/_static/widgets/tract-tsp-basic-v2.html>.

**Opdracht 3. Rekentijd**

1. Voer 5 steden in. Hoe lang duurt het om de kortste route te berekenen?

|  |
| --- |
| Antwoord 5 steden: +/-1,2 seconden (ter info: Antwoord 6 steden: +/- 6 sec) |

1. Hoe lang schat je dat het zou duren om de kortste route langs twee keer zo veel steden te vinden? Voer 10 steden in (twee keer zo veel). Duurt het twee keer zo lang?

|  |
| --- |
| Antwoord 10 steden: +/- 5 uur, 2 minuten en 24 seconden. Nee, dat is veel langer dan dubbel zo lang. |

1. En als je weer twee keer zoveel steden kiest (20 steden)? En 50 steden? Kan je schatten hoe lang het zou duren?

|  |
| --- |
|  |

# Deel B: Oorzaak van de groei

Zoals je ziet groeit de tijd dat het kost om de beste oplossing door te rekenen harder dan het aantal steden. Dat heeft te maken met het aantal combinaties dat steeds veel groter wordt als je een locatie toe voegt. Hoe dat zit bekijken we nu.

**Opdracht 4. Aantal mogelijkheden**

Als je twee steden hebt, A en B, dan is een mogelijke route ABA (want de handelsreiziger moet ook weer terug naar zijn startpunt).

1. Teken een kaart met 3 steden: A, B en C. Hoeveel routes zijn er? Schrijf de routes op.

|  |
| --- |
| Antwoord:  2 steden A en B: 1 mogelijke route, namelijk ABA  3 steden A, B en C: 2 mogelijke routes (namelijk ABCA, ACBA) |

1. Doe hetzelfde voor 4 en 5 steden.

|  |
| --- |
| 4 steden A,B,C en D : 6 mogelijke routes (ABCDA, ABDCA, ACBDA, ACDBA, ADBCA, ADCBA)    5 steden zijn (5-1)! = 4! = 24 mogelijke routes |

1. Welke patronen herken je in de antwoorden bij de vorige vraag? Kan je bedenken hoeveel mogelijkheden er zijn voor 6 steden zonder ze allemaal op te schrijven?

|  |
| --- |
| Antwoord: (6-1)! =5! = 120 mogelijke routes |

1. Kijk even terug naar de vorige vraag waar je de mogelijkheden voor 6 steden moest bepalen. Waarom zou het niet handiger zijn geweest om alle mogelijkheden op te schrijven in plaats van deze te berekenen?

|  |
| --- |
| Antwoord: het is ondoenlijk om alle 120 mogelijkheden op te schrijven! |

1. Maak een algemene formule om het aantal mogelijke routes langs *n* steden uit te rekenen. Test je formule door een paar getallen in te vullen (kijk even terug naar vragen 1, 2 en 3).

|  |
| --- |
| Antwoord: Voor n steden zijn er (n-1)! mogelijke paden |

1. Gebruik jouw algemene formule om te berekenen hoeveel mogelijke routes er zijn om 20 steden te bezoeken.

|  |
| --- |
| Antwoord: Voor 20 steden zijn er (20-1)!= 19! = 1x1017 mogelijke routes:  100 000 000 000 000 000 |

1. Gebruik jouw algemene formule om te berekenen hoeveel mogelijke routes er zijn om 100 steden te bezoeken. Probeer dit met je rekenmachine uit te rekenen. Kun je uitleggen wat er gebeurt?

|  |
| --- |
| Antwoord: voor 100 steden zijn er 99! Mogelijke routes. Dat is zo’n groot getal dat een (grafische) rekenmachine deze niet eens kan uitrekenen.  Will je het toch uitrekenen? Dan kan dat hier: <http://csfieldguide.org.nz/en/interactives/big-number-calculator/index.html>  Het antwoord is: 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000 |

1. Voer 20 steden in op de kaart van <http://www.csfieldguide.org.nz/releases/1.9.9/_static/widgets/tract-tsp-basic-v2.html>. Hoe lang duurt het om de kortste route uit te rekenen? Waarom is dit geen praktische oplossing voor een handelsreiziger die een groot aantal steden moet bezoeken?

Antwoord: Meer dan 192,000,000 jaar! Dat is niet praktisch want dan is hij allang dood.

1. Met 1 computer duurt het dus wel erg lang om uit te rekenen. Hoe meer computers of processoren aan het probleem rekenen, hoe sneller een oplossing uitgerekend kan worden. We gaan kijken of we dan wel in een redelijke tijd het resultaat kunnen uitrekenen.

Op de volgende website kun je uitrekenen hoe lang het duurt als meerdere computers gelijktijdig aan de berekening werken: <http://www.csfieldguide.org.nz/releases/1.9.9/_static/widgets/tract-scaling-v2.html>

Voer je probleem met 20 steden in, dus N=20. Ga uit van 1,000,000 operaties per seconde, en algoritme ***N!***.

Hoelang duurt het om een oplossing te vinden met 1 processor? Druk je antwoord uit in jaren. En 10 processoren? Hoeveel processoren heb je nodig om binnen een dag tot een oplossing te komen? Hoeveel keer zo lang duurt het als je 1 stad toevoegt?

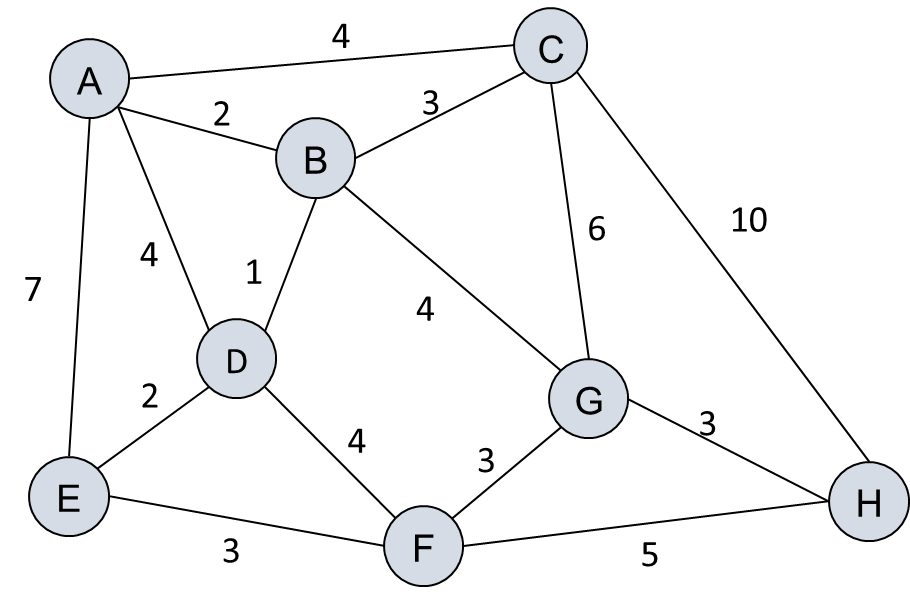
|  |
| --- |
| Antwoord: Voor N = 20, 1 000 000 operaties per seconde en 1 processor duurt het 77,3056 jaar. Met 10 processoren duurt het nog steeds erg lang: 7,731 jaar. Om binnen 1 dag een oplossing te vinden heb je 30 000 000 processoren nodig. Dat zijn er wel erg veel!  Na 1 stad toevoegen (bij N=21) duurt het 20 keer zo lang. |

|  |
| --- |
| **Waar gaat dit eigenlijk over?**  Voor een **praktisch onoplosbare probleem** bestaat er geen efficiënte algoritme.  **Complexiteit** is een belangrijk concept als je wilt kijken naar de efficiëntie van een algoritme. Hiermee kun je verschillende algoritmen voor eenzelfde problemen vergelijken. Is een bepaald algoritme sneller dan een ander? We kijken daarbij naar het aantal **stappen** dat nodig is om een probleem op te lossen, in plaats van de hoeveelheid “tijd”. Zo kunnen we namelijk redeneren over wat er gebeurt met het aantal stappen als we het probleem groter maken, bijvoorbeeld door het aantal steden te verdubbelen.  Een aantal voorbeelden van stappen die je kunt bekijken zijn:   * Voor de handelsreizigersprobleem dat je zojuist hebt onderzocht, zijn *(n-1)!* stappen nodig het bereken van een route langs *n* steden. * Als je door een stapel van *n* bladeren zoekt, van boven naar beneden, dan neemt je algoritme *n* stappen. Voor een stapel van 100 bladeren zal je hooguit 100 stappen nodig hebben om datgene te vinden dat je zocht. * Een algoritme dat elk paar waarden in een lijst met *n* waarden vergelijkt, zal *n\*(n-1)*vergelijkingen, en dus stappen, moeten maken.   Dit geeft ons veel informatie over hoe goed een algoritme is zonder in details te treden van op welke computer het draaide, welke taal en hoe goed het programma geschreven was. De complexiteit wordt dus gebruikt om de snelheid van algoritmes aan te geven en met elkaar te vergelijken. |

# Deel C: Heuristieken

Hoe lossen we zo’n praktisch onoplosbare probleem dan op? Er is geen efficiënte algoritme op de beste oplossing te vinden voor een NP-Compleet probleem. Maar je kunt wel op een efficiënte manier een suboptimale oplossing vinden. Zo’n suboptimale strategie heet een **heuristiek**. Het levert niet de beste resultaat op, maar geeft wel in aanzienbare tijd een oplossing en is daardoor wel praktisch. Je kunt het gebruiken in situaties waarin het brute force uitrekenen van alle mogelijkheden geen bruikbare aanpak is. We gaan nu kijken naar twee concrete voorbeelden van heuristieken om een optimale route voor het handelsreizigersprobleem te benaderen.

We bekijken daarbij de volgende probleem. Een postbezorger moet een route vinden langs elke locatie en weer terug naar zijn basisstation.



## Heuristiek 1: Beste-buur algoritme

**Stap 1**: Begin in een willekeurige stad.

**Stap 2**: Als je nog niet elke stad bezocht hebt, ga naar de dichtstbijzijnde nog niet bezochte stad.

**Stap 3**: Als alle steden bezocht zijn, ga dan terug naar het beginpunt.

**Opdracht 5. Evaluatie van je oplossing**

*Je hebt het beste-buur algoritme gebruikt. Wat was de oplossing?*

|  |
| --- |
| Antwoord: DBACGHFED. Afstand is daarmee 28. |

*Kun je zelf een betere oplossing vinden?*

|  |
| --- |
| Antwoord: DEFHGCABD. Afstand is dan 24. |

*Wat kan je zeggen over het beste-buur algoritme?*

|  |
| --- |
| Antwoord: Het beste-buur algoritme heeft wel tot een oplossing geleid. Het resultaat is niet per se optimaal. |

|  |
| --- |
| **Beste-buur algoritme**  Eén van de meest voor de hand liggende heuristieken begint bij een gegeven punt en gaat dan direct naar het dichtstbijzijnde volgende punt. Vanaf daar bepaal je de dichtstbijzijnde punt, waar je nog niet geweest bent. Dit herhaal je totdat je het eindpunt bereikt hebt.  Deze benadering staat bekend als het dichtstbijzijnde-buur algoritme (in het Engels "Nearest Neighbor") en is een voorbeeld van een **gretig** heuristisch algoritme. Deze kun je toepassen op meerdere situaties. In plaats van afstanden kun je ook andere maten gebruiken, zoals kosten of tijd. Vandaar de naam beste-buur algoritme.  Deze neemt altijd de beslissing die op dat moment het beste lijkt. Hoewel het in eerste instantie lijkt alsof deze algoritme **optimaal** is, laten we zien waarom dat niet het geval is. Bekijk de volgende graaf. De getallen zijn afstanden in kilometers.    Als je met de beste-buur algoritme van A naar C wilt, dan zal vanuit A eerst gekozen worden voor B, omdat 2 kleiner is dan 4. Vanuit B wordt dan gekozen worden voor C. Met de beste-buur algoritme is de route 5 km lang. Ze ziet zelf wel in dat de kortste pad van A naar C 4km is. |

## Heuristiek 2: Gebaseerd op minimale opspannende boom

**Stap 1**. Bepaal eerst de minimale opspannende boom met behulp van Kruskal’s gretige algoritme.

1. Maak een lijst van alle lengtes, van klein naar groot.
2. Schets een kaart met alle knopen (locaties).
3. Kies de kortste lengte, teken de verbinding op je kaart en streep de haal de lengte uit de lijst (streep die door).
4. Pak vervolgens weer de kortste lengte. Teken de verbinding op je kaart, zolang deze niet tot een gesloten circuit leidt. Haal de lengte uit de lijst, ook als je het niet ingetekend hebt. (het maakt niet uit of je hem hebt ingekleurd of niet).
5. Herhaal totdat er geen waardes meer in je lijst staan.
6. Bereken de totale lengte.

**Stap 2.** Maak van elke pad op je kaart twee eenrichtingspaden. Dus een pad van A naar B wordt nu twee lijnen met pijltjes: één van A naar B en één van B naar A. Bereken de totale lengte van de route. Bedenk dat je over de paden heen en weer moet gaan.

**Stap 3.** Zoek optimalisaties en bepaal dan een nieuwe totale lengte van een route. Als voorbeeld, in plaats van dat je langs dezelfde route teruggaat kun je er voor kiezen om een andere, kortere route te kiezen.

**Opdracht 6. Evaluatie van je oplossing**

Je hebt misschien niet de meest optimale route gevonden, maar vast wel een redelijke, en dat zonder dat je brute force alle mogelijkheden bent afgegaan.

*Je hebt het algoritme van Kruskal gebruikt. Wat weet je van deze algoritme en wat kun je de zeggen over je resultaat uit stap 1?*

|  |
| --- |
| Antwoord: het levert een optimale minimale opspannende boom op. |

*Als je naar je antwoord bij stap 2 bekijkt, hoe verhoud zich dat tot het antwoord uit stap 1?*

|  |
| --- |
| Antwoord:. Omdat de paden vervangen zijn door twee eenrichtingspaden is mijn oplossing bij b 2x resultaat uit minimaal opspannende boom algoritme. |

*Wat is het effect van wat je bij stap 3 hebt gedaan om de lengte van de pad? Wat kun je zeggen over je uiteindelijke resultaat voor het handelsreizigersprobleem?*

*Gebruik in je uitleg zinnen als:*

*“Het Kruskal algoritme levert altijd …. op.”*

*“Ik heb daarna …. gedaan”*

*“Dat betekend dat mijn oplossing ….. in vergelijking met het resultaat van Kruskal.”*

|  |
| --- |
| Antwoord: Kruskal levert een optimale oplossing voor de minimaal opspannende boom. In stap 2 is dat verdubbeld. Omdat ik daarna nog een verbeterslag heb gemaakt is mijn uiteindelijke oplossing **hooguit** 2x resultaat uit minimaal opspannende boom algoritme. |

|  |
| --- |
| **Waar gaat dit eigenlijk over?**  Een onhandelbaar probleem is een probleem waarvoor geen efficiënte algoritme bestaat. Het handelsreizigersprobleem is hier een voorbeeld van. Zo’n probleem noemen we **NP-Compleet**. Je kunt met **brute force** alle mogelijkheden af gaan. Dit geeft je wel altijd de beste oplossing. Maar omdat de hoeveelheid werk om de beste oplossing te vinden enorm toeneemt als de hoeveelheid gegevens (aantal steden) stijgt, duurt het erg lang om deze uit te rekenen, zelfs voor de snelste computers. Hierdoor wordt het onpraktisch dit te gebruiken.  Onderzoekers hebben veel tijd besteed aan het zoeken naar efficiënte oplossingen voor dit soort problemen, maar hebben nog geen algoritme gevonden om dit binnen redelijke tijd op te lossen.  Om toch binnen een redelijke tijd tot een oplossing te komen worden heuristieken toegepast. Dit zijn algoritmen dit een optimale resultaat benaderen. Het levert dus waarschijnlijk niet de beste oplossing op, maar doet dit wel in een redelijke tijd. Je kunt het gebruiken in situaties waarin het brute force uitrekenen van alle mogelijkheden geen bruikbare aanpak is.  **Waarom is het handig om dit te weten?**  Dit is een heel praktisch probleem dat in de echte wereld vaak voorkomt. Een koeriersdienst zoals TPG die pakketten naar verschillende adressen bezorgt wil natuurlijk zo snel mogelijk alle pakketten bezorgen. Het zelfde geld voor een frisdrankbedrijf die hun groot aantal blikjesautomaten opnieuw wil bevoorraden. Of een rockband die in verschillende steden een concert wil geven. Maar je hoeft niet altijd met ‘tijd’ te rekenen. Een vrachtwagen van de Kruidvat dat de winkels wil bevoorraden wil natuurlijk zo min mogelijk brandstof verbruiken om de kosten laag te houden. Dit zijn nog steeds voorbeelden van het handelsreizigersprobleem, weliswaar in een ander jasje. Het kan ook voorkomen bij problemen die helemaal geen betrekking hebben op reizen, bijvoorbeeld een efficiënte manier te vinden om DNA te maken.  Veel bedrijven zijn bereid om geld te investeren in het vinden van betere oplossingen voor hun eigen variant van het handelsreizigersprobleem (en de nauw verwante problemen die zich voordoen wanneer extra beperkingen zoals snelheidsbeperkingen en wegblokkades worden toegevoegd). |

**Opdracht 7. Reflectie**

Schrijf een conclusie waarin je in je eigen worden verteld wat je bij delen A, B en C gedaan hebt en wat iemand aan het resultaat heeft. Leg uit welke algoritmen en heuristieken je gebruikt hebt.

1. Bron: [www.csfieldguide.org.nz](http://www.csfieldguide.org.nz) [↑](#footnote-ref-1)