Zoekalgoritmes

Classic Computer Science Algorithms

Stijn Lievens Pieter-Jan Maenhaut Koen Mertens Lieven Smits

AJ 2021–2022

HO GENT

Zoekalgoritmes

HO
GENT

Inhoud

HO GENT

De omgeving

- eenpersoons
- compleet observeerbaar
- deterministisch
- statisch
- discreet

Agent bevindt zich in bepaalde beginpositie.

Moet toestand bereiken waar één of andere voorwaarde voldaan is.

Agent kan de acties op voorhand bepalen!

HO GENT

Definitie Zoekprobleem

Een ZOEKPROBLEEM bestaat uit de volgende elementen:

- Een TOESTANDSRUIMTE S die alle mogelijke toestanden bevat.
- Een verzameling van mogelijke acties A.
- Een TRANSITIEMODEL:

$$T : (S, A) \to S : (s, a) \mapsto s'.$$

Hierbij wordt s' een *opvolger* van s genoemd.
Het uitvoeren van een actie op een bepaalde toestand heeft meestal een bepaalde KOST:

$$c : (S, A, S) \to \mathbb{R} : (s, a, s') \mapsto c(s, a, s').$$

Opmerking: deze definitie van kost kan ook gebruikt worden in stochastische omgevingen.

HO GENT

Definitie Zoekprobleem (Vervolg)

- Een initiële toestand $s_0 \in S$; dit is de toestand van waaruit het zoeken zal vertrekken.
- een DOELTEST. Dit is een functie die voor elke toestand s aangeeft of het doel bereikt is of niet. Een toestand waarvoor de doeltest voldaan is noemen we een DOELTOESTAND.

HO GENT

Toestandsruimtegraaf

- In de toestandsruimtegraaf stellen de knopen de toestanden voor.
- Elke toestand komt m.a.w. juist één keer voor.
- Twee knopen zijn adjacent als de ene toestand de opvolger is van de andere toestand.
- De gewichten van de bogen = de kost van de acties.
- Opletten: toestandsruimtegraaf wordt snel zeer groot!

HO GENT

De 8-puzzel

HO
GENT

Wat zijn hier de componenten?

1. S?
2. Acties?
3. Transitiemodel?
4. Initiële toestand?
5. Doeltest?

HO GENT

De 8-puzzel

De 8-puzzel is een typisch voorbeeld van een zoekprobleem:

- Toestandsruimte = alle mogelijke configuraties.
- Acties: nl. `Boven`, `Onder`, `Links` en `Rechts`.
  - Beschreven in termen van lege vakje.
- Transitiemodel: zoals in de fysica van het probleem.
  - Actie `Rechts` wisselt `8` en blanco van plaats om in linkse puzzel op voorgaande slide.
- De kost van elke actie is één.
- Doeltest: verifiëren dat bepaalde vastgelegde configuratie (bv. deze rechts op de figuur) werd bereikt.

HO GENT

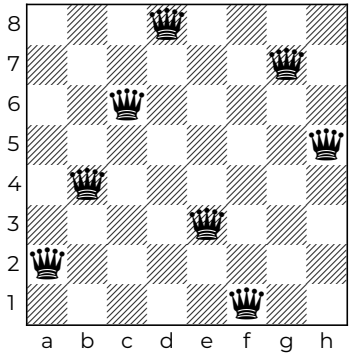
8/15/24/...-puzzel

- Aantal toestanden wordt snel zeer groot wanneer men grotere puzzels beschouwt.
- 8-puzzel heeft $9! = 362880$ toestanden. Doenbaar.
- 15-puzzel heeft $16! \approx 2.1 \times 10^{13}$ toestanden. Niet doenbaar.
- 24-puzzel heeft $25! \approx 1.6 \times 10^{25}$ toestanden. Niet doenbaar.

Voor grotere puzzels is het zeker niet mogelijk om de volledige toestandsruimtegraaf op te bouwen.

HO GENT

8-koninginnenprobleem



Een mogelijke oplossing van het 8-koninginnenprobleem.

HO
GENT

Eerste formulering als zoekprobleem

- S bevat alle configuraties met hoogstens 8 koninginnen
- Transitiemodel: voeg koningin toe aan leeg vakje wanneer minder dan 8 koninginnen op het bord.
- Kost van acties is irrelevant en wordt $= 0$ genomen.
- Initiële toestand = leeg bord
- Doeltest: controleer dat er 8 koninginnen op het bord staan en dat er geen paar elkaar aanvalt.
  - Dit is dus *niet* zomaar een opsomming van doeltoestanden!

HO GENT

Vraag

Hoeveel toestanden kan je beschouwen met deze formulering?

HO GENT

Vraag

Hoeveel toestanden kan je beschouwen met deze formulering?

$$\binom{64}{0} + \binom{64}{1} + \cdots + \binom{64}{8} \approx 5.13 \times 10^9,$$

wat met 8 bytes per toestand reeds 40 GB aan hoofdgeheugen vraagt!

HO GENT

Tweede formulering als zoekprobleem

- Toestandsruimte bevat configuraties met n koninginnen op het bord, één in elk van de n eerste kolommen. Geen paar koninginnen mag elkaar aanvallen.
- Acties: voeg koningin toe aan eerste vrije kolom op een niet bedreigd vakje.
- Transitiemodel: zoals men verwacht.
- Kost van acties: nog steeds nul.
- Initiële toestand en doeltest zoals voorheen.

Aantal toestanden is nu 2057, dus vinden oplossing is triviaal.

HO GENT

8-koninginnenprobleem: Moraal van het verhaal

Probleemformulering

De formulering van de toestanden en acties kan een grote invloed hebben op de mogelijkheid om al dan niet een oplossing te vinden.

HO GENT

Routeprobleem

In een routeprobleem wil men een weg vinden tussen twee van n steden.
We formuleren een zoekprobleem:

- Er zijn n toestanden, de toestand i betekent “in stad i”.
- Beschikbare actie: `RijdNaar` actie.
- Wanneer men in toestand i de `RijdNaar`(j) actie onderneemt, dan is de volgende toestand die waarin men zich in locatie j bevindt. Dit is het transitiemodel.
- Kost: bv. de afstand in km zijn tussen i en j, of de gemiddelde reistijd of het gemiddeld benzineverbruik, **...**
- De initiële toestand is een willekeurige stad.
- De doeltest bestaat uit verifiëren dat men in de gewenste stad is aangekomen.

HO GENT

Rondreisprobleem

Elke stad minstens éénmaal bezoeken en terugkeren naar de eerste stad.
Toestanden: ook bijhouden *waar men geweest is*!

- Toestand: huidige stad en bezochte steden: $n \times 2^n$ toestanden, dus **exponentieel** in aantal steden.
- Acties: als voorheen.
- Transitiemodel: niet enkel huidige stad aanpassen maar ook verzameling bezochte steden.
- Starttoestand: we zijn in een bepaalde stad en geen enkele stad werd reeds bezocht.
- Doeltest: terug in startstad en alle steden bezocht.

HO GENT

Rondreisprobleem

Handelsreizigersprobleem is een speciaal geval van rondreisprobleem: elke stad moet juist éénmaal worden bezocht.

HO GENT

Oplossing Zoekprobleem

Definitie

*Een OPLOSSING VAN ZOEKPROBLEEM bestaat uit een **sequentie van acties** zodanig dat startend vanuit de initiële toestand een doeltoestand wordt bereikt.*
De KOST van een oplossing is de som van de kosten van de individuele acties.
Een OPTIMALE OPLOSSING is een oplossing waarvoor de kost minimaal is onder alle mogelijke oplossingen.

HO GENT

NP-complete problemen

- Oplossen van een $n^2 - 1$-puzzel en handelsreizigersprobleem zijn NP-complete problemen.
- Kunnen geformuleerd worden als zoekproblemen.
- We kunnen *niet* verwachten dat de oplossingsmethodes binnen de AI algoritmes zullen opleveren die alle instanties efficiënt kunnen oplossen!

HO GENT

Voorbeeld oplossing

7	2	4
5		6
8	3	1

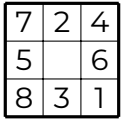
	1	2
3	4	5
6	7	8

Optimale oplossing heeft kost 26:

L, B, R, 0, R, 0, L, L, B, R, R, 0, L, L, B, R, R, B, L, L, 0, R, R, B, L, L

HO GENT

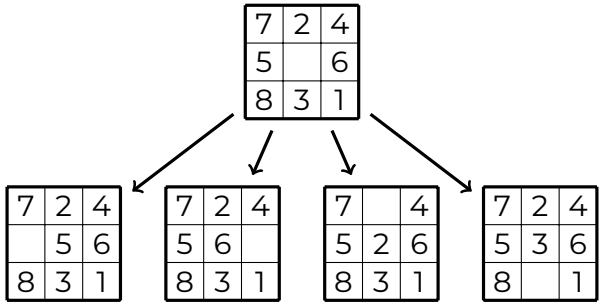
Boomgebaseerd Zoeken: Voorbeeld 8-puzzel



- Initieel bevat de *open lijst* enkel de begintoestand.
- Kies (en verwijder) een element van de open lijst en *expandeer*.
  - Expanderen = genereren en toevoegen van alle opvolgers aan de open lijst.

HO GENT

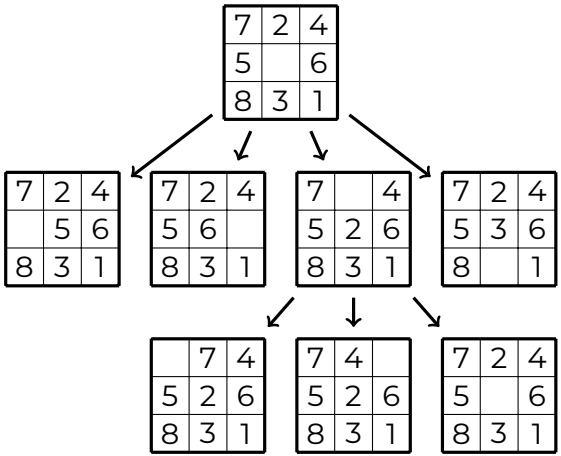
Boomgebaseerd Zoeken: voorbeeld 8-puzzel



- Open lijst bevat nu 4 elementen.
- De puzzel met de blanco bovenaan wordt gekozen voor expansie.

HO GENT

Boomgebaseerd Zoeken: Voorbeeld 8-puzzel



- 6 plannen op open lijst
- Zelfde toestand (begintoestand) komt reeds tweemaal voor.

HO GENT

Boomgebaseerd Zoeken

Proces van kiezen uit open lijst en expanderen herhaalt zich tot

- Een doeltoestand wordt gekozen voor expansie;
- *of* de open lijst leeg is.

HO GENT

Boomgebaseerd Zoeken: Pseudocode

Invoer Een zoekprobleem P.
Uitvoer Een sequentie van acties die een oplossing is van het zoekprobleem of error wanneer er geen oplossing werd gevonden.

```
1:  function TreeSearch(P)
2:      f ← nieuwe lege lijst                                  # De open lijst
3:      f.add(nieuw plan gebaseerd op initiële toestand P)
4:      while f ≠ ∅ do
5:          c ← f.chooseAndRemovePlan                          # Kies volgend plan
6:          if P.goalTest(c.getState) = true then
7:              return getActionSeq(c)
8:          else
9:              for (s, a) ∈ c.getState.getSuccessors do
10:                 f.Add(nieuw plan gebaseerd op (s, a) en c)
11:     return error: geen oplossing gevonden                  # Open lijst is leeg.
12: end function
```

Implementatie van een Plan

Een plan kan geïmplementeerd worden als een klasse met 4 velden:

- De huidige toestand.
- De laatst gekozen actie a.
- De *voorganger* of *ouder* van dit plan. Een referentie naar het plan waarvan dit plan is afgeleid door het toepassen van de huidige actie a.
- De totale kost van dit plan. Traditioneel wordt deze kost met g genoteerd.
  - Opmerking: in principe kan g ook berekend worden, maar omdat g door sommige algoritmes vaak wordt gebruikt wordt deze hier opgeslaan.

HO GENT

Criteria voor Zoekalgoritmen

Definitie

Een zoekalgoritme is COMPLEET wanneer het algoritme, voor elk zoekprobleem met een oplossing, effectief een oplossing vindt.

Definitie

Een zoekalgoritme is OPTIMAAL wanneer het niet enkel een oplossing vindt maar steeds een optimale oplossing teruggeeft voor elk zoekprobleem met een oplossing.

HO GENT

Criteria voor Zoekalgoritmen

Definitie

De TIJDSCOMPLEXITEIT van een zoekalgoritme bepaalt de uitvoeringstijd van het algoritme. We nemen aan dat de uitvoeringstijd evenredig is met het aantal gegenereerde toppen.

Definitie

De RUIMTECOMPLEXITEIT van een zoekalgoritme bepaalt de hoeveelheid geheugen die het algoritme nodig heeft tijdens de uitvoering. Dit wordt meestal uitgedrukt als het maximaal aantal toestanden dat gelijktijdig moet worden bijgehouden.

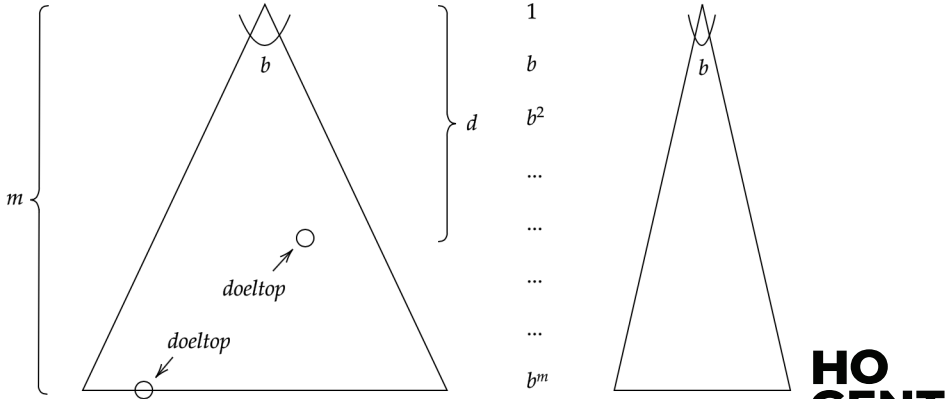
HO GENT

Grootheden voor Ruimte- en Tijdscomplexiteit

- De VERTAKKINGSFACTOR b: Deze geeft het maximaal aantal opvolgers van een top in de zoekboom.
- d: diepte van de meest ondiepe doeltop.
- m: maximale lengte (= aantal acties) van een pad in de toestandsruimte.

HO GENT

Grootheden voor Ruimte- en Tijdscomplexiteit



HO GENT

Eigenschappen Zoekboom

Eigenschap

Het aantal toppen in een zoekboom met vertakkingsfactor b en maximale diepte m wordt gegeven door

$$\frac{b^{m+1} - 1}{b - 1} = O(b^m).$$

Bovendien: de laatste laag bevat b^m toppen. Dat is méér dan alle voorgaande lagen samen!

HO GENT

Graafgebaseerd Zoeken: Herhaalde Toestanden

Boomgebaseerd zoeken onthoudt niet waar het reeds geweest is.

			3			
		3	2	3		
	3	2	1	2	3	
3	2	1	0	1	2	3
	3	2	1	2	3	
		3	2	3		
			3			

d	aantal toestanden op afstand d	aantal toppen in zoekboom op diepte d
0	1	1
1	4	4
2	8	16
3	12	64
...	...	...
d	$4d$	4^d

Lineair vs. exponentieel!

HO GENT

Graafgebaseerd zoeken

Basisidee Graafgebaseerd Zoeken

Expandeer elke *toestand* hoogstens éénmaal.

Hoe? Houd een lijst (verzameling) bij van geëxpandeerde toestanden: de GESLOTEN LIJST.

HO GENT

Pseudocode

Invoer Een zoekprobleem P.
Uitvoer Een sequentie van acties of error

```
1:  function GraphSearch(P)
2:      f ← nieuwe lege lijst                                  # De open lijst
3:      closed ← ∅                          # Verzameling geëxpandeerde toestanden
4:      f.add(nieuw plan gebaseerd op initiële toestand P)
5:      while f ≠ ∅ do
6:          c ← f.chooseAndRemovePlan                          # Kies het volgende plan
7:          if P.goalTest(c.getState) = true then
8:              return getActionSeq(c)
9:          else
10:             if c.getState ∉ closed then
11:                 closed ← closed ∪ c.getState
12:                 for (s, a) ∈ c.getState.getSuccessors do
13:                     f.Add(nieuw plan gebaseerd op (s, a) en c)
14:     return error: geen oplossing gevonden
15: end function
```

HO GENT

Blinde Zoekmethoden

Blinde zoekmethoden: kunnen enkel gebruikmaken van de informatie die verschaft wordt door de definitie van het zoekprobleem.
De blinde zoekmethoden die we zien zijn:

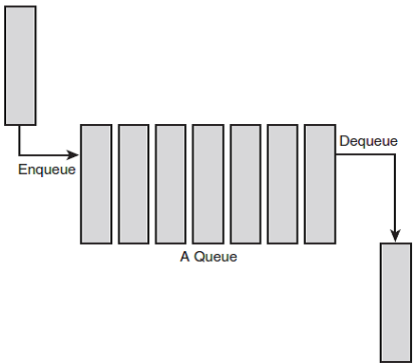
- Breedte Eerst Zoeken
- Diepte Eerst Zoeken
- Diepte Gelimiteerd Zoeken/Iteratief Verdiepen
- Uniforme Kost Zoeken

Veel hiervan is zeer gelijkaardig aan de algoritmes die we reeds zagen in het hoofdstuk m.b.t. grafen.

HO GENT

Breedte Eerst Zoeken

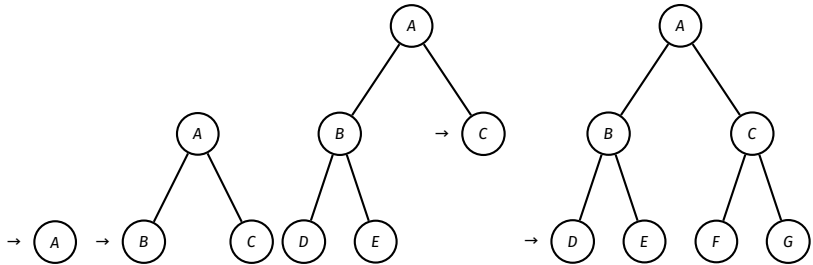
- De open lijst is een *wachtrij*, i.e. een FIFO structuur.
- De zoekboom wordt systematisch laag per laag opgebouwd.



HO GENT

Illustratie Breedte Eerst op Binaire Boom

De doeltoestand is ***D***.



Eigenschappen Breedte Eerst

- Compleet?

HO
GENT

Eigenschappen Breedte Eerst

- Compleet? Ja, want laag per laag.
- Optimaal?

HO
GENT

Eigenschappen Breedte Eerst

- Compleet? Ja, want laag per laag.
- Optimaal? Vindt steeds oplossing met minimaal aantal acties. Is niet noodzakelijk de laagste kost wanneer acties verschillende kosten hebben.
- Tijdscomplexiteit:

HO GENT

Eigenschappen Breedte Eerst

- Compleet? Ja, want laag per laag.
- Optimaal? Vindt steeds oplossing met minimaal aantal acties. Is niet noodzakelijk de laagste kost wanneer acties verschillende kosten hebben.
- Tijdscomplexiteit: exponentieel: $O(b^{d+1})$.
- Ruimtecomplexiteit:

HO GENT

Eigenschappen Breedte Eerst

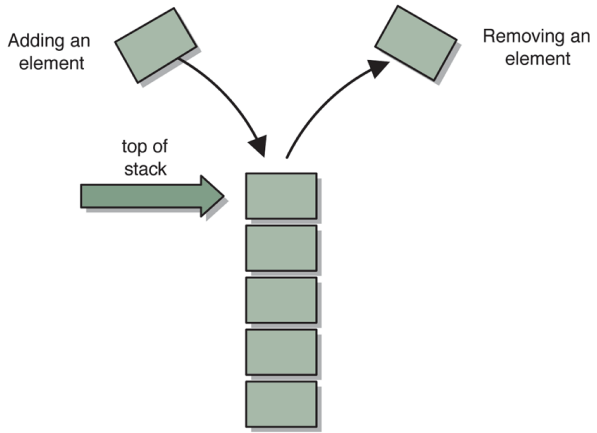
- Compleet? Ja, want laag per laag.
- Optimaal? Vindt steeds oplossing met minimaal aantal acties. Is niet noodzakelijk de laagste kost wanneer acties verschillende kosten hebben.
- Tijdscomplexiteit: exponentieel: $O(b^{d+1})$.
- Ruimtecomplexiteit: volledige laag op diepte $d + 1$: $O(b^{d+1})$.

Graafgebaseerd zoeken kan veel tijd winnen bij herhaalde toestanden.
Breedte eerst meestal in graafgebaseerde versie.

HO GENT

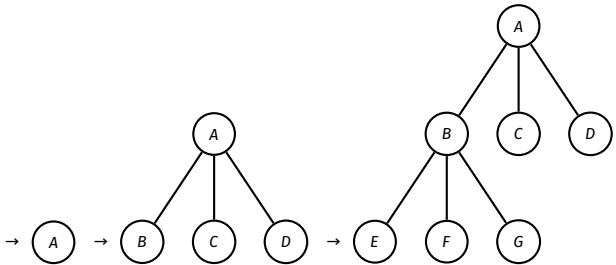
Diepte Eerst Zoeken

- Duaal aan breedte eerst: gebruikt LIFO structuur voor open lijst.



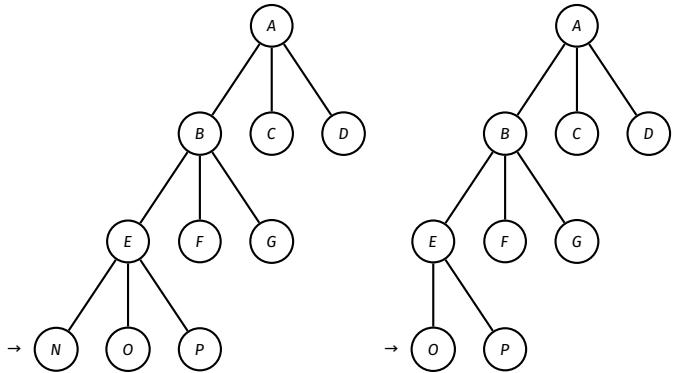
HO GENT

Diepte Eerst Zoeken



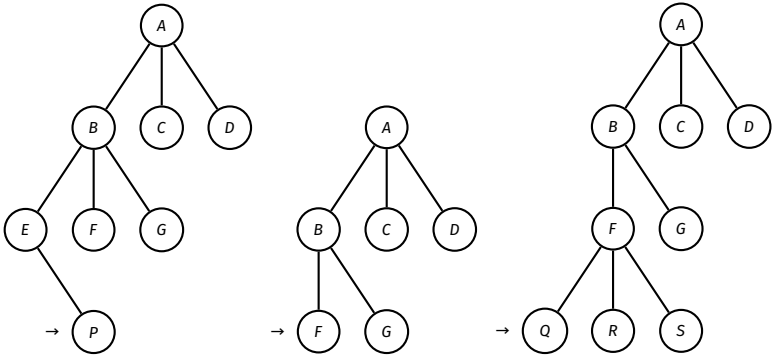
HO GENT

Diepte Eerst Zoeken



HO GENT

Diepte Eerst Zoeken



HO GENT

Diepte Eerst Zoeken

- Compleet?

HO
GENT

Diepte Eerst Zoeken

- Compleet? Neen: kan vastraken in oneindige lus; kan oneindig lange actiesequenties nemen.
- Optimaal?

HO GENT

Diepte Eerst Zoeken

- Compleet? Neen: kan vastraken in oneindige lus; kan oneindig lange actiesequenties nemen.
- Optimaal? Neen. Vindt steeds “meest linkse” doelknoop.
- Tijdscomplexiteit:

HO GENT

Diepte Eerst Zoeken

- Compleet? Neen: kan vastraken in oneindige lus; kan oneindig lange actiesequenties nemen.
- Optimaal? Neen. Vindt steeds “meest linkse” doelknoop.
- Tijdscomplexiteit: exponentieel, nl. $O(b^m)$ in slechtste geval.
- Ruimtecomplexiteit:

HO GENT

Diepte Eerst Zoeken

- Compleet? Neen: kan vastraken in oneindige lus; kan oneindig lange actiesequenties nemen.
- Optimaal? Neen. Vindt steeds “meest linkse” doelknoop.
- Tijdscomplexiteit: exponentieel, nl. $O(b^m)$ in slechtste geval.
- Ruimtecomplexiteit: **lineair**: $O(b \cdot m)$.

Diepte eerst wordt normaalgezien uitgevoerd in boomgebaseerde vorm.

HO GENT

Iteratief Verdiepen

- DIEPTE GELIMITEERD ZOEKEN: breek zoekproces diepte eerst af als vooraf bepaalde diepte d werd bereikt.
  - Doe alsof toppen op diepte d *geen opvolgers* hebben.
- Vaak weet men geen goede grens voor d.
  - Oplossing: d systematisch verhogen. Dit is ITERATIEF VERDIEPEN.

HO GENT

Iteratief Verdiepen

Invoer Een zoekprobleem P
Uitvoer Een sequentie van acties wanneer een oplossing werd gevonden of “error” wanneer er geen oplossing werd gevonden.

```
1: function IterativeDeepening(P)
2:     d ← 0
3:     sol ← DepthLimitedSearch(P, d)
4:     while sol = “hit boundary” do
5:         d ← d + 1
6:         sol ← DepthLimitedSearch(P, d)
7:     return sol
8: end function
```

Oplossing of error

HO GENT

Diepte Gelimiteerd Zoeken

Invoer Een zoekprobleem P, een maximale diepte d.
Uitvoer Een sequentie van acties wanneer een oplossing werd gevonden met diepte d of minder; een “hit boundary” conditie wanneer tijdens het zoekproces de maximale diepte werd bereikt of “error” wanneer er geen oplossing werd gevonden.

```
1: function DepthLimitedSearch(P, d)
2:     c ← nieuw plan gebaseerd op initiële toestand P
3:     return DlsRecursive(c, P, d)
4: end function
```

HO GENT

Diepte Gelimiteerd Zoeken

Invoer Een huidig plan c, een zoekprobleem P en een maximale diepte d.

Uitvoer Een sequentie van acties wanneer een oplossing werd gevonden met diepte d of minder startend vanaf het huidig plan; een “hit boundary” conditie wanneer tijdens het zoekproces de maximale diepte werd bereikt of “error” wanneer er geen oplossing werd gevonden.

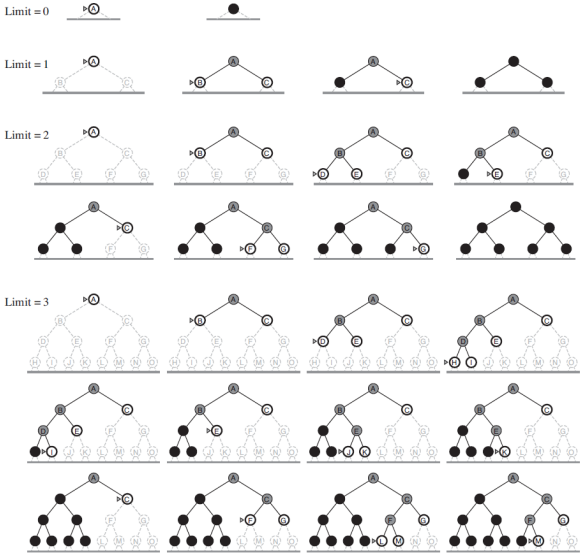
HO GENT

Diepte Gelimiteerd Zoeken

```
1:  function DlsRecursive(c, P, d)
2:      if P.goalTest(c.getState) = true then
3:          return getActionSeq(c)                          # Oplossing gevonden
4:      if d = 0 then
5:          return "hit boundary"                           # Grens bereikt
6:      boundaryHit ← false                  # Grens bereikt in één van de rec. oproepen?
7:      for (s, a) ∈ c.getState.getSuccessors do
8:          child ← nieuw plan gebaseerd op (s, a) en c
9:          sol ← DlsRecursive(child, P, d – 1)             # Recursieve oproep
10:         if sol = "hit boundary" then
11:             boundaryHit ← true
12:         else
13:             if sol ≠ "error: geen oplossing gevonden" then
14:                 return sol                              # Effectieve oplossing gevonden
15:     if boundaryHit = true then
16:         return "hit boundary"
17:     else
18:         return "error: geen oplossing gevonden"
19: end function
```

HO GENT

Illustratie Iteratief Verdiepen



HO GENT

Iteratief Verdiepen: Hoeveelheid “Overbodig” Werk?

Omdat de laatste laag het meeste aantal toppen bevat blijft de hoeveelheid werk dat “te veel” wordt gedaan binnen de perken.
Cijfervoorbeeld: voor $d = 10$ en $b = 4$.
Aantal toppen gegenereerd in laatste iteratie:

$$\frac{4^{11} - 1}{4 - 1} = 1\,398\,101.$$

Aantal toppen gegenereerd in alle *voorgaande* iteraties

$$\sum_{i=0}^{9} \frac{4^{i+1} - 1}{4 - 1} = 466\,030.$$

HO GENT

Eigenschappen Iteratief Verdiepen

- Compleet? Ja.
- Optimaal? Steeds meest ondiepe oplossing, dus niet optimaal als acties een verschillende kost kunnen hebben.
- Tijdscomplexiteit? Exponentieel $O(b^d)$
- Ruimtecomplexiteit? **lineair**!

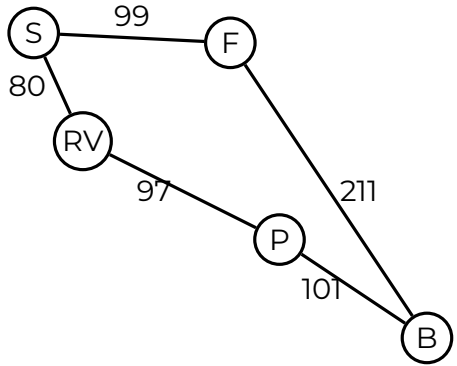
HO GENT

Uniforme Kost Zoeken

- Implementeer open lijst als prioriteitswachtrij.
  - Lagere padkost = hogere prioriteit.
  - Dus, plan met laagste g-waarde komt als volgende aan de beurt. Zelfde idee als algoritme van Dijkstra.
- Plaats doeltest in het algoritme is hier zeer belangrijk!

HO GENT

Uniforme Kost Zoeken: Belang Plaats Doeltest



Pas graafgebaseerd uniforme kost zoeken toe om van *S* naar *B* te gaan.

HO GENT

Geïnformeerde Zoekmethoden

Basisidee

Gebruik **domeinkennis** (heuristieken) om het zoeken efficiënter te laten verlopen.

HO GENT

Heuristieken

Definitie

Een HEURISTIEK h *is een afbeelding van de verzameling toestanden* S *naar de verzameling van niet-negatieve reële getallen* $\mathbb{R}^+$, *i.e.*

$$h : S \to \mathbb{R}^+ : s \mapsto h(s).$$

Opmerking: een heuristiek heeft in de context van zoekalgoritmes dus een specifieke betekenis.

HO GENT

Eigenschappen van Goede Heuristiek

- Een goede heuristiek geeft een goede indicatie voor de werkelijke optimale kost naar het doel.
- Een heuristiek moet ook *snel* te berekenen zijn.

HO GENT

Heuristieken voor 8-Puzzel

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

Hoeveel acties zijn deze puzzels van elkaar verwijderd?

HO GENT

Heuristieken voor 8-Puzzel

h_1 = aantal niet-lege vakjes dat niet op zijn juiste plaats staat

of ook

$$h_2 = \sum_{i=1}^{8} (\text{Manhattan afstand van vakje } i \text{ tot zijn correcte plaats}).$$

Wat is de waarde van h_1 en h_2 voor de zonet gegeven puzzels?

HO GENT

Toelaatbare Heuristieken

Definitie

Een heuristiek $h : S \to \mathbb{R}^+$ is TOELAATBAAR als voor elke toestand s geldt dat $h(s) \leq C^*(s)$ waarbij C^* de kost van een optimale oplossing voorstelt van s naar een doeltoestand.

Eigenschap

Wanneer de heuristiek h toelaatbaar is, dan is $h(g) = 0$ voor elke doeltoestand g.

HO GENT

Consistente Heuristieken

Definitie

Een heuristiek $h : S \to \mathbb{R}^+$ *is* CONSISTENT *als voor elke doeltoestand* g *geldt dat* $h(g) = 0$ *en als bovendien voor elke toestand* s *en elke actie* a *op* s *met* $s' = T(s, a)$ *geldt dat*

$$h(s) \leq c(s, a, s') + h(s').$$

HO GENT

Verband Toelaatbaarheid en Consistentie

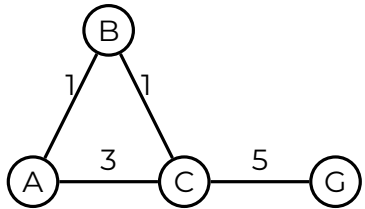
Eigenschap

Als een heuristiek consistent is, dan is ze ook onmiddellijk toelaatbaar.

Omgekeerd is dit niet waar! Er bestaan toelaatbare heuristieken die niet consistent zijn.

HO GENT

Voorbeeld Toelaatbare maar Inconsistente Heuristiek



s	G	A	B	C
$h(s)$	0	4	6	1

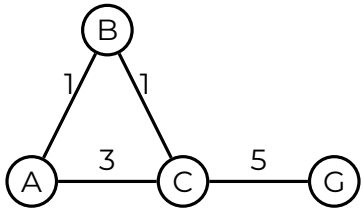
HO GENT

Gulzig Beste Eerst

Open lijst is prioriteitswachtrij en kleinere waarde voor heuristiek h betekent een grotere prioriteit.

HO GENT

Gulzig Beste Eerst is niet Optimaal

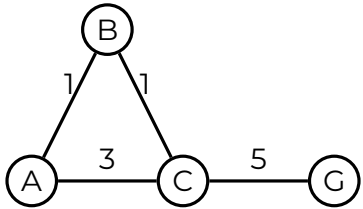


s	G	A	B	C
$h(s)$	0	7	6	5

Ga van A naar G m.b.v. het gulzig beste eerst algoritme.

HO GENT

Gulzig Beste Eerst is Niet Compleet



s	G	A	B	C
$h(s)$	0	3	4	5

Ga van A naar G met boomgebaseerd gulzig beste eerst.
Wat gebeurt er?

HO GENT

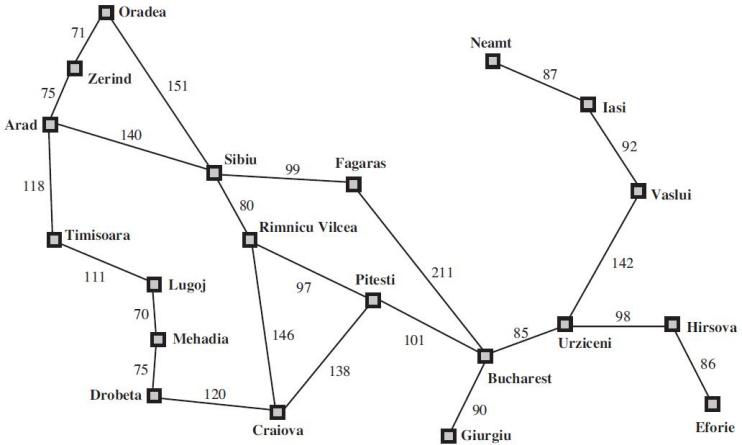
A*-Zoeken

- Uniforme Kost Zoeken: houdt enkel rekening met afgelegde weg g
- Gulzig Beste Eerst: houdt enkel rekening met heuristiek h.

Bij A*-zoeken worden beide gecombineerd. Het plan met de laagste waarde voor $f = g + h$ wordt als eerste geëxpandeerd.

HO GENT

A* met Toelaatbare Heuristiek



Ga van Arad naar Bucharest. Gebruik de afstand in vogelvlucht tot Bucharest als heuristiek.

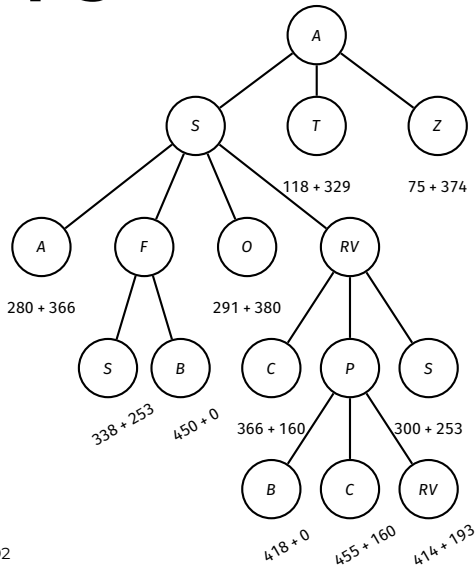
HO GENT

Afstand in Vogelvlucht tot Bucharest

Stad	Afstand	Stad	Afstand
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

HO GENT

Opgebouwde Zoekboom

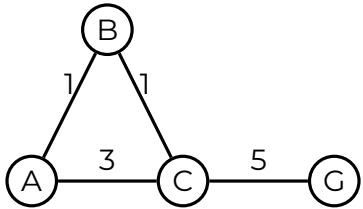


Geëxpandeerde plannen:

- $(A, 0 + 366)$
- $(S \rightarrow A, 140 + 253 = 393)$
- $(RV \rightarrow S \rightarrow A, 220 + 193 = 413)$
- $(F \rightarrow S \rightarrow A, 239 + 176 = 415)$
- $(P \rightarrow RV \rightarrow S \rightarrow A, 317 + 100 = 417)$
- $(B \rightarrow P \rightarrow RV \rightarrow S \rightarrow A, 418 + 0 = 418)$ (doeltest geslaagd)

HO GENT

A* met een Ontoelaatbare Heuristiek



s	G	A	B	C
$h(s)$	0	4	8	1

Ga van A naar G m.b.v. A*. Wat merk je op?

HO GENT

Boomgebaseerde A* met een Toelaatbare Heuristiek is Optimaal

Eigenschap

Wanneer boomgebaseerde A gebruikmaakt van een toelaatbare heuristiek* h *en wanneer alle acties een kost hebben groter of gelijk aan een zekere strikt positieve* ϵ*, dan is A* compleet en optimaal, i.e. dan vindt het algoritme steeds een optimale oplossing wanneer die bestaat.*

HO GENT

Kern van het Bewijs

1. Er is steeds een uitbreidbaar plan.
2. De f-waarde van zo'n plan p is hoogstens de optimale kost vanaf de starttoestand:

$$
\begin{aligned}
f(p) &= g(p) + h(p) \\
&\leq g(p) + C^*(p) && \text{toelaatbaardheid } h \\
&= C^*(s_0). && s_0 \text{ is de starttoestand}
\end{aligned}
$$

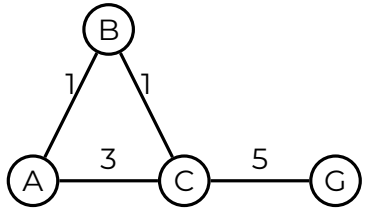
3. Een plan n met een suboptimaal pad naar een doeltoestand wordt nooit gekozen:

$$
\begin{aligned}
f(n) &= g(n) + h(n) \\
&= g(n) && n \text{ is doeltoestand en } h \text{ is toelaatbaar} \\
&> C^*(s_0) && \text{want } n \text{ is suboptimaal} \\
&\geq f(p) && \text{zie voorgaande berekening}
\end{aligned}
$$

4. Het algoritme eindigt want aantal plannen met kost $\leq C^*(s_0)$ is eindig.

HO GENT

Toelaatbaarheid *Niet* Voldoende bij Graafgebaseerde A*



s	G	A	B	C
$h(s)$	0	4	6	1

Voer graafgebaseerd A* uit om van A naar G te gaan.

Eerste geëxpandeerde pad naar C is suboptimaal. Wegens graafgebaseerd zoeken wordt het optimale pad niet verder uitgewerkt.

HO GENT

Consistentie Garandeert Optimaliteit bij Graafgebaseerde A*

Eigenschap

Wanneer graafgebaseerde A gebruikmaakt van een consistente heuristiek* h *en wanneer alle acties een kost hebben groter of gelijk aan een zekere strikt positieve* ϵ*, dan is A* compleet en optimaal, i.e. dan vindt het algoritme steeds een optimale oplossing wanneer die bestaat.*

HO GENT

A*-Zoeken: Conclusie

- Onder “milde” restricties op heuristiek is A* compleet en optimaal.
- Tijds- en ruimtecomplexiteit kan in slechtste geval exponentieel zijn.
  - In het algemeen sneller tekort aan geheugen dan aan tijd.
- Welke toppen worden geëxpandeerd? Alle toppen met

$$f(n) = g(n) + h(n) < C^*(s_0).$$

Geen enkele top met $f(n) > C^*(s_0)$ wordt geëxpandeerd. Toppen met $f(n) = C^*(s_0)$: sommige wel, sommige niet.

  - Hoe beter (groter) de heuristiek hoe minder toppen worden geëxpandeerd, maar let op ….

HO GENT

Ontwerpen van Heuristieken

Een goede heuristiek kan positieve invloed hebben op tijds- en ruimtecomplexiteit.

We bekijken twee manieren om heuristieken te ontwerpen:

1. Gebruik van vereenvoudigde problemen.
2. Gebruik van patroondatabanken.

HO GENT

Vereenvoudigde Problemen: Voorbeeld Doolhof

Vereenvoudigd probleem voor doolhof: denk de muren weg.

Oplossing nu onmiddellijk gekend: Manhattan-afstand tot het doel.

Dit is een aanvaardbare (en consistente) heuristiek voor het oorspronkelijke probleem.

HO GENT

Vereenvoudigde Problemen: de 8-Puzzel

Regels 8-puzzel: A kan naar B worden verplaatst als

1. de vakjes A en B aangrenzend zijn; **en**
2. het vakje B is het lege vakje.

Laat nu één of meer van deze restricties weg:

1. A kan naar B worden verplaatst als A en B aangrenzend zijn.
2. A kan naar B worden verplaatst als B het lege vakje is.
3. A kan naar B worden verplaatst (zonder voorwaarden).

Welke (gekende) heuristieken krijgen we?

HO GENT

Patroon Databanken

Bekijk een deelprobleem en sla waarde oplossing deelprobleem op in databank.

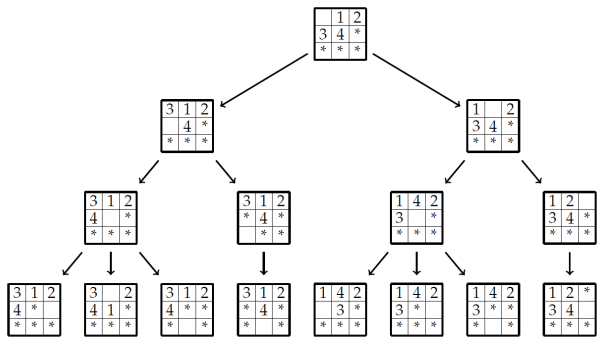
*	2	4
*		*
*	3	1

	1	2
3	4	*
*	*	*

HO GENT

Patroondatabank Opbouwen?

Door achterwaarts (omkeerbare acties) te zoeken, bv. met breedte eerst:



HO GENT

Heuristieken Combineren

Heuristieken kunnen gecombineerd worden m.b.v. max-operator.

$$h(s) = \max\big(h_1(s), h_2(s)\big).$$

Dit levert een nieuwe, betere, heuristiek op.

Aanvaardbaar als h_1 en h_2 aanvaardbaar zijn.

HO GENT

Oefening 1

(Oefening 3.4, AIMA) Beschouw twee vrienden die in verschillende steden wonen, bv. in Roemenië. Bij elke actie kunnen we elke vriend simultaan naar een naburige stad op de kaart verplaatsen. De hoeveelheid tijd nodig om zich van stad i naar de aanpalende stad j te verplaatsen is gelijk aan de afstand $d(i, j)$. Bij elke actie moet de vriend die eerst aankomt wachten tot de andere ook aankomt. De twee vrienden willen zo vlug als mogelijk samenkomen.

- Geef een gedetailleerde beschrijving van het zoekprobleem.
- Beschouw $D(i, j)$ als de afstand in vogelvlucht tussen de twee steden i en j. Welke van volgende heuristieken zijn toelaatbaar wanneer de eerste vriend zich in stad i en de tweede zich in stad j bevindt.
  - $D(i, j)$
  - $2 \cdot D(i, j)$
  - $D(i, j)/2$
- Zijn er toestanden (in de wetenschap dat er een pad is tussen alle steden op de kaart) waarvoor geen oplossing bestaat? Leg uit.

HO GENT

Oefening 2

Een aantal robots (bv. k) leven in een rooster waarin sommige locaties muren zijn. Twee robots kunnen zich nooit op dezelfde locatie bevinden. Elke robot heeft zijn eigen bestemming. Bij elke tijdseenheid verplaatsen de robots zich simultaan naar een aanpalend (vrij) vierkant of blijven ze staan. Twee robots die zich naast elkaar bevinden kunnen niet van plaats wisselen in één tijdseenheid. Elke tijdseenheid kost één punt. Beantwoord de volgende vragen:

- Geef een minimale correcte voorstelling van een toestand in de toestandsruimte.
- Schat de grootte van de toestandsruimte in voor een rooster met als afmetingen $M \times N$.

HO GENT

Vervolg oefening 2

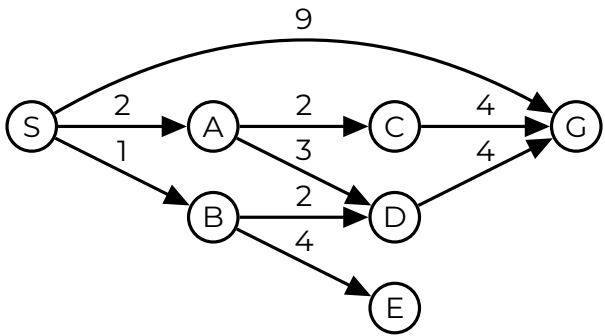
- Welke van volgende heuristieken zijn toelaatbaar? Beargumenteer je antwoord, meer bepaald: geef een situatie waarvan je aangeeft dat de gegeven heuristiek niet toelaatbaar is wanneer je dit beweert.

    - Som van de Manhattan afstanden voor elke robot tot zijn doellocatie.
    - Som van de kosten van de optimale paden indien de robots alleen in de omgeving voortbewegen, m.a.w. zonder obstructie door andere robots.
    - Maximum van de Manhattan afstanden vanuit elke robotpositie tot zijn doelpositie.
    - Maximum van de kosten van de optimale paden indien de robots alleen in de omgeving voortbewegen, m.a.w. zonder obstructie door andere robots.
    - Aantal robots die zich niet op hun doellocatie bevinden.

HO GENT

Oefening 3

Beschouw de volgende toestandsruimtegraaf. De starttoestand is steeds *S* en de doeltoestand is *G*. Voer een aantal zoekalgoritmes uit op deze toestandsruimtegraaf. Indien er ergens “random” een plan uit de open lijst moet gekozen worden, neem je het plan dat eindigt in de lexicografisch kleinste toestand. Hierdoor wordt de oplossing steeds uniek.



HO GENT

Vervolg oefening 3

De heuristiek gebruikt door de geïnformeerde zoekmethoden staat gegeven in onderstaande tabel:

toestand	S	A	B	C	D	E	G
h	6	0	6	4	1	10	0

Geef voor onderstaande blinde en geïnformeerde zoekmethodes het pad naar de doeltoestand. Geef ook aan welke toppen geëxpandeerd werden en dit in de juiste volgorde van hun expansie. De geïnformeerde zoekmethoden gebruiken de heuristiek h.

- Diepte-eerst (boomgebaseerd)
- Gulzig beste-eerst (boomgebaseerd)
- Uniforme kost zoeken (boomgebaseerd)
- A^* (boomgebaseerd)
- A^* (graafgebaseerd)

HO GENT

Oefening 4

Pas het gulzig beste eerst algoritme toe om van Arad naar Bucharest te gaan op de kaart van Roemenië. Gebruik de afstand in vogelvlucht naar Bucharest als heuristiek.

HO
GENT

Oefening 5

Gebruik A* om van Lugoj naar Bucharest te gaan op de kaart van Roemenië met de afstand in vogelvlucht als heuristiek Teken de opgebouwde zoekboom en geef aan in welke volgorde de plannen verwijderd worden van de open lijst. Los deze oefening eerst op voor boomgebaseerd zoeken en vervolgens voor graafgebaseerd zoeken.

HO
GENT

Oefening 6

Rush Hour wordt gespeeld op een $n \times n$ spelbord. Op het spelbord staan een aantal auto's, die elk twee aaneengrenzende vakjes beslaan, en een aantal vrachtwagens, die elk drie aaneengrenzende vakjes beslaan. Het spelbord bevat aan één van de zijden een uitgang. De spelregels zijn heel eenvoudig: je kan bij elke beurt een willekeurige auto of vrachtwagen een aantal vakjes voorwaarts of achterwaarts verplaatsen in de richting waarin deze geplaatst is op het spelbord. Uiteraard kunnen voertuigen elkaar niet overlappen en kunnen ze niet over elkaar springen. Het probleem is dat een specifieke auto (op de figuur gemarkeerd met X) naar de uitgang van het spelbord moet geleid worden. Hierbij moet de totale afgelegde afstand (som van alle afstanden door verplaatsen van auto's en/of vrachtwagens) zo laag mogelijk gehouden worden. Geef minstens twee aanvaardbare heuristieken voor dit probleem verschillend van de triviale aanvaardbare heuristiek $h = 0$.

HO GENT

Figuur Rush Hour

HO
GENT