Inhoudsopgave

[6.1 Inleiding 2](#_Toc91622571)

[De 8-puzzel 3](#_Toc91622572)

[Algemene Zoekalgoritmen 4](#_Toc91622573)

[6.2.1 Boomgebaseerd Zoeken 4](#_Toc91622574)

[6.2.2 Criteria voor Zoekalgoritmen 5](#_Toc91622575)

[6.2.3 Graafgebaseerd zoeken 7](#_Toc91622576)

[6.3 Blinde Zoekmethoden 9](#_Toc91622577)

[6.3.1 Breedte Eerst zoeken 9](#_Toc91622578)

[6.3.1.1 Tijdscomplexiteit 9](#_Toc91622579)

[6.3.2 Diepte Eerst Zoeken 10](#_Toc91622580)

[6.3.3 Iteratief verdiepen 12](#_Toc91622581)

[6.3.4 Uniforme Kost zoeken 15](#_Toc91622582)

[6.4 Geïnformeerde Zoekmethoden 15](#_Toc91622583)

[6.4.1 Heuristieken 15](#_Toc91622584)

[6.4.2 Gulzig Best Eerste 18](#_Toc91622585)

[6.4.3 A\* Zoekalgoritme 19](#_Toc91622586)

[6.5 Ontwerpen van Heuristieken 22](#_Toc91622587)

[6.5.1 Gebruik van Vereenvoudigede problemen 22](#_Toc91622588)

[6.5.2 Patroon databanken 22](#_Toc91622589)

Hoofdstuk 6 Zoekalgoritmes

# 6.1 Inleiding

Hierin bespreken we hoe een agent een zoekprobleem kan oplossen. Voor een zoekprobleem nemen we aan dat de agent zich bevindt in een eenpersoons omgeving die compleet observeerbaar determinitisch, statisch en discreet is. Hij bevindt zich steeds in een bepaalde beginpositie en de bedoeling is dat de agent acties onderneemt die hem in een toestand brengen waar een of andere voorwaarde voldaan is.

**Definitie:** een zoekprobleem bestaat uit:

* **Een toestandsruimte S** die alle mogelijke toestanden bevat.
* Een **verzameling** van mogelijke **acties** **A**
* **Een transitiemodel T** dat zegt wat het effect is van het uitvoeren van een actie op een bepaalde toestand:

Wanneer s’ bereikt wordt door het uitvoeren van een actie a op een toestand s dan wordt s’ een opvolger van s genoemd.

Het uitvoeren van een actie op een bepaalde toestand heeft meestal een bepaalde KOST C:



De kost kan dus afhangen van zowel s, de gekozen actie a, als van de opvolger s’. In deterministische omgevingen ligt de opvolger s’ vast wanneer men s en a weet, maar deze definitie kan ook gebruikt worden in een stochastische omgeving waar s’ onzeker is.

* Een initiële toestand s 0 ∈ 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**.

De 8-puzzel

**Voorbeeld:** De 8-puzzel is een typisch zoekprobleem. Hierbij is een vierkant rooster gegeven met 9 vakjes: 8 vakjes zijn genummerd en een vakje is leeg. Men kan een genummerd vakje verschuiven naar het lege vakje, waardoor de originele plaats van het genummerd leeg komt te staan.

We beschrijven de verschillende onderdelen van het zoekprobleem:

* De verzameling S bestaat uit alle mogelijke configuraties van de puzzel
* De 4 acties, boven onder links rechts.
* Het transitiemodel is een eenvoudige vertaling van wat er gebeurt in de fysieke puzzel. Wanneer men bv de actie rechts uitvoert op de linker puzzel, dan bekomt men een nieuwe puzzel waarbij het lege en het vakje 6 omgewisseld zijn. De kost van elke actie is 1.
* De initiële toestand is een willekeurige configuratie van de puzzel
* De doeltest bestaat uit verifiëren of de vooral vastgestelde configuratie werd bereikt.

Afbeelding met tafel

Automatisch gegenereerde beschrijving

# Algemene Zoekalgoritmen

## 6.2.1 Boomgebaseerd Zoeken

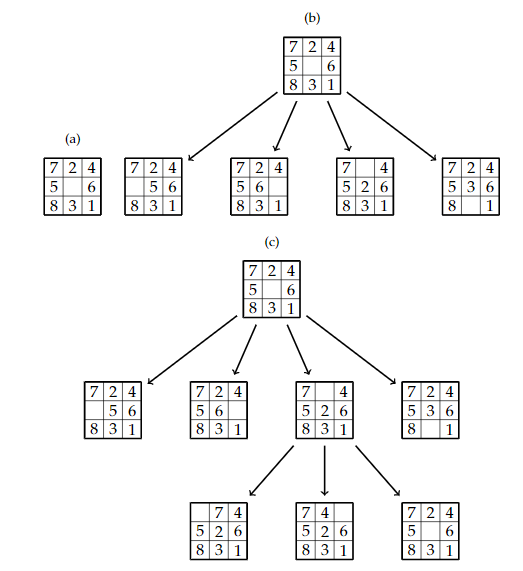
Het algemene algoritme gekend als boomgebaseerd zoeken houdt een lijst bij van mogelijke partiële oplossingen die nog verder uitgewerkt moeten worden. Deze lijst wordt de open lijst genoemd.

Bij de start van de uitvoering bestaat deze open lijst enkel uit het plan corresponderend met de initiële toestand van het zoekprobleem. Bij elke iteratie van het algoritme wordt een plan gekozen uit deze lijst. Wanneer de toestand van het gekozen plan voldoet aan de doeltest dan stopt het algoritme. Wanneer dit niet het geval is dan worden de plannen van alle opvolgers van de toestand van het gekozen plan toegevoegd aan de open lijst. Wanneer de open lijst op een bepaald moment leeg is dan geeft het algoritme aan dat er geen oplossing gevonden werd.

Conceptueel bouwen we dus een zoekboom op. Elke top van deze zoekboom stelt een sequentie van acties voor. Het is de bedoeling om de zoekproblemen op te lossen aan de hand van een zo klein mogelijke zoekboom. In een zoekboom stelt elke top een plan voor dat o.a. de huidige toestand bijhoudt. Dezelfde toestand kan (en zal) in het algemeen meerdere malen voorkomen op de open lijst.

**Implementatie van een plan**

Conceptueel stelt elke top van de zoekboom een sequentie van acties voor om een bepaalde toestand te bereiken startend vanaf de initiële toestand van het zoekprobleem.



Afbeelding met tekst

Automatisch gegenereerde beschrijving

* De huidige toestand
* De laatst gekozen actie a: deze is enkel leeg voor het plan geassocieerd met de initiële toestand.
* 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. Strikt genomen kunnen we deze kost ook berekenen door het volgen van de voorganger-referenties. Deze berekening heeft echter een uitvoeringstijd die lineair is in het aantal acties van het plan.

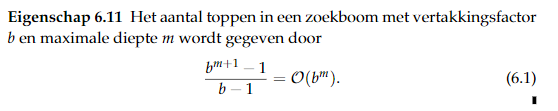
## 6.2.2 Criteria voor Zoekalgoritmen

Zoekalgoritmen kunnen op verschillende manieren worden geëvalueerd. De volgende vier criteria worden vaak gebruikt.

1. Een zoekalgoritme is compleet wanneer het algoritme, voor elk zoekprobleem met een oplossing, effectief een oplossing vindt.
2. Een zoekalgoritme is optimaal wanneer het niet enkel een oplossing vindt, maar steeds een optimale oplossing teruggeeft voor elk zoekprobleem met een oplossing.
3. De tijdscomplexiteit van een zoekalgoritme bepaalt de uitvoeringstijd van het algoritme. We nemen aan dat de uitvoeringstijd evenredig is met het aantal gegeneerde toppen.
4. De ruimte complexiteit 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.

De volgende maten worden vaak gebruikt om de tijds- en ruimtecomplexiteit van zoekalgoritmen uit te drukken.

* **Vertakkingsfactor b**. => deze geeft het maximaal aantal opvolgers van een top in de zoekboom.
* **De diepte d** van de meest ondiepe top waarvan de toestand een doeltoestand is (kortweg de doeltop genoemd).
* **Maximale lengte m** => met m duidt men de maximale lengte (gemeten als het aantal genomen acties) van een pad in de toestandsruimte aan.



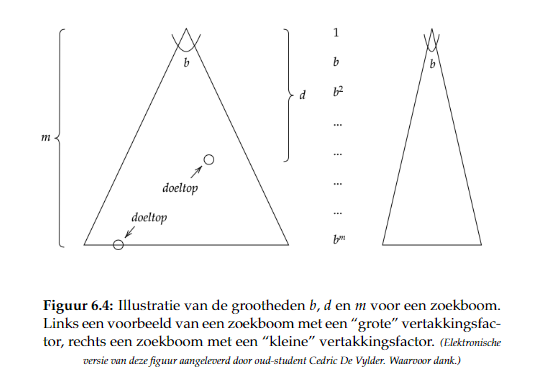
**Bewijs:** we nemen aan dat elke top exact b opvolgers heeft.

Het aantal toppen op diepte 0 is 1, het aantal toppen op diepte 1 is b, het aantal toppen op diepte 2 is b²,… . Het totaal aantal toppen is dus:

Een gesloten formule voor deze som wordt inderdaad gegeven door 6.1

**Opmerking** De laag met diepte m in een zoekboom met vertakkingsfactor b bevat b^m toppen. Die is 1 meer dan alle voorgaande lagen samen, deze bevatten in totaal slechts:





## 6.2.3 Graafgebaseerd zoeken

Het grootste probleem van boomgebaseerd zoeken is dat dit algoritme niet onthoudt waar het reeds geweest is. Dit zorgt ervoor dat we in sommige gevallen (bv diepte eerst zoeken) te maken krijgen met oneindige lussen, en dat we in veel andere gevallen een grote hoeveelheid werk herhaaldelijk uitvoeren.

**Voorbeeld:** een vaak voorkomende situatie is die van een agent die leeft in een grid met 4 acties: boven onder links rechts. De vertakkingsfactor b is in dit geval gelijk aan 4. Vergelijken we nu eventjes het aantal toestanden op afstand d van een willekeurig toestand met het aantal toppen van de zoekboom op diepte d bij boomgebaseerd zoeken.

Afbeelding met tafel

Automatisch gegenereerde beschrijving

Toppen: vb: naar rechts en omlaag: 1 toestand, 1 top | omlaag en naar rechts, 1 top, maar zelfde toestand

We zien dat het aantal verschillende toestanden lineair toeneemt met de diepte terwijl het aantal toppen exponentieel toeneemt met de diepte

De oplossing voor het probleem van de herhaalde toestanden bestaat erin om eenvoudigweg te onthouden welke toestanden reeds geëxpandeerd zijn, in een gesloten lijst.

Merk op dat de gesloten lijst toestanden bevat, terwijl de open lijst plannen bevat.

Bij graafgebaseerd zoeken wordt elke toestand hoogstens eenmaal geëxpandeerd. Wanneer een plan van de open lijst wordt gehaald dat een toestand bevat die reeds geëxpandeerd is, dan wordt deze niet opnieuw geëxpandeerd.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

# 6.3 Blinde Zoekmethoden

Blinde zoekmethoden kunnen enkel gebruikmaken van de informatie die verschaft wordt door de definitie van het zoekprobleem. Ze beschikken niet over extra informatie die hen kan helpen bij het zoekproces. We bespreken 4 blinde zoekmethoden en hun eigenschappen.

## 6.3.1 Breedte Eerst zoeken

Het breedte eerst zoeken wordt voor de open lijst een wachtrij gebruikt. Dit is een FIFO datastructuur. Bij breedte eerst wordt de zoekboom systematisch laag per laag opgebouwd.

Omdat het systematisch de lagen in de zoekboom onderzoeken zal het algoritme steeds een oplossing vinden voor elk zoekprobleem dat effectief een oplossing heeft.

Het algoritme vindt steeds de meest ondiepe doeltop, i.e. het retourneert een oplossing met een minimaal aantal acties. Wanneer acties een verschillende kost hebben is dit niet noodzakelijk een oplossing met de kleinste kost. Breedte eerst is m.a.w. niet optimaal. Het is wel optimaal als alle acties dezelfde kost hebben.

### 6.3.1.1 Tijdscomplexiteit

We onderzoeken nu de **tijdscomplexiteit** van breedte eerst zoeken. Veronderstel dat het zoekprobleem een oplossing heeft en dat de meest ondiepe doelknoop diepte d heeft. Als deze doeltop de “meest rechtste” top is dan worden alle toppen op de dieptes 0 t.e.m d geëxpandeerd. Het aantal gegenereerde toppen is m.a.w. gelijk aan:

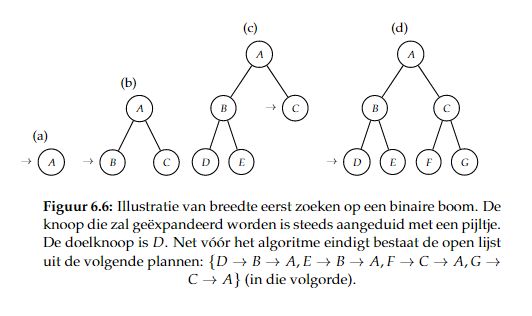


De tijdscomplexiteit is met andere woorden exponentieel in de diepte van de meest ondiepe doeltop.

Het maximaal aantal toppen dat moet worden bijgehouden in de open lijst wordt bereikt, wanneer men de doeltop expandeert. Op dit moment wordt zo goed als de volledige laag op diepte d+1 bijgehouden in de open lijst. Deze laag bevat b¨d+1 toppen. De ruimtecomplexiteit is gelijk aan:

## 6.3.2 Diepte Eerst Zoeken

Diepte eerst zoeken is in zekere zin duaal aan breedte eerst zoeken: hier gebruikt men een LIFO structuur voor het behouden van de open lijst. Deze stapel zorgt ervoor dat men zo snel mogelijk zo diep mogelijk in de boom afdaalt.

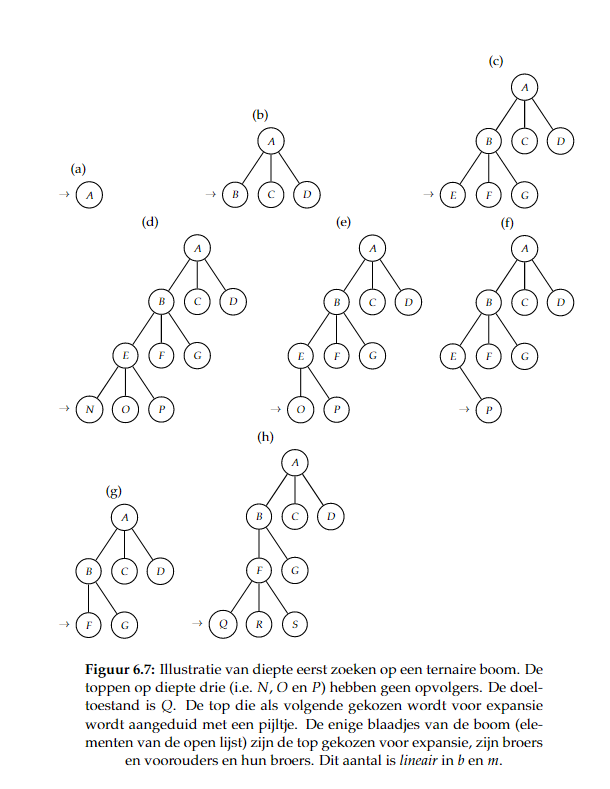


Diepte eerst zoeken genereert steeds een linkerdeel van de boom. Wanneer m eindig is en de enige doeltop helemaal rechts onderaan in de boom zit dan worden alle toppen van de boom gegenereerd. In het slechtste geval is de tijdscomplexiteit . Dit is dezelfde exponentiële tijdscomplexiteit als bij breedte eerst.

Diepte eerst kan ook in bepaalde gevallen in een oneindige lus geraken. Het is dus niet compleet en ook niet optimaal. Zelfs wanneer deze een oplossing vind kan die niet optimaal zijn.

Diepte eerst is wel goed op het vlak van geheugengebruik. Wanneer een bepaalde top geëxpandeerd wordt, dan behoren enkel de broers van zijn voorouders tot de open lijst.

Aangezien er hoogstens m broers zijn is de ruimtecomplexiteit van diepte eerst van de orde . Dit is een lineaire functie van b.



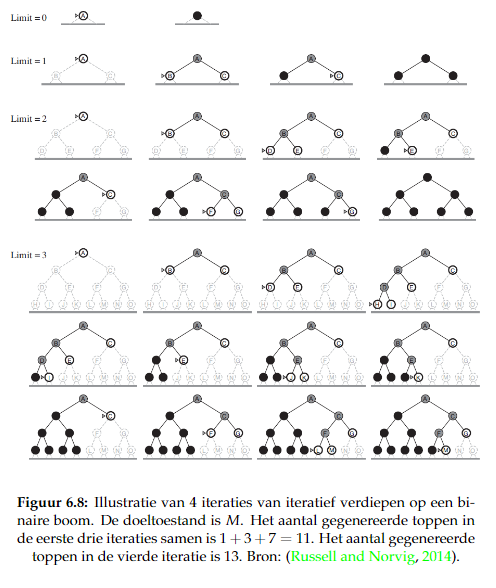
## 6.3.3 Iteratief verdiepen

het is geen directe toepassing van het boomgebaseerd zoekalgoritme. Het is in essentie een lus rond diepte eerst zoeken waarbij het zoekproces wordt afgebroken, wanneer een bepaalde diepte wordt bereikt. => dit noemt men diepte-gelimiteerd zoeken.

Het algoritme wordt recursief geïmplementeerd en bij elke recursieve oproep wordt de maximale toegelaten diepte met 1 verminderd. Wanneer de toegelaten diepte 0 bereikt wordt het meegegeven plan niet verder geëxpandeerd.

Het algoritme heeft een bijzondere returnwaarde: ‘hit boundary’ om aan te geven dat er geen oplossing gevonden werd binnen de opgegeven diepte. Dit geeft aan dat er eventueel wel een oplossing kan gevonden worden wanneer de maximale toegelaten diepte wordt verhoogd.

Bij elke iteratie van iteratief verdiepen, wordt de maximaal toegelaten diepte met 1 verhoogd. Het algoritme stopt wanneer dat er een oplossing gevonden is OF wanneer het duidelijk is dat er geen oplossing mogelijk is. Op die manier vermeiden we het probleem dat we in een oneindige lus terecht kwamen.



Afbeelding met tekst

Automatisch gegenereerde beschrijving

Afbeelding met tekst

Automatisch gegenereerde beschrijving

We bekijken nu hoeveel toppen er gegenereerd worden bij elke oproep van diepte-gelimiteerd zoeken, waarna we al deze gegenereerde toppen optellen.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Als we nu de som uitvoeren, zien we dat de term 1 in totaal d + 1 keer voorkomt, de term b komt d keer voor, de term b² komt d-1 keer voor. Het totaal gegenereerde toppen is dus



Het is interessant om het aantal toppen gegenereerd in de laatste iteratie te vergelijken met het totaal aantal toppen gegenereerd in alle iteraties ervoor.

* Het aantal toppen gegenereerd in de laatste iteratie:
* Het algemeen aantal toppen dat gegenereerd is:



* Afbeelding met tekst, klok

  Automatisch gegenereerde beschrijvingHet aantal toppen die gegenereerd zijn buiten de laatste iteratie:

## 6.3.4 Uniforme Kost zoeken

Uniforme kost zoeken tracht het probleem dat breedte eerst niet noodzakelijk optimaal is (wanneer acties een verschillende kost hebben) op te lossen door steeds het plan te expanderen waarvoor de totale kost van dit plan minimaal is. => de openlijst wordt hier geïmplementeerd adhv een prioriteitswachtrij.

* Hoe kleiner de kost hoe groter de prioriteit

Uniforme kost zoeken is een compleet en optimaal algoritme wanneer:

* Alle acties een kost hebben die groter of gelijk is aan een of andere positieve *ϵ*

De tijd- en ruimtecomplexiteit is  waarbij C\* de kost van de optimale oplossing voorstelt.

# 6.4 Geïnformeerde Zoekmethoden

De blinde of niet geïnformeerde zoekmethoden hebben geen toegang tot domeinkennis om het zoeken meer efficiënt te laten verlopen. Daardoor gaan ze vaak toestanden expanderen die wij als mens “stom” vinden.

Vb: wanneer we een weg moeten plannen tussen twee steden in een gebied waar we de weg niet kennen, gaan we eerst kijken naar de steden die al “in de juiste richting” liggen => steden waarvoor de afstand in vogelvlucht tot de doelstad klein is. We beschouwen dat als een goede indicator of schatting.

We gebruiken een heuristiek om het zoekproces efficiënter te laten verlopen.

## 6.4.1 Heuristieken

**Definitie:** een heuristiek h is een afbeelding van de verzameling toestanden S naar de verzameling van niet negatieve reële getallen IR:

Het is de bedoeding dat de heuristiek een goede schatting is voor de werkelijke kost naar het doel, wanneer h(s) klein is dan is dit een goede indicator dat s ook effectief niet ver van een doeltoestand verwijderd is.

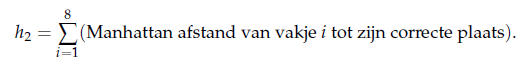
Aangezien dat h zal gebruikt worden in de zoekalgoritmes is het ook noodzakelijk dat h snel te berekenen is.

**Voorbeeld 8 puzzel**: er kan gekeken worden naar het aantal vakjes dat niet op de juiste plaats staan in vergelijking met doel

h1= aantal niet-lege vakjes dat niet op zijn juiste plaats staat

* Hoe meer vakjes er verkeerd staan, hoe meer zetten er nodig zijn om de puzzel correct te maken

We moeten niet enkel rekening houden met het aantal verkeerde vakjes, maar ook met hun afstand tot aan het doel:



**Definitie:** Een heuristiek h: S -> IR is toelaatbaar als voor elke toestand s geldt dat  waarbij C\* de kost van een optimale oplossing voorstelt van s naar een doeltoestand.

**Stelling**: wanneer een heuristiek h toelaatbaar is, dan is h(g) = 0 voor elke doeltoestand g

**Bewijs:** Voor een doeltoestand g geldt dat de kost van de optimale oplossing gelijk is aan 0 zodat:

 waaruit onmiddellijk volgt dat h(g) = 0.

**Definitie:** een heuristiek h: S -> IR is consistent als voor elk doeltoestand g geldt dat h(g) = 0 en als voor elke toestand s en elke actie a op s met s’ = T(s,a) geldt dat

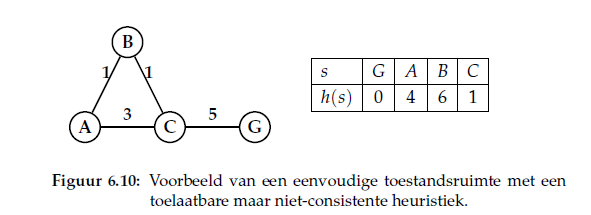
**Stelling** als een heuristiek consistent is, dan is ze ook onmiddellijk toelaatbaar.

**Bewijs:** we kunnen voor elke actie a die een toestand s omzet in s’ de heuristische kost definiëren als h(s) – h(s’). voor een consistente heuristiek geldt dat



Met andere woorden, de kost is steeds kleiner of gelijk aan de werkelijke kost.

(p172 (181/244)) bewijs nog typen



Toelaatbare maar inconsistente heuristiek: het is niet zo dat elke toelaatbare heuristiek ook consistent is. Beschouw de toestandsruimte, waarbij G de doeltoestand voorstelt. We zien op het zicht dat de kost van de optimale oplossing voor de verschillende toestanden gegeven wordt door:

Afbeelding met tafel

Automatisch gegenereerde beschrijving

Wanneer we dit vergelijken met de heuristiek h uit 6.10 dan zien we dat voor elke toestand s geldt dat: 

* De heuristiek is dus toelaatbaar

Opdat de heuristiek h consistent zou moeten zijn voor elke toestand s en voor elke opvolger s’ gelden dat:

Afbeelding met tafel

Automatisch gegenereerde beschrijving

We zien dat in twee gevallen de ongelijkheid niet voldaan is. Zo is bv voor s = B en s’ = A de heuristische kost 6-4=2, en dit is groter dan de werkelijke kost. (1).

## 6.4.2 Gulzig Best Eerste

Deze zoekmethode maakt gebruik van een heuristiek h. De methode kiest steeds de top met de kleinste waarde van h als de volgende top die wordt geëxpandeerd. Net zoals bij uniforme kost zoeken, wordt de openlijst geïmplementeerd als een prioriteitswachtrij. Een kleinere waarde voor h betekent een grotere prioriteit.

**Voorbeeld** niet-optimaliteit gulzig best eerst): we passen het gulzig best eerst algoritme toe op de toestandsruimtegraaf (6.10) waarbij we de volgende consistente heuristiek gebruiken.

Afbeelding met tafel

Automatisch gegenereerde beschrijving

De gebruikte heuristiek is in zekere zin “perfect” omdat ze voor elke toestand s gelijk is aan C\*(s). de initiële toestand is A en de doeltoestand is G.

Afbeelding met tafel

Automatisch gegenereerde beschrijving

Het algoritme gaat recht op het doel af. In dit geval behoort ook elke geëxpandeerde toestand effectief tot de geretourneerde oplossing. Helaas is de gevonden oplossing niet de optimale oplossing. In dit geval komt dit omdat het algoritme geen rekening houdt met de kost om vanuit A de toestand C te bereiken. Het houdt enkel rekening met het feit dat C er beter uitziet dan B op basis van het heuristische waarden. Afbeelding met tafel

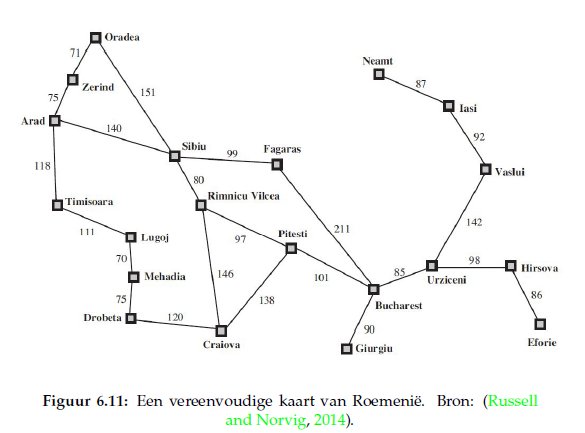
Automatisch gegenereerde beschrijving

## 6.4.3 A\* Zoekalgoritme

Het probleem van de gulzig beste eerst zoekmethode is dat er enkel rekening wordt gehouden met de waarde van de heuristiek en niet met de kost van de reeds afgelegde weg. Er wordt dus waardevolle informatie genegeerd.

Bij de A\* zoekmethode wordt de open lijst nog steeds geïmplementeerd als een prioriteitswachtrij, maar de volgende top die wordt geëxpandeerd is de top (plan) n waarvoor minimaal is. Hierbij is g(n) de totale kost van het plan n, de kost van de reeds gekozen acties, en is h(n) de waarde van de heuristiek voor de toestand die hoort bij deze top.

**Voorbeeld:** we proberen op d e kaart van Roemenië, een weg te vinden van Arad naar Bucharest. We gebruiken het A\* algoritme met als heuristiek de afstand in vogelvlucht. De opbouw van de zoekboom kan je volgen in fig 6.12.



Dit enige plan wordt van de open lijst gehaald. Aangezien Arad niet voldoet aan de doeltest wordt dit plan geëxpandeerd. De plannen S->A, t->A en Z->A worden aan de lijst toegevoegd.



Aangezien het plan (S->A, 140 +243 = 393) de kleinste f-waarde heeft wordt dit als volgende van de open lijst gehaald. Het doel is nog niet bereikt en dus wordt dit plan geëxpandeerd. De volgende plannen worden toegevoegd aan de open lijst.

Afbeelding met tekst

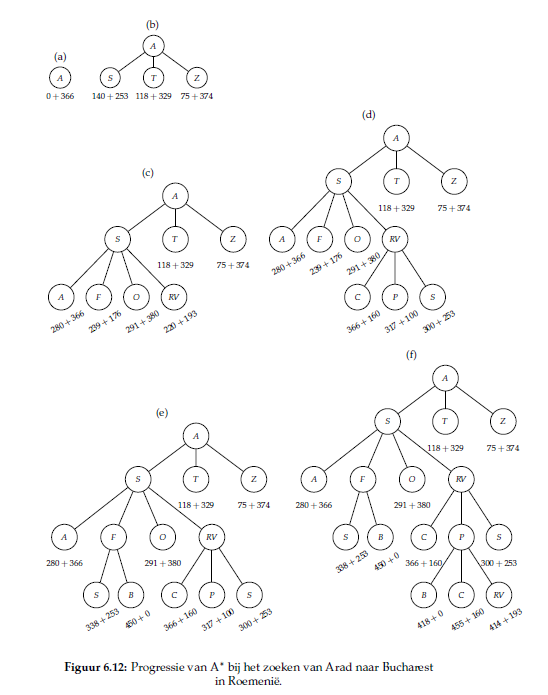
Automatisch gegenereerde beschrijving

De open lijst bestaat nu uit 6 plannen waarvan het plan RV->S ->A de kleinste f-waarde heeft. Dit plan wordt van de open lijst gehaald, en aangezien het doel niet werd bereikt, wordt het geëxpandeerd.

……. (alle overige stappen)

Het plan B->P->RV->S->A is het plan met de laagste f-waarde. Dit plan wordt van de open lijst gehaald. De doeltest slaagt voor dit plan en het algoritme eindigt en geeft de oplossing terug:





**Stelling** Wanneer boomgebaseerde A∗gebruikmaakt van een toelaatbare heuristiekℎen 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.

**Bewijs:**

Afbeelding met tekst

Automatisch gegenereerde beschrijving

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

Afbeelding met tekst

Automatisch gegenereerde beschrijving

# 6.5 Ontwerpen van Heuristieken

Het is duidelijk dat een goede heuristiek een grote positieve invloed kan hebben op de tijds- en ruimtecomplexiteit van het algoritme zoals A\*.

## 6.5.1 Gebruik van Vereenvoudigede problemen

Veronderstel dat de agent zich in een grid beweegt en dat de agent zich door middel van acties: B O L R naar een bepaalde locatie in zijn grid moet bewegen. Echter zijn sommige locaties muren, zodat het grid een doolhof voorstelt.

* denk de muren weg, dan is de oplossing meteen te berekenen. Het aantal benodigde acties is niets anders dan de Manhattan-afstand tussen de huidige locatie en de doellocatie.

## 6.5.2 Patroon databanken

Een aanvaardbare heuristiek kan ook gevonden worden als de kost van een optimale oplossing voor een deelprobleem. Veronderstel dat we in de 8 puzzel de vakjes 5 t.e.m 8 vervangen door een identiek symbool, bv een sterretje. In 6.13 zie je de instanties van dit soort puzzel wanneer dit proces wordt toegepast. Het is duidelijk dat het minimum aantal stappen nodig om de puzzel aan de linkerkant van fig 6.13 om te zetten naar de puzzel rechts, een ondergrens is voor het aantal stappen nodig om de originele puzzel op te lossen.

Afbeelding met tafel

Automatisch gegenereerde beschrijving

Uit de boom in fig 6.14 lezen we af dat de optimale oplossing kost van

