Computationele Intelligentie

Heuristisch zoeken

Heuristisch zoeken

- een algoritme voor heuristisch zoeken doorzoekt de zoekruimte van een probleem op een systematische wijze, gestuurd door kennis van het probleem;
- enkele voorbeelden zijn:
  - best-first search;
  - heuristische depth-first search;
  - A en A^*.

Een heuristische functie op de toestandsruimte

Op de toestandsruimte van een zoekprobleem wordt een functie aangebracht:

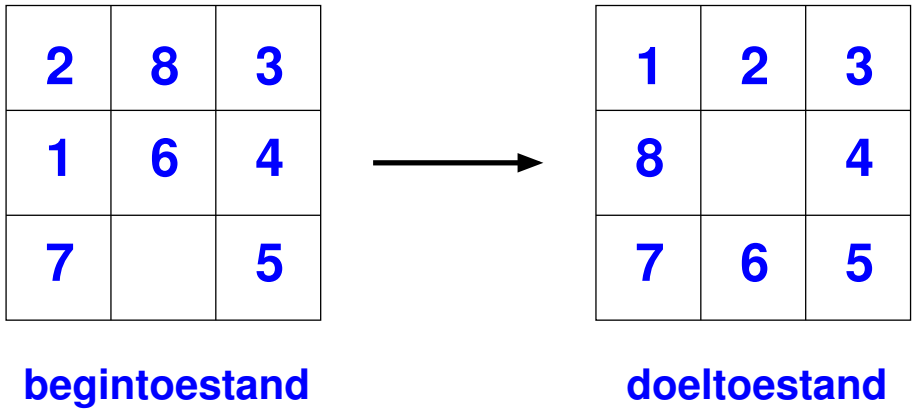
- de functie is gebaseerd op kennis van het probleem;
- de functie geeft een inschatting van de afstand van een toestand tot de dichtstbijzijnde doeltoestand.

Zo'n functie wordt een heuristische functie genoemd en wordt gebruikt om het zoekproces te sturen.

Voorbeelden van kennis

Voor de 8-puzzel geven de volgende functiewaarden een schatting van de afstand van een toestand tot de dichtstbijzijnde doeltoestand:

- het aantal tegels dat zich nog niet op hun doelpositie bevindt;
- de som van de kortste afstanden van de huidige posities van de tegels tot hun doelposities.



Een heuristische functie

Definitie

Zij T de toestandsruimte van een zoekprobleem. Een heuristische functie op T is een functie $h: T \to \mathbb{R}$.

Een heuristische functie h moet de volgende eigenschappen hebben:

- de heuristische waarde $h(t)$ van een toestand $t \in T$ is uitsluitend bepaald door
  - kennis van de doeltoestanden van het probleem;
  - kennis van de toestand t zelf;
- een heuristische waarde $h(t)$ is eenvoudig te berekenen.

Een heuristische functie voor de 8-puzzel

Op de toestandsruimte T van de 8-puzzel definiëren we een heuristische functie $h: T \to \mathbb{N}$ met voor elke toestand $t \in T$:

$$h(t) = \text{het aantal tegels in } t \text{ dat zich nog niet op hun doelpositie bevindt}$$

2	8	3
1	6	4
7		5

h(t1) = 4

1	2	3
8		4
7	6	5

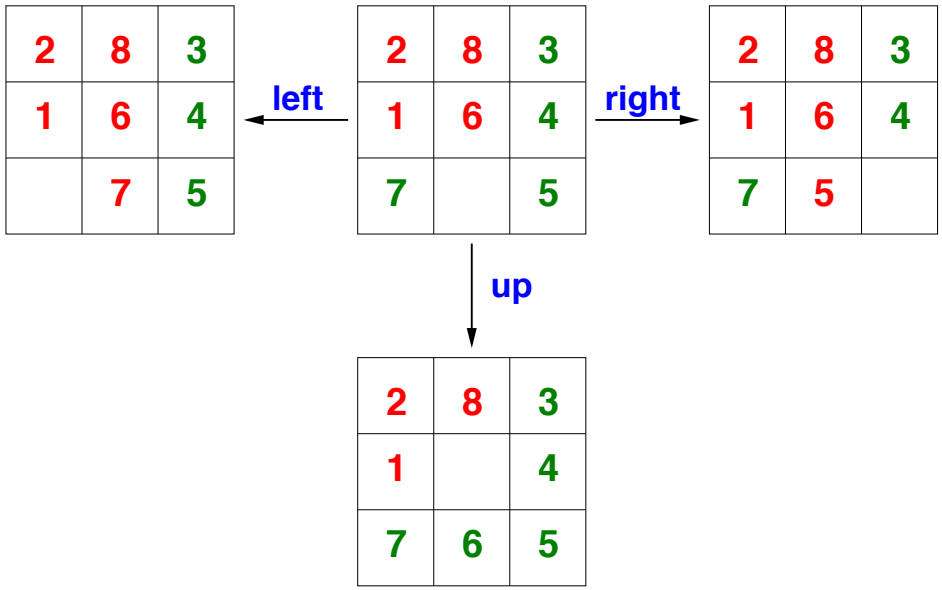
h(t2) = 0

2		3
1	8	4
7	6	5

h(t3) = 3

Het gebruik van de heuristische functie

De heuristische functie h wordt gebruikt om een operator te selecteren die het aantal incorrect gepositioneerde tegels minimaliseert:



Voor de gegeven toestand zal het zoekalgoritme voor de operator up kiezen.

Best-first search

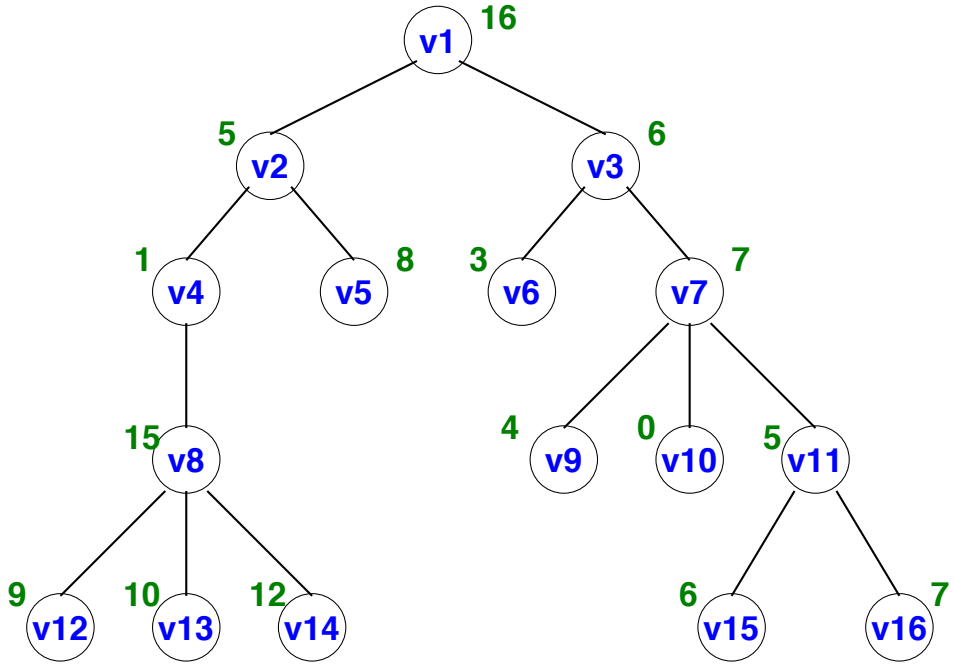
De essentie van best-first search is in pseudocode

```
procedure best-first(h,L) returns t:
  if empty(L) then return nil
  else
    t ← first(L);
    if goal(t) then return t
    else
      L ← insert(butfirst(L),successors(t));
      L ← sort-in-increasing-order(h,L);
      best-first(h,L)
endprocedure
```

De procedure `best-first` wordt aangeroepen voor een (dynamische) zoekboom T met een heuristische functie h en een lijst L met initieel alleen de wortel van T, en geeft een toestand (en een pad) terug.

Een voorbeeld

Beschouw de volgende zoekboom met bijbehorende heuristische waarden:



De lijst L heeft achtereenvolgens de volgende waarden:

$$L_1 = (\mathrm{v1});$$
$$L_2 = (\mathrm{v2}, \mathrm{v3});$$
$$L_3 = (\mathrm{v4}, \mathrm{v3}, \mathrm{v5});$$
$$L_4 = (\mathrm{v3}, \mathrm{v5}, \mathrm{v8});$$
$$L_5 = (\mathrm{v6}, \mathrm{v7}, \mathrm{v5}, \mathrm{v8});$$
$$L_6 = (\mathrm{v7}, \mathrm{v5}, \mathrm{v8});$$
$$L_7 = (\mathrm{v10}, \mathrm{v9}, \mathrm{v11}, \mathrm{v5}, \mathrm{v8}).$$

De vier eigenschappen

Als de zoekboom van een zoekprobleem eindig is en tenminste één doeltoestand bevat, dan geldt:

- best-first search vindt altijd een doeltoestand;
- de gevonden doeltoestand is niet noodzakelijk een doeltoestand met minimale afstand tot de wortel van de boom;
- best-first search kost gemiddeld exponentieel veel tijd en ruimte.

Als de zoekboom niet eindig is, dan vindt best-first search niet noodzakelijk een doeltoestand.

Het geheugenbeslag van best-first search is ruwweg vergelijkbaar met dat van dynamische breadth-first search; het precieze geheugenbeslag is echter sterk afhankelijk van de verdeling van de heuristische waarden.

Heuristische depth-first search

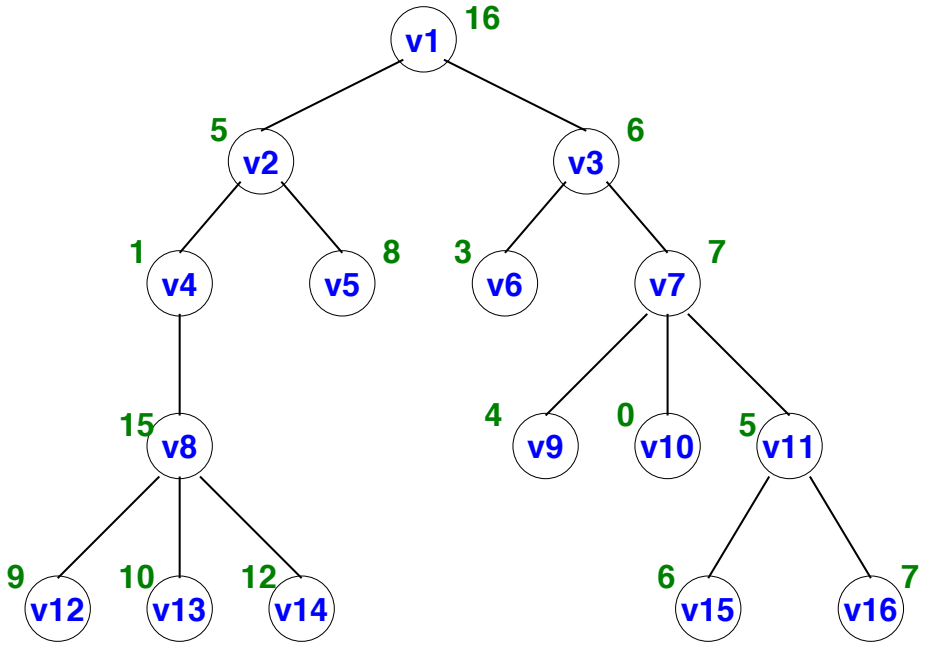
De essentie van heuristische depth-first search is in pseudocode

```
procedure heuristic-dfs(h,L) returns t:
  if empty(L) then return nil;
  else
    t ← pop(L);
    if goal(t) then return t
    else
      list ← sort-in-increasing-order(successors(T,t),h);
      L ← push(L,list);
      heuristic-dfs(h,L)
endprocedure
```

De procedure `heuristic-dfs` wordt aangeroepen voor een (dynamische) zoekboom T met een heuristische functie h en een stack L met initieel de wortel van T, en geeft een toestand (en een pad) terug.

Een voorbeeld

Beschouw de volgende zoekboom met bijbehorende heuristische waarden:



De stack L heeft achtereenvolgens de volgende waarden:

$$L_1 = (\mathrm{v}1); \qquad \vdots$$
$$L_2 = (\mathrm{v}2, \mathrm{v}3); \qquad L_{10} = (\mathrm{v}6, \mathrm{v}7);$$
$$L_3 = (\mathrm{v}4, \mathrm{v}5, \mathrm{v}3); \qquad L_{11} = (\mathrm{v}7);$$
$$L_4 = (\mathrm{v}8, \mathrm{v}5, \mathrm{v}3); \qquad L_{12} = (\mathrm{v}10, \mathrm{v}9, \mathrm{v}11).$$

Heuristisch geoorloofde zoekalgoritmen

Definitie

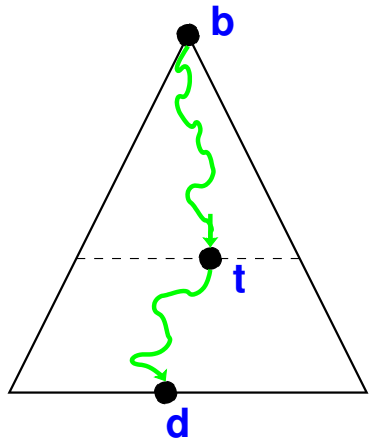
Een zoekalgoritme heet heuristisch geoorloofd als het

- voor elk (oplosbaar) zoekprobleem
- voor elke begintoestand

termineert met een pad naar een doeltoestand van minimale lengte.

Een evaluatiefunctie — inleiding

Beschouw een toestand t in een statische zoekboom :



De lengte van het werkelijk kortste pad van de begintoestand b naar een doeltoestand via toestand t is

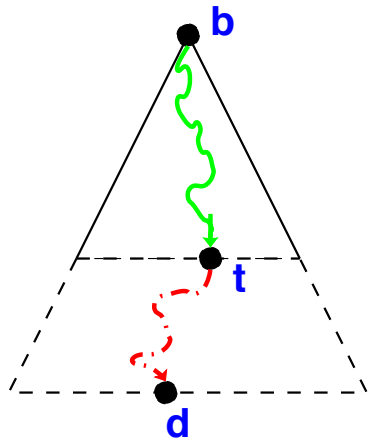
$$f^*(t) = g^*(t) + h^*(t)$$

waarin

- $g^*(t)$ is de lengte van het werkelijk kortste pad van begintoestand b naar toestand t;
- $h^*(t)$ is de lengte van het werkelijk kortste pad van toestand t naar een doeltoestand.

Een evaluatiefunctie — inleiding

Beschouw nu een toestand t in een dynamische zoekboom:



Een schatting van de lengte van het kortste pad van de begintoestand b naar een doeltoestand via toestand t is

$$f(t) = g^*(t) + h(t)$$

waarin

- $g^*(t)$ is de lengte van het werkelijk kortste pad van begintoestand b naar toestand t;
- $h(t)$ is een schatting van de lengte van het kortste pad van toestand t naar een doeltoestand.

Een evaluatiefunctie

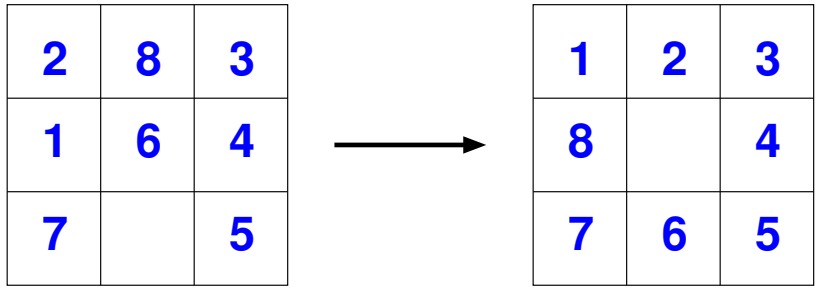
Beschouw voor een gegeven zoekprobleem nogmaals de functie

$$f(t) = g^*(t) + h(t)$$

- de functie $h : T \to \mathbb{R}$ is een heuristische functie als voorheen;
- de functie $f : T \to \mathbb{R}$ heet een evaluatiefunctie voor het probleem — men zegt dat de evaluatiefunctie f de heuristische functie h inbedt.

Een voorbeeld

Beschouw de 8-puzzel met de volgende begin- en doeltoestand:



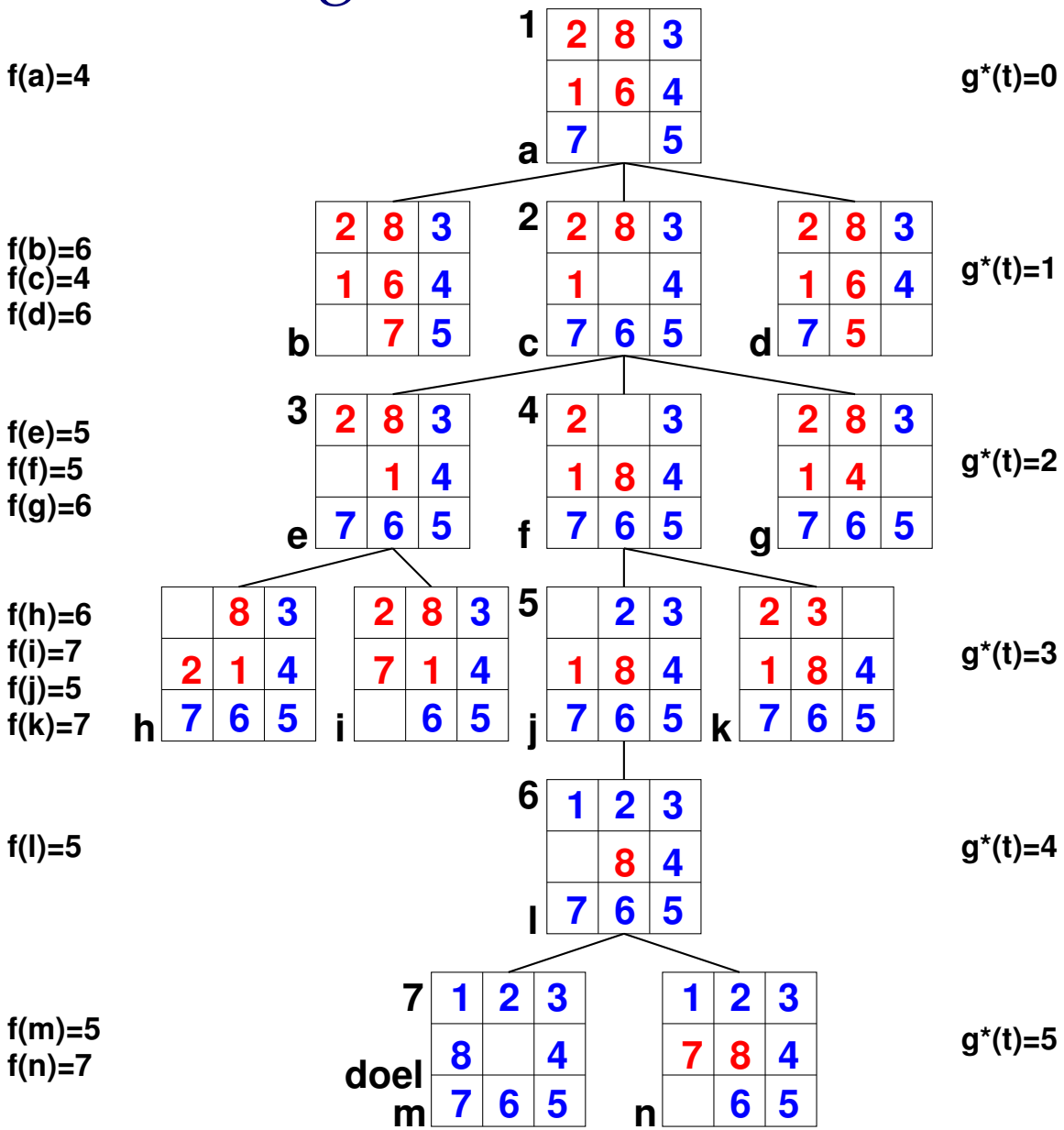
Beschouw voorts de evaluatiefunctie:

$$f(t) = g^*(t) + h(t)$$

op de toestandsruimte van de 8-puzzel, waarin

- $g^*(t)$ is de werkelijke diepte van toestand t in de dynamische zoekboom;
- $h(t)$ is het aantal incorrect geplaatste tegels in t.

Een voorbeeld — vervolg



Een geoorloofde heuristische functie

Definitie

Zij T de toestandsruimte van een zoekprobleem. Zij $h: T \to \mathbb{R}$ een heuristische functie op T en zij $h^*: T \to \mathbb{R}$ de werkelijke afstandsfunctie op T. De functie h heet geoorloofd als

$$0 \leq h(t) \leq h^*(t)$$

voor alle $t \in T$.

- voor het zoeken naar een route tussen twee posities in een ruimte met obstakels is de hemelsbrede afstandsfunctie een geoorloofde functie;
- voor de 8-puzzel is de som van de kortste afstanden van de huidige posities van de tegels tot hun doelposities een geoorloofde functie.

De zoekalgoritmen A en A*

- best-first search met een evaluatiefunctie van de vorm

$$f(n) = g^*(n) + h(n)$$

heet een A-zoekalgoritme;

- best-first search met een evaluatiefunctie van de vorm

$$f(n) = g^*(n) + h(n)$$

met h een geoorloofde heuristische functie, heet een A*-zoekalgoritme.

Geoorloofdheid van A*

Stelling

Een A*-zoekalgoritme is heuristisch geoorloofd:

- een A*-zoekalgoritme termineert voor elke begintoestand;
- een A*-zoekalgoritme vindt een doeltoestand op minimale afstand van de begintoestand.

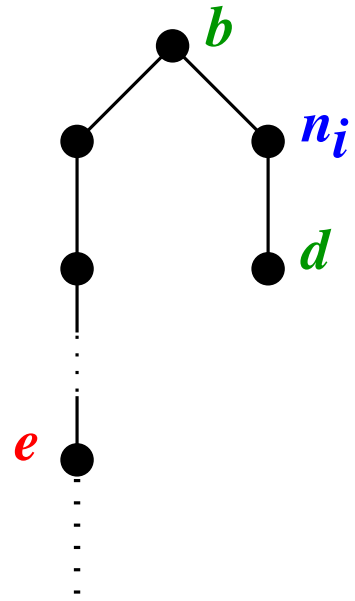
Terminatie van A* — bewijsschets

Zij $b = n_1, \ldots, n_q = d$ een eindig pad in de zoekboom van de begintoestand b naar een doeltoestand d. Zij n_i de toestand op dat pad met $n_i \in L$.

Voor de toestand n_i geldt dat

$$f(n_i) = g^*(n_i) + h(n_i) \leq f^*(n_i)$$

waarin $f^*(n_i)$ naar boven is begrensd.



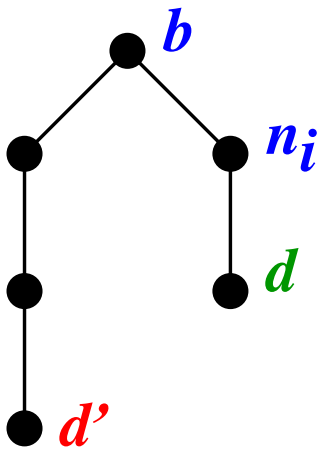
Voor de toestanden e op een oneindig pad vanaf b geldt dat $g^*(e) \to \infty$ en dus dat $f(e) \to \infty$.

Het algoritme zal dus eens voor expansie van een toestand n_i op een eindig pad kiezen en termineren.

Optimaliteit van A* — bewijsschets

Veronderstel dat het A*-zoekalgoritme termineert met het niet optimale pad $b = m_1, \ldots, m_p = d'$. Er geldt dan:

$$f(d') = g^*(d') + 0 > f^*(b)$$



Zij nu $b = n_1, \ldots, n_q = d$ een optimaal pad in de zoekboom. Zij n_i de toestand op dat pad met $n_i \in L$. Voor n_i geldt dat

$$f(n_i) = g^*(n_i) + h(n_i) \leq f^*(b)$$

en dus dat

$$f(n_i) < f(d')$$

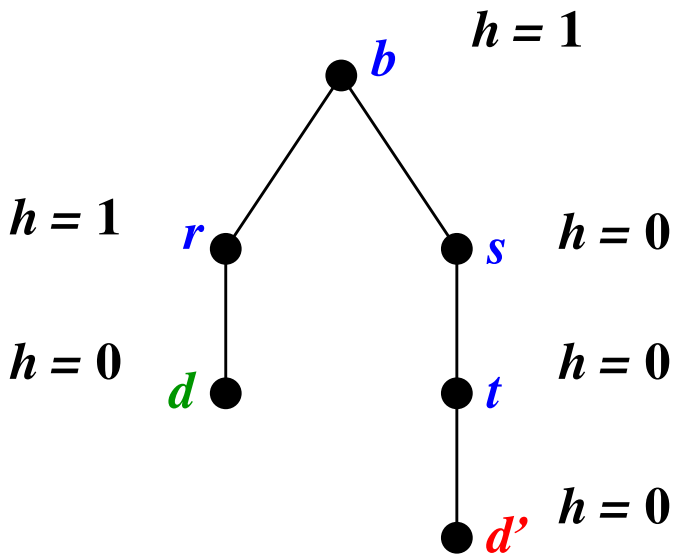
Het algoritme zal de toestand n_i voor expansie kiezen en niet toestand d'. Een tegenspraak volgt.

Het moment van testen

Een A*-zoekalgoritme is alleen geoorloofd als pas bij expansie van een toestand getest wordt of het een doeltoestand is !

Voorbeeld

Beschouw

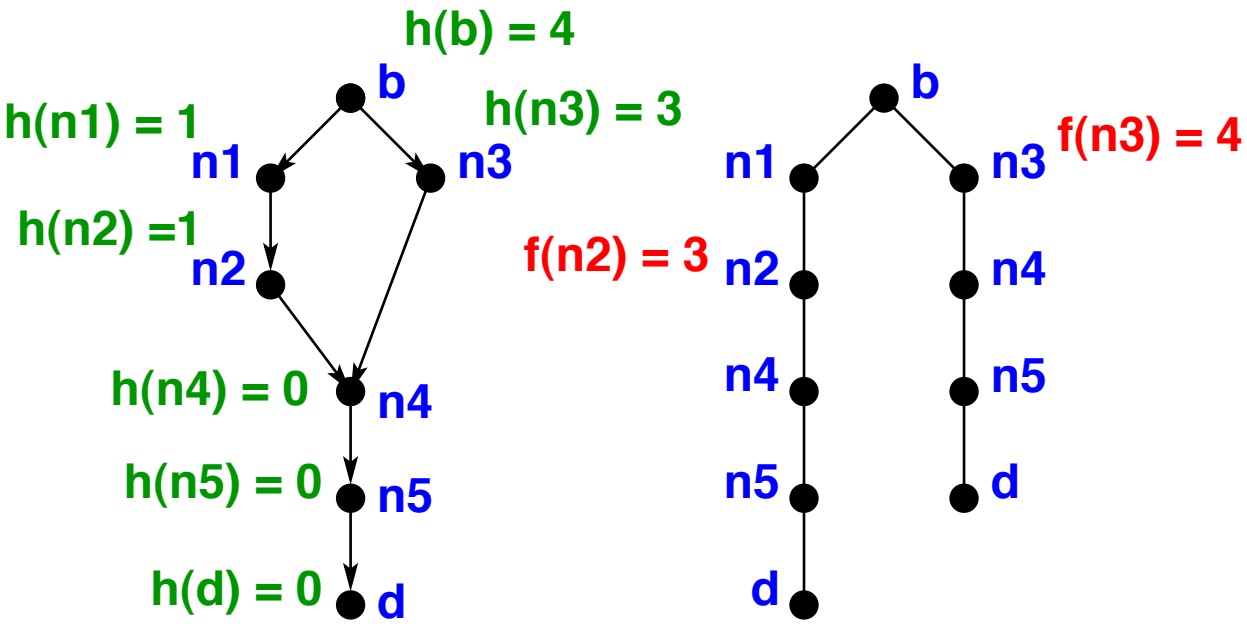


Als een toestand bij generatie al getest wordt, kan het A*-zoekalgoritme de toestanden in de volgende volgorde onderzoeken:

$$b, r, s, t, d'$$

Een optimalisatie — inleiding

De zoekboom voor een gegeven begintoestand wordt door een zoekalgoritme dynamisch gegenereerd uit de zoekgraaf van het probleem:



- een toestand kan meer dan éénmaal gegenereerd, onderzocht en geëxpandeerd worden;
- het eerstgevonden pad naar een toestand is niet noodzakelijk optimaal.

Een optimalisatie

Een A*-zoekalgoritme vindt altijd een pad van minimale lengte naar een doeltoestand, maar

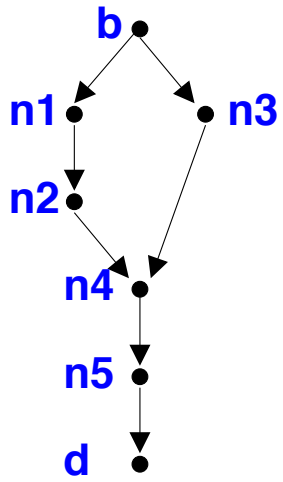
- het algoritme kan onnodig veel tijd vergen;
- het algoritme kan onnodig veel ruimte vergen.

Het A*-zoekalgoritme wordt uit efficiëntie-overwegingen geoptimaliseerd:

- elke toestand wordt ten hoogste eenmaal in de dynamische zoekboom opgenomen;
- op elk moment is het pad in de zoekboom naar een toestand het kortste tot dan toe gevonden pad naar die toestand.

Een voorbeeld

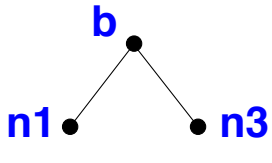
Beschouw de zoekgraaf voor een probleem met begintoestand b en doeltoestand d:



h(b)=4
h(n1)=1
h(n2)=1
h(n3)=3
h(n4)=0
h(n5)=0
h(d)=0

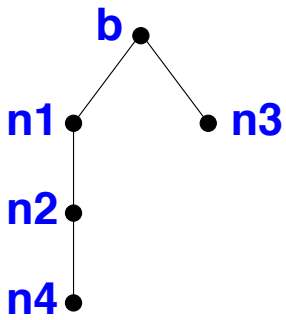
Het A*-algoritme met de gegeven heuristische functie h genereert de zoekboom als volgt:

- na initialisatie wordt toestand b geëxpandeerd:



f(b)=0+4=4
f(n1)=1+1=2
f(n3)=1+3=4

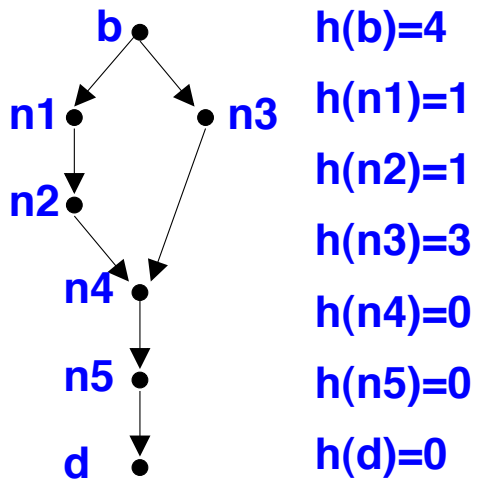
- achtereenvolgens worden n_1 en n_2 geëxpandeerd:



f(b)=4
f(n1)=2
f(n2)=3
f(n3)=4
f(n4)=3

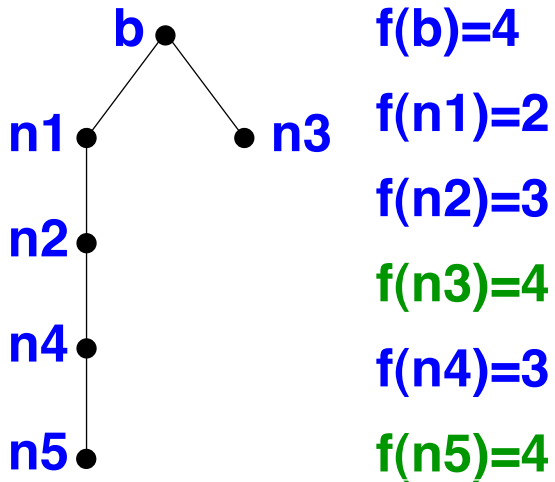
Een voorbeeld — vervolg

Beschouw nogmaals de zoekgraaf voor een probleem met begintoestand b en doeltoestand d:



Het A*-algoritme verloopt verder als volgt:

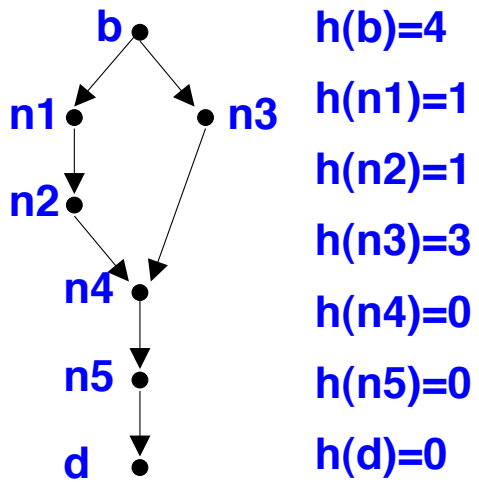
- na expansie van n_4 resulteert de volgende boom:



- bij expansie van n_3 detecteert het algoritme dat n_4 al eerder is gegenereerd en dat het nieuw gevonden pad korter is — het algoritme past de zoekboom aan;

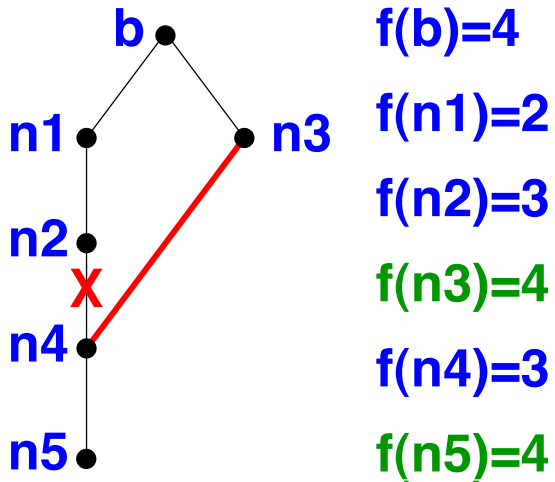
Een voorbeeld — vervolg

Beschouw nogmaals de zoekgraaf voor een probleem met begintoestand b en doeltoestand d:



Het A*-algoritme verloopt verder als volgt:

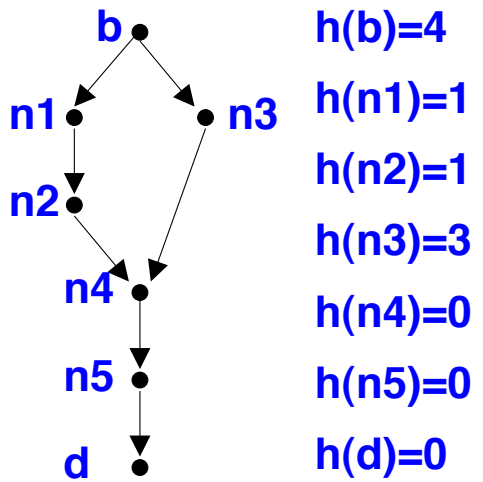
- na expansie van n_4 resulteert de volgende boom:



- bij expansie van n_3 detecteert het algoritme dat n_4 al eerder is gegenereerd en dat het nieuw gevonden pad korter is — het algoritme past de zoekboom aan;

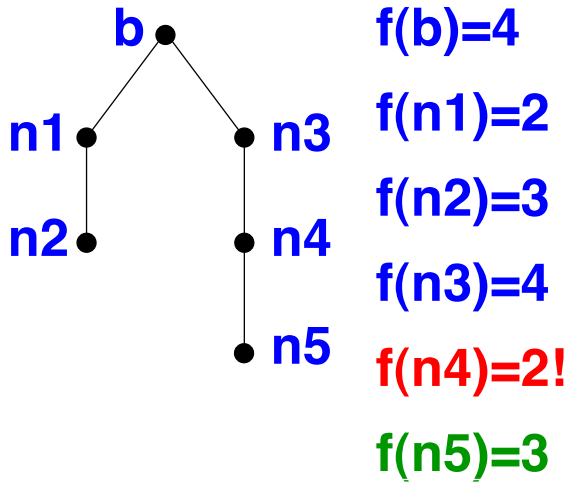
Een voorbeeld — vervolg

Beschouw nogmaals de zoekgraaf voor een probleem met begintoestand b en doeltoestand d:



Het A*-algoritme verloopt verder als volgt:

- na expansie van n_3 resulteert de volgende aangepaste boom:



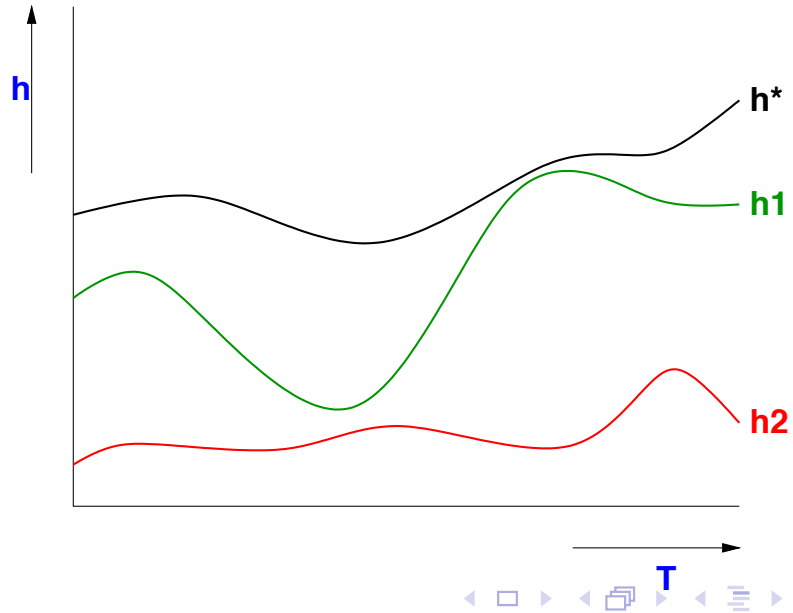
- toestand n_5 is nu de enige toestand in de zoekboom die voor expansie in aanmerking komt . . .

De hoeveelheid informatie

Definitie

Zij P een zoekprobleem met toestandsruimte T. A_i^*, $i = 1, 2$, zijn A^*-algoritmen met de evaluatiefuncties $f_i(t) = g(t) + h_i(t)$ op T:

- A_1^* heet meer geïnformeerd dan A_2^* als voor elke toestand $t \in T$ geldt dat $h_1(t) \geq h_2(t)$;
- A_1^* heet strikt meer geïnformeerd dan A_2^* als $h_1(t) > h_2(t)$ voor elke $t \in T, n \notin D$.



Een voorbeeld

Beschouw de 8-puzzel en de evaluatiefuncties $f_i(n) = g(n) + h_i(n)$, $i = 1, 2$, voor elke toestand t van de puzzel, met

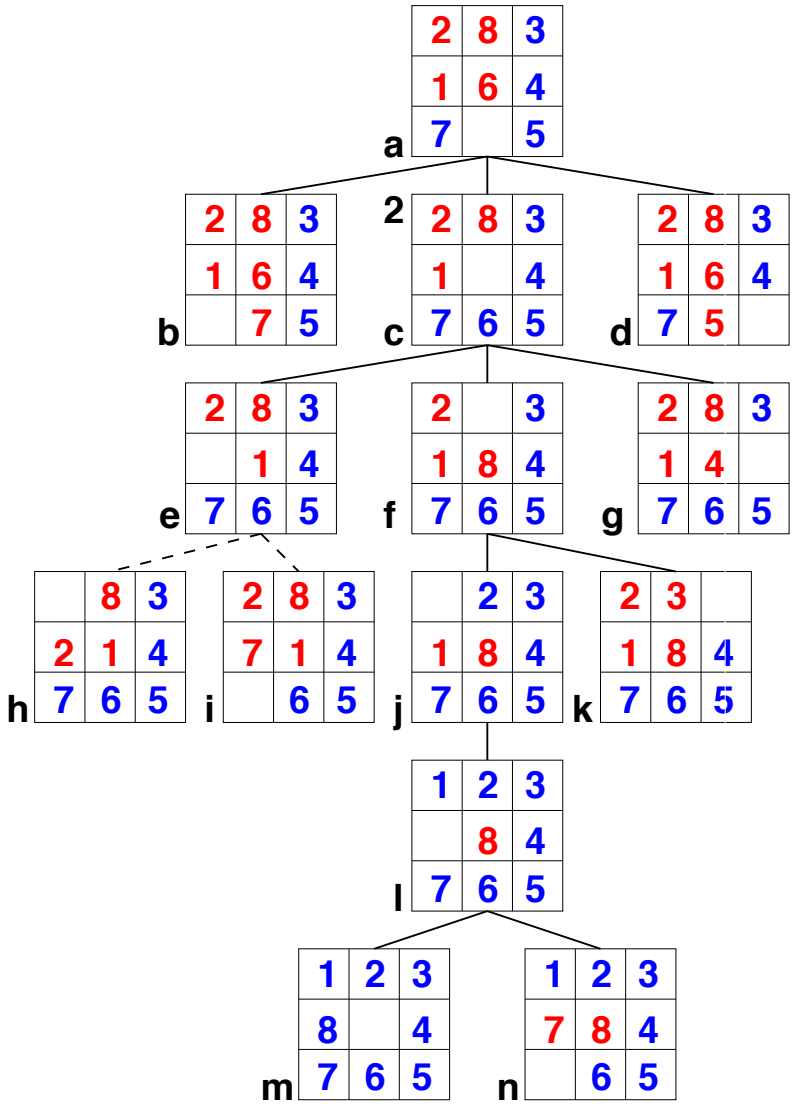
$h_1(t)$ = het aantal incorrect geplaatste tegels in t

$h_2(n)$= de som van de minimale afstanden van de tegels in t tot hun doelpositie

Het A*-zoekalgoritme A_2^* met de evaluatiefunctie f_2 is meer geïnformeerd dan het A*-zoekalgoritme A_1^* met de functie f_1.

Een voorbeeld — vervolg

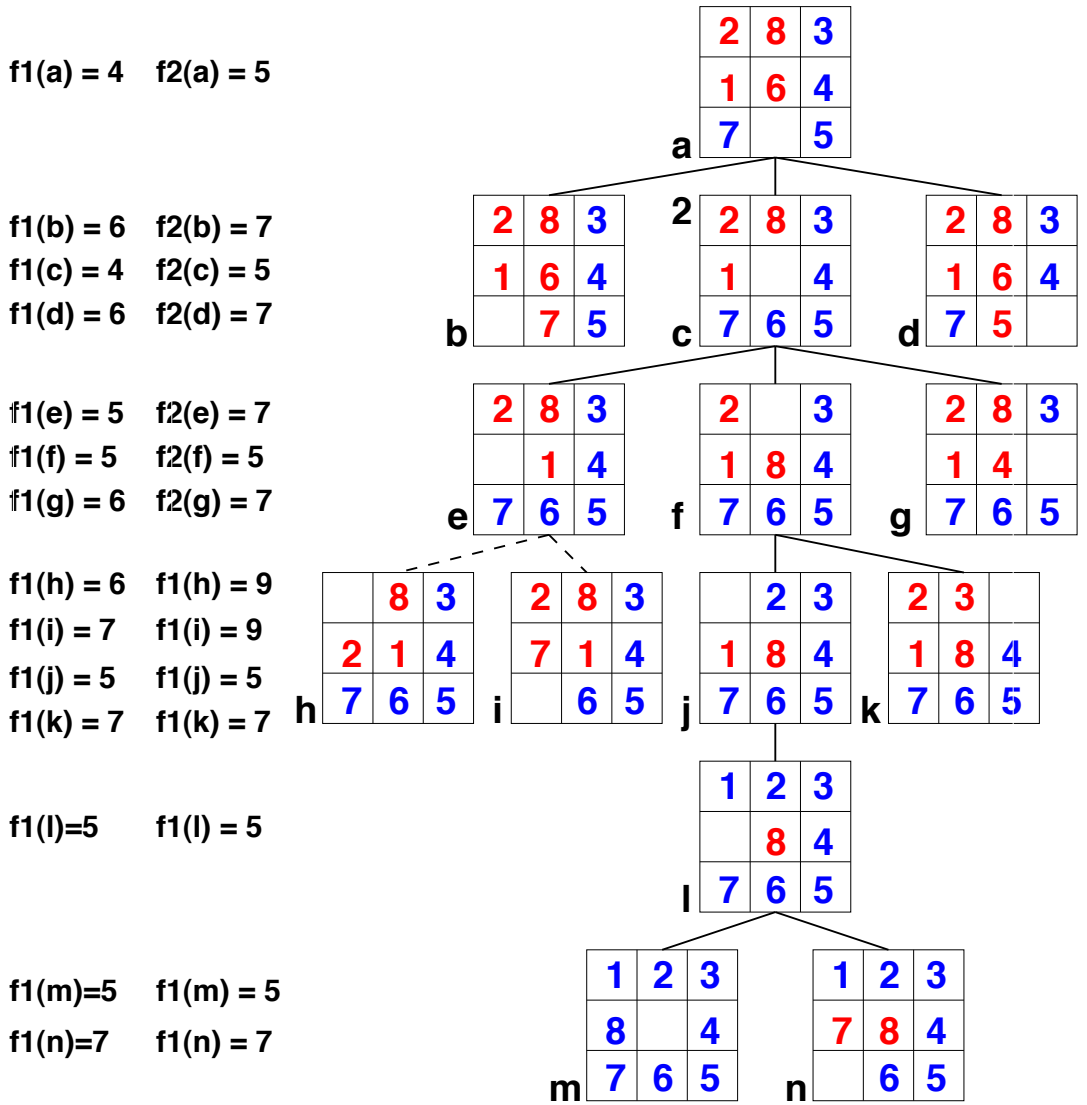
Beschouw de 8-puzzel als voorheen. Beschouw voorts de dynamische zoekboom die door het A*-zoekalgoritme A_1^* met f_1 werd gegenereerd:



Het A*-zoekalgoritme A_2^* met de functie f_2 zal de toestand e niet expanderen.

Een voorbeeld — vervolg

Beschouw de 8-puzzel als voorheen. Beschouw voorts de dynamische zoekboom die door het A*-zoekalgoritme A_1^* met f_1 werd gegenereerd:



Het A*-zoekalgoritme A_2^* met de functie f_2 zal de toestand e niet expanderen.

De kracht van informatie — inleiding

Beschouw nogmaals de 8-puzzel en de A*-zoekalgoritmen A_1^* en A_2^*. De gemiddelde aantallen toestanden die door de twee algoritmen worden gegenereerd voor verschillende oplossingsdiepten zijn:

diepte	A_1^*	A_2^*	*iteratief verdiepen*
2	6	6	10
4	13	12	112
6	20	18	680
8	39	25	6384
10	93	39	47127
12	227	73	364404
14	539	113	3473941

Een eigenschap

Lemma

Een A*-zoekalgoritme met evaluatiefunctie f expandeert in een dynamische zoekboom alleen toestanden t met $f(t) \leq f^*(b)$.

Bewijsschets

Stel dat het algoritme een toestand m expandeert met

$$f(m) > f^*(b)$$

Zij nu $b = n_1, \ldots, n_q = d$ een optimaal pad in de zoekboom en zij n_i de toestand daarop met $n_i \in L$. Voor n_i geldt dat

$$f(n_i) \leq f^*(n_i) = f^*(b)$$

Omdat $f(n_i) < f(m)$ zal het algoritme n_i voor expansie selecteren en niet m. Een tegenspraak volgt. $\square$

De effectieve vertakkingsfactor

Definitie

Zij P een zoekprobleem. Zij A_0 een zoekalgoritme en zij S de dynamische zoekboom die door A_0 voor P wordt gegenereerd. De effectieve vertakkingsfactor van A_0 voor P, notatie: $\beta(\mathrm{A}_0)$, is gedefinieerd als

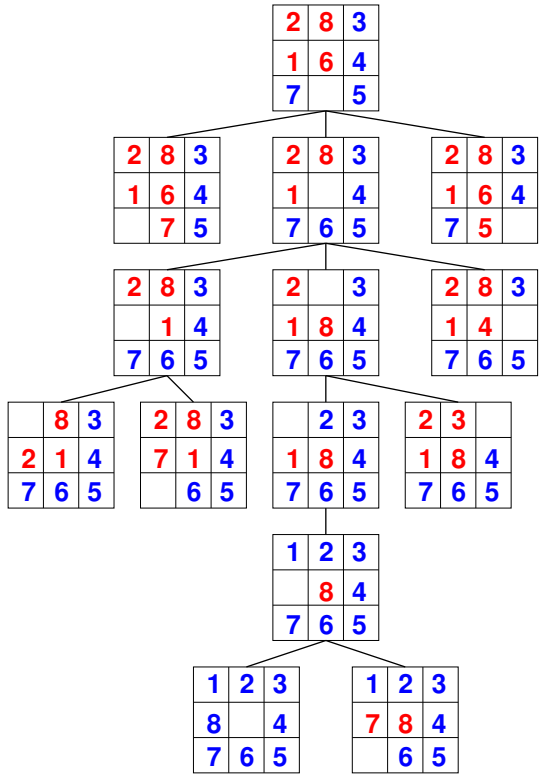
$$n = \sum_{i=0}^{l} \beta(\mathrm{A}_0)^i$$

waarin

n = het aantal toestanden in de dynamische zoekboom S;
l = de lengte van het gevonden pad in S van de begintoestand naar een doeltoestand.

Een voorbeeld

Beschouw de 8-puzzel. Beschouw voorts het A*-algoritme A_0 met de heuristische functie die het aantal incorrect geplaatste tegels in beschouwing neemt:



De effectieve vertakkingsfactor van A_0 volgt uit:

$$1 + \beta(A_0) + \beta(A_0)^2 + \beta(A_0)^3 + \beta(A_0)^4 + \beta(A_0)^5 = \frac{\beta(A_0)^6 - 1}{\beta(A_0) - 1} = 14$$

zodat $\beta(A_0) \approx 1.35$.

Een voorbeeld — vervolg

Beschouw nogmaals de 8-puzzel en de A*-zoekalgoritmen A_1^* en A_2^*. De gemiddelde effectieve vertakkingsfactor voor verschillende oplossingsdiepten is:

diepte	A_1^*	A_2^*	*iteratief verdiepen*
2	1.79	1.79	2.45
4	1.48	1.45	2.87
6	1.34	1.30	2.73
8	1.33	1.24	2.80
10	1.38	1.22	2.79
12	1.42	1.24	2.78
14	1.44	1.23	2.83

De doordringingskracht

Definitie

Zij P een zoekprobleem. Zij A_0 een zoekalgoritme en zij S de dynamische zoekboom die door A_0 voor P wordt gegenereerd. De doordringingskracht van A_0 voor P, notatie: $\kappa(\mathrm{A}_0)$, is gedefinieerd als

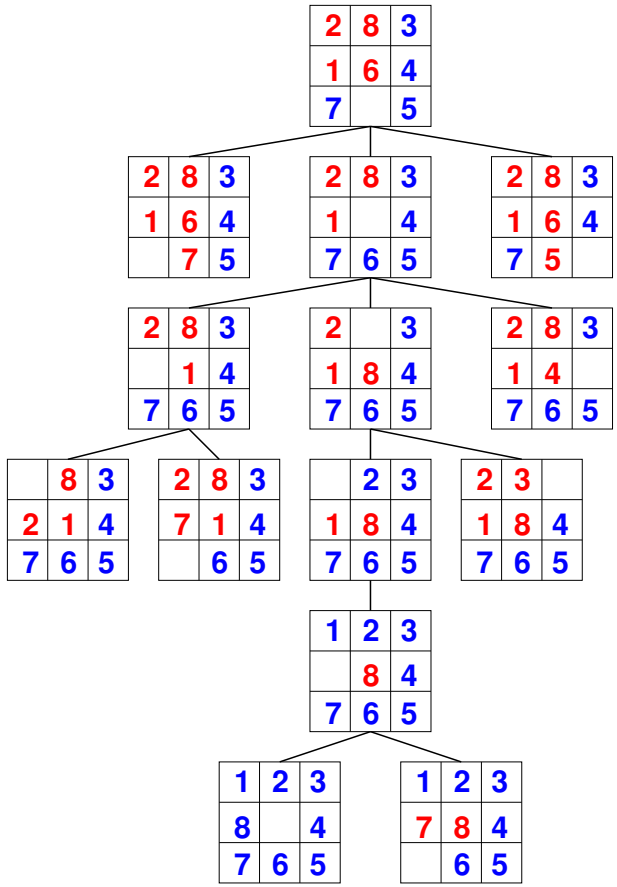
$$\kappa(\mathrm{A}_0) = \frac{l}{n-1}$$

waarin

n = het aantal toestanden in de dynamische zoekboom S;
l = de lengte van het gevonden pad in S van de begintoestand naar een doeltoestand.

Een voorbeeld

Beschouw de 8-puzzel. Beschouw voorts het A*-algoritme A_0 met de heuristische functie die het aantal incorrect geplaatste tegels in beschouwing neemt:



De doordringingskracht van A_0 is

$$\kappa(A_0) = \frac{5}{14-1} = \frac{5}{13} = 0.385$$

Een voorbeeld — vervolg

Beschouw nogmaals de 8-puzzel en de A*-zoekalgoritmen A_1^* en A_2^*. De gemiddelde doordringingskracht voor verschillende oplossingsdiepten is:

diepte	A_1^*	A_2^*	*iteratief verdiepen*
2	0.400	0.400	0.222
4	0.333	0.364	0.036
6	0.316	0.353	0.009
8	0.211	0.333	0.001
10	0.109	0.263	–
12	0.053	0.167	–
14	0.026	0.125	–

De relatie tussen $\kappa(A_0)$ en $\beta(A_0)$

De doordringingskracht en de effectieve vertakkingsfactor drukken de effectiviteit van een zoekalgoritme uit in een enkel getal.

Zij P, A_0, S, l en n als voorheen. De doordringingskracht van A_0 voor P is

$$\kappa(A_0) = \frac{l}{n-1}$$

en de effectieve vertakkingsfactor van A_0 voor P is

$$n = \sum_{i=0}^{l} (\beta(A_0))^i = \frac{\beta(A_0) \cdot ((\beta(A_0))^l - 1)}{\beta(A_0) - 1} + 1$$

Hieruit volgt dat

$$\kappa(A_0) = \frac{l \cdot (\beta(A_0) - 1)}{\beta(A_0) \cdot (\beta(A_0)^l - 1)}$$

Het ontwerpen van heuristische functies

Bij het ontwerpen van een heuristische functie h voor een gegeven zoekprobleem worden de volgende factoren tegen elkaar afgewogen:

- de geoorloofdheid van de functie h;
- de doordringingskracht van een zoekalgoritme dat van h gebruik maakt;
- de computationele kosten van de berekening van functiewaarden van h.

Enkele richtlijnen

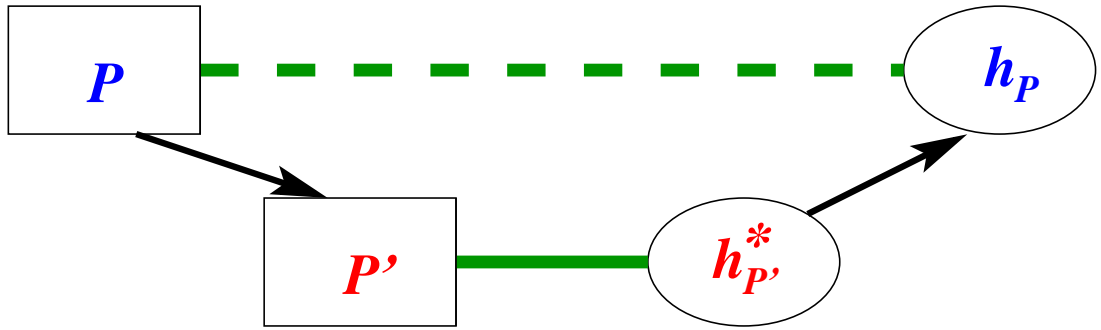
Het ontwerpen van een heuristische functie is een creatief proces. Enkele richtlijnen zijn:

- het gebruik van een variantprobleem;
- het combineren van enkele bestaande heuristische functies.

Variantproblemen — inleiding

Het gebruik van een variantprobleem voor een zoekprobleem P komt neer op:

- construeer voor P een eenvoudiger op te lossen verwant zoekprobleem P';
- gebruik de werkelijke afstandsfunctie $h^*_{P'}$ voor P' als heuristische functie h_P voor P.



Een variantprobleem

Definitie

Beschouw een zoekprobleem $P = (T, B, D, O)$. Een variantprobleem voor P is een zoekprobleem $P' = (T', B', D', O')$ met

- $T \subseteq T'$;
- $B' = B$;
- $D' = D$;
- voor elke $o \in O$ geldt dat er een $o' \in O'$ is met $o \subseteq o'$.

Het variantprobleem is verkregen door één of meer restricties op de operatoren van het oorspronkelijke zoekprobleem te laten vallen.

Een voorbeeld

Beschouw de 8-puzzel. De operatoren van de puzzel beschrijven de volgende spelregel:

- een tegel kan van positie A naar positie B verplaatst worden als A aan B grenst en B leeg is.

Mogelijke variantproblemen voor de 8-puzzel hebben de volgende spelregel:

- een tegel kan van positie A naar positie B verplaatst worden als A aan B grenst;
- een tegel kan van positie A naar positie B verplaatst worden als B leeg is;
- een tegel kan van positie A naar positie B verplaatst worden.

Een voorbeeld — vervolg

Beschouw de variant van de 8-puzzel met de volgende spelregel:

- een tegel kan van positie A naar positie B verplaatst worden.

De afstandsfunctie h^* van het variantprobleem is

$$h^*(t) = \quad \text{het aantal incorrect geplaatste tegels in } t$$

voor elke toestand t.

Een eigenschap

Beschouw een zoekprobleem P en een variantprobleem P' voor P. Dan geldt:

- elke oplossing van P is een oplossing van P';
- de lengte van de optimale oplossing van P' is kleiner dan of gelijk aan de lengte van de optimale oplossing van P.

De werkelijke afstandsfunctie $h^*_{P'}$ van P' is dus een geoorloofde heuristische functie voor P.

Het combineren van functies

Beschouw een zoekprobleem P en de geoorloofde heuristische functies $h_1, \ldots, h_m$, $m \geq 1$, voor P. Zij nu h de functie met

$$h(t) = \max\{h_i(t) \mid i = 1, \ldots, m\}$$

voor elke toestand t van P. Dan geldt:

- de functie h is een geoorloofde heuristische functie voor P;
- een zoekalgoritme met h is meer geïnformeerd dan een (vergelijkbaar) zoekalgoritme met h_i, $i = 1, \ldots, m$.