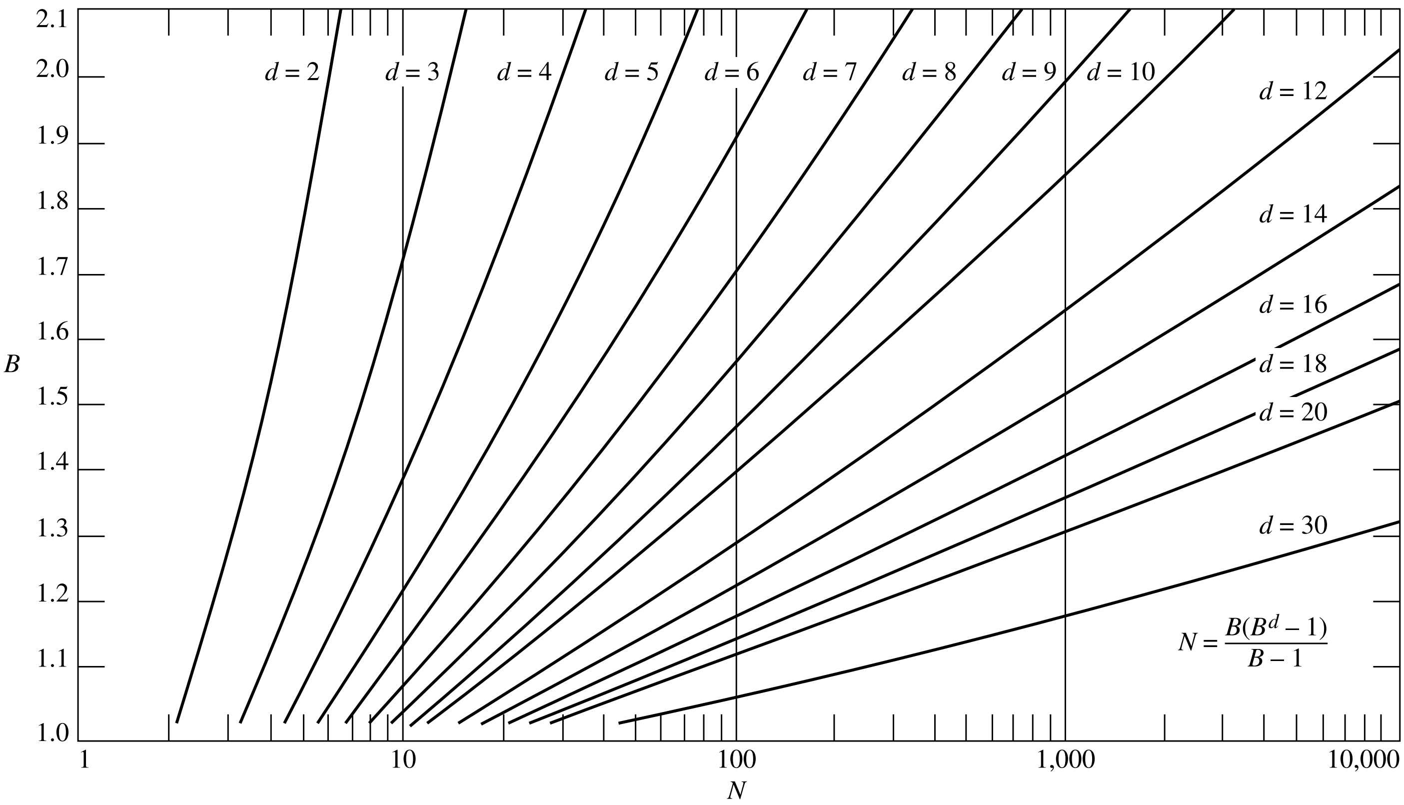


Kuva 35: $B \approx b^*$ vs. N eri arvoilla d [Nil98, kuva 9.11].

Heuristiikka h_2 *dominoi* heuristiikkaa h_1, koska (niiden määritelmistä nähdään, että)

$$h_2(n) \geq h_1(n) \text{ kaikilla } n.$$

Silloin on *aina* parempi käyttää heuristiikkaa h_2 kuin h_1: **A*** laajentaa aina *kaikki* solmut n joilla

$$f(n) < f^* \text{ mikä on yhtälön (9) nojalla}$$
$$h(n) < f^* - g(n).$$

Siis h_1 laajentaa aina *vähintään samat solmut* kuin h_2.

Jos on kaksi heuristiikkaa h' ja h'' joista kumpikaan ei dominoi toista, voidaan valita

$$h(n) = \max(h'(n), h''(n))$$

ja saadaan uusi heuristiikka, joka dominoi kumpaakin. Voidaan siis kehittää ongelmalle erilaisia heuristiikkoja, ja valita niiden antamista arvioista aina suurin.

Heuristiikka olkoon siis optimistinen, mutta vain vähän!

Heuristiikat h_1 ja h_2 saatiin *rentouttamalla (relax)* alkuperäisen ongelman vaatimusta “tiili voi siirtyä paikasta A paikkaan B jos B on tyhjä ja A sen naapuri” *helpompaan muotoon*

1. "tiili A voi siirtyä *aina mihin tahansa* paikkaan B" tai
2. "tiili A voi siirtyä *aina naapuripaikkaansa* B".

Yksi tapa keksiä heuristiikkoja onkin rentouttaa ongelmaa unohtamalla osa sen vaatimuksista ja kehittämällä mitta tämän helpotetun ongelman tapausten etäisyydelle ratkaisusta.

Sovellusalueen *taitajat* saattavat myös tarjota heuristiikkoja. Esimerkiksi korttipelin Bridge teoreetikot ovat kehittäneet pisteytyssääntöjä käden pelivoiman arviointiin:

- Kuvakortti K antaa $K - 10$ pistettä.
- Maa, josta ei ole lainkaan kortteja antaa 3 pistettä valttipelissä, mutta ei valtittomassa.
- Maa, jossa on vain yksi kortti antaa 2 pistettä valttipelissä, mutta sen kortin kuvapisteitä ei enää saa laskea ellei se ole ässä.

⋮

Tällaiset säännöt poimivat ongelman keskeisiä *piirteitä (features)*, joiden arvojen perusteella voi suhteellisen luotettavasti päätellä, onko tämä tilanne hyvä vai huono. Bridgessä tällaiset säännöt on johdettu seuraavasti:

- Onko omassa kädessäni ainakin $\frac{1}{3}$ pakan kaikista pisteistä?

- Onkohan partnerini käteen päätynyt ainakin $\frac{1}{3}$ pakan jäljellä olevista pisteistä?

  Tämä selvitetään peliin kuuluvassa tarjouskilpailussa.

- Silloin meillä on yhteensä ainakin

$$\frac{1}{3} + \frac{1}{3} \cdot \frac{2}{3} = \frac{5}{9} > \frac{1}{2}$$

  pakan kaikista pisteistä.

  Silloin me olemme vahvemmilla kuin vastustajaparimme, joten meidän kannattaa ottaa pelissä aloite.

Joskus tärkeitä piirteitä voidaan kyllä valita, mutta niiden tarkkojen numeeristen painoarvojen määrääminen on vaikeampaa: minkä arvoinen on shakkipelin sotilas — ja miten sen arvo kehittyy pelin myötä?

Numeroarvoja voidaan yrittää päätellä *tilastotieteen* tai *koneoppimisen* [RN95, osa VI] keinoin aineistona aiemmat saman tehtävän ratkaisut. Näitä saadaan "historiasta" (muistiinmerkityt "hyvät" shakkipelit) tai tietokonesimulaatioilla (pannaan tietokone pelaamaan itseään vastaan erilaisia loppupelimuunnelmia ja katsotaan tuloksia).

Tilastot tai historia eivät kuitenkaan takaa arvioiden luvallisuutta.

3.2.4. Muistin säästäminen algoritmissa A*

Luvussa 3.2.2 todettiin, että optimikäyrälle f^* jää yleensä eksponentiaalisen monta solmua. Ne kaikki ovat etsintäpuun lehtiä, joten algoritmin **A*** olisi pidettävä ne muistissa. Tämä on taas käytännössä hyvin raskasta.

Tutustutaan siis seuraavaksi esimerkein (yksityiskohdat sivuuttaen) kahteen tekniikkaan, joilla voi vähentää algoritmin **A*** muistivaatimuksia:

- Jos sovelletaan luvun 3.1.5 iteratiivisen syventämisen ideaa, niin saadaan algoritmi **IDA* (Iterative Deepening A*)**.
- Algoritmi **IDA*** säästää kuitenkin muistia jo silloin kun sitä sitä olisi vielä jäljellä.

  Tehokkaampaa on alkaa säästötoimet vasta kun muisti todellakin loppuu.

  Näin toimii algoritmi **SMA* (Simplified Memory-Bounded A*)**.
- Algoritmissa **SMA*** käytetään sellaista tekniikkaa, jolla voidaan parantaa luvun 3.2 yleistä "paras ensin" -etsintää.

  Näin saadaan algoritmi **RBFS (Recursive Best-First Search)** [Nil98, luku 9.2.5].

  Tutustutaan siis tähän tekniikkaan ja algoritmiin ennen algoritmia **SMA***.

Jokaisessa unohdetaan työmuistin loppuessa sellaisia solmuja, jotka eivät nyt näytä lupaavilta, mutta kuitenkin siten, että niihin voidaan myöhemmin palata.

IDA*

Kun iteratiivinen syventäminen tapahtuu aina seuraavalle korkeuskäyrälle f (kuvassa 33) seuraavan syvyystason sijasta, saadaan seuraava algoritmi:

```
function IDA*(problem) returns a solution sequence
   inputs: problem, a problem
   static: f-limit, the current f-COST limit
           root, a node

   root ← MAKE-NODE(INITIAL-STATE[problem])
   f-limit ← f-COST(root)
   loop do
       solution, f-limit ← DFS-CONTOUR(root, f-limit)
       if solution is non-null then return solution
       if f-limit = ∞ then return failure; end
---------------------------------------------------------------------
function DFS-CONTOUR(node, f-limit) returns a solution sequence and a new f-COST limit
   inputs: node, a node
           f-limit, the current f-COST limit
   static: next-f, the f-COST limit for the next contour, initially ∞

   if f-COST[node] > f-limit then return null, f-COST[node]
   if GOAL-TEST[problem](STATE[node]) then return node, f-limit
   for each node s in SUCCESSORS(node) do
       solution, new-f ← DFS-CONTOUR(s, f-limit)
       if solution is non-null then return solution, f-limit
       next-f ← MIN(next-f, new-f); end
   return null, next-f
```

Algoritmi 5: Iteratiivisesti syventävä **A*** (**IDA***) [RN95, kuva 4.8].

Sen ominaisuudet ovat muuten samanlaiset kuin algoritmin **A***, mutta syvyyssuuntaisena sen tilavaatimus on enää $O(\frac{bf^*}{\delta})$ solmua.

Useimmissa tapauksissa syvyyshaun $O(bd)$ solmua onkin hyvä likiarvo tilavaatimukselle.

Algoritmin **IDA*** aikavaatimus riippuu vahvasti siitä *montako eri arvoa funktio f saa ennen kuin saavutetaan tavoitearvo f^**. Tämä näet kertoo iterointien lukumäärän.

Esimerkiksi luvun 3.2.3 heuristiikalla h_2 on vain muutama mahdollinen arvo, jolloin tyypillisesti f muuttuu vain muutaman kullakin ratkaisupolulla. Silloin myös **IDA*** iteroi vain muutaman kerran.

Pahimmillaan f saattaa antaa eri arvon jokaiselle etsintäpuun solmulle, jolloin jos **A*** käsittelisi N solmua, niin **IDA*** käsittelisikin

$$1 + 2 + 3 + \cdots + N = \frac{N(N+1)}{2}$$

solmua.

Jos N oli niin suuri, ettei **A*** mahtunut toimimaan koneen muistissa, niin $O(N^2)$ on vuorostaan liian kauan!

Tämä voidaan korjata syventämällä etsintää joka kierroksella vähintään ennalta valitun vakion $\epsilon > 0$ verran.

Tällöin korjatulle algoritmille kierroksia tulee

$$O\left(\left\lceil \frac{f^*}{\epsilon} \right\rceil\right)$$

kappaletta.

Toisaalta korjattu algoritmi saattaa vastata minkä tahansa niistä ratkaisuista, joiden hinta on $\leq f^* + \epsilon$.

Toisin sanoen, tällöin korjattu algoritmi hyväksyy vakion ϵ verran epäoptimaalisuutta.

Tällaista "hallitusti leväperäistä" algoritmia kutsutaan ϵ-*luvalliseksi*.

RBFS

Unohdetaan hetkeksi luvun 3.2.2 vaatimus, että heuristikan h pitäisi olla luvallinen.

Silloin etsintä voi tuottaa *iloisia yllätyksiä:* jokin alipuu saattaakin tarkemmin tutkittaessa osoittautua lupaavammaksi — pienempiä f-arvoja omaavaksi — kuin h sen juuressa epäili.

Välitetään nämä hyvät uutiset puussa ylöspäin *peruutusarvoina (backed-up values):*

- Kun solmu n laajennetaan, kunkin syntyvän lehden m_i alkuperäinen peruutusarvo on alkuperäisen ohjausfunktion arvo $f(m_i)$.
- Sisäsolmun peruutusarvo on aina sen *lapsisolmujen peruutusarvoista pienin*.
  Sisäsolmussa on siis tieto sen *parhaasta jälkeläisestä* eikä enää siitä itsestään.
- Algoritmina:
  Kun n on laajennettu, korvataan sen vanha peruutusarvo sen lasten peruutusarvojen minimillä.
  Siirrytään solmun n isäsolmuun, ja tehdään sielläkin samoin.

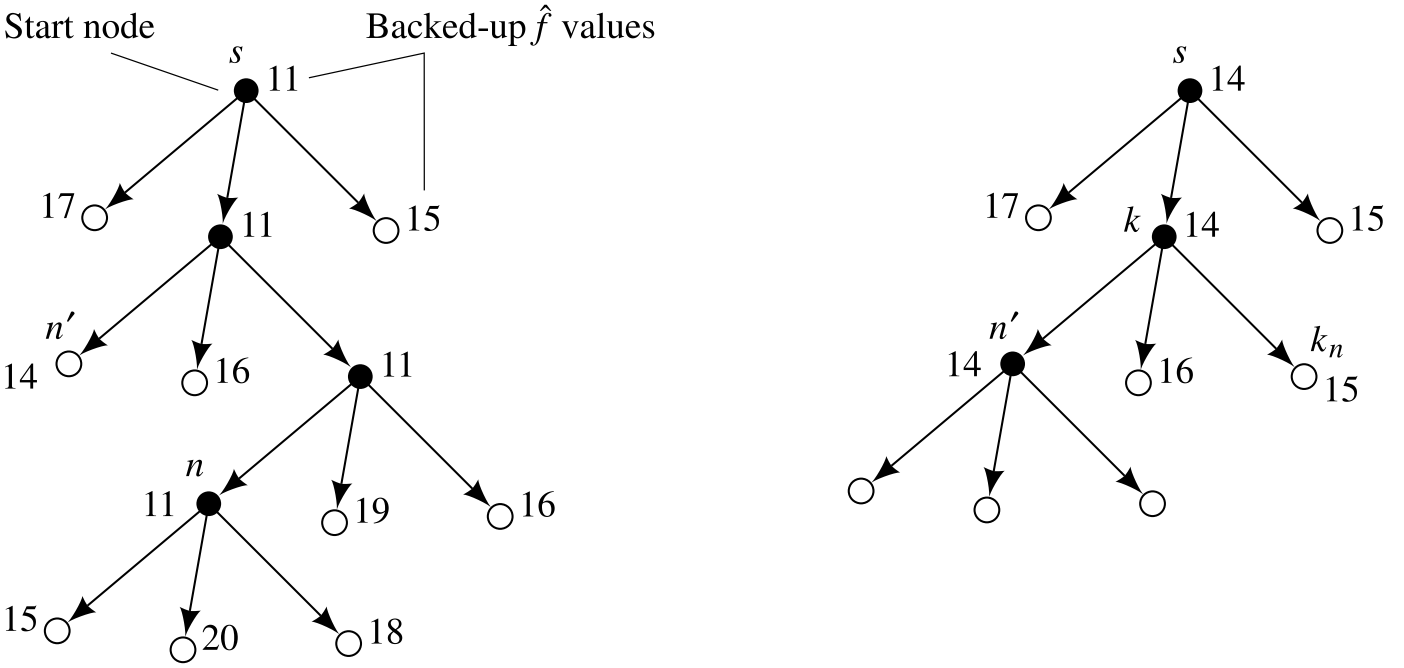
Algoritmin **RBFS** muistinsäästöstrategia on *pitää muistissa sisäsolmuista vain ne, joiden kautta on saatu parhaat uutiset*.

Algoritmina:

- Jos peruutusarvoja päivitettäessä päädytään sellaiseen sisäsolmuun k_n, jonka isäsolmu k saa jostakin toisesta lapsisolmustaan n' vielä parempia uutisia kuin tästä solmusta k_n saadut uutiset u, niin:
- Lopetetaan peruutusarvojen päivitys tähän.
  Solmun n' paremmat uutiset on jo aiemmin kerrottu koko jäljellä olevalle polulle.

- Tehdään sisäsolmusta k_n taas lehti poistamalla siitä alkanut alipuu T.
- Lehden k_n peruutusarvoksi jää se peruutusarvo u joka siihen puusta T ehti nousta.
- Seuraavaksi laajennetaan lupaavin lehtisolmu n'.

Jos myöhempi etsintä toteaakin, että solmusta n' laajennetun puun T' tuottamat lisäuutiset ovatkin huonompia kuin solmun k_n uutiset u, niin etsintä poistaa puun T' ja palaa jatkamaan etsintää solmusta k_n.



(a) RBFS has just expanded node n but has not yet backed up the $\hat{f}$ values of its successors

(b) $\hat{f}$ values have been backed up, the subtree below k_n has been discarded, and search continues below n'

Kuva 36: Esimerkki algoritmin **RBFS** toiminnasta [Nil98, kuva 9.9].

Algoritmi **RBFS**

- muistaa hieman enemmän
- laajentaa vähemmän

solmuja kuin algoritmi **IDA***.

Toinen tapa päätyä algoritmiin **A*** onkin kysyä "millaisilla h tämä peruutusarvojen välitys on turhaa?"

SMA*

Algoritmin **IDA*** ongelmana on, että kustakin iteraatiosta muistetaan vain yksi luku f-$limit$, vaikka muistia olisi pidempäänkin etsintään.

Tehdään siis algoritmista **A*** sellainen versio **SMA*** **(Simplified Memory-Bounded A*)**, jossa annetaan etukäteen *yläraja muistissa pidettävien solmujen lukumäärälle*.

Jos tämä raja on jo saavutettu, ja muistiin pitäisi lisätä uusi solmu, niin jokin muistissa jo oleva solmu luovuttaa sille oman paikkansa ja poistetaan muistista.

Tällöin tarjotusta muistista otetaan kaikki hyöty irti:

- **SMA*** pitää kaiken sille annetun muistin koko ajan täynnä.

  IDA* tyhjentää muistinsa iteraatioiden välillä.
- **SMA*** laajenee tarvittaessa eliminoimaan toistuvia tiloja luvun 3.1.7 globaalilla menetelmällä 2 siinä määrin kun rajalliseen muistiin mahtuu.

  Syvyyssuuntaisena **IDA*** laajenee vain lokaaliin toistojen poistoon menetelmällä 1.
- **SMA*** on täydellinen, jos muistia on tarpeeksi *matalimpaan (shallowest)* ratkaisuun vievälle polulle.
- **SMA*** on optimaalinen, jos muistia on tarpeeksi matalimpaan *optimaaliseen* ratkaisupolkuun.

  Muuten palautetaan paras ratkaisu joka vielä mahtui kerralla muistiin.

  Samalla voidaan palauttaa varoitus, ettei ratkaisu ehkä ole paras.
- **SMA*** on optimaalisen tehokas kuten **A*** luvussa 3.2.2, jos muistia on riittävästi.

  Ei kuitenkaan vielä tiedetä onko **SMA*** optimaalisen tehokas kaikkien muistirajoitettujen algoritmien joukossa.

```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
     if Queue is empty then return failure
     n ← deepest least-f-cost node in Queue
     if GOAL-TEST(n) then return success
     s ← NEXT-SUCCESSOR(n)
     if s is not a goal and is at maximum depth then
        f(s) ← ∞
     else
        f(s) ← MAX(f(n), g(s)+h(s))
     if all of n's successors have been generated then
        update n's f-cost and those of its ancestors if necessary
     if SUCCESSORS(n) all in memory then remove n from Queue
     if memory is full then
        delete shallowest, highest-f-cost node in Queue
        remove it from its parent's successor list
        insert its parent on Queue if necessary
     insert s on Queue
  end
```

Algoritmi 6: Muistirajoitettu **A*** (**SMA***) [RN95, kuva 4.12].

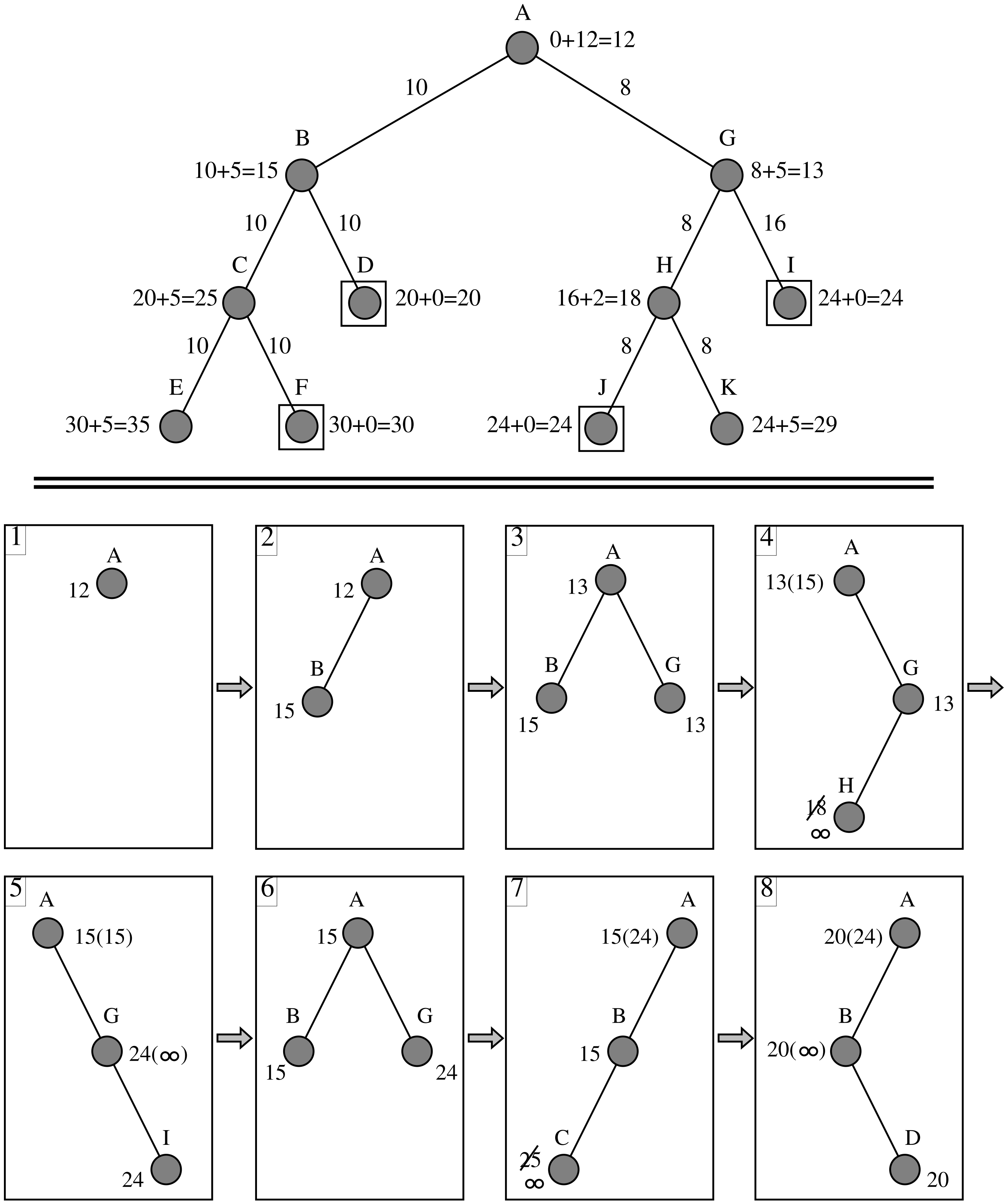
- **SMA*** päivittää solmujensa peruutusarvoja kuten **RBFS**.
  Merkitään solmun n nykyistä peruutusarvoa $f(n)$.
- **SMA*** pitää solmussa n kirjaa myös parhaasta luovutetun lapsen peruutusarvosta.

Algoritmin **SMA*** muistinluovutusstrategia on seuraava:

1. Valitse luovuttajaksi lehtisolmu, jonka f-arvo on suurin.
   Sehän on vähiten lupaava nykyisistä lehdistä.
2. Jos vaihtoehtoja on useita, niin valitse mahdollisimman matala.
   Muistiin yritetään näin mahduttaa mahdollisimman pitkä polku.

Algoritmin **SMA*** solmunlaajennusstrategia on seuraava:

1. Lisää seuraava puuttuva lapsi sille solmulle, jonka f-arvo on pienin.
2. Jos vaihtoehtoja on useita, niin valitse mahdollisimman syvä.



Kuva 37: Algoritmin **SMA*** eteneminen [RN95, kuva 4.12].

Kuvan 37 esimerkki on seuraava:

1. Juuren A ensimmäinen (vasen) lapsi B ilmestyy muistiin.
2. Nyt $f(A) < f(B)$, joten A laajenee toisen kerran ja saa myös jälkimmäisen (oikean) lapsensa G muistiin.

   Koska juuri A on nyt laajennettu kokonaan, asetetaan $f(A) = \min(f(B), f(G)) = 13$.

   Kolmen (3) solmun kokoinen muisti täyttyy.
3. Seuraavaksi laajennetaan G.
4. Koska muisti on täynnä, täytyy strategian nojalla isoveljen B luopua muistipaikastaan ensimmäisen veljenpoikansa H edestä.

   Luopuvan solmun B f-arvo 15 viedään sen isään A (sulkuihin).

   Valitettavasti lisätty H on maksimisyvyydellä, eli yhtä syvällä kuin muistin koko, eli muistiin mahtuu vain polku juuresta A tähän solmuun H.

   Koska H ei ole maalisolmu, asetetaan $f(H) = \cancel{1}8 = \infty$ eli "muisti ei riitä jatkamaan etsintää solmusta H eteenpäin".
5. G valitaan vielä kerran laajennettavaksi, ja muistin luovuttaa vasta lisätty H.

   Muistiin tulee I, joka on jälleen maksimisyvyydellä mutta nyt maalisolmu, eli säilyttää oman arvonsa $f(I) = 24$.

   Samalla solmun G molemmat lapset on nähty, joten asetetaan $f(G) = \min(\infty, f(I)) = 24$.

   Tämä muutos kerrotaan ylöspäin sen isään A, jossa $f(A) = \min(15, f(G)) = 15$ saadaan (suluissa olevasta) luopuneen lapsen B arvosta ja muutoksesta.

   Ratkaisu $A \to G \to I$ olisi nyt muistissa, mutta $f(A) = 15$ vihjaa, että parempaankin saatetaan vielä pystyä.
6. Lähdetään peräämään tätä lupausta laajentamalla juuren A välillä muistista poistunut vasen lapsi B uudelleen.

   Muistin luovuttaa I, ja näin unohtuu äskeinen ratkaisu.

   Solmujen A ja G f-arvot ovat kasvaneet sitten vaiheen 3, joten algoritmi ei ole ikuisessa silmukassa.

Lisäksi solmun A kaikki lapset (B ja G) ovat taas muistissa, joten (suluissa oleva) kirjanpito luopuneiden lasten parhaasta f-arvosta pyyhitään pois.

7. Solmun B vasen lapsi C tuodaan muistiin, jonka luovuttaa G.

   Tämä luopuminen asettaa sen isän A luopuneiden lasten (suluissa olevaksi) pienimmäksi f-arvoksi $f(G) = 24$.

   C on kuitenkin maksimisyvyydellä muttei maalisolmu, joten asetetaankin $f(C) = \cancel{25} = \infty$.

8. Seuraavaksi lisätään solmun B oikea lapsi D muistiin, jonka luovuttaa C.

   Luovutus lähettää arvon $f(C) = \infty$ (sulkuihin) sen isään B, mutta sen isoisässä A on pienempi arvo 24, joka ei muutu.

   D on maksimisyvyydellä mutta maalisolmu, joten se säilyttää oman arvonsa $f(D) = 20$.

   Koska sen isän B molemmat lapset on nyt käsitelty, asetetaan $f(B) = \min(\infty, 20) = 20$.

   Tämä muutos etenee myös sen isoisään A, jossa asetetaan $f(A) = \min(24, 20) = 20$.

9. Lopuksi valitaan laajennettavaksi D, joka on maalisolmu, ja etsintä päättyy vastaukseen $A \to B \to D$, joka onkin parempi kuin vaiheen 5 ratkaisuehdokas.

Esimerkistä nähdään, että jo nyt mutkikas algoritmin listaus 6 ei vielä sisälläkään läheskään kaikkia yksityiskohtia. . .

Jos muistia on edes kohtuullisesti, **SMA*** pystyy ratkomaan paljon mutkikkaampia ongelmia kuin **A***.

Hyvin vaikeissa ongelmissa **SMA*** alkaa hyppelehtiä ratkaisupolulta toiselle, jolloin se tuhlaa aikaa tuomalla samoja solmuja muistiin yhä uudelleen ja uudelleen.

Vaikka aika vs. tila -kysymykset ovatkin vaikeita, näyttää siltä, että jos halutaan optimiratkaisu, niin (ainakin) toista kuluu. . .

Viitteet

[AK83] J.F. Allen and J.A. Koomen. Planning using a temporal world model. In *International Joint Conference on Artificial Intelligence*, pages 741–747, 1983.

[Bod90] M.A. Boden, editor. *The Philosophy of Artificial Intelligence*. Oxford Readings in Philosophy. Oxford University Press, 1990.

[Car97] B. Carpenter. *Type-Logical Semantics*. The MIT Press, 1997.

[Cho97] N. Chomsky. B. F. Skinnerin teos *verbal behaviour*. In P. Raatikainen, editor, *Ajattelu, kieli, merkitys*, pages 279–310. Gaudeamus, 1997. Suom. R. Vilkko.

[CL73] C-L. Chang and R.C-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[CR91] W. Clinger and J. Rees. Revised^4 report on the algorithmic language Scheme, 1991.

[CSC90] P.M. Churchland and P. Smith Churchland. Could a machine think? *Scientific American*, pages 26–31, 1990.

[Dam99] A. Damasio. *The Feeling of What Happens*. William Heinemann, 1999.

[DD88] H.L. Dreyfus and S.E. Dreyfus. Making a mind versus modelling the brain: Artificial intelligence back at a branch-point. *Artificial Intelligence*, 1988.

[Dea98] T.W. Deacon. *The Symbolic Species: The Co-Evolution of Language and the Human Brain*. Penguin Books, 1998.

[MP43] W.S. McCulloch and W.H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–137, 1943. [Bod90, §1].

[Myr74a] L. Myrberg. *Differentiaali- ja integraalilaskenta*, volume 1. Kirjayhtymä, 1974.

[Myr74b] L. Myrberg. *Differentiaali- ja integraalilaskenta*, volume 2. Kirjayhtymä, 1974.

[Nil98] N.J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan-Kaufmann, 1998.

[NKN01] M.K. Nowak, N.L. Komarova, and P. Niyogi. Evolution of universal grammar. *Science*, 291:114–118, 2001.

[Pen89] R. Penrose. *The Emperor's New Mind*. Oxford University Press, 1989.

[Pin94] S. Pinker. *The Language Instinct*. William Morrow and Company, Inc., 1994.

[Pin97] S. Pinker. *How the Mind Works*. W. W. Norton & Company, Inc., 1997.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice-Hall, 1995.

[Saa85] E. Saarinen. *Länsimaisen filosofian historia huipulta huipulle*. Werner Söderström oy, 1985.

[Sea80] J. Searle. Minds, brains, and programs. *The Behavioral and Brain Sciences*, 3:417–424, 1980. [Bod90, §3].

[Sea90] J. Searle. Is the brain's mind a computer program? *Scientific American*, pages 20–25, 1990.

[TE88] T. Tononi and G.M. Edelman. Consciousness and complexity. *Science*, pages 1846–1851, 1988.

[Tur50] A.M. Turing. Computing machinery and intelligence. *Mind*, LIX(2236):433–460, 1950. [Bod90, §2].